# ECEN 240 - Lab 6 – Arithmetic Logic Units (ALUs) and Multiplexers

Name: Brodric Young

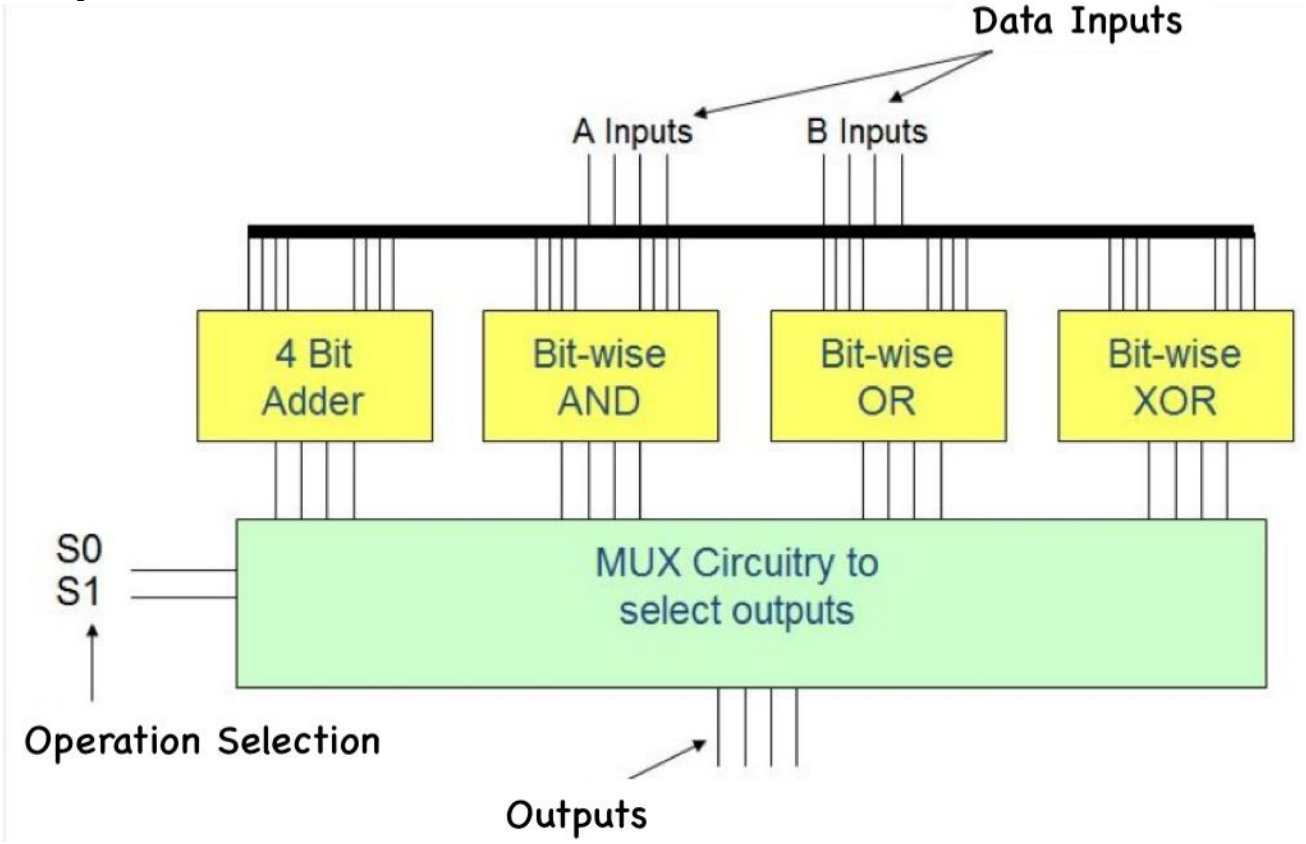
---

## Purposes:

1. Learn how to build multiple-bit multiplexers.
2. Become familiar with the design and operation of common ALU components.
3. Learn how to design complex circuits using a hierarchical approach.
4. Learn about the basic functionality of an ALU
5. Learn the SystemVerilog "Dataflow" programming style (assign statements)

## Procedure:

In this lab, you will design and built an ALU that can take two, four-bit numbers (nibbles) as inputs and generate one four-bit number as the output. The ALU will be able to take the two input nibbles and ADD them together, OR them together, AND them together, or XOR them together. The output multiplexer will select only one of these four functions to send to the output of the ALU. The block diagram of the completed ALU is shown below:



# Lab 6 Part 1 - Getting started with the ALU circuit

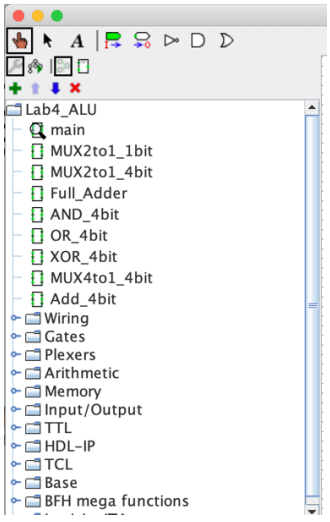
## Building a Hierarchical Design with Subcircuits

There is a lot of logic required to implement this ALU. The complete circuit is too cumbersome to draw on only one page. The use of subcircuits will make the over-all circuit more manageable. An ALU template has been created with "Lab6_ALU" as the main circuit.

