Team Members:
Saveetha Venkatesan (SXV200028), Vishnuvaradhan Moganarengam (VXM210090)

EEDG6302: Microprocessor and Embedded Systems

Wednesday Lab Report

Part 2 Task: Designing and simulating synthesizable ALU, Register File and required Multiplexors

Week 2 Summary:

The goal of this week's task is to verify the Verilog module synthesizable ALU (Arithmetic Logic Unit), Register File and required Multiplexors individually.

We have written Verilog module and tested register file module. As given in project description, it takes clk, Address A, B address, D address, 8-bit input data and clock along with write enable. If the write is enabled the input data will be written to address given by D address register. Data output A and B are controlled by address register A and B.

For Multiplexors, we have developed 4 different modules and tested each. Multiplexor A and B has 2 inputs, hence used a 2:1 Multiplexor to build it. We passed random values to check the functionality of the MUX with select lines. Similarly, we built 4:2 multiplexor for Multiplexor C and D. To verify those, we passed the values for inputs and verified with the respective select line outputs.

ALU perform the arithmetic and logical operations. It takes function select as input which determines which operation to be performed. Along with this it takes shift value, two ALU input A and B. it gives 8-bit output F along status flag like carry, zero, negative and overflow. We have included addition, subtraction, multiplication, division, OR, AND, NOR, NAND, NOT, XOR, greater comparison, equal comparison, logical shift left and logical shift right. To test it we have passed arguments in A and B, we get correct result in ALU output.

Problems Encountered During Design:

- We tried to write the code efficiently by using assign statement rather than switch case for MUX.
- To find the logic for different flags of ALU.

1. There are two completely different ways to write ALU in HDLs: (1) using ifthen-else template or (2) using switch-case (VHDL and Verilog) or with-select (only in VHDL) templates; and both are functionally correct. Why do you have to prevent using if-then-else in ALU coding? Explain briefly in 3 to 5 lines.

Using if-then-else statements in ALU coding can lead to problems with readability and maintainability, especially in complex designs. Additionally, if-then-else statements can result in larger, less efficient hardware implementations, compared to other constructs like switch-case or with-select. The latter two constructs are more commonly used in ALU coding because they allow for clear concise, and efficient code that is easier to understand and maintain. However, ultimately the choice between if-then-else and switch-case/with-select depends on the specific requirements of the design, and the design style and preferences of the designer.

2. What does overflow I arithmetic operation mean? When does it happen? When do you use each of different overflows in your ALU design? Explain briefly in 2 to 3 lines.

Overflow in arithmetic operations refers to a situation where the result of an arithmetic operation is too large to be represented in the specified number of bits, leading to loss of information. An overflow occurs when the result of an addition or subtraction operation exceeds the maximum or goes below the minimum representable value for the given number format. For example, in a two's complement

Team Members:
Saveetha Venkatesan (SXV200028), Vishnuvaradhan Moganarengam (VXM210090)

8-bit representation, the range of representable numbers is -128 to 127, so if we add two numbers whose sum is 128 or greater, an overflow will occur.

There are different types of overflows that can be used in ALU design, including:

Wrap-around overflow: In this type of overflow, the result "wraps around" to the minimum or maximum representable value.

Saturation overflow: In this type of overflow, the result is set to the minimum or maximum representable value, regardless of the actual result of the operation.

Flag overflow: In this type of overflow, a flag is set to indicate that an overflow has occurred, without altering the result of the operation.

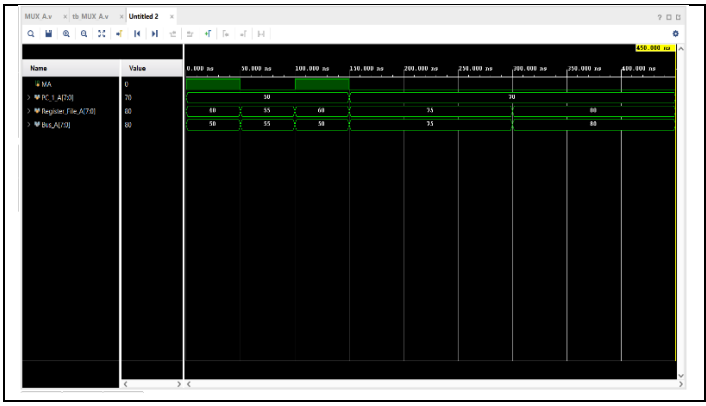
The choice of which type of overflow to use depends on the requirements of the specific design and the desired behaviour when an overflow occurs. For example, wrap-around overflow may be appropriate in some cases where the loss of information is acceptable, while saturation overflow may be more appropriate in others where it is important to limit the range of representable values. Flag overflow is useful when it is important to detect an overflow, but the result should still be used in further computations.

