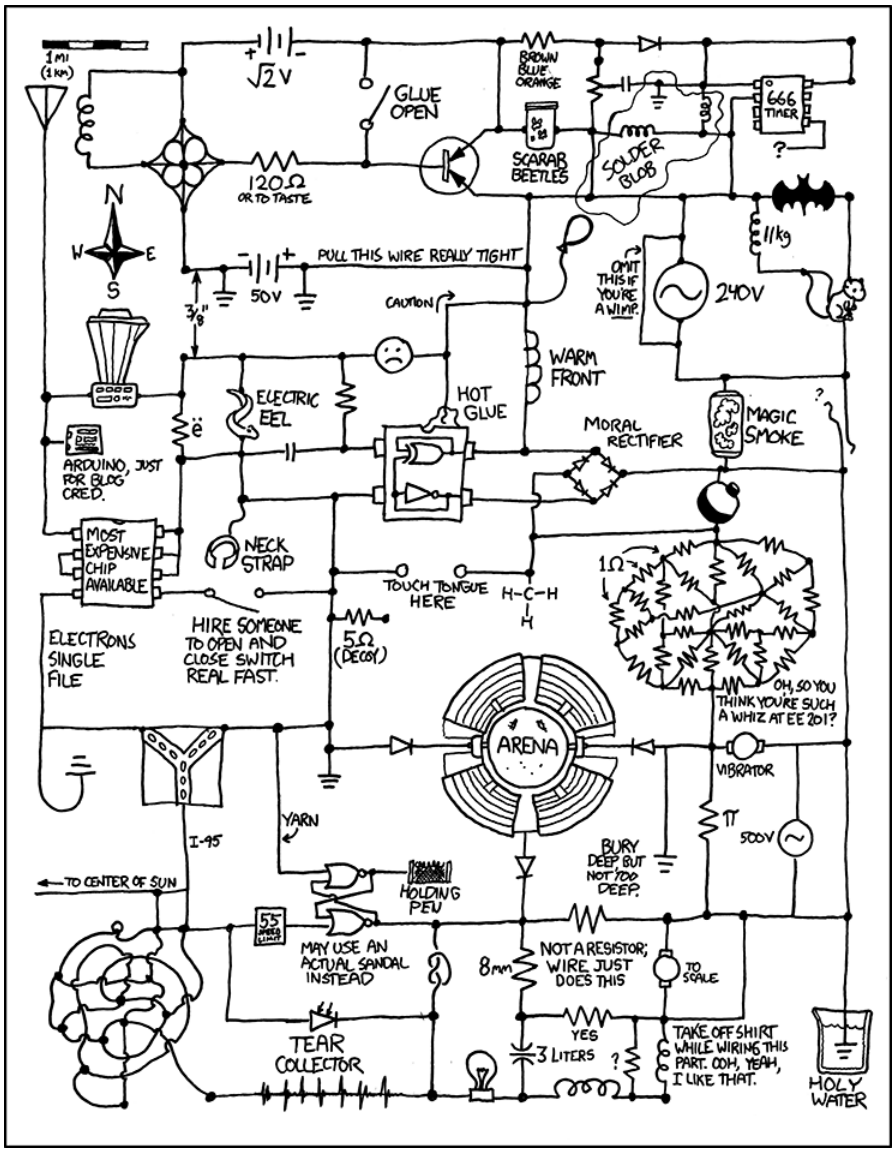
CS51 - Assignment 4
Circuit Design

Due: October 2nd, at 11:59pm



https://xkcd.com/730/

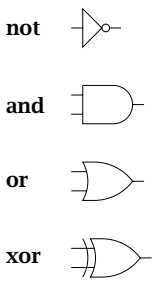
For this assignment, we will:

- Learn how to use Logisim-Evolution, a visual tool that simulates circuits using gates and digital logic.
- Design increasingly complex circuits that simulate core functionality of CPU circuitry.
- See in action how left and right shift are implemented.

1 Background

This assignment is about circuits and Boolean logic. It will give you an idea about how computers are organized as physical (or electrical) objects. We will be using a tool called Logisim-Evolution for drawing circuits. See the appendix, later in this assignment, for some pointers about Logisim-Evolution and for information on where to obtain a copy.

A circuit is assembled from gates. Consult the past lectures to refresh your memory on circuits. As a reminder, you can find below the symbols for some of the most fundamental gates. In this assignment, we will create a few small circuits, which are themselves used as larger components in assembling a whole computer.

