ECE2883 HP: Lab 3- non-Logic Experts

Combinational logic primer

The essentials of a prerequisite logic course: Part 1

We should be starting to realize that the "SMEs" in this course are just a specific type of SME – they are "logic experts." Everyone has relevant skills, but SMEs have a background in logic. So this track of learning is now called non-LE, instead of non-SME, and we are trying to get *you* to be LEs, to some extent.

In the previous logic course of the Logic Experts, a lot of material was covered, but only two major areas are essential to begin designing logic circuits of all kinds:

1. Combinational logic
2. Sequential logic

We have already touched upon combinational logic, and this module will go into more detail in that area. We will return to sequential logic in a couple weeks with a similar learning module, but we can make the distinction between the two clear at this time:

- Combinational logic is just the generalization of Boolean algebra expressions, including the examples that we have begun discussing, from the light switch examples to other simple examples given in Wills and Wills.
- Sequential logic can be thought of as logic that behaves differently at different times, even with the same inputs. At one instant, the logic produces a certain output, but at some time in the future, perhaps a tiny fraction of a second later or minutes or hours later, it could produce a different output because of the *history* of events that have occurred.

Think of it this way. You may think that your feelings about ice cream are simple combinational logic:

```
I like ice cream = True
```

But, if you've been eating ice cream for every meal for a month, you might realize that your feelings are better described by sequential logic:

```
I like ice cream = True
I like ice cream = True
I like ice cream = True
…
I like ice cream = False
```

This was sloppy sequential logic, because the expression doesn't really just change its definition as this would imply. But it is a function of something not explicitly shown here, like time or the number of times something has happened.

But that will come later, and for now let's focus on combinational logic.

Basics of logic

Homework assignments 1 and 2 included reading assignments in Wills and Wills. If any of that was unclear, reread, or better yet consider the optional reading assignment in Harris and Harris (Chapter 1, stopping after section 1.5).

Wills & Wills used switches and light bulbs to show a way to implement logic circuits, as in Figure 1, where a lit bulb could be viewed as True or False (equivalently 0 or 1), and pressing a switch was equivalent to providing a True (or 1) input. So, if the light bulb in the figure represents a Boolean variable Out, then the circuit represents

Out = A AND (B OR C)

since the bulb will light when A is pressed **and** either B **or** C is pressed.

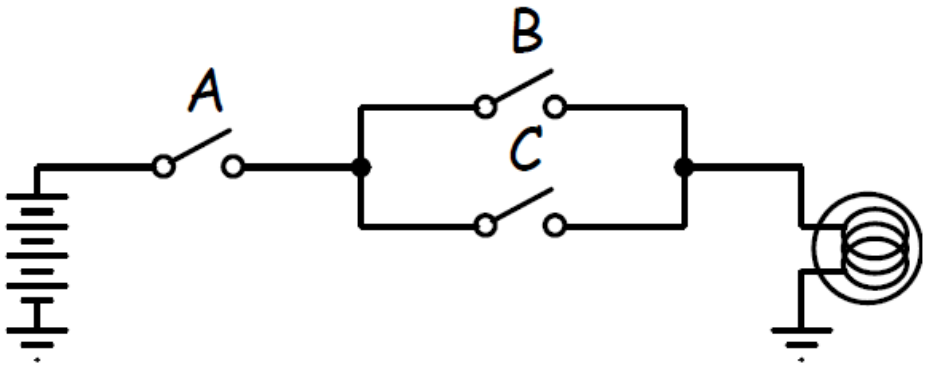

Figure 1. Example from Wills & Wills..

Wills & Wills also showed that you could list all of the possibilities for A, B, and C in something called a truth table, putting the value of Out in the last column.

The first exercise here is a self-test of that knowledge.

Exercise 1

On the answer sheet at the end of this document, draw the circuit with a light bulb that is lit under **either or both** of the following conditions:

1) Switch A **or** Switch C is pressed, **and** Switch B is pressed, or

2) Switch D is pressed.

You also learned that these Boolean equations had standard operators, where a dot means AND and a plus sign means OR. The proper equation for Figure 1 is $Out = A \cdot (B + C)$, where the dot symbol can be omitted, just as the dot symbolizing multiplication is often omitted in "regular" algebra.

Exercise 2

On the answer sheet, write the proper equation for the circuit you drew in Exercise 1, using dot and plus symbols as needed. Note that if you omit parentheses, then AND is evaluated in precedence to OR, just as in "normal" algebra multiplication is evaluated before addition. As an example: $x \cdot y + w \cdot z$ is evaluated as $(x \cdot y) + (w \cdot z)$.

The last major thing that you learned about Boolean logic in the Wills & Wills reading was that there is a way to write these equations as diagrams, with specific symbols for the major operations of AND, OR, and NOT. When drawn this way, these are called gates, and they correspond exactly to electronic devices. For example, if you have a gate that takes in two inputs and produces the OR of the inputs as the output (i.e., a two-input OR gate), this corresponds to an electronic device that will produce a logic high (high voltage) whenever either (or both) of the inputs are logic high.