The following subcircuits are placed in "Lab6_ALU" in the following hierarchy:

- Add_4bit (this subcircuit will consist of four "Full_Adder" subcircuits)
  - Full_Adder
    - One 3-input XOR gate
    - Three 2-input AND gates
    - One 3-input OR gate
- AND_4bit
  - Four 2-input AND gates
- OR_4bit
  - Four 2-input OR gates
- XOR_4bit
  - Four 2-input XOR gates
- MUX4to1_4bit
  - Three MUX2to1_4bit subcircuits
    - Four MUX2to1_1bit subcircuits

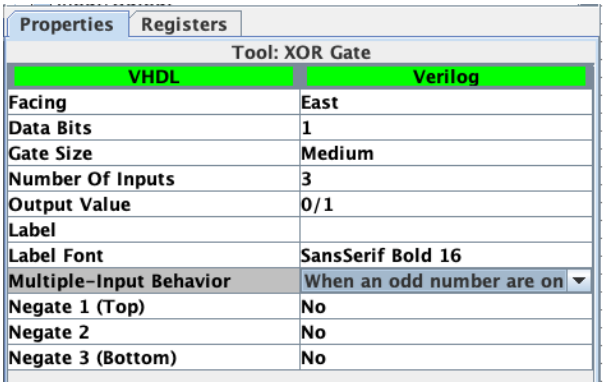
As a starting point for this lab, all the subcircuits of this ALU project have been created are contained in the template file called "Lab6_ALU". Each subcircuit is empty (except for the inputs and output) and you are tasked with completing each subcircuit and the main circuit. Double-click on them to edit them or single-click on them to add them to another higher-level circuit (you could have made these subcircuits yourself using the "Project/Add Circuit" menu at the top of Logisim Evolution).



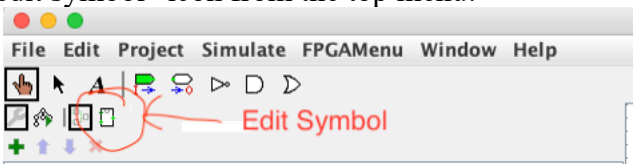
## Building the "Full_Adder" circuit

Before building the complete 4-bit adder, you must first build the subcircuit for the one-bit adder:

- Double click on the "Full_Adder" circuit in the left menu
- Implement the one-bit adder circuit as shown in figure 9.2 of the textbook.
- The XOR gate has an option called "Multiple-Input Behavior". This needs to be set to "When an odd number are on":




- The input pin names are: A, B, Cin
- The output pin names are: Cout, S
- At some point, you will be placing the symbol for this subcircuit in the four-bit adder. If you wish, you may make the symbol look more like the full-adder symbols used in figure 9.1 of the textbook, or you may prefer a different symbol arrangement. To edit the symbol, select the "edit symbol" icon from the top menu:

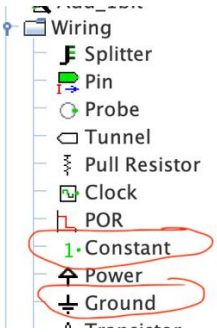



- There is no test vector for this, so you should verify that your circuit is functioning correctly by comparing its operation to the truth table of figure 9.2.

## Building the Complete Four-Bit Adder

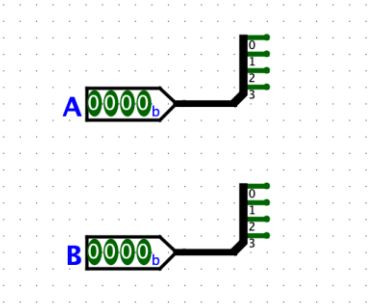
The four-bit adder is constructed of four, Full_Adder subcircuits similar to figure 9.1 of the textbook.

- Double click on "Add_4bit" in the left menu
- Implement the four-bit adder by cascading four, one-bit adders (similar to figure 9.1 of the textbook).
- The first full-adder that contains the LSB should have the Cin input connected to a constant zero or to ground because there is no need to carry information into the first adder:




- The carry out (Co) of the last full adder that implements the MSB is not connected.

- When connecting to the splitter wires, take note of the small number that labels each of the bits. The "0" wire is the LSB.