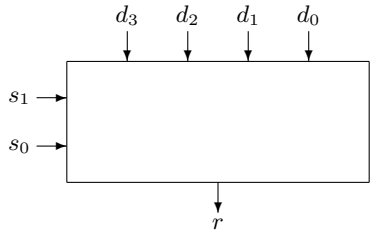


You will submit one file, created in Logisim-Evolution, named `assignment4.circ`, the `assignment4.tex` and resulting `assignment4.pdf` in the usual way. The file will contain several circuits, one for each problem on the assignment. Please read and follow the instructions later in this assignment for creating and naming individual circuits. Label your input and output pins as shown in the block diagrams.

2 Multiplexers and Minterms

Recall the 2-bit decoder from the lecture on Combinational Circuits, a circuit that takes 2 inputs and converts them to up to 4 outputs. One can use the decoder pattern to create a *multiplexer,* a circuit that acts as a switch that chooses a single output from among several possible values based on the values of selection signals. Specifically, a k:1 multiplexer receives 2^k inputs and gives as an output a single one of them based on the values of k selection signals.

Here is a block diagram of a $4:1$ multiplexer with four inputs, d_0, d_1, d_2, and d_3 and two selection signals, s_0 and s_1.



This is a truth table showing how the multiplexer would produce the output r based on the values of the select signals s_0 and s_1. Note that each of the four possible combinations leads to the selection of one of the d_0, d_1, d_2, or d_3 input signals. For example, when the input value $s_1 s_0$ is 01, for example, the input d_1 is passed to the output r.

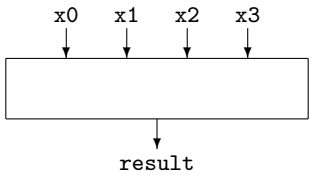
s_1	s_0	r
0	0	d_0
0	1	d_1
1	0	d_2
1	1	d_3

2.1 Design a 4:1 multiplexer [2 points]

Create a circuit named `multiplexer_4_1` using only the basic `and`-, `or`-, and `not`-gates. Use the 2-bit decoder for inspiration. Label the input and output pins as they appear in the block diagram above.

2.2 Minterms [2.5 points]

Start by creating the truth table and then use the minterm expansion technique we used to calculate the disjunctive normal form of a Boolean function to create a circuit that has four inputs and one output. The output of the circuit is to be 1 when *exactly two* of the inputs are 1. Name your circuit `exactly_two`.



If you were to create the K-map, you would quickly see we can't do better than that.

You should include the truth table, disjunctive normal form, and the circuit in `assignment4.tex`.

3 Circuits from a CPU

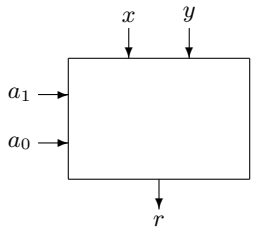
The central processing unit (CPU) of a computer is responsible for carrying out arithmetic and logic operations that are dictated by a program. One of its components, the arithmetic logic unit or ALU applies a provided operator to two operands. For example, the ALU is responsible for adding, subtracting, and performing bitwise arithmetic on two numbers, in addition to performing logical and arithmetic shifts.

3.1 Baby ALU [2 points]

For simplicity, we will focus only on the scenario of being given two bits, x and y, and applying either i) bitwise *and*, ii) bitwise *or*, or bitwise *xor*, based on two bits a_1, a_0 to produce a result r that applies the following truth table. When a_1 and a_0 are equal to 0, we apply a bitwise *and*. When a_1 is 0 and a_0 is 1, we apply a bitwise *or*. And when a_1 is 1 and a_0 is 0, we apply a bitwise *xor*. Here is a sample, but you will want to build the full truth table in `assignment4.tex`:

a_1	a_0	x	y	r
0	0	0	1	0
0	1	0	1	1
1	0	0	1	1
1	0	1	1	0

Create a circuit named `baby_ALU` that has the structure shown in the block diagram below. The value `r` will be the result of one of the three bitwise logical operations we just saw applied to `x` and `y`. Obviously, only three of the four possible values of a_1 and a_0 will be relevant. The output of your circuit may take the red "error" value for the unused values. You may use your multiplexer from the previous problem and any of the basic gates. Label the input and output pins as shown in the diagram. Add the screenshot of the circuit in `assignment4.tex` after expanding the full truth table.

