EEDG/CE 6302

Microprocessors and Embedded Systems

Electrical and Computer Engineering

Erik Jonsson School of Engineering and Computer Science

The University of Texas at Dallas

Dr. Tooraj Nikoubin

Project 1 (Part 2): Register File, Multiplexor and ALU

Submitted By:

Muripa Uppaluri (MXU220008)

Chandanam Sai Nived (SXC210186)

Table of Contents

Content	Page Number
Multiplexor	**3**
Schematic View and Waveforms of Multiplexor	**3**
Design Summary Report	**5**
8 8bit Registers	**5**
Schematic View and Waveforms of Registers	**6**
ALU	**7**
Schematic View and Waveforms of ALU	**7**
Design Summary Report	**8**
Lab Handout Questions and Answers	**9**

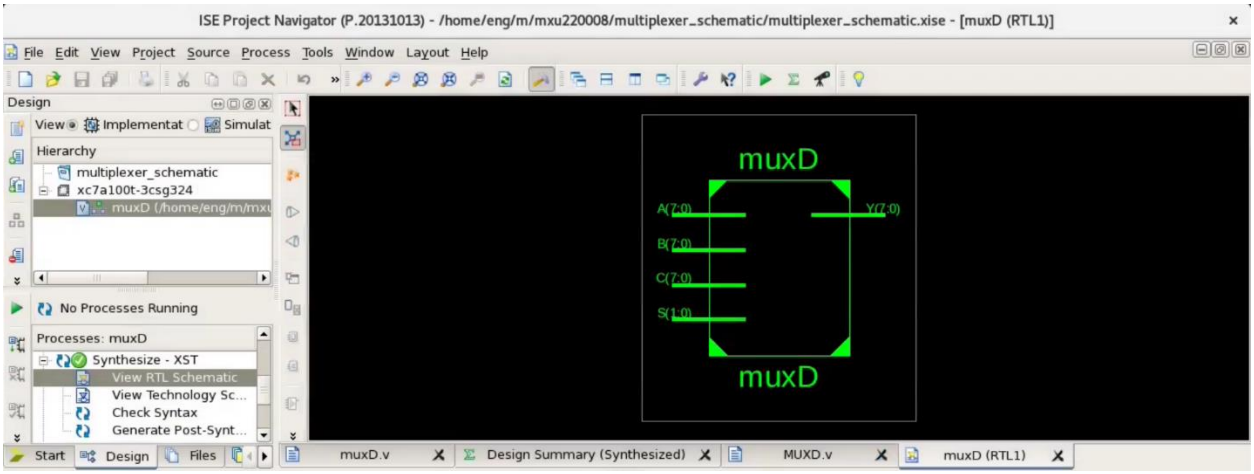
In this second part of first lab project, we've designed a 4 multiplexors, 8 8bit registers and an ALU. We will be explaining about the implementation of each in the following sections.

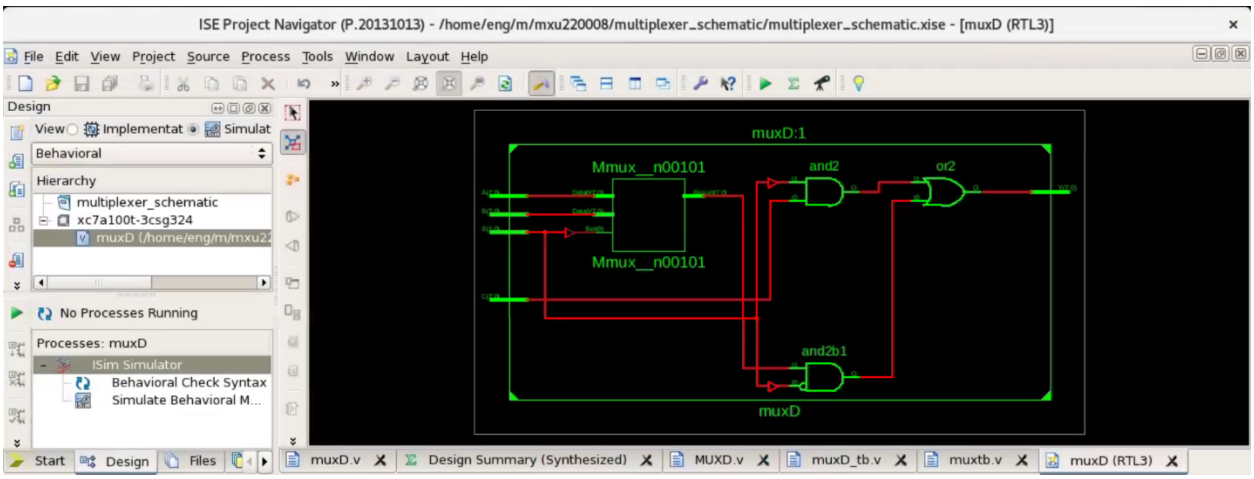
Multiplexor:

A multiplexor transfers data from one of the N inputs to a single output based on the select lines. In this project, we have 4 multiplexers, MUXA, MUXB, MUXC, MUXD. MUX A and MUX B have 2 input ports each and each input port is 8 bit wide. Since we have 2 input ports, we considered 1 select line that would select between the input ports and transfer the data to output port Y. MUXC has 4 input ports with 8 bits each and 2 select lines to select between the 4 different data input ports. MUXD has 3 input ports and 2 select lines.

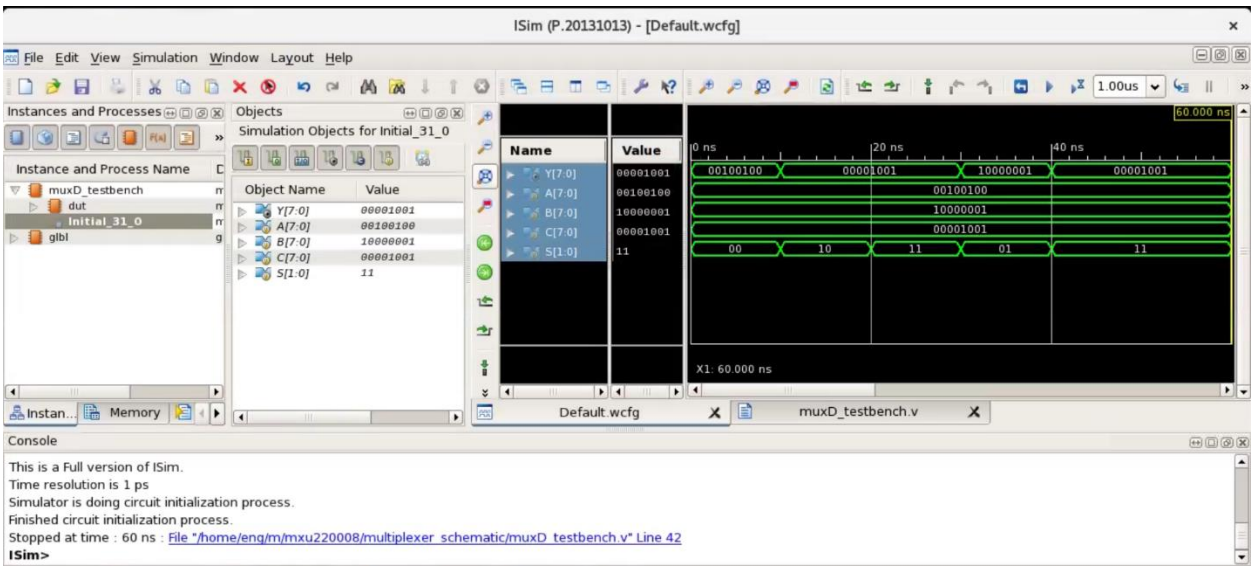
We've created a simple testbench to verify this multiplexor. In this testbench module, we've declared all **the input ports as reg datatype** so that they can hold the changing values of stimulus and the output port as wire data type. The device under test is instantiated using **port-map instantiation**. We are driving random values to the input ports using $random and then changing the value of select lines to check the outputs. After checking all values, $finish stops the simulation.

