Project 1 - Digital Logic

Christopher Turko, Sankaet Cheemalamarri, Sameer Suri, Udit Subramanya, Tanya Qin

Fall 2022

Contents

1 Introduction

This project is designed to strengthen your knowledge of basic hardware elements and circuitry. The sections build on each other, so work down the document linearly, otherwise you may not understand what is going on.

Please read through the entire document before starting. Oftentimes things are elaborated on below where they are introduced, so reading the entire document can give you a better grasp on things. **Start early** and if you get stuck, there's always EdDiscussion or come visit us in office hours.

2 Setup

The software you will be using for this project and all future circuit based assignments is called CircuitSim - an interactive circuit simulation package.

In order to use CircuitSim, you must have Docker up and running. If you do not have Docker, follow the instructions laid out in the installation guide found at `https://gt-cs2110.github.io/cs2110docker/`.

CircuitSim comes pre-installed on the Docker image, and should be accessible via the desktop. Please only use the CircuitSim through Docker to build your circuits as it is the correct version. **CircuitSim downloaded elsewhere may not be compatible with our grader. You have been warned.**

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.

All .sim files should be opened in CircuitSim.

3 Part 1 — Introduction to CircuitSim

The first part of this project is designed to get you up to speed with CircuitSim. If you do not have CircuitSim running through Docker, see the setup section of this document to get there.

3.1 Read Resources

Read through the following resources

- CircuitSim Wires Documentation `https://ra4king.github.io/CircuitSim/docs/wires/`
- Tutorial 0: My First Circuit `https://ra4king.github.io/CircuitSim/tutorial/tut-1-beginner`

3.2 Complete Tutorial 1

In the existing file `tutorial1.sim` and subcircuit `Tutorial 1`, complete the following tutorial. **Do not rename the subcircuit or file**. Be sure to label your two inputs `a` and `b`, and your output as `c`.

Complete Tutorial 1 `https://ra4king.github.io/CircuitSim/tutorial/tut-2-xor`

3.3 Complete Tutorial 2

In the existing file `tutorial2.sim` and subcircuit `Tutorial 2`, complete the following tutorial. **Do not rename the file or subcircuit.** Name the input `in`, and the output `out`.

Complete Tutorial 2 `https://ra4king.github.io/CircuitSim/tutorial/tut-3-tunnels-splitters`

4 Part 2 — ALU

Now that we have the basics down, we can move on to more complex circuits and logic. All computer processors have a very important component known as the Arithmetic Logic Unit (ALU). This component allows the computer to do, as the name suggests, arithmetic and logical operations. For this part, you're going to build an ALU of your own, along with creating some of the gates.

For this part of the project you will:

1. Create the standard logic gates (NAND, NOR, NOT, AND, OR)
2. Create an 8-input multiplexer and an 8-output decoder
3. Create a 1-bit full adder
4. Create an 8-bit full adder using your 1-bit full adder
5. Use your 8-bit full adder and other components to construct an 8-bit ALU

When building these circuits, restrictions have been placed on what you can use. These restrictions are listed at the beginning of each section. **Using anything not listed will result in heavy deductions.** You also need to have everything in its correctly named sub-circuit. More information on the sub-circuits is given below.

Use tunnels where necessary to make your designs more readable, but do not overdo it! For gates, multiplexers, decoders, and adders you can often make clean circuits by placing your components strategically rather than using tunnels everywhere.

4.1 1-Bit Logic Gates

Allowed Components: Wiring Tab and Circuits Tab

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

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

Remember for this section that you are only allowed to use the components listed in the Wiring section, along with any of the logic gates you are implementing in CircuitSim. 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 of the subcircuit tabs can be the easiest option.

Hint: Start by creating the NAND and NOT gates from transistors. Then use this gate as a subcircuit for implementing the others.

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

4.2 Muxes

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

All of the circuits are found in the `muxes.sim` file.

4.2.1 Decoder

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. `000` should correspond to `A`, `001` should correspond to `B`, etc.

4.2.2 Multiplexer, commonly abbreviated as "mux"

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. `000` should correspond to `A`, `001` should correspond to `B`, etc.

4.3 Adders

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

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

4.3.1 1-Bit Adder

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
```

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

4.3.2 8-Bit Adder

You should 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`.

4.4 8-Bit ALU

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

You will create an 8-bit ALU with the following operations. Feel free to use the circuits you made in previous parts of this lab to implement the following operations. They will be under the **Circuits** tab in CircuitSim. You may also create subcircuits for each ALU operation.

000 `add`	`[A + B]`
001 `sub`	`[A - B]`
010 `min`	`[min(A, B)]`
011 `numEqualBits`	`See below`
100 `leastSignificantOne`	`See below`
101 `isMultipleOf4`	`[A % 4 == 0]`
110 `multiplyBy9`	`[A * 9]`
111 `isPositive`	`[A > 0]`

`isMultipleof4` and `isPositive` should return either `00000000` for `false` or `00000001` for `true`

`multiplyBy9` should not account for overflow

`numEqualBits:` For this operation, you will need to implement a circuit which counts the number of equal bits between `A` and `B`. Your circuit should compare each bit in `A` with its corresponding bit in `B`, and if they are equivalent, increment the number of equal bits by 1.

Examples:

