**CS1050 – Computer Organizing and Digital Design**

**Nano Processor**

Table of Contents

[Lab Tasks 4](#_Toc199179655)

[Components 8](#_Toc199179656)

[1. Slow Clock 8](#_Toc199179657)

[1.1. Design Source File 8](#_Toc199179658)

[1.2. Elaborate design 9](#_Toc199179659)

[1.3. Simulation File 9](#_Toc199179660)

[1.4. Timing Diagram 11](#_Toc199179661)

[2. Decoder 3 to 8 11](#_Toc199179662)

[2.1. Design Source File 11](#_Toc199179663)

[2.2. Elaborate design 13](#_Toc199179664)

[2.3. Simulation File 13](#_Toc199179665)

[2.4. Timing Diagram 15](#_Toc199179666)

[3. Register 16](#_Toc199179667)

[3.1. Design Source File 16](#_Toc199179668)

[3.2. Elaborate design 17](#_Toc199179669)

[3.3. Simulation File 17](#_Toc199179670)

[3.4. Timing Diagram 20](#_Toc199179671)

[4. Register Bank 20](#_Toc199179672)

[4.1. Design Source File 20](#_Toc199179673)

[4.2. Elaborate design 24](#_Toc199179674)

[4.3. Simulation File 24](#_Toc199179675)

[4.4. Timing Diagram 27](#_Toc199179676)

[5. Multiplexer 8 to 1 4bit 27](#_Toc199179677)

[5.1. Design Source File 27](#_Toc199179678)

[5.2. Elaborate design 29](#_Toc199179679)

[5.3. Simulation File 29](#_Toc199179680)

[5.4. Timing Diagram 32](#_Toc199179681)

[6. Multiplexer 2 to 1 4bit 32](#_Toc199179682)

[6.1. Design Source File 32](#_Toc199179683)

[6.2. Elaborate design 34](#_Toc199179684)

[6.3. Simulation File 34](#_Toc199179685)

[6.4. Timing Diagram 37](#_Toc199179686)

[7. Program Counter 37](#_Toc199179687)

[7.1. Design Source File 37](#_Toc199179688)

[7.2. Elaborate design 39](#_Toc199179689)

[7.3. Simulation File 39](#_Toc199179690)

[7.4. Timing Diagram 42](#_Toc199179691)

[8. Program Rom 42](#_Toc199179692)

[8.1. Design Source File 42](#_Toc199179693)

[8.2. Elaborate design 44](#_Toc199179694)

[8.3. Simulation File 44](#_Toc199179695)

[8.4. Timing Diagram 47](#_Toc199179696)

[9. Adder-Subtractor Unit 4bit 47](#_Toc199179697)

[9.1. Design Source File 47](#_Toc199179698)

[9.2. Elaborate design 49](#_Toc199179699)

[9.3. Simulation File 49](#_Toc199179700)

[9.4. Timing Diagram 52](#_Toc199179701)

[10. Instruction Decoder 52](#_Toc199179702)

[10.1. Design Source File 52](#_Toc199179703)

[10.2. Elaborate design 55](#_Toc199179704)

[10.3. Simulation File 55](#_Toc199179705)

[10.4. Timing Diagram 58](#_Toc199179706)

[11. Nano Processor 58](#_Toc199179707)

[11.1. Design Source File 58](#_Toc199179708)

[11.2. Elaborate design 65](#_Toc199179709)

[11.3. Simulation File 65](#_Toc199179710)

[11.4. Timing Diagram 68](#_Toc199179711)

[12. LUT 16\_7 68](#_Toc199179712)

[12.1. Design Source File 68](#_Toc199179713)

[12.2. Elaborate design 69](#_Toc199179714)

[12.3. Simulation File 70](#_Toc199179715)

[12.4. Timing Diagram 72](#_Toc199179716)

[13. Rom Address Select (3 bit adder and Mux 2 way 3 bit) 72](#_Toc199179717)

[13.1. Design Source File 72](#_Toc199179718)

[13.2. Elaborate design 74](#_Toc199179719)

[13.3. Simulation File 74](#_Toc199179720)

[13.4. Timing Diagram 77](#_Toc199179721)

[Optimization 77](#_Toc199179722)

[Constrain File 79](#_Toc199179723)

[Workload Distribution 86](#_Toc199179724)

[Difficulties we face 87](#_Toc199179725)

[Conclusion 88](#_Toc199179726)

# Lab Tasks

The primary objective of this lab was to design and implement a simple 4-bit *nanoprocessor* capable of executing a predefined set of instructions. The tasks were divided among team members to ensure collaborative development and integration of multiple components. The following tasks were completed as part of this lab:

1. Design of 4-bit Arithmetic Unit

* Developed a 4-bit adder/subtractor using 2’s complement representation.
* Modified the 4-bit Ripple Carry Adder (RCA) from Lab 3 to perform both addition and subtraction based on a control signal.

2. Program Counter and Incrementor

* Implemented a 3-bit Program Counter (PC) using D Flip-Flops with a clear/reset input to allow program restarts.
* Designed a 3-bit adder using a modified 4-bit RCA to increase the PC value on each clock cycle.

3. Multiplexer Design

* Built the following multiplexers using previously developed components:
  + 2-way 3-bit multiplexer
  + 2-way 4-bit multiplexer
  + 8-way 4-bit multiplexer (based on the 8-to-1 multiplexer from Lab 4)
* Explored alternative implementation using tri-state buffers.

4. Register Bank

* Developed a register bank containing eight 4-bit registers (R0–R7).
* Hardcoded R0 to hold the constant value 0.
* Incorporated D Flip-Flops with reset inputs to allow global resetting via a pushbutton.
* Used a 3-to-8 decoder (from Lab 4) for register selection.

5. Program ROM

* Extended the ROM-based LUT from Lab 7 to store machine code instructions.
* Hardcoded the Assembly program into ROM to allow instruction fetching during execution.

6. Instruction Decoder

* Designed the instruction decoder to interpret binary instructions and activate appropriate processor components.
* Ensured precise control signal generation for:
  + Data movement (e.g., MOVI)
  + Arithmetic operations (e.g., ADD, NEG)
  + Conditional jumps (e.g., JZR)
* Used minimal activation logic to prevent conflicting operations during instruction execution.

7. Bus System Integration

* Employed 3-bit, 4-bit, and 12-bit labeled buses to connect major components.
* Streamlined wiring and simplified circuit design using clearly labeled buses such as D(3 downto 0), I(11 downto 0), M(3 downto 0), and R(3 downto 0).

8. Clock and Control Logic

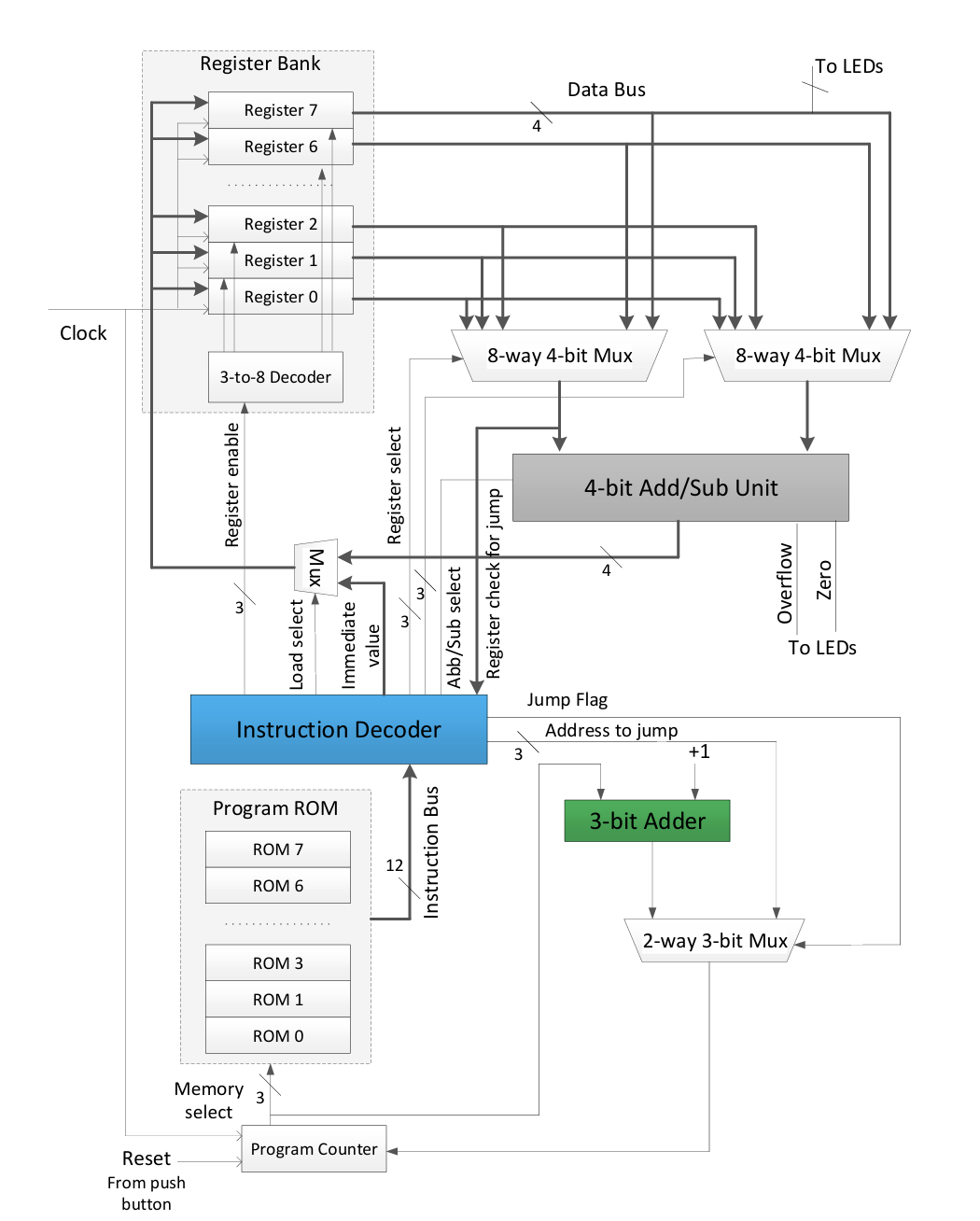
* Integrated a slow clock (2–3 seconds per tick) to observe the execution of each instruction step-by-step.
* Mapped a push button to reset the PC and register bank simultaneously, allowing a fresh start to the program.

9. Team Collaboration

* Tasks were distributed among the five team members to encourage parallel development.
* Regular integration sessions were conducted to ensure seamless merging of independently developed modules.
* Communication and documentation were maintained throughout the development cycle for effective coordination

Instruction Set

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



**High-level diagram of the *nanoprocessor*.**

# Components

## 1. Slow Clock

### 1.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for Slow\_Clk

-- It takes an input clock (Clk\_in) and generates a slower clock (Clk\_out)

entity Slow\_Clk is

Port ( Clk\_in : in STD\_LOGIC; -- Input clock signal

Clk\_out : out STD\_LOGIC); -- Output slower clock signal

end Slow\_Clk;

-- Architecture definition for the behavior of Slow\_Clk

architecture Behavioral of Slow\_Clk is

-- 26-bit counter => 2^26 = 67,108,864 (e.g., for 1 Hz from 50 MHz clock)

signal counter : unsigned(25 downto 0) := (others => '0');

signal clk\_div : STD\_LOGIC := '0';

begin

process(Clk\_in)

begin

if rising\_edge(Clk\_in) then

counter <= counter + 1;

clk\_div <= counter(25); -- Use MSB as divided clock

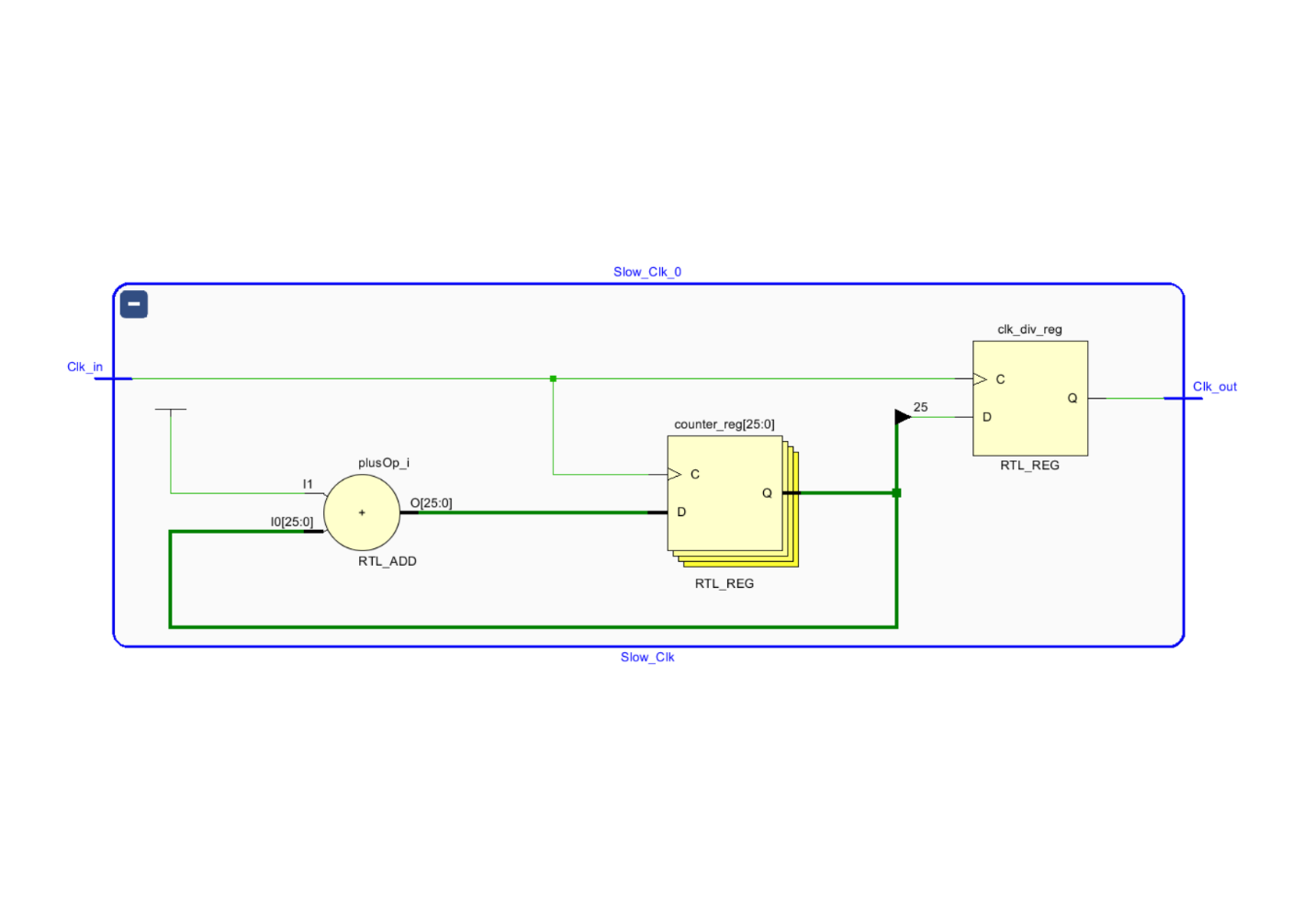
end if;

end process;

Clk\_out <= clk\_div;

end Behavioral;

### 1.2. Elaborate design



### 1.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/26/2025 06:40:10 AM

-- Design Name:

-- Module Name: Sim\_Slow\_Clk - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: Testbench to simulate the Slow\_Clk module

--

-- Dependencies: Requires the Slow\_Clk component

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity declaration for the testbench

entity Sim\_Slow\_Clk is

-- No ports needed for a testbench

end Sim\_Slow\_Clk;

-- Behavioral architecture of the testbench

architecture Behavioral of Sim\_Slow\_Clk is

-- Component declaration of the design under test (Slow\_Clk)

component Slow\_Clk

Port (

Clk\_in : in STD\_LOGIC; -- Input clock signal

Clk\_out : out STD\_LOGIC -- Output slower clock signal

);

end component;

-- Internal signals to connect to the Slow\_Clk component

signal Clk\_in, Clk\_out: STD\_LOGIC;

begin

-- Instantiation of the Slow\_Clk module

uut: Slow\_Clk

port map(

Clk\_in => Clk\_in,

Clk\_out => Clk\_out

);

-- Clock generation process

process

begin

-- Generate a clock signal with 10 ns period (5 ns high, 5 ns low)

Clk\_in <= '0';

wait for 5 ns;

Clk\_in <= '1';

wait for 5 ns;

end process;

end Behavioral;

### 1.4. Timing Diagram

A screen shot of a computer

AI-generated content may be incorrect.

