

Lab 1: Intro to Logic Simulation

Due Friday, April 26th ,2024 11:59 PM on Gradescope

Minimum Submission Requirements

- Your lab1 work must contain the following files:
 - part_a.dig
 - part_a.txt
 - part_b.dig
 - part_b.txt
 - part_c.dig
 - part_c.txt
- Upload your files to the Lab1 Gradescope submission portal




Objective



This lab will introduce you to a schematic entry logic simulation program, [Digital](#). In this lab you will practice creating truth tables and implementing logic based on those truth tables. To install Digital

1. Ensure you have a Java Runtime Environment on your computer. A good one is [AdoptOpenJDK](#), which has installation options for Mac, Windows and Linux or just the normal [Java](#).
2. Download [Digital.zip](#) and uncompress it.
3. In the Digital folder open Digital.jar (it's that simple!)

Tutorial

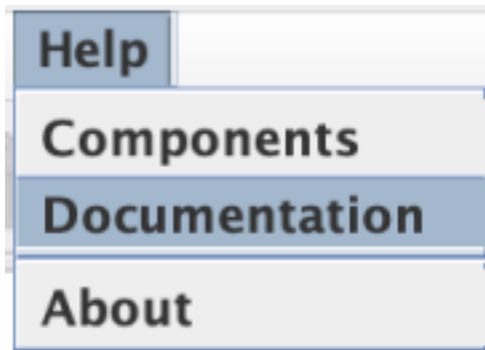
Before starting the lab assignment, follow this tutorial on how to build a very simple circuit.

- Start a new circuit by inserting an input  into the graph from Components->IO.
- Place a second input below the first. You can either select "input" again from Components->IO or you can click on the input icon in the upper right toolbar, which displays recently used components.
- Select an "Exclusive Or" (XOr)  from Components->Logic and place it to the right of the two inputs. **NOTE: If XOR gate has not yet been discussed in class as of you reading this sentence, proceed to the section [Appendix](#).**
- Select an output  from Components->IO and place it to the right of the XOr gate.
- Connect the inputs to the XOr gate by clicking on the red dot of the input and then one of the XOr's blue dots.
- Connect your XOr gate to your output in the same way.

- Now that you have a functioning circuit, you can start a simulation by clicking the Play button  in the toolbar. Test some different inputs to ensure the XOR gate works the way you want it to.
- Once you have explored the functioning circuit, stop the simulation with the Stop button  in the toolbar.
- Finally, label your inputs and output by right clicking on them (control-click on Mac.)

Resources

You can find the English documentation for Digital under Help->



Documentation:

You can also find the documentation in various other languages

[here](#). Finally, a video walkthrough of the tutorial can be found

[here](#).

Specification

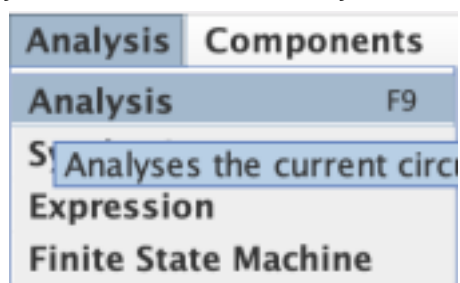
For all parts of the design, assume an ON LED represents “1” and an OFF LED represents “0.”

Part A

- For Part A, implement the truth table below using either **CANONICAL** Sum of Products (SOP) or Product of Sums (POS) using gates, not transistors. **Label your inputs and outputs exactly as is shown below (wrong case or spaces will fail the auto-grade), as they will be referenced in grading. You MUST not simplify the SOP/POS expression at this stage!**

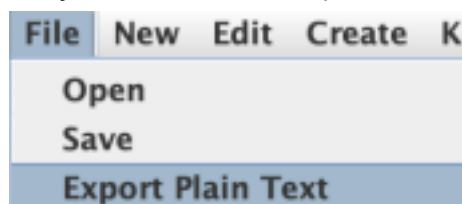
in_0	in_1	in_2	out_0
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

To quickly check if your circuit is correct, you can generate a truth table for your current circuit by selecting Analysis->Analysis:



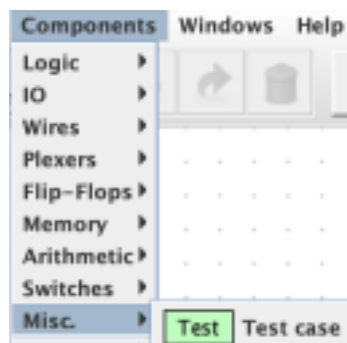
This will bring up the truth table that your circuit currently represents.

For this assignment, you will be submitting the plain text output of your truth table as well as the digital file. Once you have correctly implemented a particular circuit, you can save the output of the automated analysis by selecting Analyses -> Analyses -> File -> Export Plain Text.