```
A = 00001111, B = 11110000 => return 00000000
A = 11110000, B = 11110011 => return 00000110
```

`leastSignificantOne:` To elaborate, this operation should return the 8-bit representation of the value that is produced when all bits of C are cleared except for the least significant 1. Keep in mind that C is given in a 4-bit representation.

Examples:

```
C = 0110 => return 00000010
C = 1100 => return 00000100
```

Notice that `leastSignificantOne`, `isMultipleOf4`, `multiplyBy9`, and `isPositive` only operate on the `A` or `C` input. They should **NOT** rely on `B` being a particular value.

This ALU has two 8-bit inputs for `A` and `B`, one 4-bit input for `C`, and one 3-bit input for `OP`, which is the op-code for the operation in the list above. It has one 8-bit output named `OUT`.

The provided autograder will check the op-codes according to the order listed above (add (`000`), sub (`001`), etc.) and thus it is important that the operations are in this exact order. **If you ever need a constant value, use a constant pin (not an input pin). Failing to do so will interfere with the autograder.**

No partial credit will be given for incorrect outputs for logic gates, plexers, or adders. However, for the ALU, partial credit will be awarded on a per-operation basis, wherein each operation must perform successfully to be awarded credit. Because of this, we urge you to check your score before the due date.

As always, do not change/delete any of the input/output pins (You can move them).

5 Part 3 — Advanced Combinational Logic

All of the circuits are found in the `project.sim` file.

You are the most famous ice cream connoisseur in the world. For the upcoming semiannual national ice cream convention, you have been tasked with creating the penultimate sundae for the evening: a most prestigious honor. To create the perfect ice cream sundae, you consult the ICDS (Ice Cream Deliciousness Scale). The goal is to craft a sundae which scores between 8 and 12 (A 10 would be ice cream perfection). If it is under 8, it will not fully satisfy your fellow ice cream connoisseurs. If over 12, it will be more than what their taste buds can comprehend. You have narrowed down your sundae to 5 items and assigned each a point value, but still do not know which combinations will illicit a suitable sundae. To accomplish this, you craft a combinational logic circuit with 5 inputs and 1 output. The 5 inputs are your 5 items: **Mint Ice Cream, Vanilla Ice Cream, Caramel Sauce, Sprinkles, Whipped Cream** (there are vegan and gluten-free options for connoisseurs with dietary restrictions). The circuit will output a 1 if the given combination will score between 8 and 12, and a 0 otherwise.

M	Mint Ice Cream	2 points
V	Vanilla Ice Cream	4 points
C	Caramel Sauce	2 points
S	Sprinkles	6 points
W	Whipped Cream	2 points

(Note: These point values do not represent the TAs' actual opinions)

Your goal is to create a circuit which can tell you whether a given combination of items will score between an 8 and 12 on the ICDS (Ice Cream Deliciousness Scale).

An **input** value of 1 indicates the item is being included in the ice cream sundae. An **input** value of 0 indicates the item will not be included in the ice cream sundae.
An **output** value of 1 indicates the given combination of items will receive a score between 8 and 12. An **output** value of 0 indicates the given combination of items will **not** receive a score between 8 and 12.

Ex: If **M = 0**, **V = 1**, **C = 0**, **S = 1**, and **W = 0** then the output should equal 1 since `4 + 6` will result in a score of `10`. This is between the acceptable range of `8` to `12`.

Ex: If **M = 0**, **V = 1**, **C = 1**, **S = 1**, and **W = 0** then the output should equal 0 since `4 + 6 + 2 + 2` will result in `14`. This is not in the acceptable range of `8` to `12`.

As always, do not change/delete any of the input/output pins (You can move them).

5.1 Karnaugh Maps

While it is not a deliverable, making 2 K-maps and reducing boolean expressions will make this circuit significantly easier to design. To aid this, we have provided a template for both K-maps below:

M	$V'C'$	$V'C$	VC	VC'
$S'W'$				
$S'W$				
SW				
SW'				

M'	$V'C'$	$V'C$	VC	VC'
$S'W'$				
$S'W$				
SW				
SW'				

Once you fill out each K-map individually and make groups, you can stack the two K-maps on top of each other to create a 5-variable K-map. Then you may group any overlapping groups together into one big group and get rid of the `M` variable in the resulting production.

5.2 Boolean Logic Details

For this part of the project, we are asking that you approach this problem in **sum of products format**. This means that after your reductions using the K-maps, you should have something in the format

$$D = A'B + C$$

before attempting to build the circuit. Failure to do so can lead to complications in your circuit that prevent the reduction we are looking for.

5.3 Circuit Details

To prevent trivialization of this assignment, we have placed a few restrictions:

- All of this circuit should be contained in the **project.sim** file
- You should try to reduce the amount of AND and OR gates as much as possible.
  Hint: The K-maps do this for you!

6 Autograder

To run the autograder, run