## 2. Decoder 3 to 8

### 2.1. Design Source File

-- Include IEEE standard logic library

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

-- use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

-- library UNISIM;

-- use UNISIM.VComponents.all;

-- Entity declaration for 3-to-8 decoder

entity Decoder\_3\_to\_8 is

Port (

I : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit input

Y : out STD\_LOGIC\_VECTOR (7 downto 0) -- 8-bit one-hot output

);

end Decoder\_3\_to\_8;

-- Behavioral architecture

architecture Behavioral of Decoder\_3\_to\_8 is

begin

-- Process that triggers whenever input I changes

process (I) begin

-- Based on the 3-bit input, assign a corresponding one-hot value to Y

case (I) is

when "000" => Y <= "00000001"; -- Output 0 active

when "001" => Y <= "00000010"; -- Output 1 active

when "010" => Y <= "00000100"; -- Output 2 active

when "011" => Y <= "00001000"; -- Output 3 active

when "100" => Y <= "00010000"; -- Output 4 active

when "101" => Y <= "00100000"; -- Output 5 active

when "110" => Y <= "01000000"; -- Output 6 active

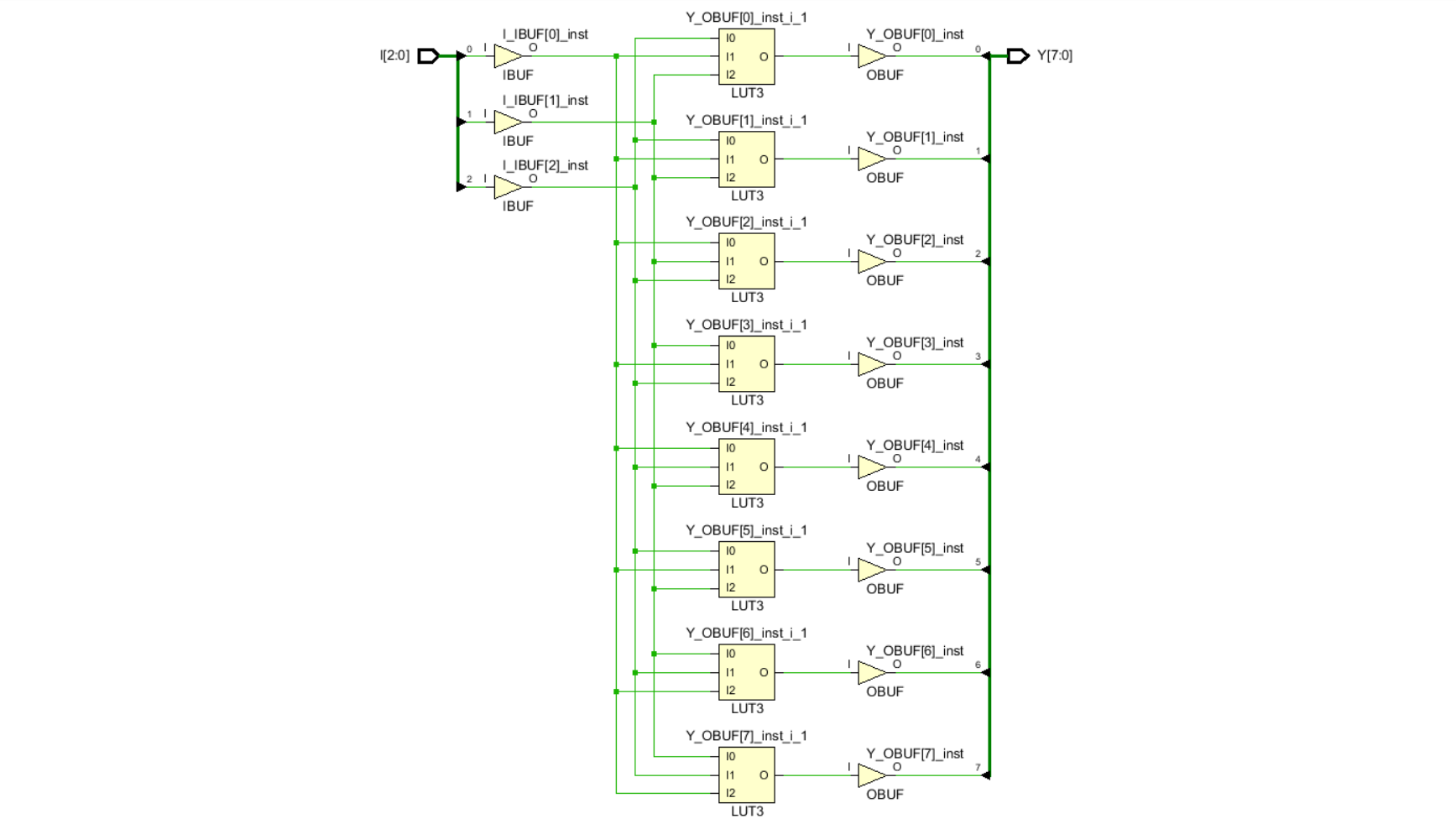
when others => Y <= "10000000"; -- Covers "111" case: Output 7 active

end case;

end process;

end Behavioral;

### 2.2. Elaborate design



### 2.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/26/2025 06:30:09 AM

-- Design Name:

-- Module Name: Sim\_Decoder\_3\_to\_8 - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: Testbench for 3-to-8 Decoder

--

-- Dependencies: Requires 'Decoder\_3\_to\_8' module

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- The entity declaration for the testbench

entity Sim\_Decoder\_3\_to\_8 is

-- No ports since this is a testbench

end Sim\_Decoder\_3\_to\_8;

-- Architecture of the testbench

architecture Behavioral of Sim\_Decoder\_3\_to\_8 is

-- Component declaration for the 3-to-8 decoder module being tested

component Decoder\_3\_to\_8

Port (

I : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit binary input

Y : out STD\_LOGIC\_VECTOR (7 downto 0) -- 8-bit one-hot output

);

end component;

-- Internal signals to connect to the decoder

signal I: STD\_LOGIC\_VECTOR(2 downto 0); -- Input signal to the decoder

signal Y: STD\_LOGIC\_VECTOR(7 downto 0); -- Output signal from the decoder

begin

-- Instantiate the decoder and map internal signals to its ports

uut: Decoder\_3\_to\_8

port map (

I => I,

Y => Y

);

-- Test process to apply input stimuli to the decoder

process

begin

-- Apply all possible 3-bit inputs one by one with 100 ns delay between each

I <= "000";

wait for 100 ns;

I <= "001";

wait for 100 ns;

I <= "010";

wait for 100 ns;

I <= "011";

wait for 100 ns;

I <= "100";

wait for 100 ns;

I <= "101";

wait for 100 ns;

I <= "110";

wait for 100 ns;

I <= "111";

wait; -- Wait indefinitely after all inputs are applied

end process;

end Behavioral;

### 2.4. Timing Diagram



## 3. Register

### 3.1. Design Source File

-- Include standard logic library

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity declaration for a 4-bit Register

entity Reg is

Port (

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input

EN : in STD\_LOGIC; -- Enable signal (active high)

Clk : in STD\_LOGIC; -- Clock signal (positive edge triggered)

Q : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit data output

);

end Reg;

-- Behavioral architecture of the 4-bit Register

architecture Behavioral of Reg is

begin

-- Process triggered on rising edge of the clock

process(Clk)

begin

-- Check for rising edge of the clock

if rising\_edge(Clk) then

-- Load input D into register only if enable signal is high

if (EN = '1') then

Q <= D;

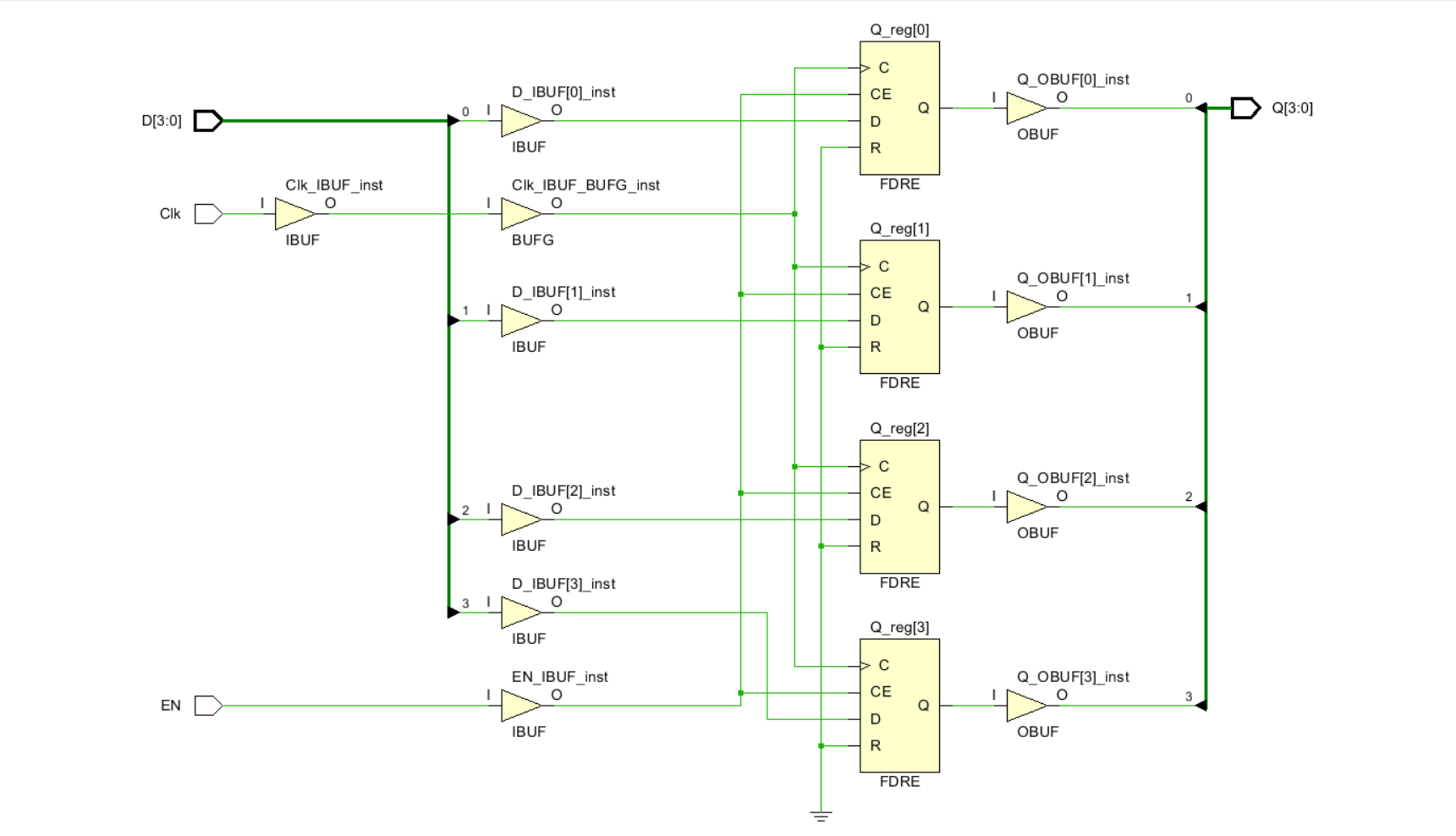
end if;

end if;

end process;

end Behavioral;

### 3.2. Elaborate design



### 3.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/26/2025 06:47:06 AM

-- Design Name:

-- Module Name: Sim\_Reg - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: Testbench for a 4-bit register with enable

--

-- Dependencies: Requires 'reg' module

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity declaration for the testbench (no ports needed)

entity Sim\_Reg is

end Sim\_Reg;

-- Architecture for the testbench

architecture Behavioral of Sim\_Reg is

-- Component declaration of the 4-bit register

component reg

Port (

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input

EN : in STD\_LOGIC; -- Enable signal

Clk : in STD\_LOGIC; -- Clock signal

Q : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit data output

);

end component;

-- Signals to connect to the register

signal D, Q : STD\_LOGIC\_VECTOR (3 downto 0); -- Input and output data lines

signal EN, Clk : STD\_LOGIC; -- Enable and Clock signals

begin

-- Instantiate the register component and map testbench signals to its ports

uut: reg

port map (

D => D,

EN => EN,

Clk => Clk,

Q => Q

);

-- Clock generation process: produces a clock with 20 ns period

Clk\_process: process

begin

Clk <= '0';

wait for 10 ns;

Clk <= '1';

wait for 10 ns;

end process;

-- Main stimulus process

Main\_process: process

begin

-- First input (write enabled)

EN <= '1'; -- Enable the register

D <= "0110"; -- Set input data to 0110

wait for 100 ns; -- Wait to allow multiple clock cycles

-- Disable writing to register

EN <= '0'; -- Disable the register

wait for 100 ns; -- Output Q should retain previous value

-- Change input while disabled (Q should not change)

D <= "1011"; -- New data on input, but EN is low

wait for 100 ns;

-- Re-enable writing (register should now load "1011")

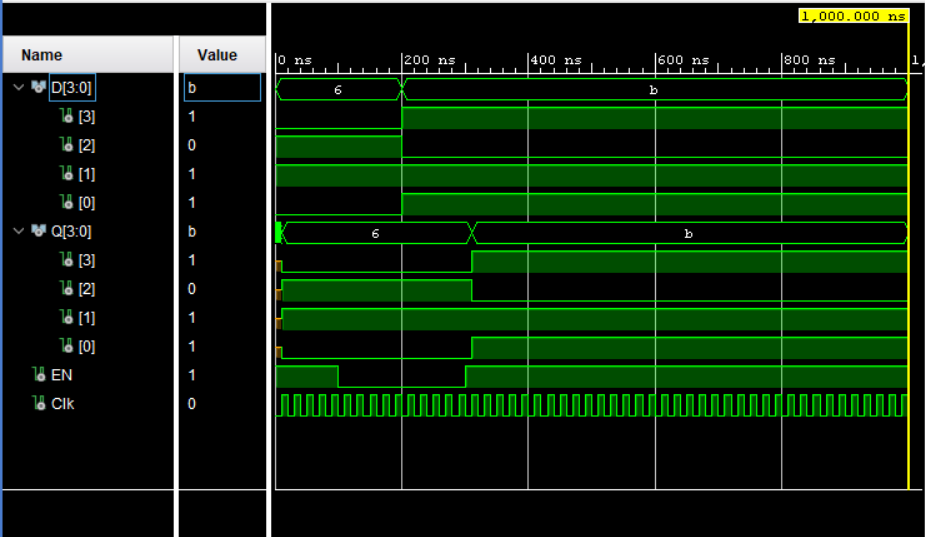
EN <= '1'; -- Enable the register again

wait; -- Wait indefinitely for simulation to end

end process;

end Behavioral;