Schematic view of the Multiplexor –

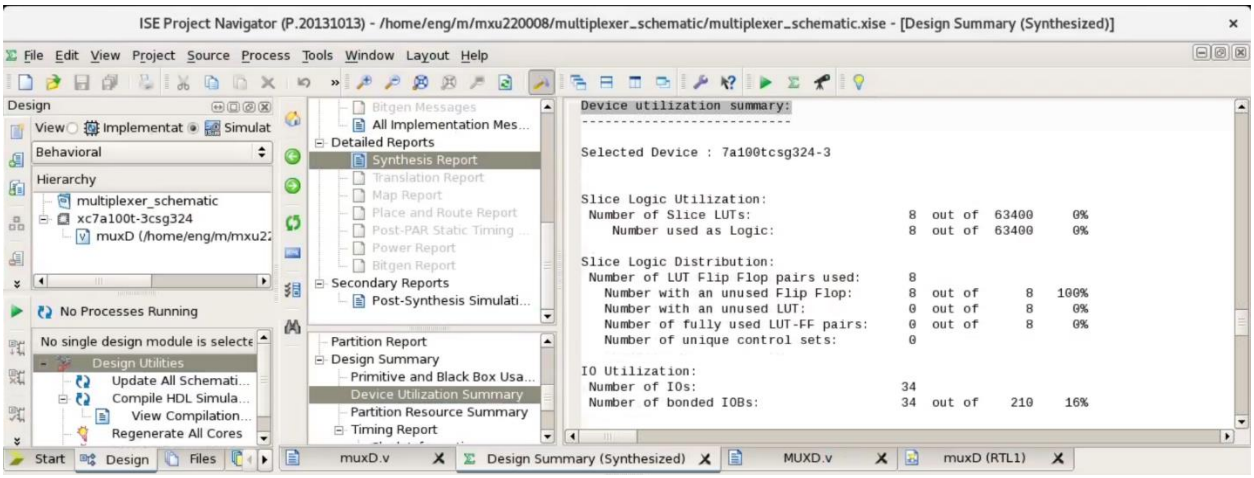


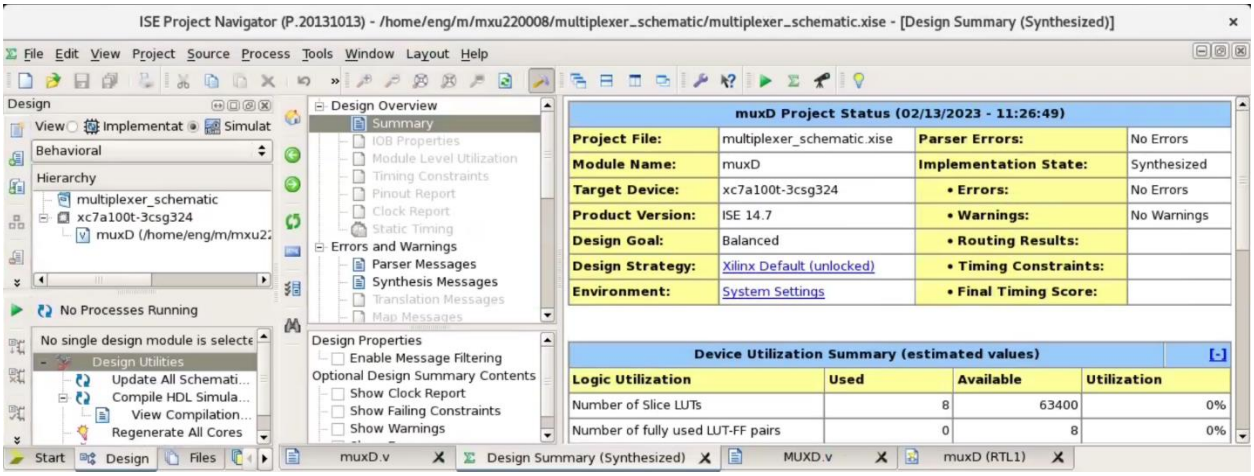


Multiplexor Waveform –



Design Summary Report:





8 8bit Registers Implementation:

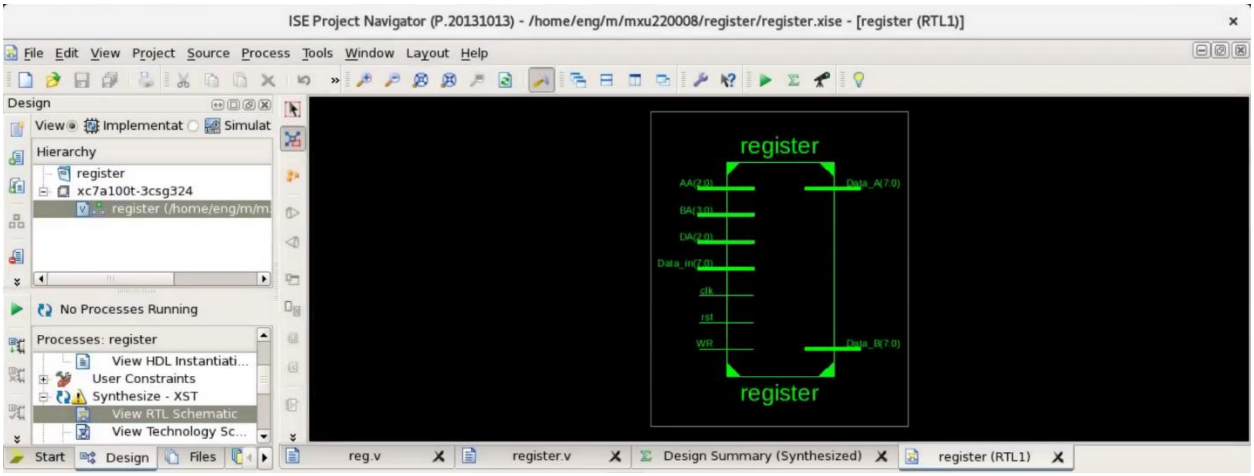
In this design we have 8 registers, each 8 bit wide to store the required data for MCU operation. We have a clock signal and reset signal that serve as inputs to the registers. We also have a single bit write/read enable signal called WR signal, which when asserted means a write operation and de-asserted means a read operation. During a write operation, the data from Data_in port is written into one of the registers selected by DA port whereas during a read operation, data from the registers selected using AA and BA input ports is transferred to the Data_A and Data_B terminals respectively. In this design, register R0 is always tied to 0.

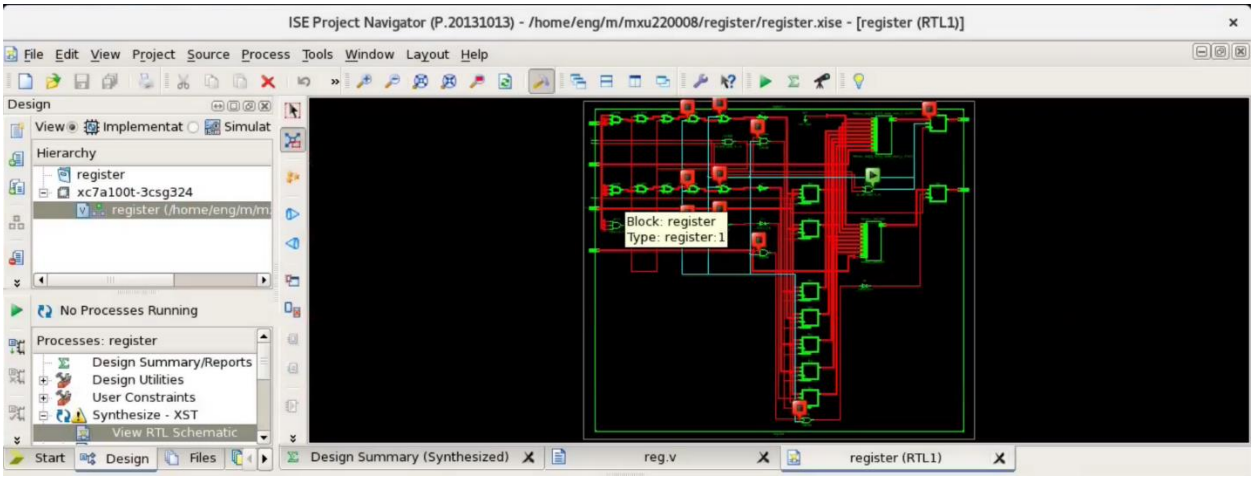
Register Testbench –

We've created a simple testbench to verify this register. In this testbench module, we've declared all the input ports as reg datatype so that they can hold the changing values of stimulus and the output port as wire data type. The device under test is instantiated using port-map instantiation. The testbench logic is