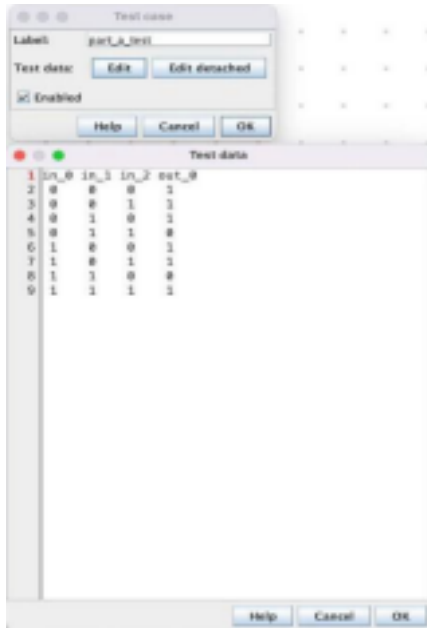


Copy paste the text from the pop up window into a text file. Save it as **part_a.txt/ part_b.txt/ part_c.txt** depending on the requirement. **Make sure the encoding of the text file is UTF-8.** If you are on Windows, I'd highly recommend [Notepad++](#) as your text editor. Simply go to Notepad++ -> Encoding and check that it is UTF-8. In general, you should use Notepad++ for coding assignments in High Level Languages since it very convenient for that purpose.

Finally, you must create an automated test to verify your results. You can create a test component by selecting Components->Misc:

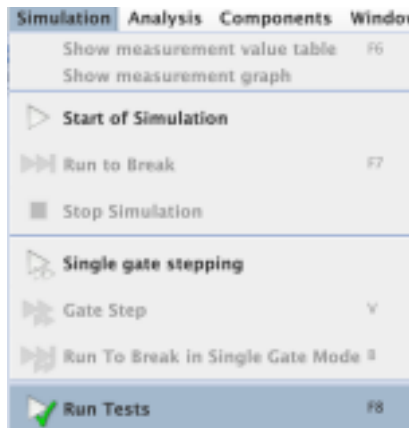


Once placed, you can right click on the test to name it, and hit "edit" to enter the expected output of your circuit. The first line should contain your input names and your output name, separated by a space, while the following lines should contain your expected truth table values:



NOTE: The truth table in the diagram above does not match the intended truth table in Part A and is only meant to serve as an example.

Name your test “part_a_test”. You can automatically run your test by selecting Simulation->Run Tests:



Once you have confirmed that your test runs correctly, save the plain text export of your automated analysis as part_a.txt, and the digital file as part_a.dig. For all your .dig files, you should make sure they are following UTF-8 encoding as well. Usually, Windows users don't need to worry about this but other systems do tend to use different encodings.

Part B

For Part B, implement a simplification of the following expression using the rules explained in class (using gates, not transistors)

$$out_0 = (\overline{in_0})(\overline{in_1})(\overline{in_2}) + (\overline{in_0})(in_1)(\overline{in_2}) + (in_0)(\overline{in_1})(\overline{in_2}) + (in_0)(in_1)(\overline{in_2}) + (in_0)(in_1)(in_2)$$

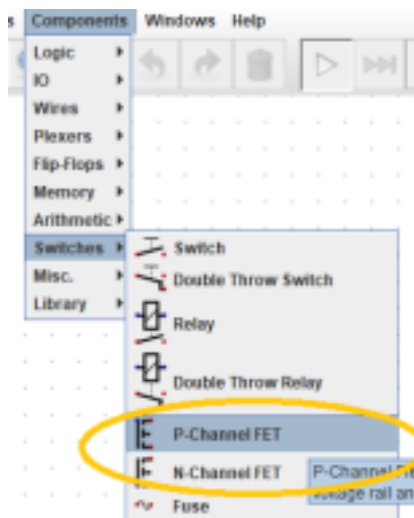
Once you have simplified the expression to its most basic form, create the circuit in Digital. Once you have created the circuit, create a test named “part_b_test” that represents the truth table of the above expression to confirm it is working.

Save the text output of the truth table as well as the digital file itself.
Save these as part_b.txt and part_b.dig, respectively.

Part C

For Part C, build the following truth table using P channel and N channel Field Effect Transistors (FETs). Note these are the same transistors which we described in lecture as PMOS or NMOS FET transistors.:

in_0	in_1	in_2	out_0
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



The FET components can be found under Components->Switches:

Once you have finished, create an automated test named “**part_c_test**” with the above truth table to confirm it is working correctly. Save the text output of the truth table as well as the digital file itself. Save these as part_c.txt and part_c.dig, respectively.

APPENDIX

The XOR gate: It’s really not a complicated gate to understand if you have fully understood how the AND, OR, NOT gates work. Just like AND, OR, the XOR gate is a multi-input (>1), single output logic gate. If I were to describe a general n input

XOR gate ($in_1, in_2, in_3, \dots, in_n$) with the output signal out, then I’d verbally say: “out is high ONLY if **odd** number of inputs are high”.

The following the truth table for a 2-input XOR gate:

in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	0

In the above truth table, out is 1 only when 1 (an odd number) of the inputs is

1. Likewise, here's the truth table for a 3-input XOR gate:

in_1	in_2	in_3	out
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

In the above truth table, out is 1 only when 1(an odd number) of the inputs, or when 3(another odd number) of the inputs, is 1. For the other cases when the number of inputs is even (0,2), out is 0. Note that in the context of digital logic, we consider the number 0 to be an even number.

GRADING

30 pt Part A implemented Correctly

30 pt Part B implemented Correctly

40 pt Part C implemented Correctly