### 3.4. Timing Diagram



## 4. Register Bank

### 4.1. Design Source File

ibrary IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for Register Bank with 8 registers

entity Register\_Bank is

Port (

Reg\_EN : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit input to select which register to enable

Clk : in STD\_LOGIC; -- Clock signal

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input

Res : in STD\_LOGIC := '0'; -- Reset signal for all registers

Reg\_0\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 0

Reg\_1\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 1

Reg\_2\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 2

Reg\_3\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 3

Reg\_4\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 4

Reg\_5\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 5

Reg\_6\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 6

Reg\_7\_out : out STD\_LOGIC\_VECTOR (3 downto 0) -- Output of Register 7

);

end Register\_Bank;

-- Behavioral architecture of the Register Bank

architecture Behavioral of Register\_Bank is

-- Component declaration for 3-to-8 decoder

component Decoder\_3\_to\_8

Port (

I : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit input

Y : out STD\_LOGIC\_VECTOR (7 downto 0) -- 8 enable outputs

);

end component;

-- Component declaration for 4-bit register

component Reg

Port (

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input

EN : in STD\_LOGIC; -- Enable signal

Clk : in STD\_LOGIC; -- Clock signal

Q : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit data output

);

end component;

-- Signal to hold outputs from the decoder

signal Y: STD\_LOGIC\_VECTOR(7 downto 0);

signal EN\_1,EN\_2,EN\_3,EN\_4,EN\_5,EN\_6,EN\_7: STD\_LOGIC;

signal Input\_Vector: STD\_LOGIC\_VECTOR(3 downto 0);

begin

Input\_Vector <= "0000" when (Res ='1') else D;

-- Instantiate the 3-to-8 decoder to generate enable signals

Decoder\_3\_to\_8\_0: Decoder\_3\_to\_8

port map(

I => Reg\_EN, -- Register select input

Y => Y -- Decoder output used to enable one of the 8 registers

);

EN\_1 <= '1' when (Res = '1') else Y(1);

EN\_2 <= '1' when (Res = '1') else Y(2);

EN\_3 <= '1' when (Res = '1') else Y(3);

EN\_4 <= '1' when (Res = '1') else Y(4);

EN\_5 <= '1' when (Res = '1') else Y(5);

EN\_6 <= '1' when (Res = '1') else Y(6);

EN\_7 <= '1' when (Res = '1') else Y(7);

-- Instantiate 8 registers, each enabled by a specific bit from decoder output

Reg\_0: Reg

port map(

D => "0000",

EN => '1',

Clk => Clk,

Q => Reg\_0\_out

);

Reg\_1: Reg

port map(

D => Input\_Vector,

EN => EN\_1,

Clk => Clk,

Q => Reg\_1\_out

);

Reg\_2: Reg

port map(

D => Input\_Vector,

EN => EN\_2,

Clk => Clk,

Q => Reg\_2\_out

);

Reg\_3: Reg

port map(

D => Input\_Vector,

EN => EN\_3,

Clk => Clk,

Q => Reg\_3\_out

);

Reg\_4: Reg

port map(

D => Input\_Vector,

EN => EN\_4,

Clk => Clk,

Q => Reg\_4\_out

);

Reg\_5: Reg

port map(

D => Input\_Vector,

EN => EN\_5,

Clk => Clk,

Q => Reg\_5\_out

);

Reg\_6: Reg

port map(

D => Input\_Vector,

EN => EN\_6,

Clk => Clk,

Q => Reg\_6\_out

);

Reg\_7: Reg

port map(

D => Input\_Vector,

EN => EN\_7,

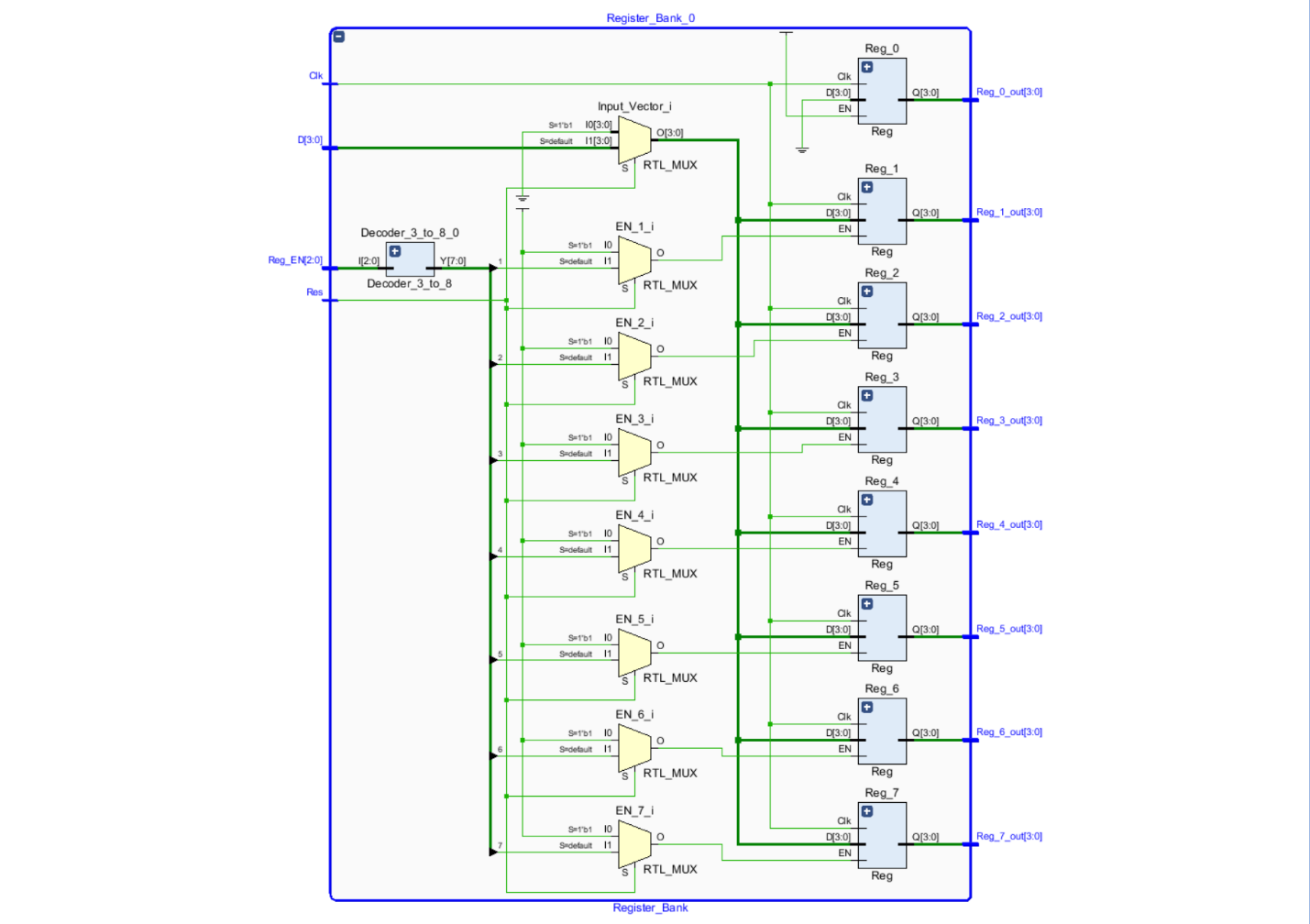
Clk => Clk,

Q => Reg\_7\_out

);

end Behavioral;

### 4.2. Elaborate design



### 4.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/05/2025 09:48:57 PM

-- Design Name:

-- Module Name: SIM\_Register\_Bank - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Testbench entity for Register\_Bank (no ports required)

entity SIM\_Register\_Bank is

-- Port ( );

end SIM\_Register\_Bank;

architecture Behavioral of SIM\_Register\_Bank is

-- Component declaration for the Register\_Bank under test

component Register\_Bank

Port (

Reg\_EN : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit register enable input

Clk : in STD\_LOGIC; -- Clock input

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input

Res : in STD\_LOGIC; -- Reset input

Reg\_out: out STD\_LOGIC\_VECTOR(31 downto 0)

);

end component;

-- Internal signals to connect to Register\_Bank inputs and outputs

signal Reg\_EN: STD\_LOGIC\_VECTOR(2 downto 0); -- Register enable select

signal Clk, Res: STD\_LOGIC; -- Clock and Reset signals

signal D, Reg\_0\_out, Reg\_1\_out, Reg\_2\_out, Reg\_3\_out, Reg\_4\_out, Reg\_5\_out, Reg\_6\_out, Reg\_7\_out: STD\_LOGIC\_VECTOR(3 downto 0);

signal Reg\_out: STD\_LOGIC\_VECTOR(31 downto 0);

begin

-- Instantiate the Unit Under Test (UUT) Register\_Bank

UUT: Register\_Bank

port map(

Reg\_EN => Reg\_EN,

Clk => Clk,

D => D,

Res => Res,

Reg\_out => Reg\_out

);

-- Process to drive input stimuli to the Register\_Bank

Register\_bank\_process: process

begin

Res <= '0';

Reg\_EN <= "000"; -- Enable Register 0

D <= "0010"; -- Data to be loaded: 2

wait for 100ns;

Reg\_EN <= "001"; -- Enable Register 1

D <= "0011"; -- Data to be loaded: 3

wait for 100ns;

Reg\_EN <= "010"; -- Enable Register 2

D <= "0000"; -- Data to be loaded: 0

wait for 100ns;

Reg\_EN <= "011"; -- Enable Register 3

D <= "0111"; -- Data to be loaded: 7

wait for 100ns;

Reg\_EN <= "100"; -- Enable Register 4

D <= "0000"; -- Data to be loaded: 0

wait for 100ns;

Reg\_EN <= "101"; -- Enable Register 5

D <= "0011"; -- Data to be loaded: 3

wait for 100ns;

Res <= '1'; -- Activate reset signal

wait; -- Wait indefinitely

end process;

-- Clock generation process, toggles clock every 10 ns (50 MHz clock)

Clk\_process: process

begin

while true loop

Clk <= '0';

wait for 10ns;

Clk <= '1';

wait for 10ns;

end loop;

end process;

end Behavioral;

### 4.4. Timing Diagram

## 5. Multiplexer 8 to 1 4bit

### 5.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for the 8-way 4-bit multiplexer

entity Mux\_8\_Way\_4\_Bit is

Port (

I0 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 0

I1 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 1

I2 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 2

I3 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 3

I4 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 4

I5 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 5

I6 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 6

I7 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Input 7

S : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit select signal

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- Output

);

end Mux\_8\_Way\_4\_Bit;

-- Architecture describing the behavior of the multiplexer

architecture Behavioral of Mux\_8\_Way\_4\_Bit is

begin

-- Process sensitive to the select signal S

process (S)

begin

-- Case statement to select the appropriate input based on S

case S is

when "000" => Y <= I0; -- Select input I0

when "001" => Y <= I1; -- Select input I1

when "010" => Y <= I2; -- Select input I2

when "011" => Y <= I3; -- Select input I3

when "100" => Y <= I4; -- Select input I4

when "101" => Y <= I5; -- Select input I5

when "110" => Y <= I6; -- Select input I6

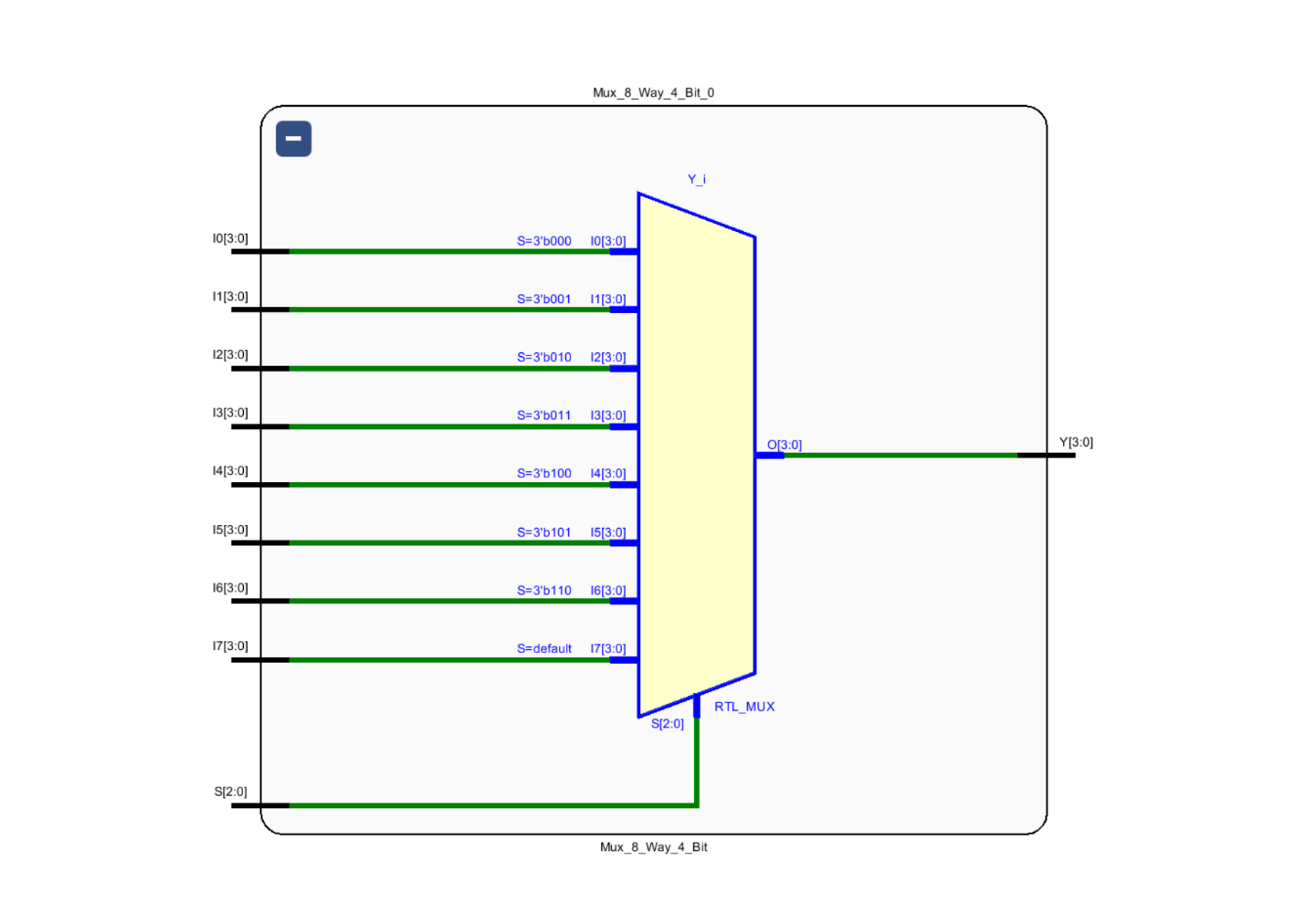
when others => Y <= I7; -- Default: select input I7 for "111" or invalid values

end case;

end process;

end Behavioral;

### 5.2. Elaborate design



### 5.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/15/2025 07:14:44 PM

-- Design Name:

-- Module Name: Sim\_Mux\_8\_Way\_4\_Bit - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Sim\_Mux\_8\_Way\_4\_Bit is

-- Port ( );

end Sim\_Mux\_8\_Way\_4\_Bit;

architecture Behavioral of Sim\_Mux\_8\_Way\_4\_Bit is

-- Component declaration for the multiplexer

component Mux\_8\_Way\_4\_Bit

Port ( I0 : in STD\_LOGIC\_VECTOR (3 downto 0);

I1 : in STD\_LOGIC\_VECTOR (3 downto 0);

I2 : in STD\_LOGIC\_VECTOR (3 downto 0);

I3 : in STD\_LOGIC\_VECTOR (3 downto 0);

I4 : in STD\_LOGIC\_VECTOR (3 downto 0);

I5 : in STD\_LOGIC\_VECTOR (3 downto 0);

I6 : in STD\_LOGIC\_VECTOR (3 downto 0);

I7 : 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 declarations for inputs, output, and select lines

signal I0,I1,I2,I3,I4,I5,I6,I7,Y : STD\_LOGIC\_VECTOR(3 downto 0);

signal S: STD\_LOGIC\_VECTOR(2 downto 0);

begin

-- Instantiate the Mux\_8\_Way\_4\_Bit component

UUT: Mux\_8\_Way\_4\_Bit

port map(

I0 => I0,

I1 => I1,

I2 => I2,

I3 => I3,

I4 => I4,

I5 => I5,

I6 => I6,

I7 => I7,

S => S,

Y => Y);

-- Testbench process to apply input stimulus

process begin

I0 <= "0000"; --0

I1 <= "0001"; --1

I2 <= "0010"; --2

I3 <= "0011"; --3

I4 <= "0100"; --4

I5 <= "0101"; --5

I6 <= "0110"; --6

I7 <= "0111"; --7

-- Apply select signal and wait for output

S <= "000"; -- Select Data from I0

wait for 100ns;

S <= "001"; -- Select Data form I1

wait for 100ns;

S <= "011"; -- Select Data from I2

wait for 100ns;

S <= "111"; -- Select Data from I7

wait;

end process;

end Behavioral;

### 5.4. Timing Diagram

A screen shot of a black screen

AI-generated content may be incorrect.