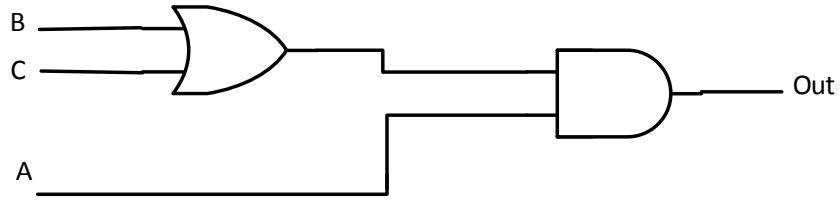

Figure 2. The gate diagram, or schematic, for the logical relation described by Figure 1..

Exercise 3

On the answer sheet, draw the gate diagram for the circuit that you drew in Exercise 1.

And finally, it is worth mentioning that sometimes NOT doing something is just as relevant as doing something. The switches shown in Figure 1 are deliberately drawn to be "normally open," so that they conduct electricity only when pressed, and "TRUE" is equivalent to pressed. They could have been drawn to show "normally closed" switches that conduct electricity as long as they are NOT pressed. This changes the logical sense of an input, and would correspond to putting an inverter (a NOT gate) on an input such as those of Figure 2.

What you missed by not having a logic theory course

Equations like the one for Figure 1, and the one you came up with for Exercise 2, can get much more complicated, either by having more variables or more operators, or both.

In general, they may not be written in their simplest form, and gate diagrams may have more gates than are needed.

It was not terribly important, but a table in Wills & Wills listed properties of Boolean algebra like commutativity and associativity (and more). These probably seem a little familiar from "regular" algebra, although there are some differences.

Engineers who specialize in the design of efficient circuits will utilize (whether they know it or not) those properties to minimize the underlying Boolean equations and use fewer gates. A fair amount of time is

spent in a logic theory course acquainting students with techniques for doing this minimization. Some of those students will eventually develop software tools that do this for practicing engineers, so this is very important. Others simply need to understand it, much as all of us benefit from understanding why arithmetic works, even though we typically use calculators to do all but the simplest computations.

For our purposes in this course, the Non-LEs do not need to concern themselves with coming up with the **best** logical equations. It is sufficient that they understand what the logic equations are telling us. There are really two reasons that being especially efficient in producing equations is not important:

1) Usually, the software tools that you will soon learn are doing this for you, much as a calculator does arithmetic for you, and
2) Even if you needed to do it manually, your project team will have an LE or two.

Number systems

Everything above was related to reading in Homework 1. There was also some reading in Homework 2 from Wills & Wills, related to number systems. Most of you probably have had some experience with binary numbers, and this reading was intended to prepare you for the exercise near the end of Lab 2 where you needed to interpret the position of slide switches as being a binary number (switch down =0, switch up =1). This will not be the last time you will need to use this knowledge. The 0/1 interpretation of voltage will be more common than the false/true interpretation, especially when we group multiple signals (bits) together to make larger numbers (bytes, words, etc.).

Some of the material covered in that section of Wills & Wills was fairly "mechanical" in nature (apologies to the MEs). Converting numbers between binary and decimal can be done by different procedures that are not especially fun, for example. In the long run, we typically use computers or calculators to do this for all but the smallest numbers. So rather than learn some efficient algorithm for conversion, focus instead on the underlying principle, so you could do it yourself on an exam. Specifically, if you want to convert a number to binary, just list all the powers of two that are smaller than the decimal number you are converting, and see which ones "fit" in your decimal number, starting with the largest.

Example:

If your decimal number were 1745, then list: 1, 2, 4, 8,16, 32, 64, 128, 256, 512, 1024. There's no point in listing 2048, the next number in the sequence, because it is bigger than the number we are converting.

Starting with the largest power of two in your list, write a 1 if you can fit it in the number you are converting, subtract it from the number, and repeat for the next-smaller power of two. (Write a 0 if you cannot fit it.) Go until you get to the last power of two.

1745 : 1024 - 1 (1024 "fits" in 1745, and after removing it, produces 721 for the next step)
721 : 512 - 1 (fits, producing 209)
209: 256 - 0 (does not fit, so still 209)
209: 128 - 1 (fits, producing 81)
81: 64 - 1 (fits, producing 17)

17: 32 - 0 (does not fit, so still 17)
17: 16 - 1 (fits, producing 1)
1: 8 - 0 (still 1)
1: 4 - 0 (still 1)
1: 2 - 0 (still 1)
1: 1 - 1 (fits, and done, since there are no more powers of two to consider)

Write the bits down in order from largest power to smallest: $11011010001_2 = 1745_{10}$.