3. Which template do you use for multiplexors? If-then-else or switch-case? Why?

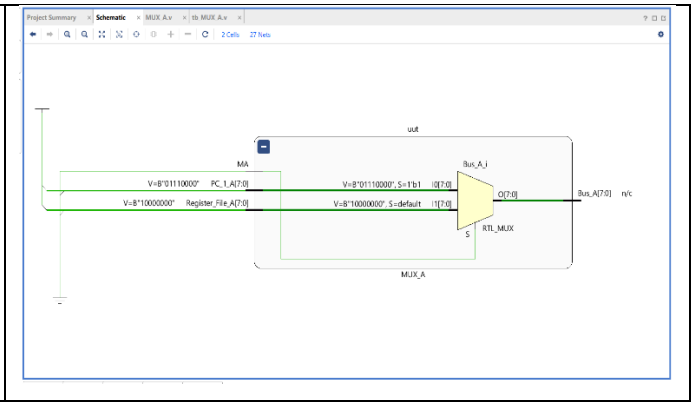
Both if-then-else and switch-case can be used to implement multiplexers in HDLs. However, switch-case provides a clearer and more concise representation of the multiplexer logic. In a switch-case construct, each case statement corresponds to one of the input signals and its corresponding output. The choice of which output to use is determined by the value of the selector signal. This makes the implementation of a multiplexer with a switch-case statement more straightforward and easier to understand than with an equivalent implementation using if-then-else statements.

MUX-A:

Simulation:



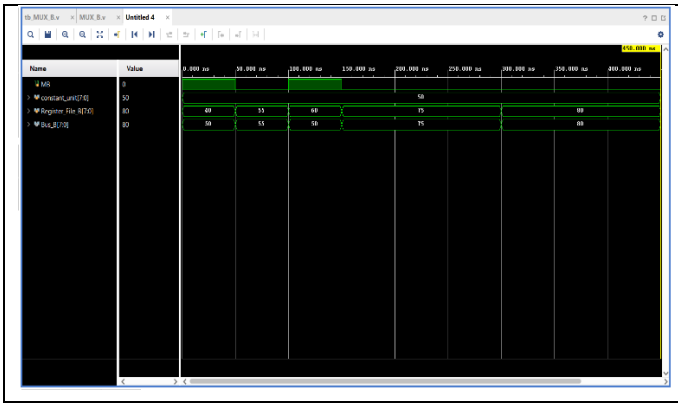
Schematic:



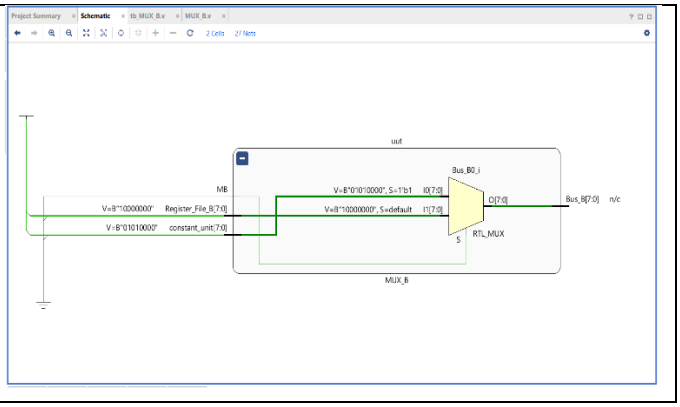
Team Members:
Saveetha Venkatesan (SXV200028), Vishnuvaradhan Moganarengam (VXM210090)

MUX-B:

Simulation:

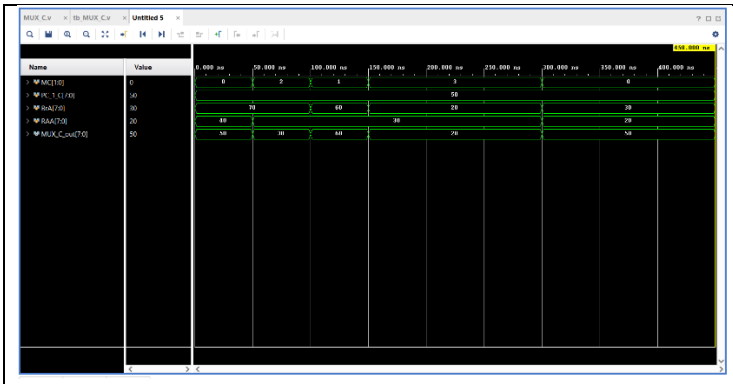


Schematic:

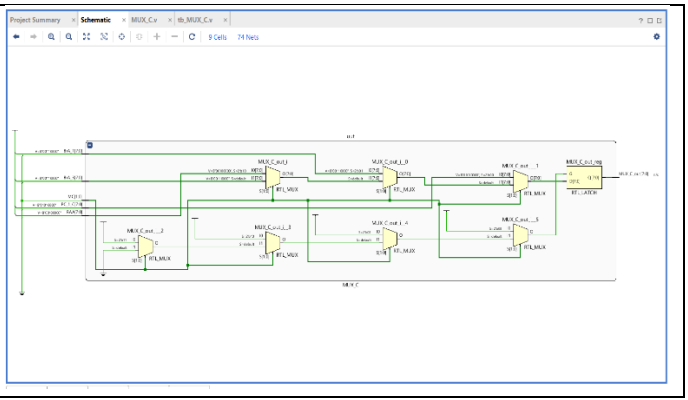


MUX-C:

Simulation:

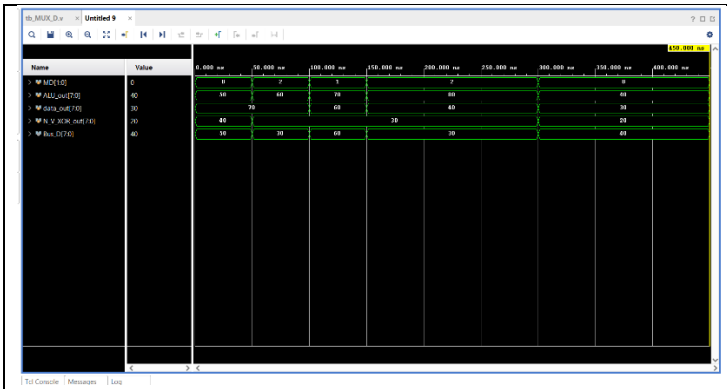


Schematic:

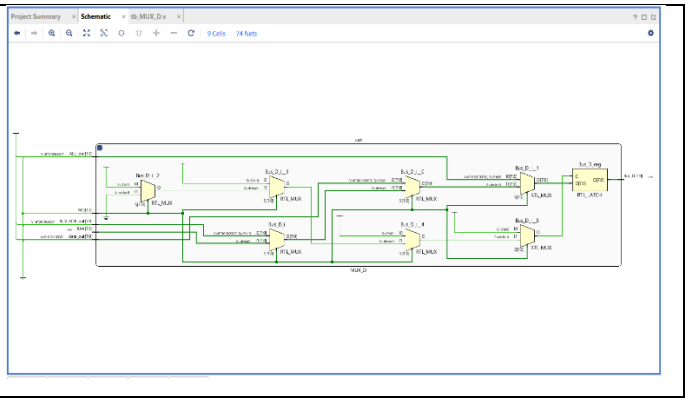


MUX-D

Simulation:



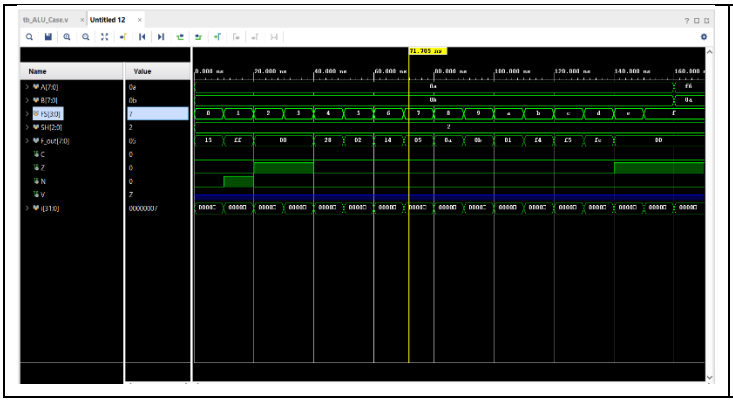
Schematic:



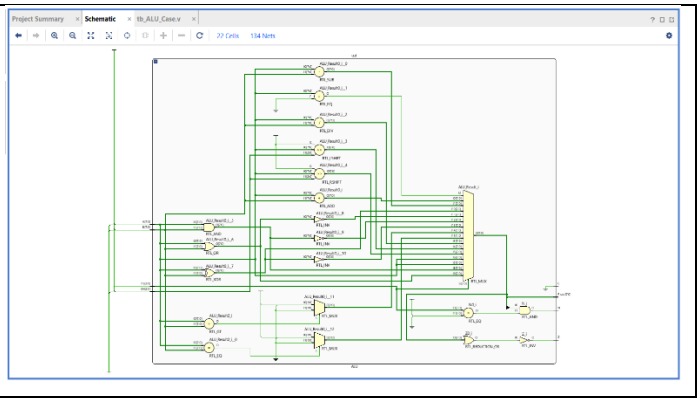
Team Members:
Saveetha Venkatesan (SXV200028), Vishnuvaradhan Moganarengam (VXM210090)

ALU:

Simulation:



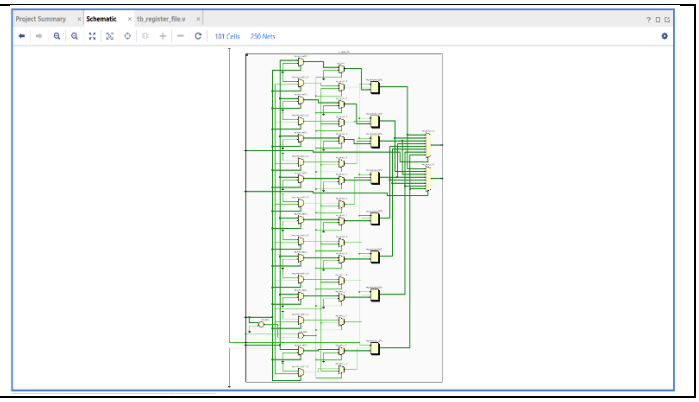
Schematic:



Register File:

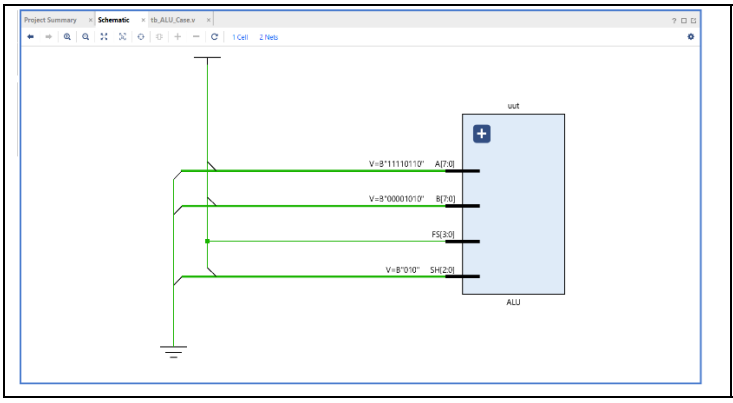
Simulation:

Schematic:

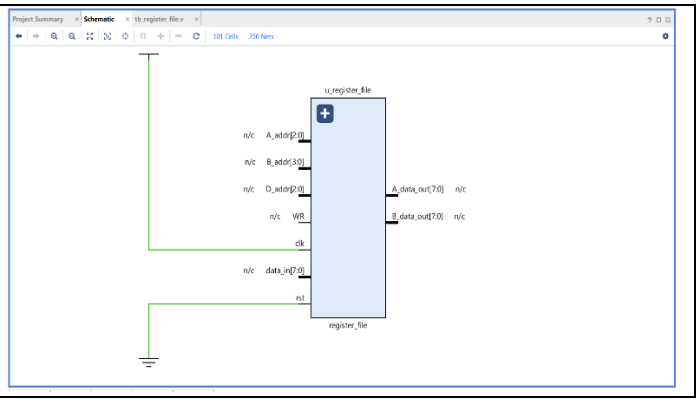


Schematic Top View:

ALU:



Register File:



Team Members:
Saveetha Venkatesan (SXV200028), Vishnuvaradhan Moganarengam (VXM210090)