## 6. Multiplexer 2 to 1 4bit

### 6.1. Design Source File

-- Include standard logic library

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity declaration for 2-way 4-bit Multiplexer

entity Mux\_2\_Way\_4\_Bit is

Port (

A : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input A

B : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input B

S : in STD\_LOGIC; -- Select signal

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit output Y

);

end Mux\_2\_Way\_4\_Bit;

-- Behavioral architecture of the multiplexer

architecture Behavioral of Mux\_2\_Way\_4\_Bit is

-- Declaration of a 1-bit 2-way multiplexer component

-- This is not instantiated or used in this architecture,

-- but might be reserved for future structural implementation

component Mux\_2\_Way\_1\_Bit

Port (

I0 : in STD\_LOGIC; -- Input 0

I1 : in STD\_LOGIC; -- Input 1

S : in STD\_LOGIC; -- Select signal

Y : out STD\_LOGIC -- Output

);

end component;

begin

-- Concurrent assignment:

-- If S = '1' then output A is assigned to Y

-- Else output B is assigned to Y

Y <= A when (S = '1') else B;

end Behavioral;

### 6.2. Elaborate design

A diagram of a computer circuit

AI-generated content may be incorrect.

### 6.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/15/2025 08:08:53 PM

-- Design Name:

-- Module Name: Sim\_Mux\_2\_Way\_4\_Bit - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for the testbench of 2-Way 4-Bit Multiplexer

entity Sim\_Mux\_2\_Way\_4\_Bit is

-- No ports needed for a testbench

end Sim\_Mux\_2\_Way\_4\_Bit;

architecture Behavioral of Sim\_Mux\_2\_Way\_4\_Bit is

-- Component declaration for the 2-Way 4-Bit Multiplexer to be tested

component Mux\_2\_Way\_4\_Bit

Port (

A : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input A

B : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input B

S : in STD\_LOGIC; -- Select signal

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit output

);

end component;

-- Internal signals to connect to the MUX under test (UUT)

signal A, B, Y : STD\_LOGIC\_VECTOR(3 downto 0); -- Inputs A, B and output Y

signal S : STD\_LOGIC; -- Select line

begin

-- Instantiate the Unit Under Test (UUT)

UUT: Mux\_2\_Way\_4\_Bit

port map(

A => A, -- Connect test signal A

B => B, -- Connect test signal B

S => S, -- Connect test select line

Y => Y -- Observe output Y

);

-- Test process to apply stimulus to the UUT

process

begin

-- Set initial values

A <= "0101"; -- A = 0101

B <= "1011"; -- B = 1011

S <= '0'; -- Select B (S = 0 ? output = B)

wait for 100ns;

S <= '1'; -- Select A (S = 1 ? output = A)

wait; -- Wait indefinitely (end of simulation)

end process;

end Behavioral;

### 6.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 7. Program Counter

### 7.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity ProgramCounter is

Port (

Clk\_in : in STD\_LOGIC; -- Clock input signal

Res : in STD\_LOGIC; -- Asynchronous reset signal (active high)

NextVal : in UNSIGNED(2 downto 0); -- Next value to be loaded into the counter (3-bit unsigned)

Q : out UNSIGNED(2 downto 0) -- Current output value of the counter (3-bit unsigned)

);

end ProgramCounter;

architecture Behavioral of ProgramCounter is

-- Internal signal to hold the current count value

signal PC : UNSIGNED(2 downto 0) := (others => '0'); -- Initialize PC to 0

begin

process(Clk\_in)

begin

if rising\_edge(Clk\_in) then -- Trigger on the rising edge of the clock

if Res = '1' then -- If reset is asserted

PC <= (others => '0'); -- Reset the program counter to 0

else

PC <= NextVal; -- Otherwise, load the next value into the counter

end if;

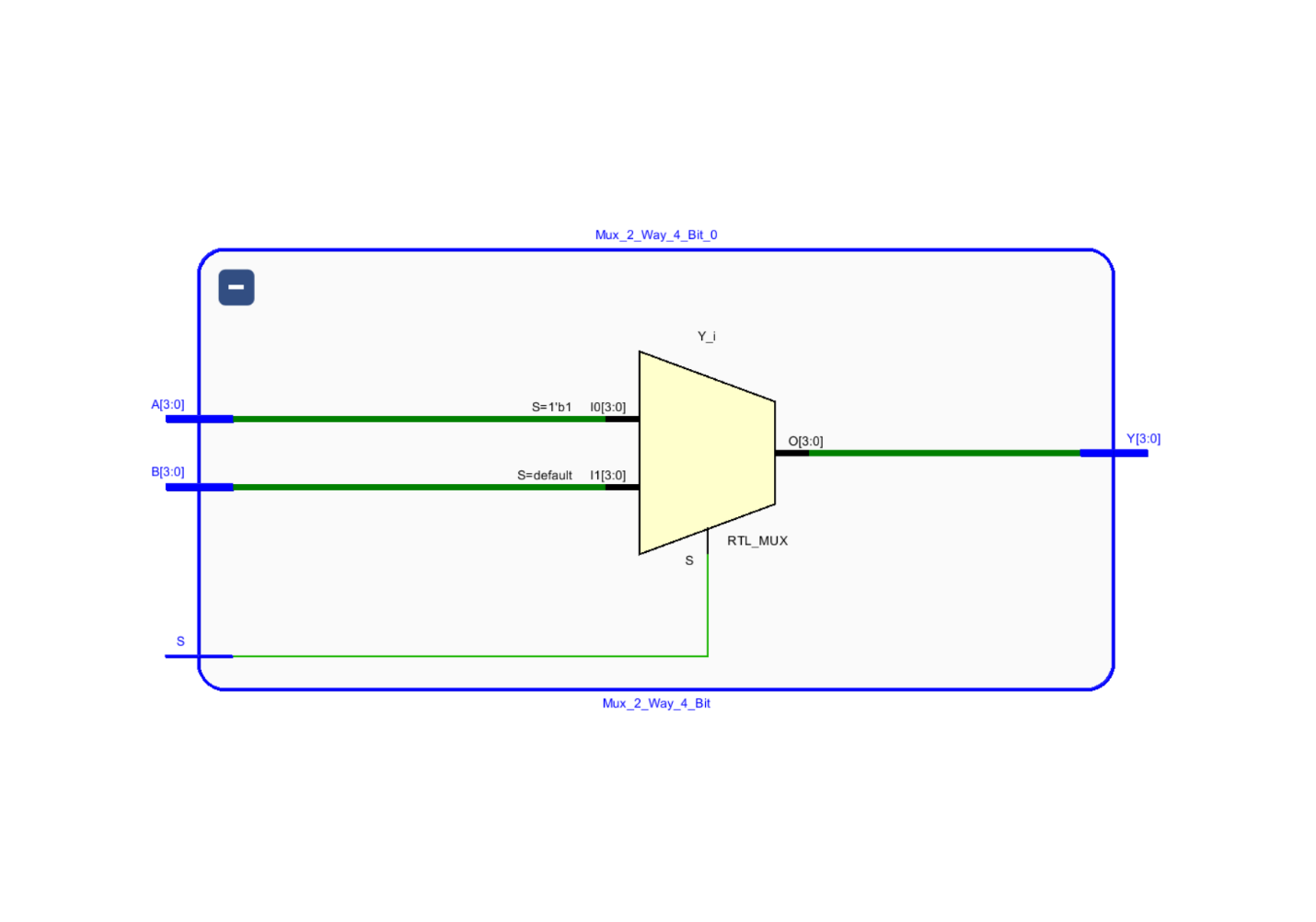
end if;

end process;

Q <= PC; -- Assign the internal counter value to the output port

end Behavioral;

### 7.2. Elaborate design



### 7.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/15/2025 08:08:53 PM

-- Design Name:

-- Module Name: Sim\_Mux\_2\_Way\_4\_Bit - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for the testbench of 2-Way 4-Bit Multiplexer

entity Sim\_Mux\_2\_Way\_4\_Bit is

-- No ports needed for a testbench

end Sim\_Mux\_2\_Way\_4\_Bit;

architecture Behavioral of Sim\_Mux\_2\_Way\_4\_Bit is

-- Component declaration for the 2-Way 4-Bit Multiplexer to be tested

component Mux\_2\_Way\_4\_Bit

Port (

A : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input A

B : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input B

S : in STD\_LOGIC; -- Select signal

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- 4-bit output

);

end component;

-- Internal signals to connect to the MUX under test (UUT)

signal A, B, Y : STD\_LOGIC\_VECTOR(3 downto 0); -- Inputs A, B and output Y

signal S : STD\_LOGIC; -- Select line

begin

-- Instantiate the Unit Under Test (UUT)

UUT: Mux\_2\_Way\_4\_Bit

port map(

A => A, -- Connect test signal A

B => B, -- Connect test signal B

S => S, -- Connect test select line

Y => Y -- Observe output Y

);

-- Test process to apply stimulus to the UUT

process

begin

-- Set initial values

A <= "0101"; -- A = 0101

B <= "1011"; -- B = 1011

S <= '0'; -- Select B (S = 0 ? output = B)

wait for 100ns;

S <= '1'; -- Select A (S = 1 ? output = A)

wait; -- Wait indefinitely (end of simulation)

end process;

end Behavioral;

### 7.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 8. Program Rom

### 8.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for a program ROM

-- Takes a 3-bit address input and outputs a 12-bit instruction

entity program\_rom is

Port (

address : in unsigned (2 downto 0); -- 3-bit address input

instruction : out STD\_LOGIC\_VECTOR (11 downto 0) -- 12-bit instruction output

);

end program\_rom;

-- Architecture defining the behavior of the ROM

architecture Behavioral of program\_rom is

-- Define a ROM type as an array of 8 elements, each 12 bits wide

type rom\_type is array (0 to 7) of std\_logic\_vector(11 downto 0);

-- Initialize ROM with predefined instructions

signal ROM : rom\_type := (

0 => "100010000001", -- MOVI R1, 1

1 => "100100000010", -- MOVI R2, 2

2 => "100110000011", -- MOVI R3, 3

3 => "101110000000", -- MOVI R7, 0

4 => "001110010000", -- ADD R7, R1

5 => "001110100000", -- ADD R7, R2

6 => "001110110000", -- ADD R7, R3

7 => "000000000000" -- NOP (or HALT)

);

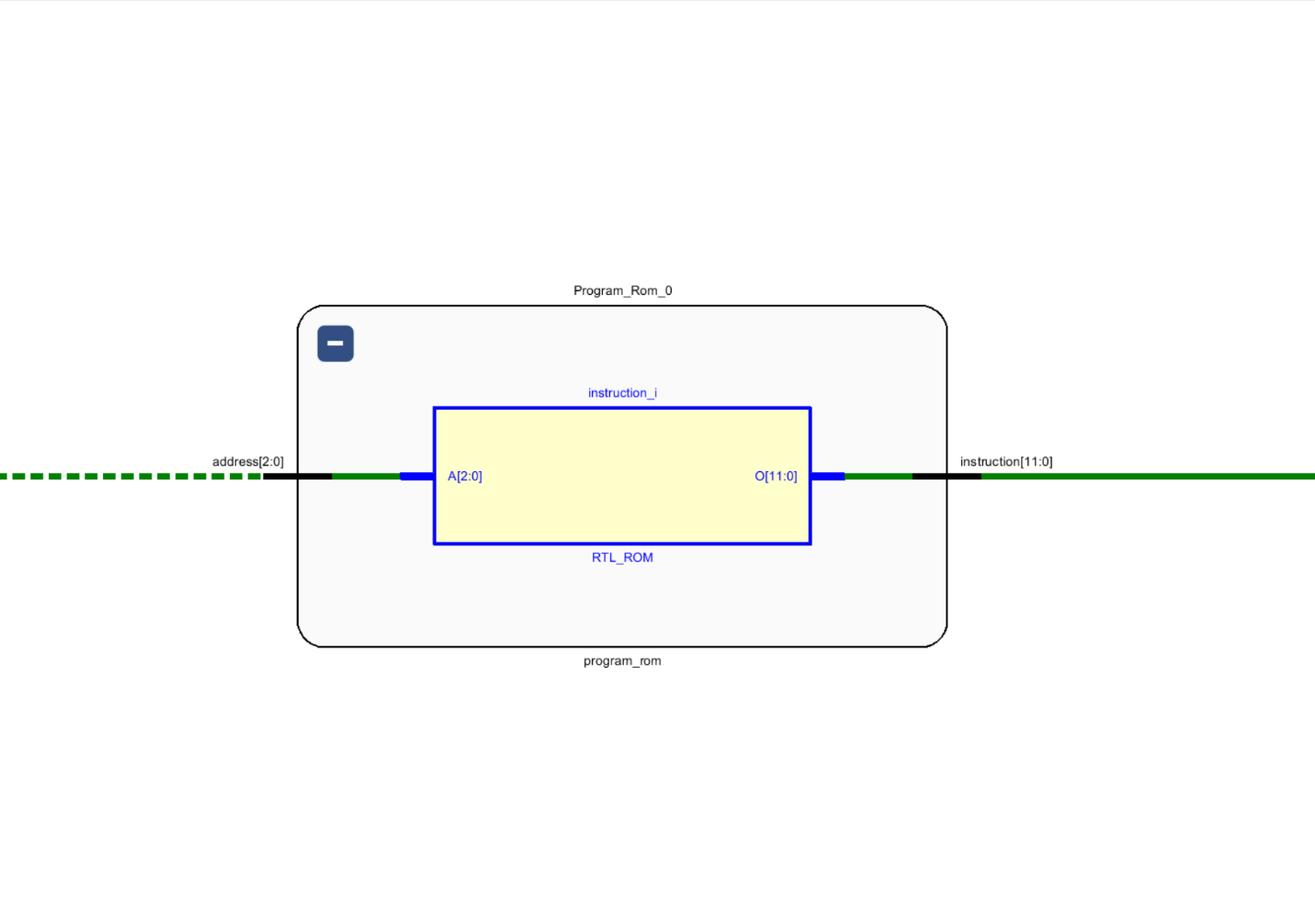
begin

-- Output the instruction at the given address

instruction<= ROM(to\_integer(address));

end Behavioral;