Converting binary to decimal is usually more obvious, since the binary "places" have clear meaning, and it is like coming home to our "normal" decimal base system. The rightmost position is the ones place. The next one to the left is the twos place, then the fours place, and so on. So converting a number like the one we just found, 11011010001_2, is straightforward. There is 1 one, 0 twos, 0 fours, 0 eights, 1 sixteen, and so on:

1(1)+0(2)+0(4)+0(8)+1(16)+0(32)+1(64)+1(128)+0(256)+1(512)+1(1024)

=1+16+64+128+512+1024

$=1745_{10}$

Exercise 4
Convert 363_{10} to binary.

Exercise 5
Convert 1100110_2 to decimal.

Motivation and revisiting project-based learning
LEs taking a typical digital design course are often in the same situation that you may be in right now. You have seen some abstract concepts in logic, and some very simple Boolean equations, and it may be unclear why this important or how it can be used.

This is where ECE2883 can really help. We can immediately see some applications relevant to the hardware components that we have been discussing in class. One of the candidate projects found by a classmate was the Waterfall Swing. If you need a refresher about how it worked, see https://youtu.be/p1uwQVtHHOQ . Let's pick out part of the swing, a cross section such as shown in Figure 3.

Here, the water is allowed to flow when a solenoid valve is opened, and the solenoid is controlled by a Boolean value V: V=1 opens the valve, V=0 closes the valve. There are a variety of ways that the position of the swing could be sense, but suppose there are several camera-like sensors that produce a Boolean value of 1 when the swing is passing over them. Figure 3 shows five such sensors, and in the exact situation shown, most of them are producing a value of 0 (e.g., B=0, C=0, …), but A is producing the value of 1.

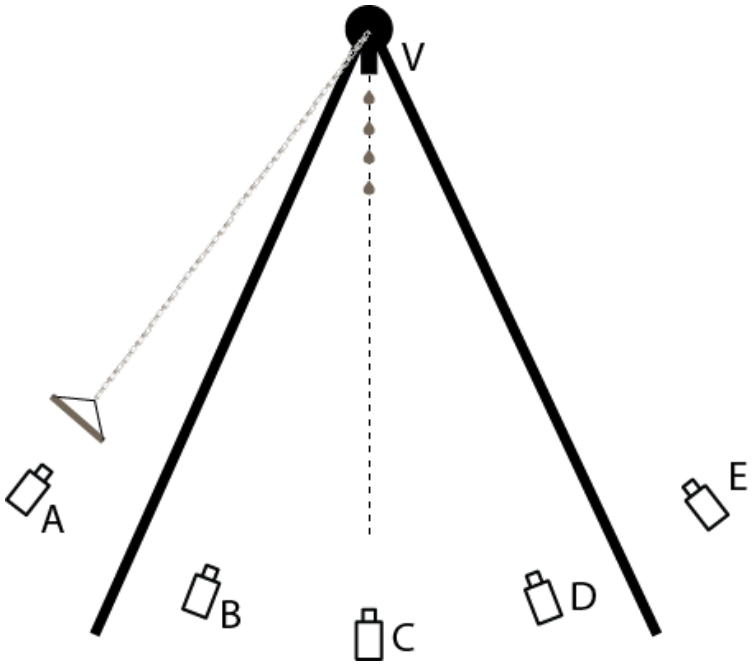

Figure 3.The Waterfall Swing could be implemented with a water valve and multiple sensors of swing position..

A simple way to control the valve, preventing water from flowing when the swing is passing through the low point, is

$$V = \bar{C}$$

In other words, V is true (open) whenever C is not true (does not sense a swing passing over it). Or V is false (closed) whenever C is true (DOES sense a swing passing over it).

This may be an inadequate solution. The swing is probably moving too fast for this strategy to stop the water soon enough. Because the water is already falling as the swing approaches the low point, the swing will be hit with water. One way to improve the solution is to stop the water whenever the swing is at the low point or even approaching the low point:

$$V = \overline{B + C + D}$$

In other words, V is false (closed) whenever either B or C or D is true.

Either of these solutions is trivial to implement with digital logic. The second solution, although better, is probably too conservative to make the swing very exciting. The water would be off most of the time. This example foreshadows the next major topic, sequential logic. Using sequential logic (and possibly more than just five sensors), it is possible to always keep track of the direction that the swing is moving, and only close the valve just early enough to stop it as the swing passes through the vertical. This is done by always using a history of sensor values, as opposed to just the current values.

Exercise 6

Read Chapter 2 of Harris and Harris, up to and including section 2.6. Don't be too concerned about the theories, axioms, and simplification of equations in section 2.3. Be prepared to discuss in class whatever you find confusing about the rest of it.

Results to turn in

The answers below are due by our seminar on Thursday, September 17.

Exercise 1

Draw the circuit, showing switches and a light bulb:

Exercise 2

Write the equation:

Exercise 3

Draw the gate diagram:

Exercise 4

363_{10} =

Exercise 5

1100110_{2} =