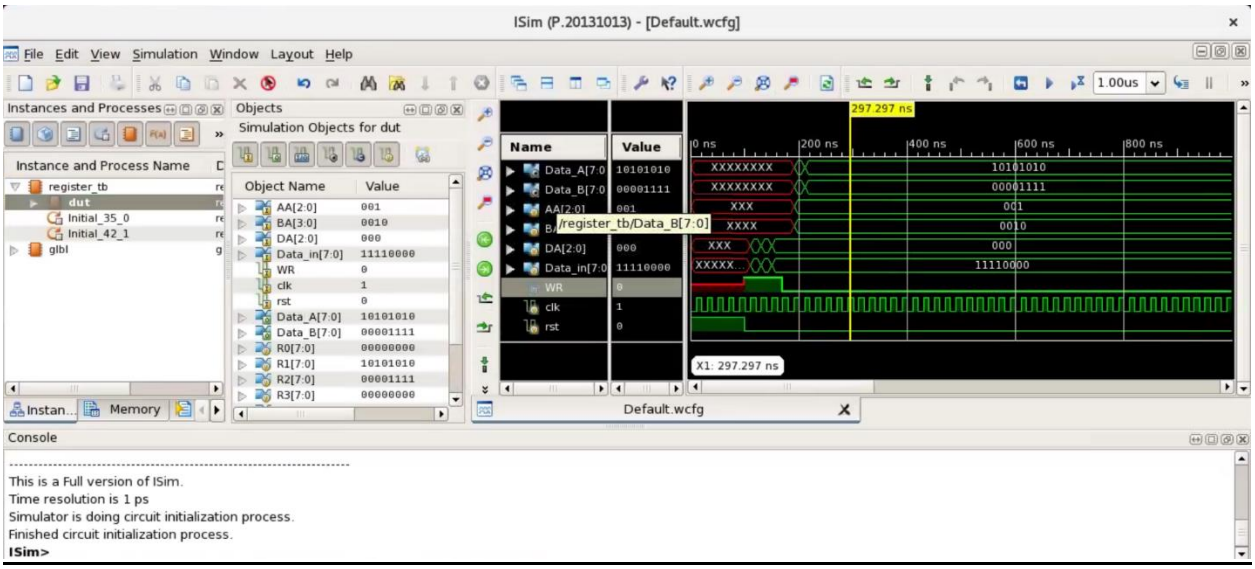
simple, when the system is out of reset and WR is asserted, we drive random data on DA to select a register and random data on Data_in to write this data into the register selected. When WR is deasserted, we drive random data onto AA and BA and check the data on Data_A and Data_B terminals respectively.