### 8.2. Elaborate design



### 8.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/11/2025 10:30:23 PM

-- Design Name:

-- Module Name: Sim\_program\_rom - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Testbench entity for the program\_rom

-- No ports are defined since this is a testbench

entity Sim\_program\_rom is

-- Port ( );

end Sim\_program\_rom;

architecture Behavioral of Sim\_program\_rom is

-- Declare the component program\_rom to be tested

component program\_rom

port (

address : in std\_logic\_vector (2 downto 0); -- 3-bit address input

instruction : out std\_logic\_vector (11 downto 0) -- 12-bit instruction output

);

end component;

-- Signals to connect to the program\_rom inputs and outputs

signal address : std\_logic\_vector (2 downto 0); -- Signal to drive the address input

signal instruction : std\_logic\_vector (11 downto 0); -- Signal to capture the instruction output

begin

-- Instantiate the Unit Under Test (UUT)

UUT : program\_rom

port map (

address => address, -- Connect testbench address signal to UUT input

instruction => instruction -- Connect UUT instruction output to testbench signal

);

-- (Note: No test process provided here yet to drive the address signal)

process

begin

address <= "000"; --- ROM 0

wait for 20 ns;

address <= "001"; --- ROM 1

wait for 20 ns;

address <= "010"; --- ROM 2

wait for 20 ns;

address <= "011"; --- ROM 3

wait for 20 ns;

address <= "100"; --- ROM 4

wait for 20 ns;

address <= "101"; --- ROM 5

wait for 20 ns;

address <= "110"; --- ROM 6

wait for 20 ns;

address <= "111"; --- ROM 7

wait for 20 ns;

wait;

end process;

end Behavioral;

### 8.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 9. Adder-Subtractor Unit 4bit

### 9.1. Design Source File

-- Include IEEE standard logic library

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

-- Entity declaration for a 4-bit Adder/Subtractor Unit

entity AdderSubtractor4bitUnit is

Port (

A : in STD\_LOGIC\_VECTOR(3 downto 0); -- 4-bit input A

B : in STD\_LOGIC\_VECTOR(3 downto 0); -- 4-bit input B

ADD\_SUB : in STD\_LOGIC; -- Operation select: '0' for add, '1' for subtract

RESULT : out STD\_LOGIC\_VECTOR(3 downto 0); -- 4-bit result output

C\_OUT : out STD\_LOGIC; -- Carry-out (not used in this architecture)

OVERFLOW : out STD\_LOGIC; -- Overflow flag output from RCA\_4

ZERO : out STD\_LOGIC -- Zero flag: '1' if result is all zeros

);

end AdderSubtractor4bitUnit;

-- Architecture definition using a 4-bit Ripple Carry Adder component

architecture Behavioral of AdderSubtractor4bitUnit is

-- Internal signals

signal A\_mod : STD\_LOGIC\_VECTOR(3 downto 0); -- Modified A input (inverted for subtraction)

signal B\_mod : STD\_LOGIC\_VECTOR(3 downto 0); -- Modified B input (zero for subtraction)

signal S : STD\_LOGIC\_VECTOR(4 downto 0); -- Output from RCA\_4, 5 bits to include carry/overflow

signal temp\_sum : unsigned(4 downto 0); -- Unsigned 5-bit internal sum to hold carry

begin

-- Logic for operand selection:

-- If ADD\_SUB = '0' ? addition: A\_mod = A, B\_mod = B

-- If ADD\_SUB = '1' ? subtraction: A\_mod = NOT A (two's complement), B\_mod = "0000"

A\_mod <= A when (ADD\_SUB = '0') else not(A); -- Invert A for subtraction

B\_mod <= B when (ADD\_SUB = '0') else (others => '0'); -- Set B to zero in subtraction

-- Perform the operation:

-- For addition: A + B

-- For subtraction: NOT A + 0 + 1 (2's complement of A), simulating -A

temp\_sum <= ('0' & unsigned(A\_mod)) + unsigned(B\_mod) + ("0000" & ADD\_SUB);

-- Assign calculated value to output signal S

S <= STD\_LOGIC\_VECTOR(temp\_sum);

-- Set 4-bit RESULT from lower 4 bits of S

RESULT <= S(3 downto 0);

-- ZERO flag: '1' if result is all zeros, else '0'

ZERO <= not (S(0) OR S(1) OR S(2) OR S(3));

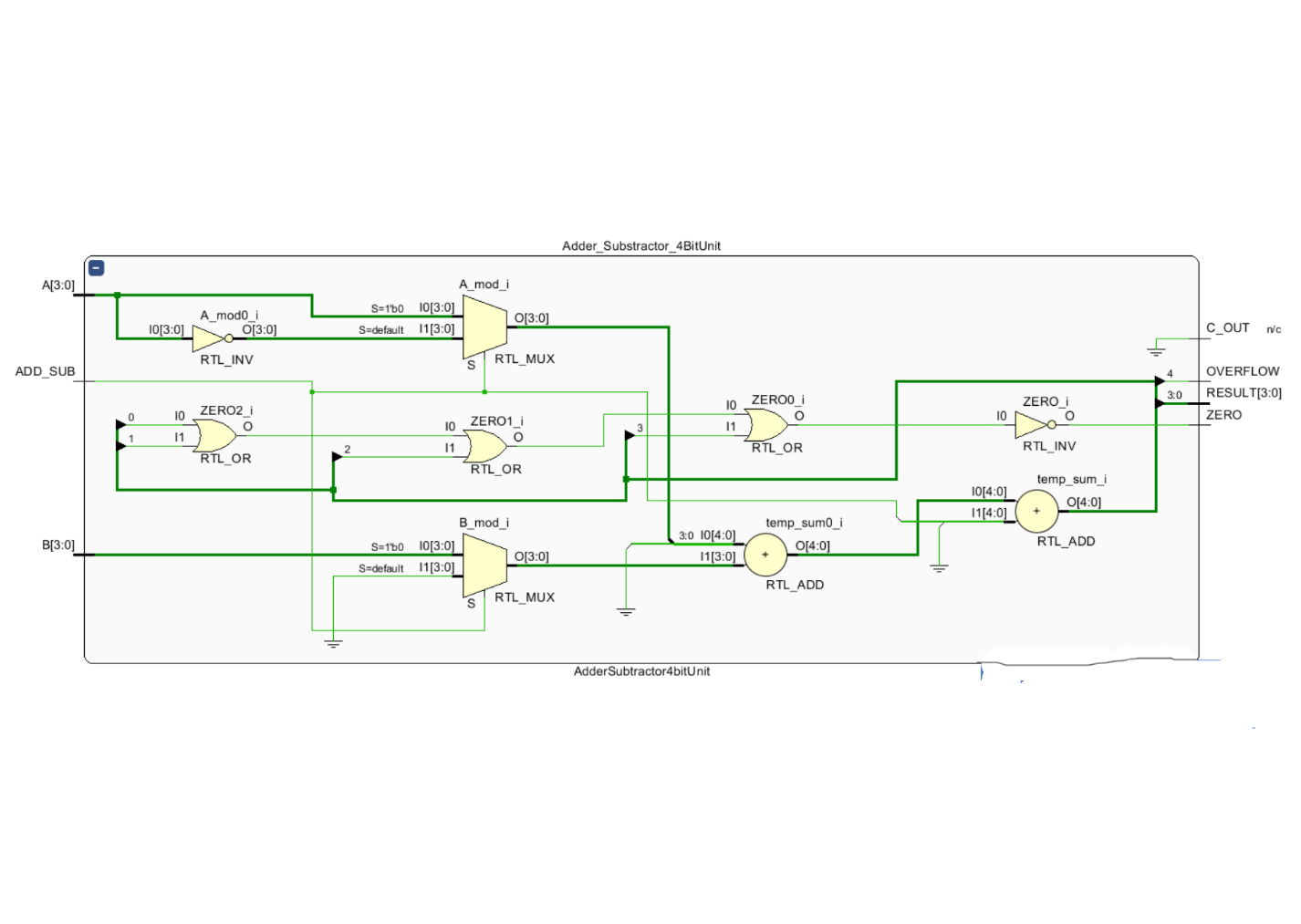
-- Overflow is indicated by the 5th bit (MSB) of the sum

OVERFLOW <= S(4);

-- Note: C\_OUT is declared in the port but is not assigned or used in this architecture

end Behavioral;

### 9.2. Elaborate design



### 9.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/14/2025 08:28:12 PM

-- Design Name:

-- Module Name: TB\_AdderSubtractor4bitUnit - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity TB\_AdderSubtractor4bitUnit is

-- Port ( );

end TB\_AdderSubtractor4bitUnit;

architecture Behavioral of TB\_AdderSubtractor4bitUnit is

component AdderSubtractor4bitUnit

Port (

A : in STD\_LOGIC\_VECTOR(3 downto 0);

B : in STD\_LOGIC\_VECTOR(3 downto 0);

ADD\_SUB : in STD\_LOGIC;

RESULT : out STD\_LOGIC\_VECTOR(3 downto 0);

OVERFLOW : out STD\_LOGIC;

ZERO : out STD\_LOGIC

);

end component;

-- Signals for test

signal A, B : STD\_LOGIC\_VECTOR(3 downto 0);

signal ADD\_SUB : STD\_LOGIC;

signal RESULT : STD\_LOGIC\_VECTOR(3 downto 0);

signal C\_OUT : STD\_LOGIC;

signal OVERFLOW : STD\_LOGIC;

signal ZERO : STD\_LOGIC;

begin

UUT: AdderSubtractor4bitUnit

port map (

A => A,

B => B,

ADD\_SUB => ADD\_SUB,

RESULT => RESULT,

OVERFLOW => OVERFLOW,

ZERO => ZERO

);

-- Test process

process

begin

-- Test case 1: 5 + 7

A <= "0101"; -- 5

B <= "0111"; -- 7

ADD\_SUB <= '0'; -- Add

wait for 10 ns;-- 1100

A <= "0111"; -- 7

B <= "0011"; -- 3

ADD\_SUB <= '1'; -- Subtract

wait for 10 ns; --1001

A <= "0010"; -- 2

B <= "0101"; -- 5

ADD\_SUB <= '1'; -- Subtract

wait for 10 ns;--1110

-- Test case 4: 8 + 8 (expect overflow)

A <= "1000"; -- 8

B <= "1000"; -- 8

ADD\_SUB <= '0';

wait for 10 ns;--0000

A <= "0100"; -- 4

B <= "0100"; -- 4

ADD\_SUB <= '1';-- Subtract

wait for 10 ns; --1100

wait; -- Stop simulation

end process;

end Behavioral;

### 9.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 10. Instruction Decoder

### 10.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

-- Entity declaration for InstructionDecoder

-- Decodes a 12-bit instruction bus into control signals and register selections

entity InstructionDecoder is

Port (

InstructionBus : in STD\_LOGIC\_VECTOR (11 downto 0); -- 12-bit instruction input

RegisterCheck : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input for register condition checking

Reg\_EN : out STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit output for register select 1

RegSelect\_A : out STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit output for register select 2

RegSelect\_B : out STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit output for register select 3

LoadSelect : out STD\_LOGIC; -- Control signal for load selection

ImmediateValue : out STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit immediate value output

OperationSelect : out STD\_LOGIC; -- Control signal for operation selection

JumpFlag: out STD\_LOGIC; -- Flag indicating if jump instruction is active

JumpAddress: out unsigned(2 downto 0) -- 3-bit jump address output

);

end InstructionDecoder;

architecture Behavioral of InstructionDecoder is

-- Signal to hold the 2-bit opcode extracted from the instruction bus

signal Opcode : STD\_LOGIC\_VECTOR(1 downto 0);

begin

process(InstructionBus, RegisterCheck)

variable opcode\_var : STD\_LOGIC\_VECTOR(1 downto 0);

begin

opcode\_var := InstructionBus(11 downto 10);

-- Default assignments:

RegSelect\_A <= InstructionBus(9 downto 7);

RegSelect\_B <= InstructionBus(6 downto 4);

ImmediateValue <= InstructionBus(3 downto 0);

JumpAddress <= unsigned(InstructionBus(2 downto 0));

OperationSelect<= InstructionBus(10);

LoadSelect <= not InstructionBus(11);

JumpFlag <= '0';

Reg\_EN <= InstructionBus(9 downto 7);

case opcode\_var is

when "10" => -- MOVI

Reg\_EN <= InstructionBus(9 downto 7);

-- LoadSelect = 0 (already set above)

when "00" => -- ADD

Reg\_EN <= InstructionBus(9 downto 7);

-- LoadSelect = 1 (already set above)

when "01" => -- NEG

Reg\_EN <= InstructionBus(9 downto 7);

-- OperationSelect already set

when "11" => -- JZR

if RegisterCheck = "0000" then

JumpFlag <= '1';

else

JumpFlag <= '0';

end if;

Reg\_EN <= "000"; -- prevent register writes during jump

when others =>

Reg\_EN <= "000";

JumpFlag <= '0';

end case;

end process;

end Behavioral;

### 10.2. Elaborate design

A computer screen shot of a diagram

AI-generated content may be incorrect.