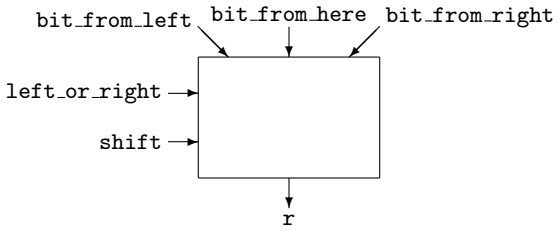


3.2 Shifting

Let's now think about how we could implement shifting. We assume that the bit a_0 determines the direction of the shift (0 for left and 1 for right), and bit a_1 determines the nature of the shift (0 for logical and 1 for arithmetic). The only thing that is missing is the magnitude of the shift. For simplicity, we will work with 4-bit numbers, so the magnitude of a shift can be specified with 2 bits.

3.2.1 1-bit shifting [1.5 point]

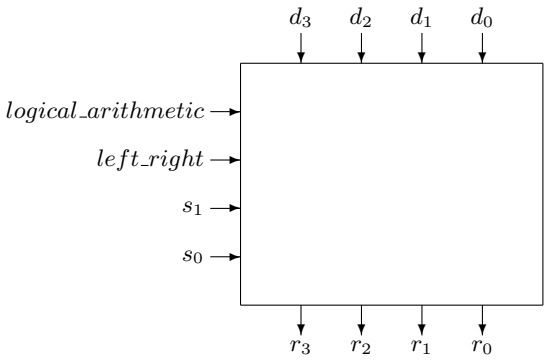
Begin by creating a special kind of multiplexer that is responsible for one bit of a shift. Call your circuit `shiftplexer` and include a screenshot of it in `assignment4.tex`.



If `shift` is 0, then there is no shift; the contents from the `bit from here` is passed directly to the result r. If `shift` is 1 and `left_or_right` is 0 signifying a left shift, then the contents of `bit_from_right` is passed to the result r. If both `shift` and `left_or_right` are 1, then the contents of `bit_from_left` is passed to the result r.

3.2.2 Barrel shifter [2 points]

Use multiple copies of your shiftplexer component to construct a 4-bit *barrel shifter* with the structure shown in the block diagram below. Name it `barrel_shifter` and label the pins as shown.



The four input bits $d_3d_2d_1d_0$ along the top are the bits to be shifted. The four input values on the left specify the type (`logical/arithmetic`), direction (`left/right`), and magnitude (s_1s_0 interpreted as a 2-digit binary number) of the shift. The 4-bit number $r_3r_2r_1r_0$ at the bottom is the result of the shift.

You will need eight `shiftplexer`s arranged in two rows. One row will have a shift distance of one and will be controlled by s_0. The other row will have a shift distance of two and will be controlled by s_1. The input bits will flow into the first row of `shiftplexer`s, then the output bits from that row will go into the next row of `shiftplexer`s, and then finally the output bits of the second row are the output bits of the circuit. In addition to the four bits to be shifted, you will need a bit to

be shifted in from the right (always 0) and another bit to be shifted in from the left (0 if it is a logical right shift or d_3 (the most significant bit) if it is an arithmetic right shift.)

Here are a few sample results:

control				input				output			
s_1	s_0	$left_right$	$logical_arithmetic$	d_3	d_2	d_1	d_0	r_3	r_2	r_1	r_0
0	0	0	0	0	1	1	0	0	1	1	0
0	1	0	0	0	1	1	0	1	1	0	0
0	1	0	1	0	1	1	0	1	1	0	0
1	0	1	0	1	0	1	1	0	0	1	0
1	1	1	0	1	0	1	1	0	0	0	1
0	1	1	1	1	0	1	1	1	1	0	1
1	1	1	1	1	0	1	1	1	1	1	1

Hints and observations

- Make sure that you understand the description above, specifically, how you can get a shift distance of between 0 and 3 positions by creating the two rows of `shiftplexer`s.
- As you do when developing code, you can take an incremental approach to building your circuit. For example, you might just wire up one row of `shiftplexer`s and then test to make sure it works. Then incrementally add the `shiftplexer`s for the next row. Do whatever makes the most sense for you, but consider doing it in a way that allows you to test as you go; Logisim-Evolution has good functionality for playing with a working circuit.
- It is easy for more complicated circuit diagrams to get messy and confusing. Follow regular patterns to keep your wires from getting too tangled. And feel free to start over when your circuit is too confusing. It is often much easier to begin with a fresh canvas than to try to fix up a bad circuit.
- Sometimes Logisim-Evolution gets in a bad state. If you find that you're seeing weird behavior (like input wires with error codes), try saving your file and then restarting Logisim. On the other hand, sometimes the problem is in your circuit!

4 Feedback

Create a file named `feedback4.txt` that answers the questions:

- How long did you spend on this assignment?
- Any comments or feedback? What things did you find interesting, challenging, or boring?

