CS 2110 Homework 2
Digital Logic and the ALU

Prabhav Gupta, Jack Keller, Arnav Mardia,
Lydia Lazor, Vy Mai, Richard So

Spring 2024

Contents

1 Overview

1.1 Purpose

You have learned about digital logic, including transistors, gates, and combinational logic. Gates (AND, OR, etc.) can be built using transistors, and combinational logic circuits (decoder, multiplexers, ALUs etc.) can be built using gates. Note how the concepts build up from transistors to gates to combinational logic. We have provided you with a tool called CircuitSim that allows you to simulate building circuits, without actually using physical hardware.

The purpose of this assignment is for you to become proficient building gates and combinational logic circuits.

1.2 Task

You will complete four CircuitSim files, and build an ALU (arithmetic logic unit) from the ground up. Please read this entire document for detailed instructions, including which digital logic components you are allowed to use or prohibited from using for each part of the assignment.

This document also contains tutorials on using CircuitSim.

The steps to complete this assignment are:

1. Create the standard logic gates (NAND, NOR, NOT, AND, OR)
2. Apply DeMorgan's Law to convert a provided circuit to one without any OR Gates
3. Create an 8-input multiplexer and an 8-output decoder
4. Use multiplexers to create a circuit that evaluates the sign of a number
5. Create a 1-bit full adder
6. Create an 8-bit full adder using the 1-bit full adder
7. Use your 8-bit full adder and other components to construct an 8-bit ALU

1.3 Criteria

You must use the provided CS 2110 version of CircuitSim, which is available via the JAR on Canvas. Other versions of CircuitSim are incompatible with our autograders.

You will submit four CircuitSim files that you produce to Gradescope: `gates.sim`, `demorgan.sim`, `plexers.sim`, and `alu.sim`. Your circuits must work properly and produce the desired results in order to receive credit. For the ALU portion, partial credit is awarded for each operation that works properly.

Be sure to avoid using components that are disallowed for each phase of this assignment.

2 Optional Tutorial

Note: This tutorial is optional and to help you get acquainted with CircuitSim before you start this homework. If you're comfortable with this software, feel free to skip ahead. CircuitSim is an interactive circuit simulation package. We will be using this program for the next couple of homework assignments. This is a tutorial to help you get acquainted with the software. CircuitSim is a powerful simulation tool designed for educational use. This gives it the advantage of being a little more forgiving than some of the more commercial simulators. However, it still requires some time and effort to be able to use the program efficiently. With this in mind, we present you with the following assignment:

2.1 Part 1 — Read Resources

Read through the following resources

- CircuitSim Wires Documentation `https://ra4king.github.io/CircuitSim/docs/wires/`
- Tutorial 1: My First Circuit `https://ra4king.github.io/CircuitSim/tutorial/tut-1-beginner`

2.2 Part 2 — Complete Tutorial 2

Complete Tutorial 2 `https://ra4king.github.io/CircuitSim/tutorial/tut-2-xor`

Instead of saving your file as **xor.sim**, save your file as **part1.sim**. As well, make sure you label your two inputs **a** and **b**, and your output as **c**, as well as rename your subcircuit to xor.

2.3 Part 3 — Complete Tutorial 3

Complete Tutorial 3 `https://ra4king.github.io/CircuitSim/tutorial/tut-3-tunnels-splitters`

Name the subcircuit **umbrella**, the input **in**, and the output **out**. Save your file as **part2.sim**.

3 Instructions

3.1 1-Bit Logic Gates

Allowed Components: Wiring Tab and Circuits Tab

All of the circuits in this file are in the `gates.sim` file.

For this part of the assignment, you will create a transistor-level implementation of the NOT, NAND, NOR, AND, and OR logic gates.

For this section you are only allowed to use components listed in the Wiring section and circuits you have already implemented. For example, once you implement the NOT gate you are free to use that subcircuit in implementing other logic gates. Implementing the gates in the order listed below will be easiest.

As a brief summary of the behavior of each logic gate:

NOT (Input: IN - Output: OUT)

A	NOT A
0	1
1	0

NAND (Inputs: A, B - Output: OUT)

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR (Inputs: A, B - Output: OUT)

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

AND (Inputs: A, B - Output: OUT)

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR (Inputs: A, B - Output: OUT)

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

All of the logic gates must be within their named sub-circuits.