### 10.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/14/2025 01:38:50 PM

-- Design Name:

-- Module Name: Sim\_Instruction\_Decoder - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Sim\_Instruction\_Decoder is

-- Port ( );

end Sim\_Instruction\_Decoder;

architecture Behavioral of Sim\_Instruction\_Decoder is

-- Component declaration for the InstructionDecoder to be tested

component InstructionDecoder

Port (

InstructionBus : in STD\_LOGIC\_VECTOR (11 downto 0); -- 12-bit instruction input

RegisterCheck : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit register check input

Reg\_EN : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register Enable

RegSelect\_A : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register select output A

RegSelect\_B : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register select output B

LoadSelect : out STD\_LOGIC; -- Load select control signal

ImmediateValue : out STD\_LOGIC\_VECTOR (3 downto 0); -- Immediate value output

OperationSelect : out STD\_LOGIC; -- Operation select control signal

JumpFlag: out STD\_LOGIC; -- Jump flag output

JumpAddress: out STD\_LOGIC\_VECTOR(2 downto 0) -- Jump address output

);

end component;

-- Signals to connect to InstructionDecoder inputs and outputs

signal InstructionBus: STD\_LOGIC\_VECTOR(11 downto 0);

signal RegisterCheck, ImmediateValue : STD\_LOGIC\_VECTOR(3 downto 0);

signal Reg\_EN, RegSelect\_A, RegSelect\_B, JumpAddress: STD\_LOGIC\_VECTOR(2 downto 0);

signal LoadSelect, OperationSelect, JumpFlag: STD\_LOGIC;

begin

-- Instantiate the Unit Under Test (UUT)

UUT : InstructionDecoder

port Map(

InstructionBus => InstructionBus,

RegisterCheck => RegisterCheck,

Reg\_EN => Reg\_EN,

RegSelect\_A => RegSelect\_A,

RegSelect\_B => RegSelect\_B,

LoadSelect => LoadSelect,

ImmediateValue => ImmediateValue,

OperationSelect => OperationSelect,

JumpFlag => JumpFlag,

JumpAddress => JumpAddress);

-- Test process to drive different instruction inputs and observe outputs

process

begin

-- Test case 1: MOVI R1, 10 or similar instruction

InstructionBus <= "100010001010";

wait for 100 ns;

-- Test case 2: NEG or arithmetic operation

InstructionBus <= "010100000000";

wait for 100 ns;

-- Test case 3: ADD or other arithmetic instruction

InstructionBus <= "000100010000";

wait for 100 ns;

-- Test case 4: Jump instruction with RegisterCheck zero (enabling jump)

InstructionBus <= "110010000011";

RegisterCheck <= "0000"; -- Must be set to enable jump in decoder

wait for 100 ns;

-- Test case 5: NOP or default instruction

InstructionBus <= "000000000000";

wait; -- Wait indefinitely to end simulation

end process;

end Behavioral;

### 10.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 11. Nano Processor

### 11.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

-- Entity declaration for the Nano Processor

entity Nano\_Processor is

Port (

Res, Clk : in STD\_LOGIC; -- Reset and Clock inputs

Reg\_7\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Output of Register 7

OverFlow, Zero: out STD\_LOGIC; -- ALU status flags: Overflow and Zero

Display\_7\_Segment: out STD\_LOGIC\_VECTOR(6 downto 0); -- Output signals to 7-segment display (7 segments)

an: out STD\_LOGIC\_VECTOR(3 downto 0) -- Anode control for 4-digit 7-segment display (active low)

);

end Nano\_Processor;

architecture Behavioral of Nano\_Processor is

-- Register bank: 8 registers, each 4-bit wide

component Register\_Bank

Port (

Reg\_EN : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit register enable address (select which register to write)

Clk : in STD\_LOGIC; -- Clock signal

D : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit data input to registers

Res : in STD\_LOGIC := '0'; -- Reset signal to reset all registers

Reg\_0\_out : out STD\_LOGIC\_VECTOR (3 downto 0); -- Outputs for each register (0 to 7)

Reg\_1\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_2\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_3\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_4\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_5\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_6\_out : out STD\_LOGIC\_VECTOR (3 downto 0);

Reg\_7\_out : out STD\_LOGIC\_VECTOR (3 downto 0)

);

end component;

-- ALU: 4-bit Adder/Subtractor Unit

component AdderSubtractor4bitUnit

Port (

A, B : in STD\_LOGIC\_VECTOR(3 downto 0); -- Two 4-bit operands

ADD\_SUB : in STD\_LOGIC; -- Operation select: 0 for add, 1 for subtract

RESULT : out STD\_LOGIC\_VECTOR(3 downto 0); -- 4-bit result output

OVERFLOW : out STD\_LOGIC; -- Overflow flag output

ZERO : out STD\_LOGIC -- Zero flag output

);

end component;

-- 2-to-1 multiplexer for 4-bit data input selection

component Mux\_2\_Way\_4\_Bit

Port (

A, B : in STD\_LOGIC\_VECTOR (3 downto 0); -- Two 4-bit inputs

S : in STD\_LOGIC; -- Select signal: 0 selects A, 1 selects B

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- Output of the mux

);

end component;

-- 2-to-1 multiplexer for 3-bit addresses (used for selecting ROM address)

component Rom\_Address\_Select

Port (

A, B : in unsigned(2 downto 0); -- Two 3-bit address inputs

S : in STD\_LOGIC; -- Select signal: 0 selects A, 1 selects B

Y : out unsigned (2 downto 0) -- Output selected address

);

end component;

-- 8-to-1 multiplexer for selecting one of 8 registers (4-bit each)

component Mux\_8\_Way\_4\_Bit

Port (

I0, I1, I2, I3, I4, I5, I6, I7 : in STD\_LOGIC\_VECTOR (3 downto 0); -- Inputs from all 8 registers

S : in STD\_LOGIC\_VECTOR (2 downto 0); -- 3-bit select to choose one register

Y : out STD\_LOGIC\_VECTOR (3 downto 0) -- Output selected register data

);

end component;

-- Instruction decoder: decodes 12-bit instruction into control signals

component InstructionDecoder

Port (

InstructionBus : in STD\_LOGIC\_VECTOR (11 downto 0); -- 12-bit instruction input

RegisterCheck : in STD\_LOGIC\_VECTOR (3 downto 0); -- Data from selected register A (for instruction checking)

Reg\_EN : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register to write enable (3 bits)

RegSelect\_A : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register A selector (3 bits)

RegSelect\_B : out STD\_LOGIC\_VECTOR (2 downto 0); -- Register B selector (3 bits)

LoadSelect : out STD\_LOGIC; -- Select load source (ALU result or immediate)

ImmediateValue : out STD\_LOGIC\_VECTOR (3 downto 0); -- Immediate 4-bit value from instruction

OperationSelect : out STD\_LOGIC; -- ALU operation select (Add/Sub)

JumpFlag : out STD\_LOGIC; -- Jump flag signal

JumpAddress : out unsigned(2 downto 0) -- Jump address for PC

);

end component;

-- Program Counter: holds current instruction address (3-bit)

component ProgramCounter

Port (

Clk\_in : in STD\_LOGIC; -- Clock input

Res : in STD\_LOGIC; -- Reset signal

NextVal : in unsigned (2 downto 0); -- Next address input

Q : out unsigned (2 downto 0) -- Current address output

);

end component;

-- Clock divider to slow down system clock for visibility / timing reasons

component Slow\_Clk

Port (

Clk\_in : in STD\_LOGIC; -- Fast clock input

Clk\_out : out STD\_LOGIC -- Slower clock output

);

end component;

-- 3-bit adder that adds 1 to the current PC value (used for PC increment)

component adder\_3bit

Port (

INPUT : in unsigned (2 downto 0); -- Input 3-bit number

OUTPUT : out unsigned (2 downto 0) -- Output incremented by 1

);

end component;

-- ROM: Program memory holding instructions (8 locations, 12-bit wide instructions)

component program\_rom

Port (

address : in unsigned (2 downto 0); -- 3-bit address input

instruction : out STD\_LOGIC\_VECTOR (11 downto 0) -- 12-bit instruction output

);

end component;

-- Lookup table to convert 4-bit binary value to 7-segment display pattern

component LUT\_16\_7

Port (

address : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input value

data : out STD\_LOGIC\_VECTOR (6 downto 0) -- 7-bit segment pattern output

);

end component;

-- Internal signals used for interconnecting components

signal Clk\_out, Jump\_Flag, LoadSelect, OperationSelect : STD\_LOGIC;

signal Instruction\_Bus\_1 : STD\_LOGIC\_VECTOR(11 downto 0);

signal Next\_Rom\_Address, Rom\_Address\_Add, Rom\_Address, Jump\_Address : unsigned(2 downto 0);

signal Reg\_EN, RegSelect\_A, RegSelect\_B : STD\_LOGIC\_VECTOR(2 downto 0);

signal Reg\_A, Reg\_B, ImmediateValue, Result, Value : STD\_LOGIC\_VECTOR(3 downto 0);

signal Reg\_0, Reg\_1, Reg\_2, Reg\_3, Reg\_4, Reg\_5, Reg\_6, Reg\_7 : STD\_LOGIC\_VECTOR(3 downto 0);

begin

-- Instantiate the slow clock generator to reduce clock speed for timing

Slow\_Clk\_0: Slow\_Clk

port map(Clk\_in => Clk, Clk\_out => Clk\_out);

-- Instantiate the Program Counter with slow clock and reset

Program\_Counter: ProgramCounter

port map(Clk\_in => Clk\_out, Res => Res, NextVal => Next\_Rom\_Address, Q => Rom\_Address);

-- PC + 1 adder to calculate next sequential instruction address (currently commented out)

--Adder\_3\_Bit: adder\_3bit

--port map(INPUT => Rom\_Address, OUTPUT => Rom\_Address\_Add);

-- Multiplexer to select between jump address and next sequential PC address

Rom\_Address\_Select\_0: Rom\_Address\_Select

port map(A => Jump\_Address, B => Rom\_Address, S => Jump\_Flag, Y => Next\_Rom\_Address);

-- Instruction memory ROM: fetch instruction based on current PC

Program\_Rom\_0: Program\_Rom

port map(address => Rom\_Address, Instruction => Instruction\_Bus\_1);

-- Instruction decoder: decode instruction into control signals and immediate values

Instruction\_Decoder: InstructionDecoder

port map(

InstructionBus => Instruction\_Bus\_1,

RegisterCheck => Reg\_A,

Reg\_EN => Reg\_EN,

RegSelect\_A => RegSelect\_A,

RegSelect\_B => RegSelect\_B,

LoadSelect => LoadSelect,

ImmediateValue => ImmediateValue,

OperationSelect => OperationSelect,

JumpFlag => Jump\_Flag,

JumpAddress => Jump\_Address);

-- Register bank: 8 registers, load enabled by Reg\_EN on rising clock edge

Register\_Bank\_0: Register\_Bank

port map(

Reg\_EN => Reg\_EN,

Clk => Clk\_out,

D => Value,

Res => Res,

Reg\_0\_out => Reg\_0,Reg\_1\_out => Reg\_1,Reg\_2\_out => Reg\_2,Reg\_3\_out => Reg\_3,Reg\_4\_out => Reg\_4,

Reg\_5\_out => Reg\_5,Reg\_6\_out => Reg\_6,Reg\_7\_out => Reg\_7);

-- Mux to select either ALU Result or Immediate value for loading into register

Mux\_2\_Way\_4\_Bit\_0: Mux\_2\_Way\_4\_Bit

port map(A => Result, B => ImmediateValue, S => LoadSelect, Y => Value);

-- Mux to select Register A data based on RegSelect\_A (used as ALU input and decoder input)

Mux\_8\_Way\_4\_Bit\_0: Mux\_8\_Way\_4\_Bit

port map(

I0 => Reg\_0, I1 => Reg\_1, I2 => Reg\_2, I3 => Reg\_3,

I4 => Reg\_4, I5 => Reg\_5, I6 => Reg\_6, I7 => Reg\_7,

S => RegSelect\_A, Y => Reg\_A);

-- Mux to select Register B data based on RegSelect\_B (used as ALU input)

Mux\_8\_Way\_4\_Bit\_1: Mux\_8\_Way\_4\_Bit

port map(

I0 => Reg\_0, I1 => Reg\_1, I2 => Reg\_2, I3 => Reg\_3,

I4 => Reg\_4, I5 => Reg\_5, I6 => Reg\_6, I7 => Reg\_7,

S => RegSelect\_B, Y => Reg\_B);

-- ALU: 4-bit Adder/Subtractor unit performs operation on Reg\_A and Reg\_B

Adder\_Substractor\_4BitUnit: AdderSubtractor4bitUnit

port map(

A => Reg\_A, B => Reg\_B, ADD\_SUB => OperationSelect,

Result => Result, OverFlow => OverFlow, ZERO => Zero);

-- Lookup table converts 4-bit Register 7 output into 7-segment display encoding

LUT\_16\_7\_0: LUT\_16\_7

port map(address => Reg\_7, data => Display\_7\_Segment);

-- Output the contents of Register 7 externally

Reg\_7\_out <= Reg\_7;

-- Enable only the first digit of the 7-segment display (active low)

an <= "1110";

end Behavioral;

### 11.2. Elaborate design

A diagram of a machine

AI-generated content may be incorrect.

### 11.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/15/2025 10:05:30 PM

-- Design Name:

-- Module Name: Sim\_Nano\_Processor - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL; -- Optional for counter-style clock generation

use IEEE.NUMERIC\_STD.ALL;

use ieee.std\_logic\_textio.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC\_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Sim\_Nano\_Processor is

-- Port ( );

end Sim\_Nano\_Processor;

architecture Behavioral of Sim\_Nano\_Processor is

-- Component declaration of the Nano\_Processor

component Nano\_Processor

Port (

Res : in STD\_LOGIC;

Clk : in STD\_LOGIC;

Reg\_7\_out : out STD\_LOGIC\_VECTOR(3 downto 0);

OverFlow : out STD\_LOGIC;

Zero : out STD\_LOGIC;

Display\_7\_Segment: out STD\_LOGIC\_VECTOR(6 downto 0)

);

end component;

-- Signals to connect to the Nano\_Processor ports

signal Res : STD\_LOGIC := '1'; -- Start with reset active

signal Clk : STD\_LOGIC := '0';

signal Reg\_7\_out : STD\_LOGIC\_VECTOR(3 downto 0);

signal OverFlow : STD\_LOGIC;

signal Zero : STD\_LOGIC;

signal Display\_7\_Segment: STD\_LOGIC\_VECTOR(6 downto 0);

constant CLK\_PERIOD : time := 20 ns; -- 50 MHz clock (for simulation)

begin

-- Instantiate the Unit Under Test (UUT)

UUT: Nano\_Processor

Port map (

Res => Res,

Clk => Clk,

Reg\_7\_out => Reg\_7\_out,

OverFlow => OverFlow,

Zero => Zero,

Display\_7\_Segment => Display\_7\_Segment

);

-- Clock generation process

Clk\_process :process

begin

while true loop

Clk <= '0';

wait for CLK\_PERIOD/2;

Clk <= '1';

wait for CLK\_PERIOD/2;

end loop;

end process;

-- Reset process

Reset\_process :process

begin

-- Hold reset active for first 2 clock cycles

Res <= '1';

wait for 2\*CLK\_PERIOD;

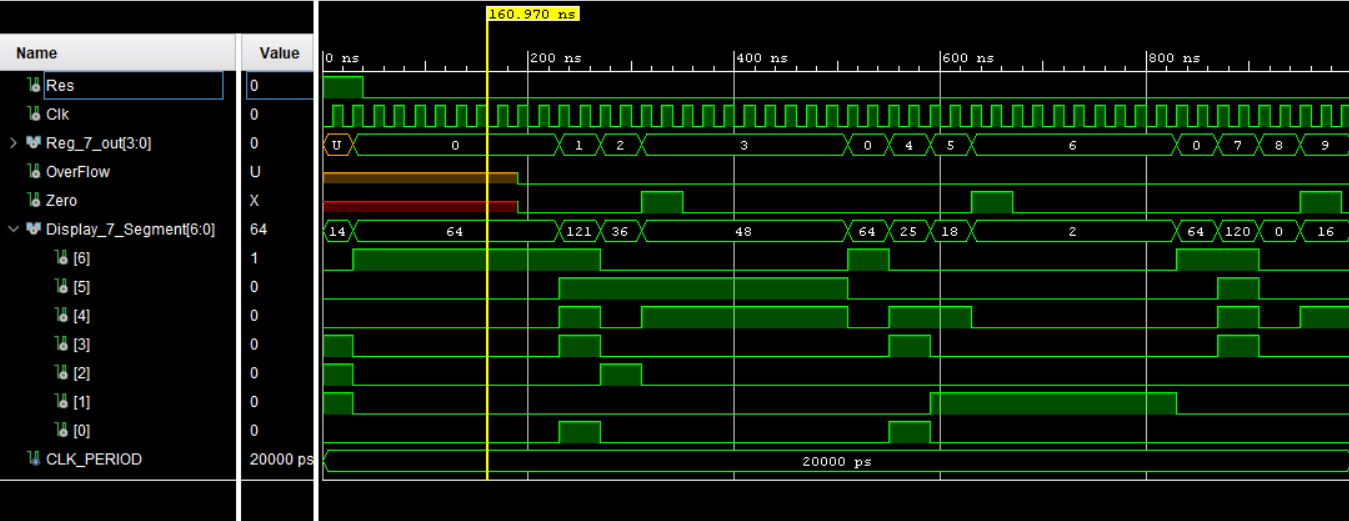
Res <= '0'; -- Release reset

wait;

end process;

end Behavioral;