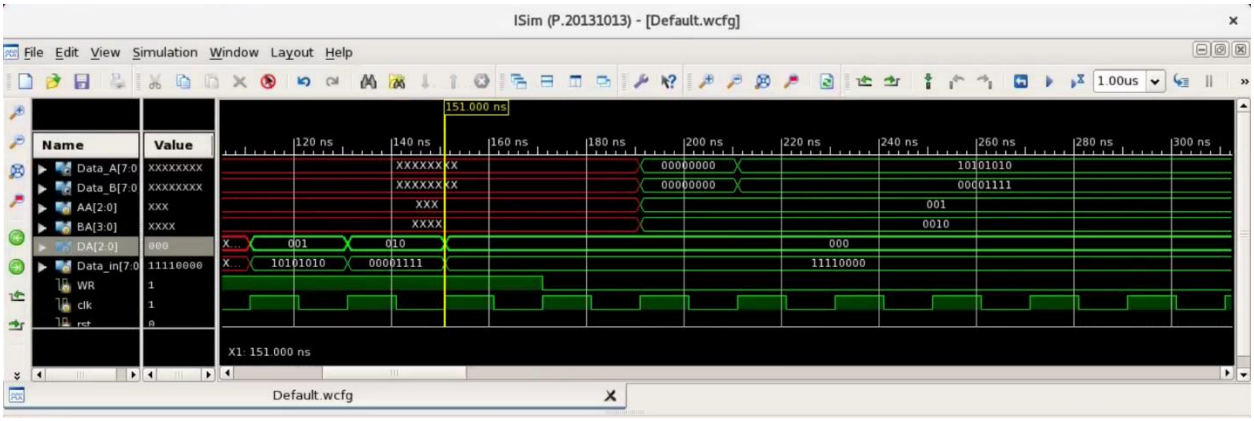
Schematic View of Registers –





Register waveform –

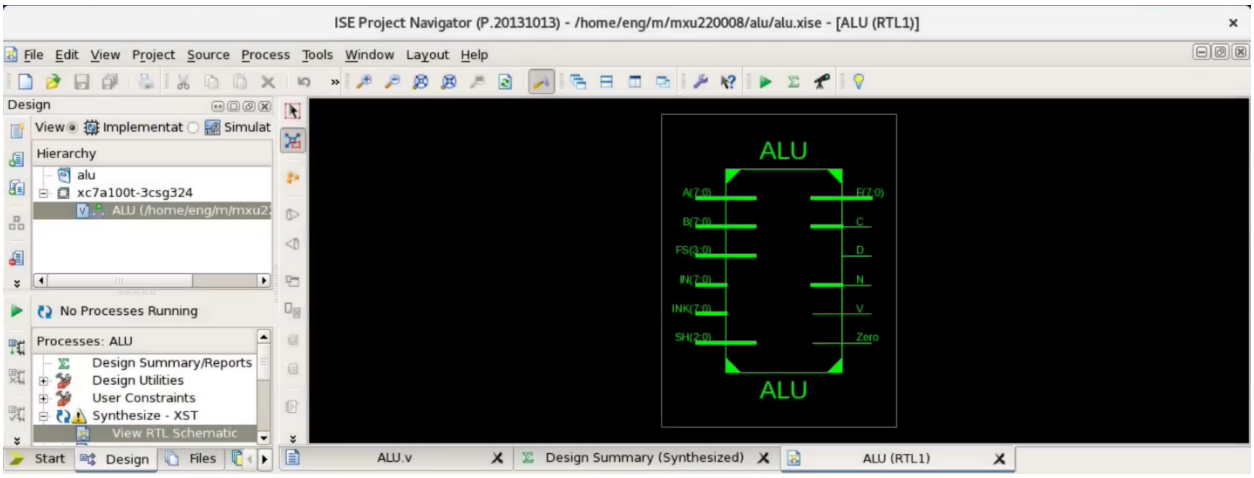




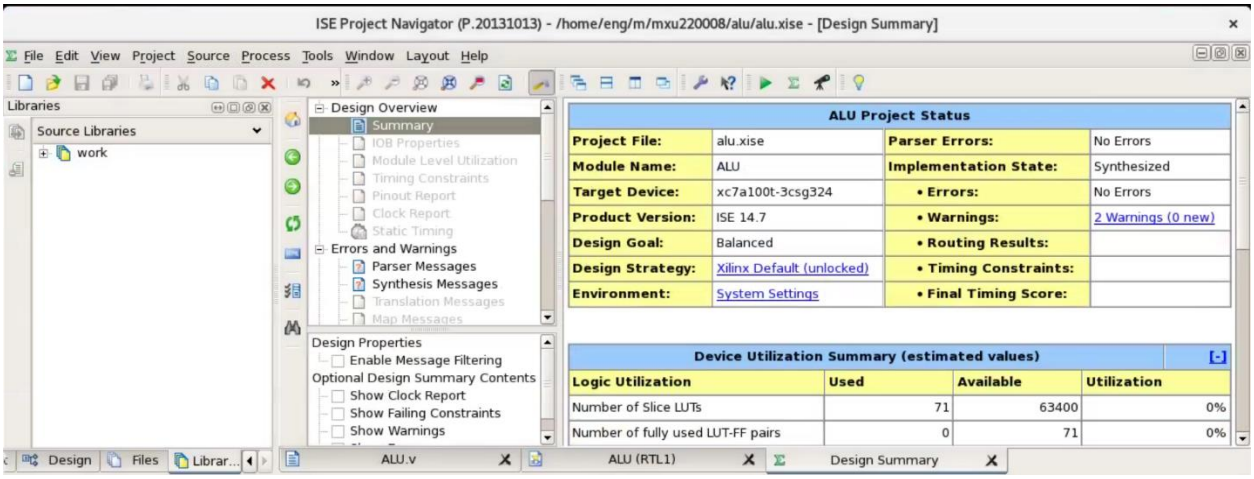
ALU –

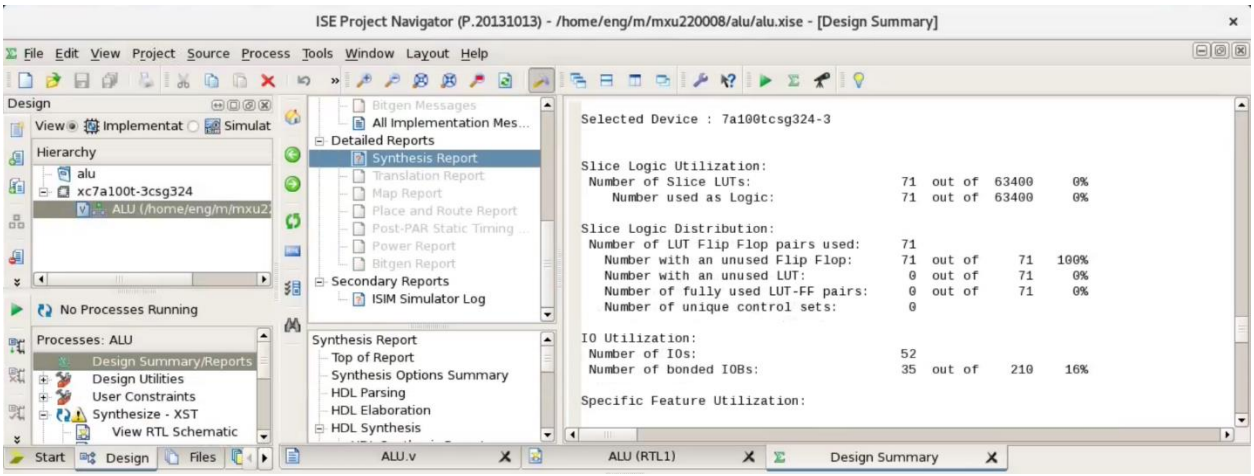
We have a pure combinational ALU in this design. The instruction is decoded based on the opcode and the operands are used for data interpretation. We have 5 status signals that output from ALU and provide the status of carry, negative result, zero result, etc. A testbench is written to verify the decoding of ALU with random opcodes.

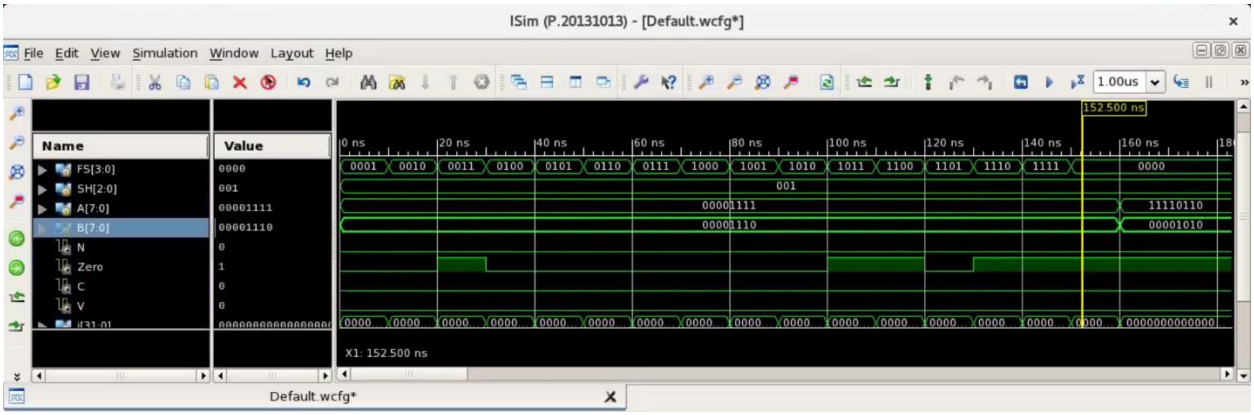
ALU Schematic Diagram:



ALU Design Summary Report:







Lab Handout Questions

1. While it is possible to implement an ALU using if-then-else constructs in HDLs, it is generally discouraged due to potential performance and maintenance issues. An if-then-else construct can result in longer and more complex code, as well as slower execution time compared to a switch-case or with-select construct. Additionally, if the number of input options for the ALU is high, using if-then-else can become cumbersome and hard to maintain.

2. Overflow in arithmetic operation occurs when the result of an arithmetic operation exceeds the range of values that can be represented by the given number of bits. In two's complement arithmetic, overflow can occur when the result of adding two positive numbers is negative, or when the result of adding two negative numbers is positive. There are two types of overflows, signed overflow and unsigned overflow. Signed overflow occurs when the result of an arithmetic operation exceeds the maximum positive or minimum negative value that can be represented by the given number of bits. Unsigned overflow occurs when the result of an arithmetic operation exceeds the maximum value that can be represented by the given number of bits.

3. Both if-then-else and switch-case constructs can be used for implementing multiplexors in HDLs, but the choice depends on personal preference and the specific requirements of the design. Switch-case constructs are generally preferred in Verilog due to their more concise syntax, while with-select is a similar construct available in VHDL. However, if-then-else constructs can also be used for simple multiplexors or for designs with a small number of input options.