Assignment 1: Digital Logic Gates and Latches
ENGN/PHYS 208—Winter 2019

Background: Logic Gates

The computation foundation of any modern machine is transistor-based ***logic gates***. For example, let's say a piece of hospital equipment is to alert a nurse of doctor if a patient's heart rate *or* temperature increases above a certain allowed limit. Notice, what we just said: the alarm bell must sound (output) if either the EKG OR the thermometer (inputs) tell it to.

Today, as a hands on introduction to logic gates, you'll investigate the various types of logic gates (**OR**, **AND**, **NOT** (inverter), **NAND** (not-AND), **NOR** (not-OR), and **XOR** (exclusive OR)) and latches. We'll soon see how to combine these gates to do some really cool stuff with computers and memory elements! But first things first...

Often you will hear the term **TTL**, which stands for **T**transistor-coupled **T**ransistor **L**ogic. In other words, hook up some transistors together in a clever configuration, and they can compute something! Fortunately, the nice folks at TI, Fairchild, etc. have packaged all of these clever configurations into easy-to-use integrated circuits (aka "chips"). Refer to Fig. 1 for the corresponding chip numbers, all of which belong to the famous 74-series.

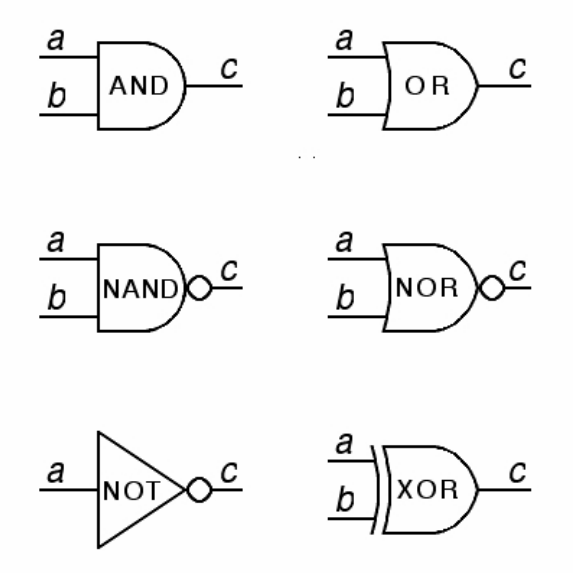

Figure 1: Collection of standard logic gates. AND = 7408. OR = 7423. NAND = 7400. NOR = 7402. NOT = 7404. XOR = 7486.

For all 74*xx* series chips listed above (see caption of Fig. 1), notice that pin outs of make them essentially interchangeable—the power connections, the inputs are on the pins, and the outputs are all on the same pin numbers regardless of logic operation. When implementing/testing logic gates, all inputs should be connected. Inputs left unconnected are termed ***floating***. Floating inputs (and outputs) typically wreak havoc in digital logic systems.

To measure the output you can use a dc multimeter. But even more fun, faster, and requiring fewer hands is to create an LED indicator. See Fig. 2 for an example. Don't forget to power your chips: Use the Vcc = +5 V and ground for your power supply connections.

1 Experiment I: Basic Logic Gate Implementation

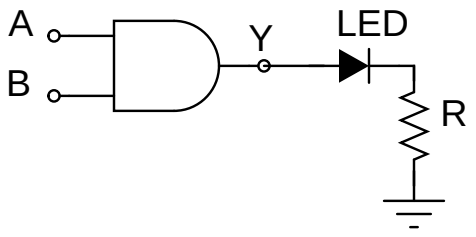

Figure 2: AND gate with inputs A and B. The output Y drives the LED. Make sure to orient it properly, or it will fry! The current limiting resistor should be ≈ 100–200 Ω. The device is powered with Vcc = +5V and Ground.

1. For all logic gates, provide an every day use of such logic. For example, take the XOR gate operating at a 4-way intersection. The logic of the stoplight might be: If there is a car heading down Main St. and there is a car on Washington St about to reach the intersection, make sure only one of them gets a green light.

2. Next, implement and test the logic operation of a few (maybe 2 or 3) different logic gates, in isolation from one another, on the breadboard. Do enough playing/testing so that you feel comfortable connecting various logic gates together in a more complicated arrangement.

2 Digital Safe Key Code:

Now it's time to integrate a bunch of logic gates and latches into a more complex device. The idea is to build the infamous Teddy KGB[1] a secure digital safe for all his chips and fortune-telling Oreos. The basic idea idea of the safe is that you must set digital bits correctly, then turn the key/push the open button. If the digital bits are set properly, then turning the key opens the safe. The safe should have a ***unique code***: It should open for one and only one code. (Otherwise, it remains a fortress for all those delicious Oreos.)

(a) Digital safe from Honeywell.

(b) Teddy KGB, from the movie *Rounders*.

Figure 3: Teddy KGB needs a digital safe. Can you help him?

Specifically, your task is to design a 4-bit (e.g., input = 1011) code based on logic gates, and implement an turn key/open button as well. The output should be a logic of 1 for *only* one set of digital bits, and 0 otherwise. Of course the fewer logic gates you use, the more obvious the solution. On the other hand, Teddy KGB greatly admires elegant solutions—so he places a hard constraint that you may use 6 logic chips maximum.

1. Carefully diagram your circuit schematic
2. Then build it to prove it works as advertise.
   (a) Again, the code should be 4 bits plus 1 "enter/open" button.
   (b) Your design must include green and red LED indicators: a green LED illuminates when the correct code has been entered, red lights up otherwise.
3. We will take turns attempting to crack each others' codes based on the hardware implementation. The one that takes the longest to crack wins a prize (a pack of oreos, of course!)

[1] Check out the movie Rounders, in particular this famous scene

3 Write-up

Due: Monday, Jan 14 2019, noon. What to submit:

1. Circuit schematic. Use a software package such as Schemeit or Eagle.
2. On your schematic, indicate the output for each logic gate, assuming the proper code to enter the safe has been set.
3. Now assume (at least) one of the bits has been set incorrectly. Indicate graphically/show logic gate output leading to safe staying locked (ultimately outputting a 0).
4. Make a brief video of your safe in action. You should show walk the viewer through a few of the test cases (there are 16 in all, but something like 4 should suffice to get the point across).