### 11.4. Timing Diagram



## 12. LUT 16\_7

### 12.1. Design Source File

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.numeric\_std.all; -- For numeric conversions

-- Entity declaration for the 16-to-7 segment LUT

entity LUT\_16\_7 is

Port (

address : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input address (0 to 15)

data : out STD\_LOGIC\_VECTOR (6 downto 0) -- 7-bit output to drive seven-segment display

);

end LUT\_16\_7;

architecture Behavioral of LUT\_16\_7 is

begin

process(address)

begin

case address is

when "0000" => data <= "1000000"; -- 0

when "0001" => data <= "1111001"; -- 1

when "0010" => data <= "0100100"; -- 2

when "0011" => data <= "0110000"; -- 3

when "0100" => data <= "0011001"; -- 4

when "0101" => data <= "0010010"; -- 5

when "0110" => data <= "0000010"; -- 6

when "0111" => data <= "1111000"; -- 7

when "1000" => data <= "0000000"; -- 8

when "1001" => data <= "0010000"; -- 9

when "1010" => data <= "0001000"; -- A

when "1011" => data <= "0000011"; -- B

when "1100" => data <= "1000110"; -- C

when "1101" => data <= "0100001"; -- D

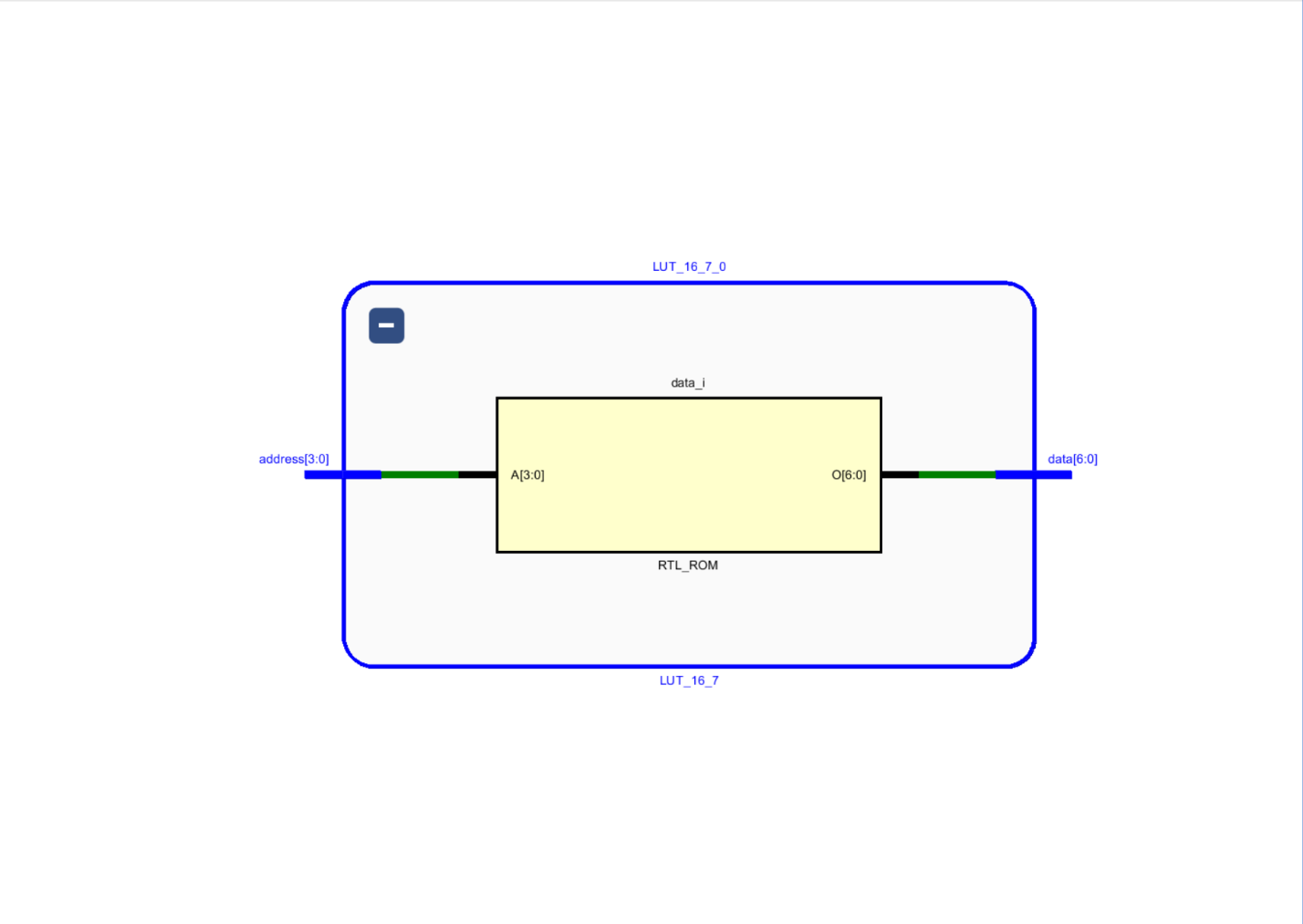
when "1110" => data <= "0000110"; -- E

when others => data <= "0001110"; -- F

end case;

end process;

### 12.2. Elaborate design



### 12.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 05/26/2025 07:25:34 AM

-- Design Name:

-- Module Name: Sim\_LUT\_16\_7 - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: Testbench for LUT\_16\_7 - simulates a lookup table that maps

-- 4-bit binary addresses to 7-bit outputs for a seven-segment display.

--

-- Dependencies: Requires LUT\_16\_7 component definition

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

-- Entity for the testbench; no ports needed

entity Sim\_LUT\_16\_7 is

end Sim\_LUT\_16\_7;

architecture Behavioral of Sim\_LUT\_16\_7 is

-- Declaration of the component under test

component LUT\_16\_7

Port (

address : in STD\_LOGIC\_VECTOR (3 downto 0); -- 4-bit input address (values 0 to 15)

data : out STD\_LOGIC\_VECTOR (6 downto 0) -- 7-bit output (usually for 7-segment display)

);

end component;

-- Signals to connect to the component's ports

signal address : STD\_LOGIC\_VECTOR(3 downto 0); -- Test input address

signal data : STD\_LOGIC\_VECTOR (6 downto 0); -- Output from the LUT

begin

-- Instantiate the unit under test (UUT)

uut: LUT\_16\_7

port map (

address => address,

data => data

);

-- Test process to apply input stimulus

process

begin

-- Apply different 4-bit address values to simulate lookup behavior

address <= "0000"; -- Expect output for 0

wait for 100 ns;

address <= "0001"; -- Expect output for 1

wait for 100 ns;

address <= "0010"; -- Expect output for 2

wait for 100 ns;

address <= "0011"; -- Expect output for 3

wait for 100 ns;

address <= "0100"; -- Expect output for 4

wait for 100 ns;

address <= "0101"; -- Expect output for 5

wait for 100 ns;

address <= "0110"; -- Expect output for 6

wait for 100 ns;

address <= "0111"; -- Expect output for 7

wait; -- End of simulation (no time delay, simulation halts here)

end process;

end Behavioral;

### 12.4. Timing Diagram

A screenshot of a computer

AI-generated content may be incorrect.

## 13. Rom Address Select (3 bit adder and Mux 2 way 3 bit)

### 13.1. Design Source File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

-- Create Date: 14.05.2025 14:43:41

-- Design Name:

-- Module Name: 2\_Way\_3\_Bit\_Mux - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: A 2-to-1 multiplexer for 3-bit inputs. Selects between input A and B

-- based on select signal S and assigns the result to output Y.

-- Dependencies: None

-- Revision:

-- Revision 0.01 - File Created

----------------------------------------------------------------------------------

-- Include standard logic library

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

-- Entity declaration for the both 2-Way 3-Bit Multiplexer and 3 bit adder to select between Jump Address and next Address

entity Rom\_Address\_Select is

Port (

A : in unsigned(2 downto 0); -- 3-bit input vector A

B : in unsigned (2 downto 0); -- 3-bit input vector B

S : in STD\_LOGIC; -- 1-bit select signal

Y : out unsigned (2 downto 0) -- 3-bit output vector Y

);

end Rom\_Address\_Select;

-- Architecture that defines the behavior of the multiplexer

architecture Behavioral of Rom\_Address\_Select is

begin

-- Multiplexer logic:

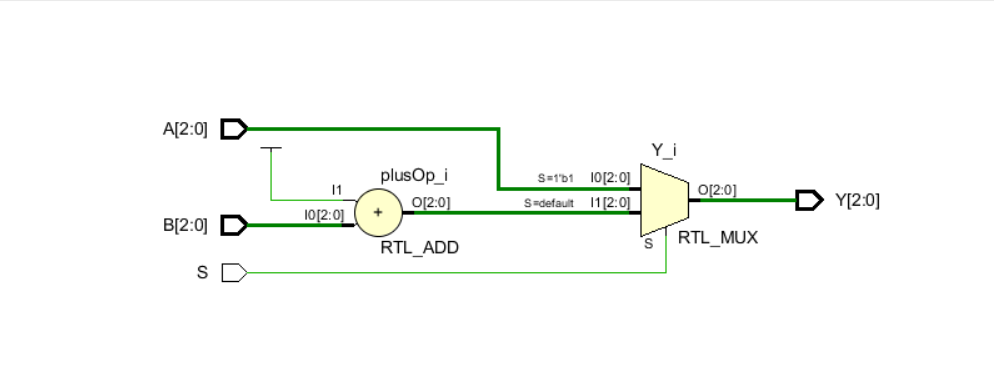
-- When select signal S = '1', output Y is assigned input A

-- When select signal S = '0', output Y is assigned input B incremented by 1

Y <= A when (S='1') else B + 1;

end Behavioral;

### 13.2. Elaborate design



### 13.3. Simulation File

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

-- Create Date: 05/26/2025 07:11:55 AM

-- Design Name:

-- Module Name: Sim\_Rom\_Address\_Select - Behavioral

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description: Testbench for Rom\_Address\_Select module

-- Dependencies: Requires the Rom\_Address\_Select component

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL; -- Required for using 'unsigned' type and arithmetic

-- Entity declaration for the testbench

entity Sim\_Rom\_Address\_Select is

-- No ports are required for the testbench

end Sim\_Rom\_Address\_Select;

architecture Behavioral of Sim\_Rom\_Address\_Select is

-- Declaration of the component under test

component Rom\_Address\_Select

Port (

A : in unsigned(2 downto 0); -- 3-bit input address A

B : in unsigned(2 downto 0); -- 3-bit input address B

S : in STD\_LOGIC; -- Select signal (if '0' select A, else select B)

Y : out unsigned(2 downto 0) -- 3-bit output address Y

);

end component;

-- Internal signals to connect to the DUT (Device Under Test)

signal A, B, Y : unsigned(2 downto 0); -- 3-bit unsigned vectors

signal S : STD\_LOGIC; -- Select signal

begin

-- Instantiation of the Rom\_Address\_Select module

uut: Rom\_Address\_Select

port map (

A => A,

B => B,

S => S,

Y => Y

);

-- Test process to apply stimulus to the module

process

begin

-- Initial inputs

A <= "110"; -- Set A to 6

B <= "000"; -- Set B to 0

S <= '0'; -- Select B (Y should become B = "000")

wait for 100 ns;

-- Change value of B while S is still '0'

B <= "011"; -- Set B to 3 (Y should be B = "011")

wait for 100 ns;

-- Change S to select A instead (Y should become A = "110")

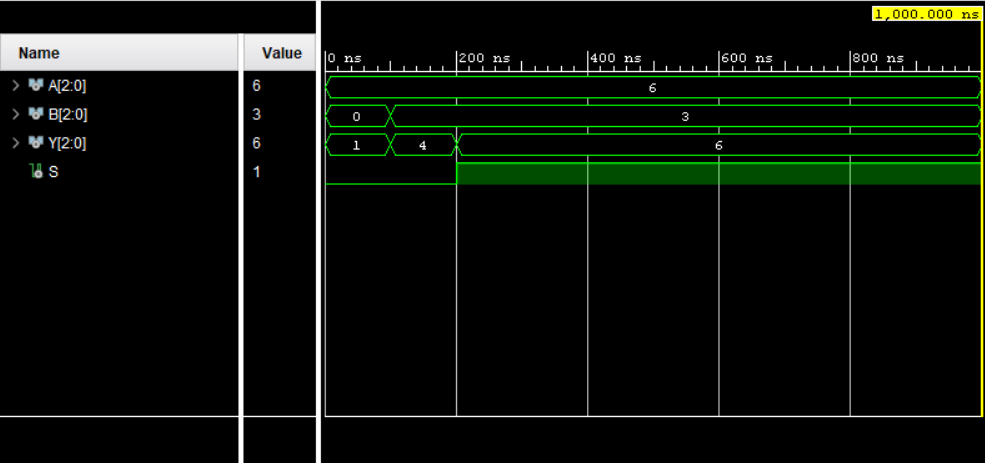
S <= '1'; -- Now select A (Y should become "110")

wait;

end process;

end Behavioral;

### 13.4. Timing Diagram



# Optimization

**Slow Clock**  
The code uses the most significant bit (MSB) of a 26-bit counter (counter(25)) as the divided clock signal. By using the MSB instead of comparing the counter to a fixed value and toggling the signal, we can generate a square wave at a frequency of Clk\_in / 2^26. This approach is beneficial because it conserves logic resources by eliminating the need for complex conditional logic or additional flip-flops.

**Program Counter**  
We utilized the UNSIGNED type from IEEE.NUMERIC\_STD for arithmetic operations and value management. This approach eliminates type mismatch issues and casting that occur with STD\_LOGIC\_VECTOR, facilitating efficient synthesis and simulation with native numeric behavior.

**Mux 2 Way 3-Bit and 3-Bit Adder (Rom\_Address\_select)**  
Y <= A when (S='1') else B + 1;  
The entire multiplexer (MUX) logic is implemented in a single line using a conditional signal assignment. This approach minimizes logic by avoiding extra processes or nested if statements, promoting combinational synthesis and clear timing. It allows synthesis tools to easily infer a MUX and an adder.

Additionally, we have included a 3-bit adder to further reduce combinational logic. We utilize unsigned integers to directly apply the addition operator (+) to increment the input by 1.

**Register Bank**

case (I) is

when "000" => Y <= "00000001"; -- Output 0 active

when "001" => Y <= "00000010"; -- Output 1 active

when "010" => Y <= "00000100"; -- Output 2 active

when "011" => Y <= "00001000"; -- Output 3 active

when "100" => Y <= "00010000"; -- Output 4 active

when "101" => Y <= "00100000"; -- Output 5 active

when "110" => Y <= "01000000"; -- Output 6 active

when others => Y <= "10000000"; -- Covers "111" case: Output 7 active

end case;

We used a case statement to decode the 3-bit input in Register Bank 3 to an 8-decoder. This approach synthesizes efficiently into logic, typically resulting in a decoder tree. It allows synthesis tools to infer minimal logic paths more easily and leads to cleaner, more maintainable code compared to using a series of if-else statements.