3.2 DeMorgan's Law

Allowed Components: Wiring Tab, NOT Gate, 9 OR Gates, and 1 NAND Gate

The circuits in this file are in the `demorgan.sim` file.

De Morgan's laws help relate conjunctions (AND) and disjunctions (OR) of propositions with negation.

In this file, you will be creating a circuit that is an alternative to the circuit that is provided in the *Provided* subcircuit (first tab). Your work will be done in the *Student* subcircuit (second tab).

3.2.1 Provided

Nothing needs to be done in this subcircuit! You may find it helpful to test some inputs and build a truth table.

3.2.2 Student

This circuit needs to be directly equivalent to the circuit provided in the *Provided* subcircuit, meaning the **same inputs** should lead to the **same outputs**.

Here's the catch: you must to rebuild this circuit with **exactly 9 OR gates, 1 NAND gate, and no other gates!** You may use NOT gates / bubbles on the inputs of gates freely.

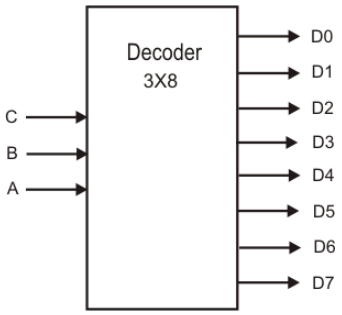
3.3 Plexers

All of the circuits in this file are in the `plexers.sim` file.

3.3.1 Decoder

Allowed Components: Wiring Tab, Circuits Tab, and Gates Tab

The decoder you will be creating has a single 3-bit selection input (`SEL`), and eight 1-bit outputs (labeled `A`, `B`, `C`, ..., `H`). The decoder uses the `SEL` input to raise a specific output line, as seen below.



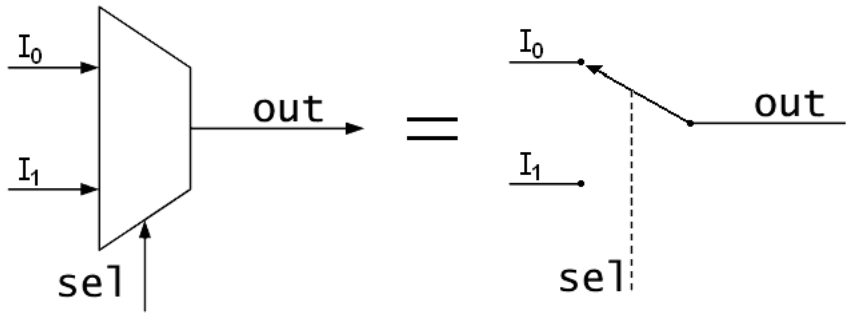
For example:

```
SEL = 000 ==> A = 1, BCDEFGH = 0
SEL = 001 ==> B = 1, ACDEFGH = 0
SEL = 010 ==> C = 1, ABDEFGH = 0
SEL = 011 ==> D = 1, ABCEFGH = 0
SEL = 100 ==> E = 1, ABCDFGH = 0
SEL = 101 ==> F = 1, ABCDEGH = 0
SEL = 110 ==> G = 1, ABCDEFH = 0
SEL = 111 ==> H = 1, ABCDEFG = 0
```

3.3.2 Multiplexer

Allowed Components: Wiring Tab, Circuits Tab, and Gates Tab

The multiplexer you will be creating has 8 1-bit inputs (labeled appropriately as `A`, `B`, `C`, ..., `H`), a single 3-bit selection input (`SEL`), and one 1-bit output (`OUT`). The multiplexer uses the `SEL` input to choose a specific input line for forwarding to the output.



For example:

```
SEL = 000 ==> OUT = A
SEL = 001 ==> OUT = B
```

```
SEL = 010 ==> OUT = C
SEL = 011 ==> OUT = D
SEL = 100 ==> OUT = E
SEL = 101 ==> OUT = F
SEL = 110 ==> OUT = G
SEL = 111 ==> OUT = H
```

3.3.3 Sign Evaluation

Allowed Components: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

In this step, you will construct a circuit that takes an 8-bit two's complement input and evaluates whether the input is negative, zero, or positive.