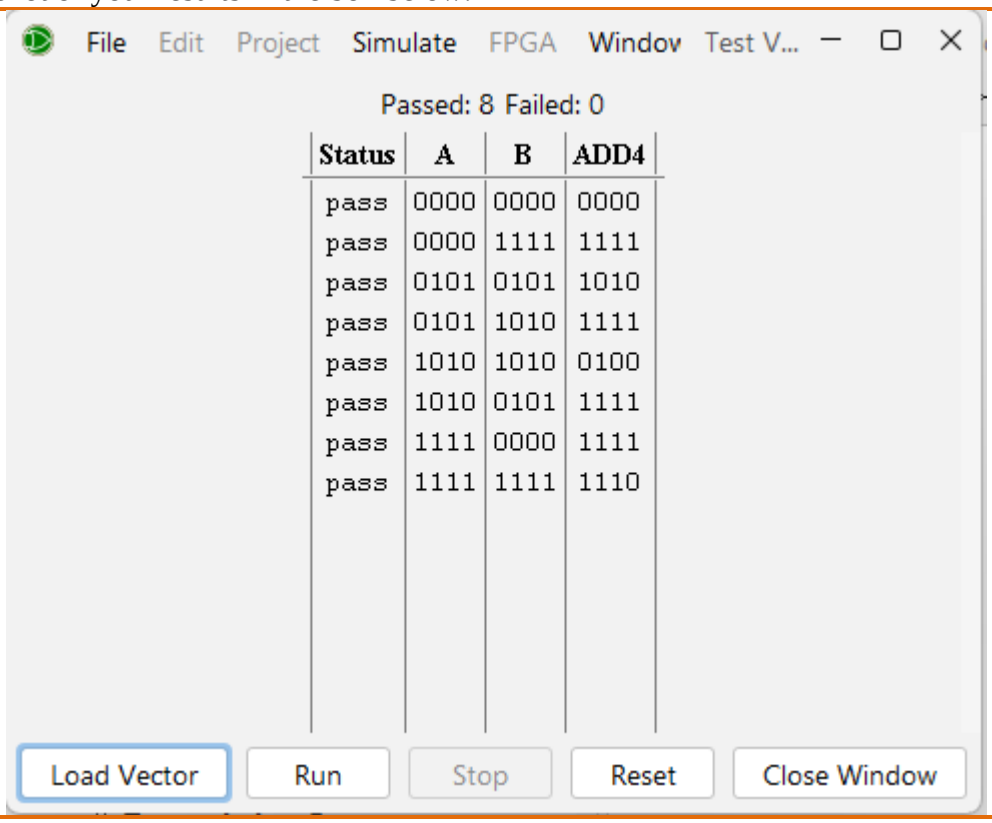



- Test your design

Expected output of the adder for the following inputs:

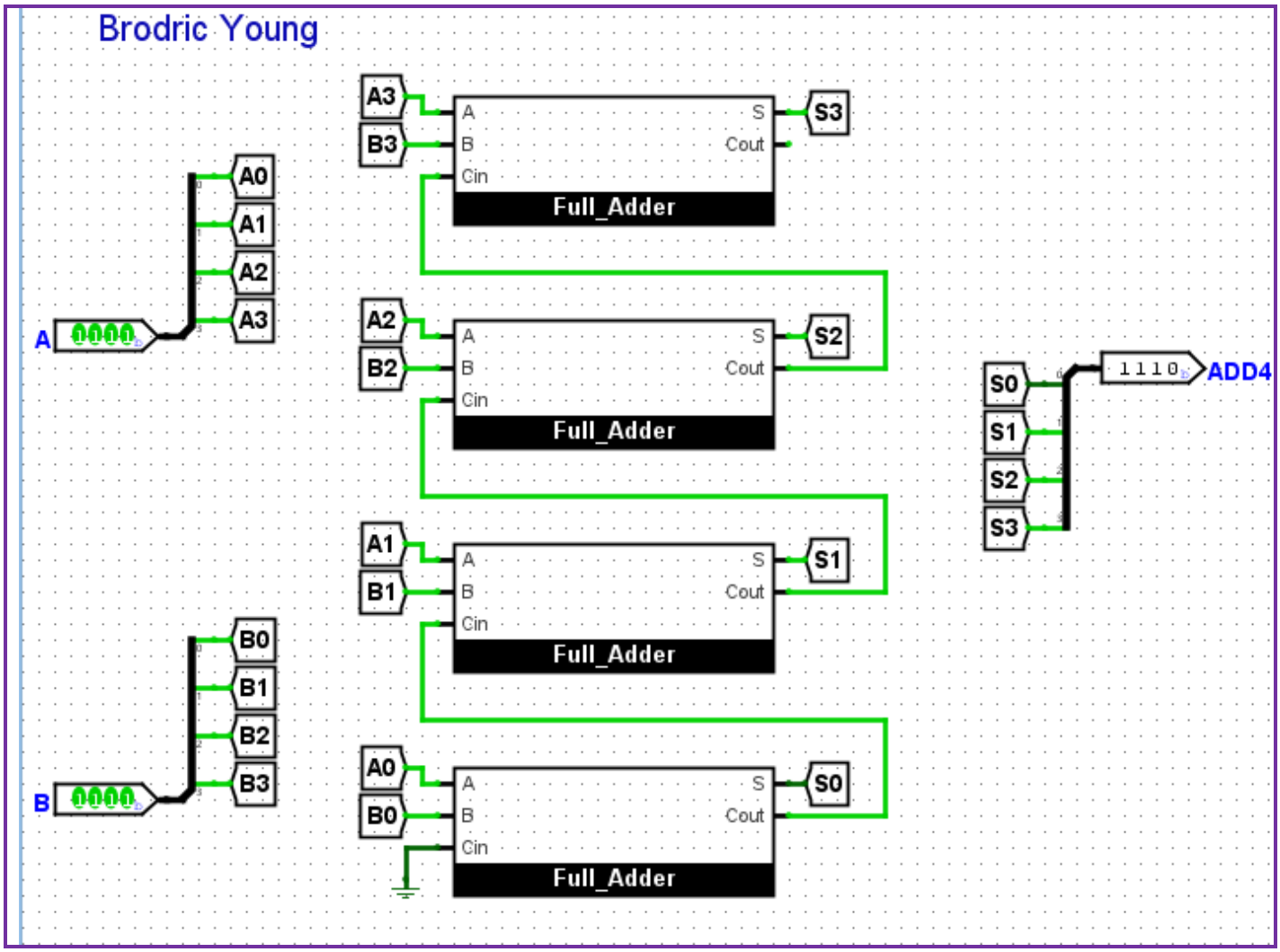
| Inputs | | Expected Output |
|---|---|---|
| A | B | ADD4 |
| 1001 | 1001 | 0010 |
| 0000 | 1111 | 1111 |
| 1111 | 1111 | 1110 |
| 1010 | 0101 | 1111 |
| 0010 | 0011 | 0101 |