In the register within the register bank, we chose not to use 4 D Flip-Flops for data storage. Instead, we directly assigned the data upon the rising edge of the clock signal. This method helps minimize the use of logic components.

**Adder/Subtractor Unit**  
We combined addition and subtraction using a single adder block by modifying the inputs based on the ADD\_SUB control signal. This approach saves hardware resources by avoiding separate adder and subtractor modules and implements subtraction using:

A\_mod <= A when (ADD\_SUB = '0') else not(A);

B\_mod <= B when (ADD\_SUB = '0') else (others => '0');

temp\_sum <= ('0' & unsigned(A\_mod)) + unsigned(B\_mod) + ("0000" & ADD\_SUB);

We did not use the RCA 4 to add 4-bit numbers, which helps minimize the use of logic expressions.

# Constrain File

## This file is a general .xdc for the Basys3 rev B board

## To use it in a project:

## - uncomment the lines corresponding to used pins

## - rename the used ports (in each line, after get\_ports) according to the top level signal names in the project

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports Clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports Clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports Clk]

## Switches

#set\_property PACKAGE\_PIN V17 [get\_ports {sw[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

#set\_property PACKAGE\_PIN V16 [get\_ports {sw[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

#set\_property PACKAGE\_PIN W16 [get\_ports {sw[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

#set\_property PACKAGE\_PIN W17 [get\_ports {sw[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

#set\_property PACKAGE\_PIN W15 [get\_ports {sw[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[4]}]

#set\_property PACKAGE\_PIN V15 [get\_ports {sw[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[5]}]

#set\_property PACKAGE\_PIN W14 [get\_ports {sw[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[6]}]

#set\_property PACKAGE\_PIN W13 [get\_ports {sw[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[7]}]

#set\_property PACKAGE\_PIN V2 [get\_ports {sw[8]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[8]}]

#set\_property PACKAGE\_PIN T3 [get\_ports {sw[9]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[9]}]

#set\_property PACKAGE\_PIN T2 [get\_ports {sw[10]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[10]}]

#set\_property PACKAGE\_PIN R3 [get\_ports {sw[11]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[11]}]

#set\_property PACKAGE\_PIN W2 [get\_ports {sw[12]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[12]}]

#set\_property PACKAGE\_PIN U1 [get\_ports {sw[13]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[13]}]

#set\_property PACKAGE\_PIN T1 [get\_ports {sw[14]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[14]}]

#set\_property PACKAGE\_PIN R2 [get\_ports {sw[15]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[15]}]

## LEDs

set\_property PACKAGE\_PIN U16 [get\_ports {Reg\_7\_out[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Reg\_7\_out[0]}]

set\_property PACKAGE\_PIN E19 [get\_ports {Reg\_7\_out[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Reg\_7\_out[1]}]

set\_property PACKAGE\_PIN U19 [get\_ports {Reg\_7\_out[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Reg\_7\_out[2]}]

set\_property PACKAGE\_PIN V19 [get\_ports {Reg\_7\_out[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Reg\_7\_out[3]}]

#set\_property PACKAGE\_PIN W18 [get\_ports {led[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[4]}]

#set\_property PACKAGE\_PIN U15 [get\_ports {led[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[5]}]

#set\_property PACKAGE\_PIN U14 [get\_ports {led[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[6]}]

#set\_property PACKAGE\_PIN V14 [get\_ports {led[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[7]}]

#set\_property PACKAGE\_PIN V13 [get\_ports {led[8]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[8]}]

#set\_property PACKAGE\_PIN V3 [get\_ports {led[9]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[9]}]

#set\_property PACKAGE\_PIN W3 [get\_ports {led[10]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[10]}]

#set\_property PACKAGE\_PIN U3 [get\_ports {led[11]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[11]}]

#set\_property PACKAGE\_PIN P3 [get\_ports {led[12]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[12]}]

#set\_property PACKAGE\_PIN N3 [get\_ports {led[13]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {led[13]}]

set\_property PACKAGE\_PIN P1 [get\_ports {Zero}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Zero}]

set\_property PACKAGE\_PIN L1 [get\_ports {OverFlow}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {OverFlow}]

##7 segment display

set\_property PACKAGE\_PIN W7 [get\_ports {Display\_7\_Segment[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[0]}]

set\_property PACKAGE\_PIN W6 [get\_ports {Display\_7\_Segment[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[1]}]

set\_property PACKAGE\_PIN U8 [get\_ports {Display\_7\_Segment[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[2]}]

set\_property PACKAGE\_PIN V8 [get\_ports {Display\_7\_Segment[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[3]}]

set\_property PACKAGE\_PIN U5 [get\_ports {Display\_7\_Segment[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[4]}]

set\_property PACKAGE\_PIN V5 [get\_ports {Display\_7\_Segment[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[5]}]

set\_property PACKAGE\_PIN U7 [get\_ports {Display\_7\_Segment[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Display\_7\_Segment[6]}]

#set\_property PACKAGE\_PIN V7 [get\_ports dp]

#set\_property IOSTANDARD LVCMOS33 [get\_ports dp]

set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]

set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]

set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]

##Buttons

set\_property PACKAGE\_PIN U18 [get\_ports Res]

set\_property IOSTANDARD LVCMOS33 [get\_ports Res]

#set\_property PACKAGE\_PIN T18 [get\_ports btnU]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnU]

#set\_property PACKAGE\_PIN W19 [get\_ports btnL]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnL]

#set\_property PACKAGE\_PIN T17 [get\_ports btnR]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnR]

#set\_property PACKAGE\_PIN U17 [get\_ports btnD]

#set\_property IOSTANDARD LVCMOS33 [get\_ports btnD]

##Pmod Header JA

##Sch name = JA1

#set\_property PACKAGE\_PIN J1 [get\_ports {JA[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[0]}]

##Sch name = JA2

#set\_property PACKAGE\_PIN L2 [get\_ports {JA[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[1]}]

##Sch name = JA3

#set\_property PACKAGE\_PIN J2 [get\_ports {JA[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[2]}]

##Sch name = JA4

#set\_property PACKAGE\_PIN G2 [get\_ports {JA[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[3]}]

##Sch name = JA7

#set\_property PACKAGE\_PIN H1 [get\_ports {JA[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[4]}]

##Sch name = JA8

#set\_property PACKAGE\_PIN K2 [get\_ports {JA[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[5]}]

##Sch name = JA9

#set\_property PACKAGE\_PIN H2 [get\_ports {JA[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[6]}]

##Sch name = JA10

#set\_property PACKAGE\_PIN G3 [get\_ports {JA[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JA[7]}]

##Pmod Header JB

##Sch name = JB1

#set\_property PACKAGE\_PIN A14 [get\_ports {JB[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[0]}]

##Sch name = JB2

#set\_property PACKAGE\_PIN A16 [get\_ports {JB[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[1]}]

##Sch name = JB3

#set\_property PACKAGE\_PIN B15 [get\_ports {JB[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[2]}]

##Sch name = JB4

#set\_property PACKAGE\_PIN B16 [get\_ports {JB[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[3]}]

##Sch name = JB7

#set\_property PACKAGE\_PIN A15 [get\_ports {JB[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[4]}]

##Sch name = JB8

#set\_property PACKAGE\_PIN A17 [get\_ports {JB[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[5]}]

##Sch name = JB9

#set\_property PACKAGE\_PIN C15 [get\_ports {JB[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[6]}]

##Sch name = JB10

#set\_property PACKAGE\_PIN C16 [get\_ports {JB[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JB[7]}]

##Pmod Header JC

##Sch name = JC1

#set\_property PACKAGE\_PIN K17 [get\_ports {JC[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[0]}]

##Sch name = JC2

#set\_property PACKAGE\_PIN M18 [get\_ports {JC[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[1]}]

##Sch name = JC3

#set\_property PACKAGE\_PIN N17 [get\_ports {JC[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[2]}]

##Sch name = JC4

#set\_property PACKAGE\_PIN P18 [get\_ports {JC[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[3]}]

##Sch name = JC7

#set\_property PACKAGE\_PIN L17 [get\_ports {JC[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[4]}]

##Sch name = JC8

#set\_property PACKAGE\_PIN M19 [get\_ports {JC[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[5]}]

##Sch name = JC9

#set\_property PACKAGE\_PIN P17 [get\_ports {JC[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[6]}]

##Sch name = JC10

#set\_property PACKAGE\_PIN R18 [get\_ports {JC[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JC[7]}]

##Pmod Header JXADC

##Sch name = XA1\_P

#set\_property PACKAGE\_PIN J3 [get\_ports {JXADC[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[0]}]

##Sch name = XA2\_P

#set\_property PACKAGE\_PIN L3 [get\_ports {JXADC[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[1]}]

##Sch name = XA3\_P

#set\_property PACKAGE\_PIN M2 [get\_ports {JXADC[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[2]}]

##Sch name = XA4\_P

#set\_property PACKAGE\_PIN N2 [get\_ports {JXADC[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[3]}]

##Sch name = XA1\_N

#set\_property PACKAGE\_PIN K3 [get\_ports {JXADC[4]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[4]}]

##Sch name = XA2\_N

#set\_property PACKAGE\_PIN M3 [get\_ports {JXADC[5]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[5]}]

##Sch name = XA3\_N

#set\_property PACKAGE\_PIN M1 [get\_ports {JXADC[6]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[6]}]

##Sch name = XA4\_N

#set\_property PACKAGE\_PIN N1 [get\_ports {JXADC[7]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {JXADC[7]}]

##VGA Connector

#set\_property PACKAGE\_PIN G19 [get\_ports {vgaRed[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaRed[0]}]

#set\_property PACKAGE\_PIN H19 [get\_ports {vgaRed[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaRed[1]}]

#set\_property PACKAGE\_PIN J19 [get\_ports {vgaRed[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaRed[2]}]

#set\_property PACKAGE\_PIN N19 [get\_ports {vgaRed[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaRed[3]}]

#set\_property PACKAGE\_PIN N18 [get\_ports {vgaBlue[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaBlue[0]}]

#set\_property PACKAGE\_PIN L18 [get\_ports {vgaBlue[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaBlue[1]}]

#set\_property PACKAGE\_PIN K18 [get\_ports {vgaBlue[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaBlue[2]}]

#set\_property PACKAGE\_PIN J18 [get\_ports {vgaBlue[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaBlue[3]}]

#set\_property PACKAGE\_PIN J17 [get\_ports {vgaGreen[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaGreen[0]}]

#set\_property PACKAGE\_PIN H17 [get\_ports {vgaGreen[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaGreen[1]}]

#set\_property PACKAGE\_PIN G17 [get\_ports {vgaGreen[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaGreen[2]}]

#set\_property PACKAGE\_PIN D17 [get\_ports {vgaGreen[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {vgaGreen[3]}]

#set\_property PACKAGE\_PIN P19 [get\_ports Hsync]

#set\_property IOSTANDARD LVCMOS33 [get\_ports Hsync]

#set\_property PACKAGE\_PIN R19 [get\_ports Vsync]

#set\_property IOSTANDARD LVCMOS33 [get\_ports Vsync]

##USB-RS232 Interface

#set\_property PACKAGE\_PIN B18 [get\_ports RsRx]

#set\_property IOSTANDARD LVCMOS33 [get\_ports RsRx]

#set\_property PACKAGE\_PIN A18 [get\_ports RsTx]

#set\_property IOSTANDARD LVCMOS33 [get\_ports RsTx]

##USB HID (PS/2)

#set\_property PACKAGE\_PIN C17 [get\_ports PS2Clk]

#set\_property IOSTANDARD LVCMOS33 [get\_ports PS2Clk]

#set\_property PULLUP true [get\_ports PS2Clk]

#set\_property PACKAGE\_PIN B17 [get\_ports PS2Data]

#set\_property IOSTANDARD LVCMOS33 [get\_ports PS2Data]

#set\_property PULLUP true [get\_ports PS2Data]

##Quad SPI Flash

##Note that CCLK\_0 cannot be placed in 7 series devices. You can access it using the

##STARTUPE2 primitive.

#set\_property PACKAGE\_PIN D18 [get\_ports {QspiDB[0]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {QspiDB[0]}]

#set\_property PACKAGE\_PIN D19 [get\_ports {QspiDB[1]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {QspiDB[1]}]

#set\_property PACKAGE\_PIN G18 [get\_ports {QspiDB[2]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {QspiDB[2]}]

#set\_property PACKAGE\_PIN F18 [get\_ports {QspiDB[3]}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {QspiDB[3]}]

#set\_property PACKAGE\_PIN K19 [get\_ports QspiCSn]

#set\_property IOSTANDARD LVCMOS33 [get\_ports QspiCSn]

# Workload Distribution

Wikramaratna S. I. P.

… Final Nano Processor Design

… Register Bank

… Instruction Decoder

… LUT\_16\_7

de Silva N. N. B.

… Adder Substractor 4 bit unit

… Program Counter

… Presentation

Bandara S. A. N. K.

… Mux 2 way 4 bit

… Mux 8 way 4 bit

… Lab Report

Padmasiri G.R.H.D.

… Rom Address Select

… Program Rom

# Difficulties we face

During the development of our nanoprocessor, we encountered several technical and collaborative challenges. Below are the key difficulties we faced and how we addressed them:

**1.Port Connection Issues**

In the initial stages of the project, we faced difficulties in establishing proper connections between ports. Although the ports were connected physically, the communication was not functioning as expected due to incorrect configurations. After multiple attempts and adjustments, we were eventually able to establish the connections correctly.

**2.High Utilization of Lookup Tables (LUTs)**

At the beginning, our design resulted in a very high usage of lookup tables, which affected performance and scalability. We carried out several rounds of optimization, including logic simplification and resource sharing. Through these efforts, we successfully reduced LUT usage by more than 50%.

**3.File Sharing and Version Control Issues**

Initially, we shared project files using WhatsApp. However, this method led to multiple problems, such as the presence of duplicate files and corrupted or outdated versions. To resolve this, we created a GitHub repository, which significantly streamlined collaboration. Using GitHub enabled better version control, easier file sharing, and efficient distribution of tasks among team members.

**4.Missing Components in Development Machines**

At times, certain necessary software components or tools were missing from the development environments on some team members’ machines. These issues were resolved collaboratively using GitHub, where updated dependencies and installation guides were made accessible to everyone, ensuring consistency across all development systems.

# Conclusion

The successful completion of the Nano Processor lab project has greatly enhanced our understanding of digital system design and the practical implementation of processor architecture. Throughout this lab course, we developed each essential component systematically—from fundamental modules like the slow clock and multiplexers to more complex systems such as the instruction decoder, register bank, and the integrated nano processor itself. This modular design approach not only reinforced theoretical concepts but also improved my skills in VHDL programming, simulation practices, and system-level thinking.

The processor’s ability to accurately execute instructions such as MOVI, ADD, NEG, and JZR clearly demonstrated the correctness and functionality of the architecture and instruction decoding. Additionally, the use of a slow clock and reset mechanisms allowed for detailed, step-by-step observation of instruction execution, facilitating a deeper analysis of control signal behavior and data flow within the processor.

Overall, this project provided us with valuable hands-on experience and practical insights into the design and debugging of digital systems. It also highlighted the importance of precise planning, iterative testing, and effective collaboration in successfully realizing complex hardware systems.

Group 37

230703N – Wikramaratna S. I. P.

230126X – de Silva N. N. B.

230081D – Bandara S. A. N. K.

230453V - Padmasiri G.R.H.D.