5 When you're done

Submit your `assignment4.circ`, `assignment4.tex`, and resulting `assignment4.pdf`, and `feedback4.txt` files online using Gradescope; be sure you're uploading them to the right assignment. Note that you can resubmit the same assignment as many times as you would like up until the deadline.

6 Grading

	points
Part 2.1: Design a 4:1 multiplexer (screenshot and circuit)	2
Part 2.2: Create the truth table	0.5
Part 2.2: Calculate the DNF	0.5
Part 2.2: Design the exactly-two circuit using DNF	1.5
Part 3.1: Create the truth table	0.5
Part 3.1: Design a baby ALU (screenshot and circuit)	1.5
Part 3.2.1: Design shiftplexer (screenshot and circuit)	1.5
Part 3.2.2: Design barrel shifter (screenshot and circuit)	2
Total	10

Appendix: Working with Logisim-Evolution

You can download Logisim-Evolution from this link in compiled form. Make sure you select the right version based on the operating system and processor your system has. Note the special instructions if you are on MacOS. If you are unsure, talk with one of the course staff. You can also use the lab computers that already have Logisim-Evolution installed.

Logisim-Evolution is (usually) an easy program to use. There is a tutorial in the Help menu that you can use to get started. The four main areas of the Logisim-Evolution window are the small Toolbar in the upper left corner, the Explorer Pane below it, the Attribute Table in the lower left, and the Canvas occupying most of the area on the right. The Canvas is where you will be using your mouse to construct the circuits.

Creating and naming circuits in Logisim-Evolution In the main menu, under Project, select Add Circuit. Then type the name of the circuit (as specified in the problem statements) in the window that opens. A new Canvas will open, and its name will show in the Explorer Pane. Do this before you start to create each circuit.

Once you have created a circuit, you can use it as a component in subsequent constructions. Just click on the name in the Explorer Pane and then in the Canvas of the new construction. Any changes you make later to the component circuit will be reflected in all the places where it is used. This will be useful when working with circuits like the barrel shifter.

Adding components and wires To add a component, click on its image in the Toolbar and then again in the canvas where you want the component to be. Once on the canvas, a component can be moved to where you want it. When a component is selected, the Attribute Pane[1] displays several options, like the orientation of the component, the size of the component, the location of the label, the content of the label, and the number of inputs. You will frequently have to change the attributes from their default values.

To add a wire, hold the mouse button down while dragging the mouse. To route wires neatly, create them in segments—either straight lines or single right angles.

Input and output pins can be found on the main menu above the canvas. If you need multiple copies of a gate or another component, it is easiest to create just one and adjust its properties. Then you can copy and paste to make identical copies and move them into place.

Logisim-Evolution has many different circuit elements that we will not use. Nearly all the gates that you will need appear in the Toolbar. Two exceptions are the constant value (as opposed to a variable input), which you can find in the Explorer Pane under Wiring, and the xor-gate, which is in the Explorer Pane under Gates.

Laying out your circuits You will see, after only a few minutes with Logisim-Evolution, how easy it is to create a tangled mess of wires and components. To the extent you can, sketch the circuit on paper before trying to construct it in Logisim-Evolution. That way, you can avoid moving existing components to make space for a new one.

It is usually best to place the components before adding wires. Line them up so that the wires are as straight as possible. Try to make your circuit as regular as possible, for example by aligning similar components, keeping distances between components uniform, and having wires with the same function follow similar paths.

Remember to arrange the pins along the edges of the circuit, in the same relative positions as they are shown in the block diagrams. For this assignment, input pins (e.g., bits of input) are at the top, control pins (which are a form of input but correspond to things like whether a shift is left or right, or how much to shift) are on the left side, and output pins are at the bottom.

If you move a component and the wires get twisted, just delete the wire segments one at a time and then add them back in. More drastically, if you somehow created a mess, it is often easier to delete the whole thing and start over. At that point, you know how the circuit will look, and you can quickly regenerate the components and connections.

Testing your circuits When you have completed a circuit, all the wires should be green. Here is Logisim-Evolution's color coding:

Dark Green: The wire is carrying a 0.

Bright Green: The wire is carrying a 1.

[1] You may want to use the smaller "narrow" gate size instead of the default of "medium."

Red: The wire has an error value, probably because it is connected to the output of a gate which is unable to determine its value.

Blue: The wire has no value, probably because it is not connected to a source.

You can change the values of the input pins using the Poke Tool (the pointing finger) and observe the values of the output pins.

Saving your circuits Select File ⟶ Save. The first time you save a circuit, you will be prompted for a file name and a directory. Make sure both are correct.

To return to an existing circuit, open Logisim-Evolution and select File ⟶ Open or File ⟶ Open Recent.