Test the 4-bit adder circuit using the "ALU_ADD_test.txt" test vector file, and paste the screenshot of your results in the box below:




Four-bit adder test results (6 points)

Paste your *Logisim* four-bit adder circuit (including your name) in the box below:




Four-bit adder *Logisim* Circuit (6 points)

***To Verify Your Work in Part 1, Take Lab 6 Quiz 1***

(Quiz is worth 6 points)

# Lab 6 Part 2 - Completing the ALU Schematics

## Bitwise Logic Circuits (AND, OR, and XOR)

Build the subcircuits for the bit-wise AND, OR, and XOR bit-wise operations.

A bit-wise AND circuit is created using four, 2-input AND gates, where each AND gate has one of the "A" inputs, and the corresponding "B" input, wired to its input pins. There will be four outputs of the bit-wise AND operation. Merge the four outputs with a splitter and call the pin "AND4".

Record the expected output of a 4-bit bit-wise AND circuit:

| Inputs | | Expected Output |
|---|---|---|
| A | B | AND4 |
| 1001 | 1001 | 1001 |
| 0000 | 1111 | 0000 |
| 1111 | 1111 | 1111 |
| 1010 | 0101 | 0000 |
| 0010 | 0011 | 0010 |

Repeat the above subcircuit creation process for the OR and XOR operations (the output names will be OR4 and XOR4, respectively).

Record the expected output of a 4-bit bit-wise OR circuit:

| Inputs | | Expected Output |
|---|---|---|
| A | B | OR4 |
| 1001 | 1001 | 1001 |
| 0000 | 1111 | 1111 |
| 1111 | 1111 | 1111 |
| 1010 | 0101 | 1111 |
| 0010 | 0011 | 0011 |

Record the expected output of a 4-bit bit-wise XOR circuit:

| Inputs | | Expected Output |
|---|---|---|
| A | B | XOR4 |
| 1001 | 1001 | 0000 |
| 0000 | 1111 | 1111 |
| 1111 | 1111 | 0000 |
| 1010 | 0101 | 1111 |
| 0010 | 0011 | 0001 |

If desired, you can test your bit-wise logic circuits with the provided test-vector files to make sure they function correctly, but you are not required to include the results in this lab document. You are also not required to place snapshots of the Logisim circuits for these 3 subcircuits. The test-vector file names are:

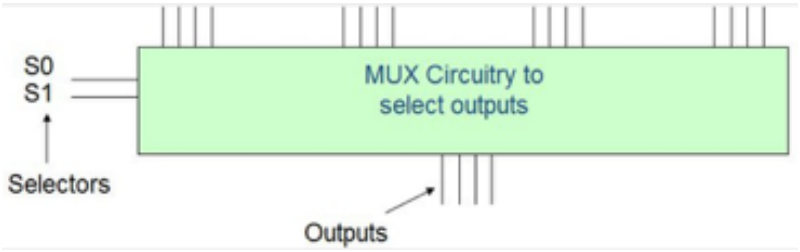
- ALU_AND_test.txt,
- ALU_OR_test.txt

- ALU_XOR_test.txt

## Four-bit Wide 4:1 MUX

The function of this subcircuit is to select a single, four-bit output from the results of the four ALU operations (ADD, AND, OR and XOR).

The following is a block diagram of the MUX, but in practice, busses (splitters) will be used to simplify the appearance of the schematic:




Build this MUX with the following subcircuits:

- MUX2to1_1bit - A 1-bit, 2 to 1 MUX as shown in figure 10.2 of the textbook.
- MUX2to1_4bit – Four MUX2to1_1bit subcircuits as shown in figure 10.5.
- MUX4to1_4bit – Three MUX2to1_4bit subcircuits as shown in figure 10.4.

Manually check the circuit to make sure it behaves as expected.

Test the MUX4to1_4bit circuit using the "ALU_MUX_test.txt" test vector file, and paste your results in the box below:

File Edit Project Simulate FPGA Window Test V...