```
java -jar project1-tester-1.0.jar
```

at a command prompt in the same directory as all your `.sim` files. This is the same autograder that Gradescope uses, but is much easier and faster to use. **The autograder should be run from the terminal emulator in docker.**

7 Deliverables

Please submit the follow files:

1. `tutorial1.sim`	
2. `tutorial2.sim`	`umbrella`
3. `gates.sim`	`NOT, NAND, NOR, AND, OR`
4. `muxes.sim`	`Decoder, MUX`
5. `alu.sim`	`1-Bit Adder, 8-Bit Adder, 8-Bit ALU`
6. `project.sim`	`Advanced Combinational Logic`
7. `collaborators.txt`	

to Gradescope under the assignment “Project 1”.

Note: The autograder may not reflect your final grade on this assignment. We reserve the right to update the autograder as we see fit when grading.

8 Demos

This project will be demoed. Demos are designed to make sure that you understand the content of the project and related topics. They may include technical and/or conceptual questions.

- Sign up for a demo time slot via Canvas **before** the beginning of the first demo slot. This is the only way you can ensure you will have a slot.
- If you cannot attend any of the predetermined demo time slots, e-mail the Head TA **before** the beginning of the first demo slot.
- If you know you are going to miss your demo, you can cancel your slot on Canvas with no penalty. However, you are **not** guaranteed another time slot. You cannot cancel your demo within 24 hours or else it will be counted as a missed demo.

- Your overall project score will be `((project_score * 0.5) + (demo_score * 0.5))`, meaning if you received a 90% on your project, but a 30% on the demo you would receive an overall score of 60%. **If you miss your demo you will not receive any of these points and the maximum you can receive on the project is 50%.**
- You will be able to makeup one of your demos at the end of the semester for half credit.

9 Rules and Regulations

9.1 General Rules

1. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.
2. Please read the assignment in its entirety before asking questions.
3. Please start assignments early, and ask for help early. Do not email us a few hours before the assignment is due with questions.
4. If you find any problems with the assignment, it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

9.2 Submission Conventions

1. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.
2. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

9.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.
2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. 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 Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.
3. You are additionally responsible for ensuring that the collaborators list you have provided in your submission is accurate.
4. Projects turned in late receive partial credit within the first 48 hours. We will take off 30% of the points for a project submitted between 0 and 24 hours late, and we will take off 50% of the points for a project submitted between 24 and 48 hours late. We will not accept projects turned in over 48 hours late. This late policy is also in the syllabus.

5. You alone are responsible for submitting your project before the assignment is due; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until the deadline.

9.4 Is collaboration allowed?

From the syllabus:

- You must submit an assignment or project as your own work. No collaboration on answers is permitted. Absolutely no code or answers may be copied from others. Such copying is Academic Misconduct.
- Using code from GitHub, via Googling, from Stack Overflow, etc., is Academic Misconduct (Honor Code: Academic Misconduct includes "submission of material that is wholly or substantially identical to that created or published by another person").
- Publishing your assignments on public repositories accessible to other students is unauthorized collaboration and thus Academic Misconduct.

Any of your peers with whom you collaborate in a conceptual manner with must be properly added to a **collaborators.txt** file. Collaborating with another student without listing that student as a collaborator is considered plagiarism.

9.5 Syllabus Excerpt on Academic Misconduct

The goal of all assignments in CS 2110 is for you to learn. Learning requires thought and hard work. Copying answers thus prevents learning. More importantly, it gives an unfair advantage over students who do the work and follow the rules.

1. **As a Georgia Tech student, you have read and agreed to the Georgia Tech Honor Code.** The Honor Code defines Academic Misconduct as "***any act that does or could improperly distort Student grades or other Student academic records.***"
2. You must submit an assignment or project as your own work. **No collaboration on answers is permitted. Absolutely no code or answers may be copied from others. Such copying is Academic Misconduct.**
3. Using code from GitHub, via Googling, from Stack Overflow, etc., is Academic Misconduct (Honor Code: Academic Misconduct includes "*submission of material that is wholly or substantially identical to that created or published by another person*").
4. Publishing your assignments on public repositories accessible to other students is unauthorized collaboration and thus Academic Misconduct.
5. Suspected Academic Misconduct will be reported to the Division of Student Life Office of Student Integrity. It will be prosecuted to the full extent of Institute policies.
6. Students suspected of Academic Misconduct are informed **at the end of the semester**. Suspects receive an *Incomplete* final grade until the issue is resolved.
7. We use accepted forensic techniques to determine whether there is copying of a coding assignment.
8. **If you are not sure about any aspect of this policy, please ask Dr. Conte.**