Based on this, this circuit will output a 3-bit bit-vector called NZP where the most significant bit is 1 *iff* the input is negative, the middle bit is 1 *iff* the input is zero, and the least significant bit is 1 *iff* the input is positive. Only 1 bit of the output should be set at any given time. Zero is not considered a positive number.

For example:

```
INPUT = 10111000 ==> NZP = 100
INPUT = 00000000 ==> NZP = 010
INPUT = 00000100 ==> NZP = 001
```

Note that you may use the plexer tab for this circuit.

Hint: Recall that in two's complement, the most significant bit can be used to determine the sign of a number!

3.4 Adders & ALUs

All of the circuits in this file are in the `alu.sim` file.

3.4.1 1-Bit Adder

Allowed Components: Wiring Tab, Circuits Tab, and Gates Tab

The full adder has three 1-bit inputs (`A`, `B`, and `CIN`), and two 1-bit outputs (`SUM` and `COUT`). The full adder adds `A + B + CIN` and places the sum in `SUM` and the carry-out in `COUT`.

For example:

```
A = 0, B = 1, CIN = 0 ==> SUM = 1, COUT = 0
A = 1, B = 0, CIN = 1 ==> SUM = 0, COUT = 1
A = 1, B = 1, CIN = 1 ==> SUM = 1, COUT = 1
```

Hint: Making a truth table of the inputs will help you.

3.4.2 8-Bit Adder

Allowed Components: Wiring Tab, Circuits Tab, and Gates Tab

For this part of the assignment, you will daisy-chain 8 of your 1-bit full adders together in order to make an 8-bit full adder.

This circuit should have two 8-bit inputs (`A` and `B`) for the numbers you're adding, and one 1-bit input for `CIN`. The reason for the `CIN` has to do with using the adder for purposes other than adding the two inputs.

There should be one 8-bit output for `SUM` and one 1-bit output for `COUT`.

3.4.3 Basic 8-Bit ALU

Allowed Components: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

You will first create a simple 8-bit ALU, using the 8-bit full adder you created previously.

For this ALU, we will be using a multiplexer. This ALU has two **8-bit** inputs for `A` and `B` and one **2-bit** input for `OP`, the op-code for the operation in the list below. It has one **8-bit** output named `OUT`. The following 4 operations will be selected from:

00. Addition	`[A + B]`
01. AND	`[A & B]`
10. NOT	`[NOT A]`
11. Pass	`[PASS A]`

The **Pass** operation simply refers to outputting the value of A unchanged.

Notice that **NOT** and **Pass** only operate on the `A` input. **They should NOT rely on `B` being a particular value.**

The provided autograder will check the op-codes according to the order listed above (Addition (`00`), AND (`01`), etc.) and thus it is important that the operations are in this exact order.

3.4.4 Intermediate 8-Bit ALU

Allowed Components: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

You will next create a different 8-bit ALU, with identical structure as the previous, but more complex operations as outlined below. Make sure to match the OP

00.	isPalindrome	`[A == reverse(A)]`
01.	isDivisibleBy32	`[A % 32 == 0]`
10.	inBetween[8, 64)	`[A is in the range of [8, 64)]`
11.	maximum(-15 * A, B)	`[max(-15 * A, B)]`

For circuits isPalindrome, isDivisibleBy32, and inBetween[8, 64) operations, return `00000001` **if the condition is true, and** `00000000` **otherwise**.

The autograder will check the op-codes according to the order listed above and thus it is important that the operations are in this exact order.

3.5 Running the Autograder

To run the autograder locally, type the following command into your terminal while in the Homework 2 directory:

```
java -jar hw02-tester.jar
```

Make sure all the tests have been passed. Keep in mind that even if you get full credit from the autograder, we reserve the right to test for more cases.

4 Disclaimers

The following are trivial things that may lead to errors on the autograder, even though your circuit is *technically* correct.