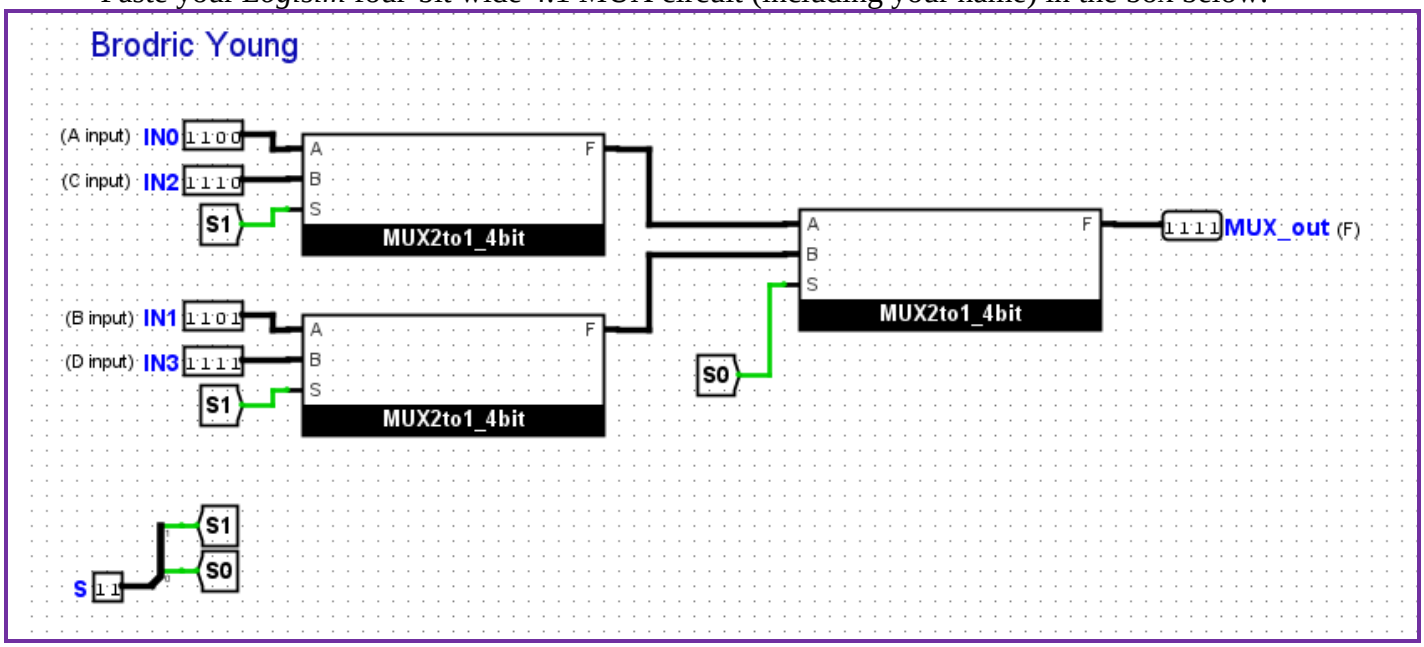
Passed: 16 Failed: 0

| Status | S | IN3 | IN2 | IN1 | IN0 | MUX_out |
|---|---|---|---|---|---|---|
| pass | 00 | 0011 | 0010 | 0001 | 0000 | 0000 |
| pass | 00 | 0111 | 0110 | 0101 | 0100 | 0100 |
| pass | 00 | 1011 | 1010 | 1001 | 1000 | 1000 |
| pass | 00 | 1111 | 1110 | 1101 | 1100 | 1100 |
| pass | 01 | 0011 | 0010 | 0001 | 0000 | 0001 |
| pass | 01 | 0111 | 0110 | 0101 | 0100 | 0101 |
| pass | 01 | 1011 | 1010 | 1001 | 1000 | 1001 |
| pass | 01 | 1111 | 1110 | 1101 | 1100 | 1101 |
| pass | 10 | 0011 | 0010 | 0001 | 0000 | 0010 |
| pass | 10 | 0111 | 0110 | 0101 | 0100 | 0110 |
| pass | 10 | 1011 | 1010 | 1001 | 1000 | 1010 |
| pass | 10 | 1111 | 1110 | 1101 | 1100 | 1110 |
| pass | 11 | 0011 | 0010 | 0001 | 0000 | 0011 |
| pass | 11 | 0111 | 0110 | 0101 | 0100 | 0111 |
| pass | 11 | 1011 | 1010 | 1001 | 1000 | 1011 |
| pass | 11 | 1111 | 1110 | 1101 | 1100 | 1111 |

Load Vector | Run | Stop | Reset | Close Window

Four-bit wide 4:1 MUX results (6 points)

Paste your *Logisim* four-bit wide 4:1 MUX circuit (including your name) in the box below:




Four-bit wide 4:1 MUX *Logisim* Circuit (6 points)

## Putting Together all of the ALU Subcircuits

Double-click on "main" and build the complete ALU by placing the necessary subcircuits in the required place. All of the interconnections will be made with multi-wire busses. Make sure that the four functions are connected to the proper mux input:

- When S = 00 the ALU outputs the bit-wise XOR
- When S = 01the ALU outputs the bit-wise AND
- When S = 10 the ALU outputs the bit-wise OR
- When S = 11 the ALU outputs the 4-bit ADD

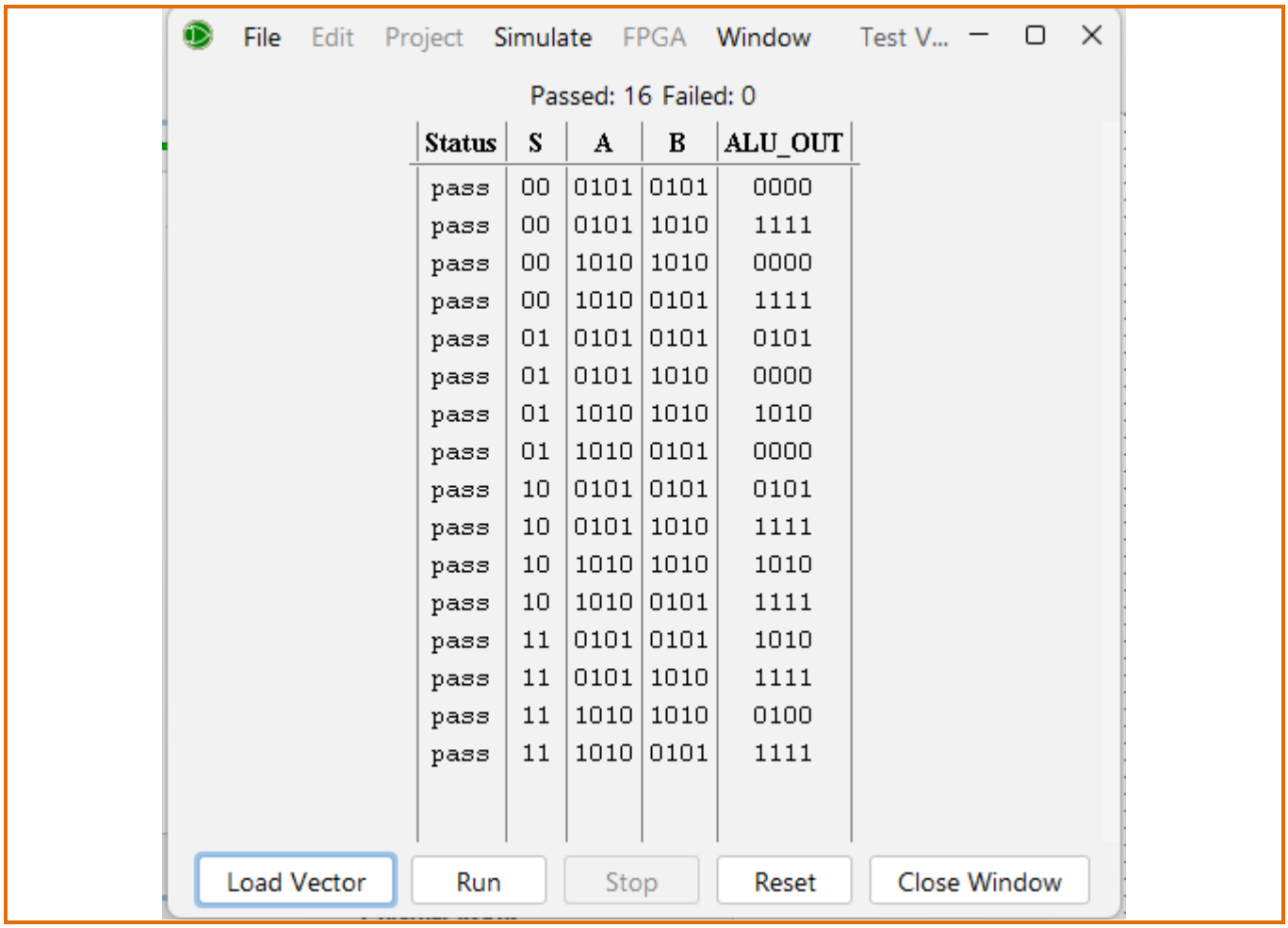
| A Input | B Input | S Input | ALU Function | Expected Output |
|---|---|---|---|---|
| 1001 | 1001 | 00 | XOR | 0000 |
| 1001 | 1001 | 01 | AND | 1001 |
| 1001 | 1001 | 10 | OR | 1001 |
| 1001 | 1001 | 11 | ADD | 0010 |
| 0000 | 1111 | 00 | XOR | 1111 |
| 0000 | 1111 | 01 | AND | 0000 |
| 0000 | 1111 | 10 | OR | 1111 |
| 0000 | 1111 | 11 | ADD | 1111 |
| 1111 | 1111 | 00 | XOR | 0000 |
| 1111 | 1111 | 01 | AND | 1111 |
| 1111 | 1111 | 10 | OR | 1111 |
| 1111 | 1111 | 11 | ADD | 1110 |
| 1010 | 0101 | 00 | XOR | 1111 |
| 1010 | 0101 | 01 | AND | 0000 |
| 1010 | 0101 | 10 | OR | 1111 |
| 1010 | 0101 | 11 | ADD | 1111 |
| 0010 | 0011 | 00 | XOR | 0001 |
| 0010 | 0011 | 01 | AND | 0010 |
| 0010 | 0011 | 10 | OR | 0011 |
| 0010 | 0011 | 11 | ADD | 0101 |

***Take Lab 6 Quiz 2***

(Quiz is worth 10 points)

### Testing Your Logisim ALU

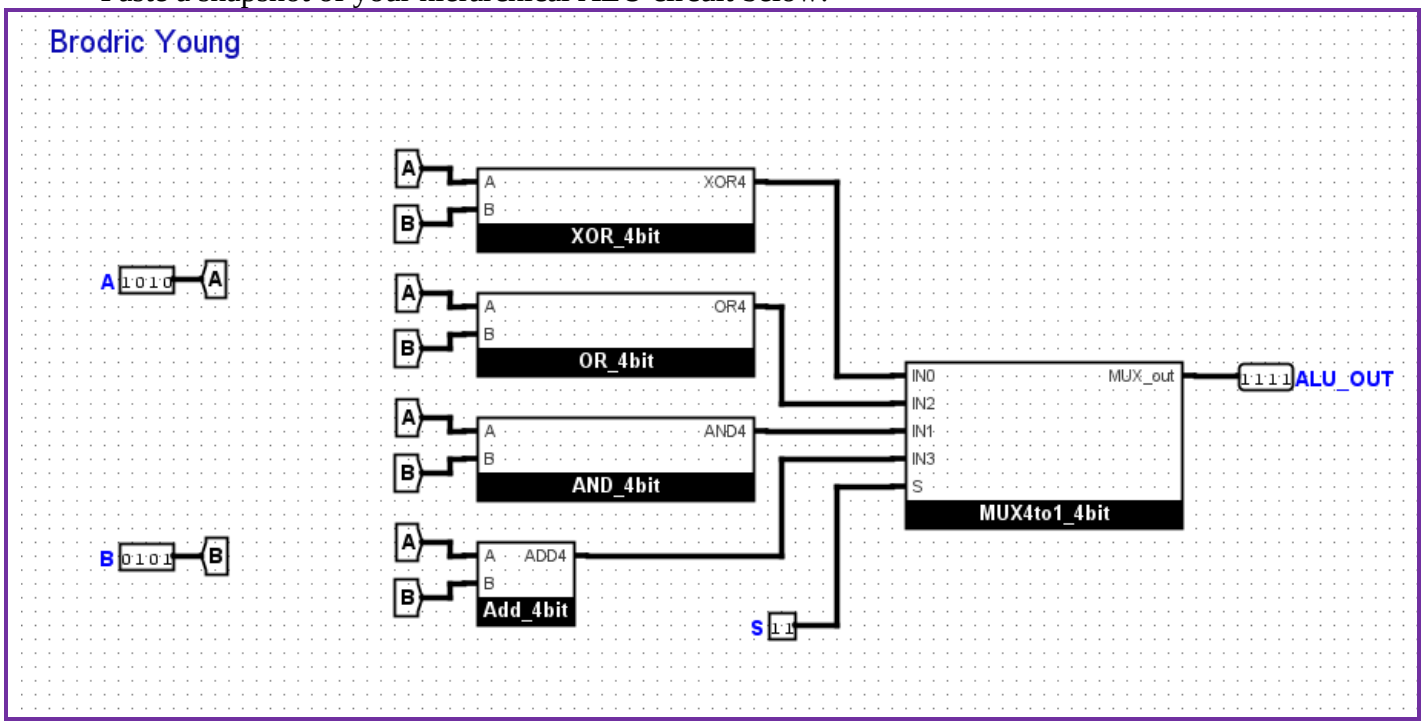
Manually test your ALU against your expected results. Once you are convinced your implementation is correct, verify its functionality using the "ALU_test.txt" test-vector file and paste your results below:




| Status | S | A | B | ALU_OUT |
|---|---|---|---|---|
| pass | 00 | 0101 | 0101 | 0000 |
| pass | 00 | 0101 | 1010 | 1111 |
| pass | 00 | 1010 | 1010 | 0000 |
| pass | 00 | 1010 | 0101 | 1111 |
| pass | 01 | 0101 | 0101 | 0101 |
| pass | 01 | 0101 | 1010 | 0000 |
| pass | 01 | 1010 | 1010 | 1010 |
| pass | 01 | 1010 | 0101 | 0000 |
| pass | 10 | 0101 | 0101 | 0101 |
| pass | 10 | 0101 | 1010 | 1111 |
| pass | 10 | 1010 | 1010 | 1010 |
| pass | 10 | 1010 | 0101 | 1111 |
| pass | 11 | 0101 | 0101 | 1010 |
| pass | 11 | 0101 | 1010 | 1111 |
| pass | 11 | 1010 | 1010 | 0100 |
| pass | 11 | 1010 | 0101 | 1111 |

ALU Test Vector Results (10 points)

Paste a snapshot of your hierarchical ALU circuit below:




*Logisim Evolution* ALU circuit (10 points)

# Lab 6 Part 3 – SystemVerilog ALU

# Using “Dataflow” Programming Style

Download the “Lab6_ALU” project zip file and follow the instructions in the “Lab6_SystemVerilog_Instructions” document.