1. Renaming or removing input pins may cause issues with the autograder because it's looking for input pins with a specific label.
2. You must use constant pins when necessary. Using any input pins for a scenario where a fixed constant value is required will lead to the autograder setting the input pin to 0 and thus cause issues for scenarios where it needed to be a constant 1. **Never add/remove input or output pins. All necessary input and output pins have been provided for you.**

5 Deliverables

Please upload the following files onto the assignment on Gradescope:

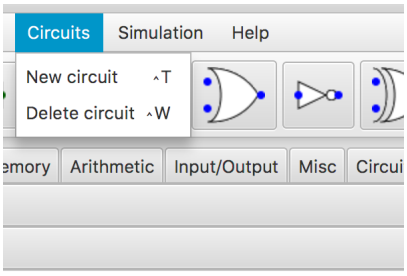
1. `gates.sim` `NOT, NAND, NOR, AND, OR`
2. `demorgan.sim` `Provided, Student`
3. `plexers.sim` `Decoder, MUX, Sign Evaluation`
4. `alu.sim` `1-Bit Adder, 8-Bit Adder, Basic ALU, Intermediate ALU`

6 Sub-Circuit Tutorial

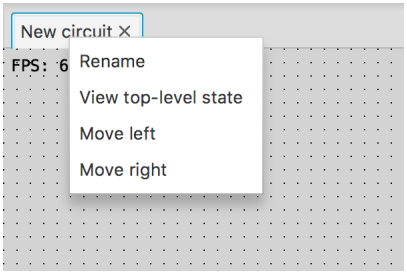
As you build circuits that are more and more sophisticated, you will want to build smaller circuits that you can use multiple times within larger circuits. Sub-circuits behave like classes in Object-Oriented languages. Any changes made in the design of a sub-circuit are automatically reflected wherever it is used. The direction of the IO pins in the sub-circuit correspond to their locations on the representation of the sub-circuit.

To create a sub-circuit:

1. Go to the "Circuits" menu and choose "New circuit"

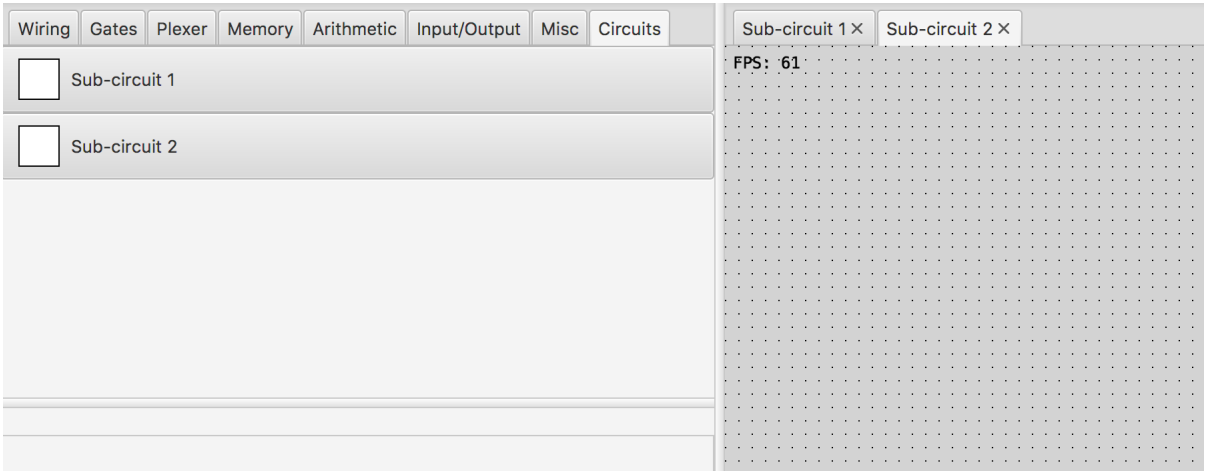


2. Name your circuit by right-clicking on the "New circuit" item and selecting "Rename"

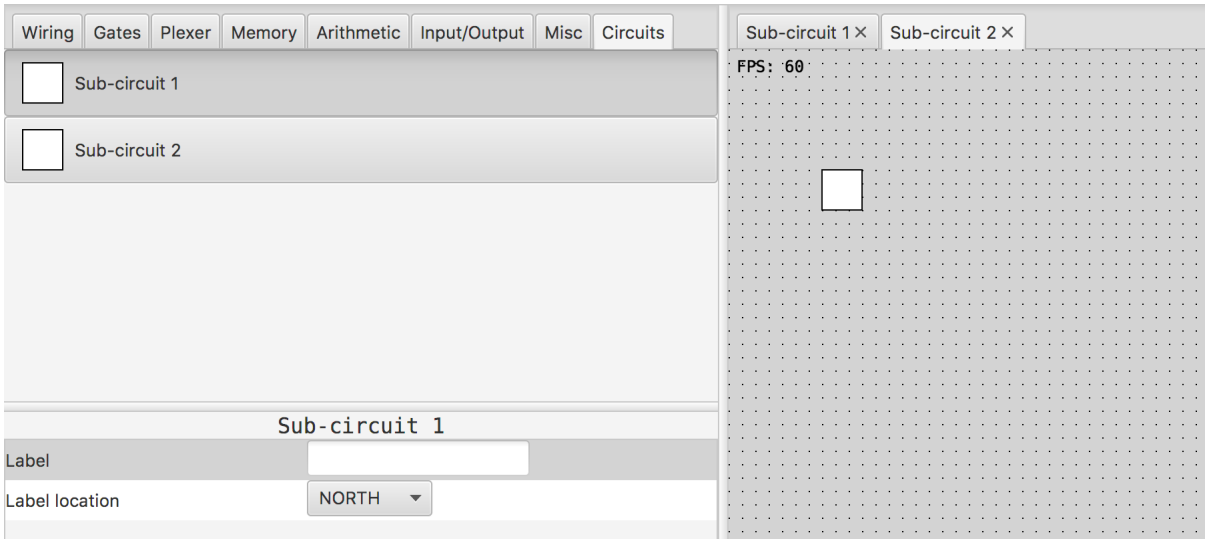


To use a sub-circuit:

1. Click the "Circuits" tab next to the "Misc" tab



2. Select the circuit you wish to use and place it in your design



7 Rules and Regulations

1. Please read the assignment in its entirety before asking questions.
2. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.
3. If you find any problems with the assignment, please report them to the TA team. Announcements will be posted if the assignment changes.
4. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency please reach out to your instructor and the head TAs **IN ADVANCE** of the due date with documentation (i.e. note from the dean, doctor's note, etc).
5. You are responsible for ensuring that what you turned in is what you meant to turn in. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Gradescope. Email submissions will not be accepted.
6. See the syllabus for information regarding late submissions; any penalties associated with unexcused late submissions are non-negotiable.

7.1 Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work. Homework assignments will be examined using cheat detection programs to find evidence of unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student. If you supply a copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any platform which would allow other parties to it (public repositories, pastebin, etc). If you would like to use version control, use a private repository on github.gatech.edu

Homework collaboration is limited to high-level collaboration. Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment.

High-level collaboration means that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code, or providing other students any part of your code.

Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

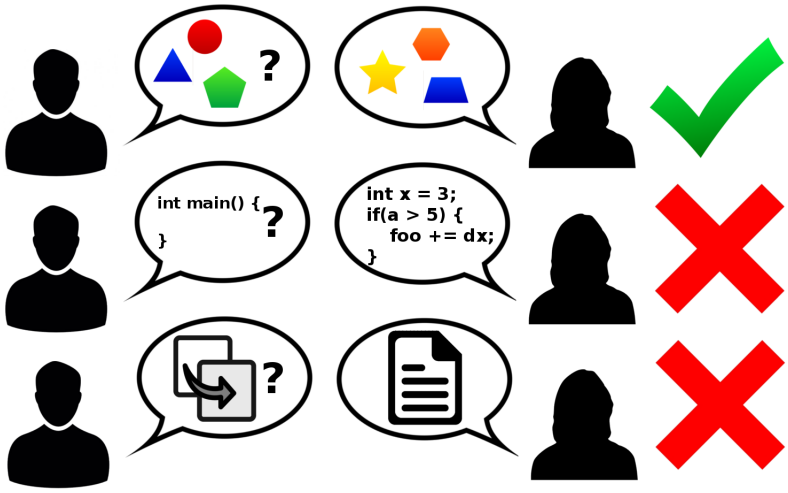


Figure 1: Collaboration rules, explained colorfully