***Have instructor or lab assistant enter password for Lab 6 Quiz 3 to pass off the SystemVerilog lab***
(15 points)

Copy and paste your SystemVerilog code here

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/25/2021 11:19:05 PM
// Design Name:
// Module Name: Lab6_ALU
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Lab6_ALU(
    input logic [3:0] A,
    input logic [3:0] B,
    input logic [1:0] S,
    output logic [3:0] ALU_OUT
    );
    //Declare the logic signals that you are using as internal signals to interconnect the modules
    logic [3:0] ADD_OUT; //Example declaration for the Adder output
    logic [3:0] AND_OUT;
    logic [3:0] OR_OUT;
    logic [3:0] XOR_OUT;

    ADD4 U0 (A, B, ADD_OUT);   //Example module instantiation for the adder.  "ADD4" is the
mdule name, U0 is the instance name
    AND4 U1 (A, B, AND_OUT);   //Module instantiation for the AND circuit.
    OR4  U2 (A, B, OR_OUT);   //Module instantiation for the OR circuit
    XOR4 U3 (A, B, XOR_OUT);   //Module instantiation for the XOR circuit
    MUX4bit_4to1 U4 (S, XOR_OUT, AND_OUT, OR_OUT, ADD_OUT, ALU_OUT);  //Module
instantiation for the MUX circuit
endmodule

module ADD4(
```

```
  input logic [3:0] A,
  input logic [3:0] B,
  output logic [3:0] out
  );
  assign out = A + B; //Example completion of the 4-bit adder

endmodule

module AND4(
  input logic [3:0] A,
  input logic [3:0] B,
  output logic [3:0] out
  );
  assign out = A & B;

endmodule

module OR4(
  input logic [3:0] A,
  input logic [3:0] B,
  output logic [3:0] out
  );
  assign out = A | B;

endmodule

module XOR4(
  input logic [3:0] A,
  input logic [3:0] B,
  output logic [3:0] out
  );
  assign out = A ^ B;

endmodule

module MUX4bit_4to1(
  input logic [1:0] S,
  input logic [3:0] in0,
  input logic [3:0] in1,
  input logic [3:0] in2,
  input logic [3:0] in3,
  output logic [3:0] out
  );
  logic [3:0] wire1;
  logic [3:0] wire2;

  assign wire1= S[1]?in2:in0;
  assign wire2 = S[1] ? in3 : in1;
  assign out = S[0] ? wire2 : wire1;

endmodule
```

SystemVerilog ALU Code (15 points)

**Conclusions Statement**

Write a brief conclusions statement that discusses the original purposes of the lab found at the beginning of this lab document.

- Are your thoughts on the hierarchical design approach? Advantages? Disadvantages?
  - Advantages of the hierarchical design is its simpler to code and read because it takes fewer lines and its easier to follow. Using modules helps keep the code organized and calling them gives you an idea of what the module does without going into the code of the module. Structural is more detailed, but overall I think hierarchical is much easier.

- You have designed a four-function ALU. What other types of functions do you think would be useful for an ALU?
  - Other functions that could be useful for us to include in an ALU would be other math operations like subtraction, multiplication and division or functions that make those operations up.

- What are the important aspects to note regarding the construction of multiple-bit multiplexers?
  - For multi-bit multiplexers, it's important to note the order and amount of the bits going in and out, keeping it ordered from the lsb to msb and the same bit width. I had made a mistake in my design where the msb and lsb of the 2 bit select input was switched around and it changed the functionality of the multiplexer until I fixed it.

- What can you say about the functionality of an ALU (for example, how does the ALU know which operation to perform?
  - The functionality of ALU is like a ROM how it takes inputs and a select value and decided which operation its going to pass through using the select value, then executes that operation on the input values.

- What are the advantages and disadvantages of using the "dataflow" programming style of SystemVerilog to implement an ALU?
  - Advantages of using the dataflow style are is easier to both program and to read. Using structural it's easy to get lost on which wire is which and what's actually going on but the dataflow style helps simplify things. Disadvantages would be that its not as detailed as structural.

Please use complete sentences and correct grammar to express your thoughts:

(The conclusions box will expand as you write)

In this lab we learned how to build multi-bit multiplexers which are used in ALU components. From what we learned about multi-bit multiplexers, we were able to design and implement using SystemVerilog a simple ALU which also helped us to learn the design and operation of an ALU. We learned how an ALU actually works by doing this and saw it function when we programmed the FPGA to act as an ALU. This was the most complex circuit we've done so far in this class, but we were able to simplify it in SystemVerilog by using the hierarchical design approach and dataflow programming style. These allowed us to use simple statements instead of a bunch of gates to make the program easier to read and less cluttered by implementing it with both fewer lines and inside modules to keep similar things together in one module.

Conclusions Statement (10 points)

Congratulations, you have completed the lab!
You may now submit this document.