ECE 385

Lab 1: Delays / Glitches with a 2x1 MUX

Geitanksha Tandon

TA: Ruihao Yao (RY)

Geitanksha Tandon
ECE 385
Spring 2022

Purpose of Circuit

The circuit we create in this Lab is a design that creates a 2-to-1 MUX (essentially like a digital switchbox) in order to highlight the design delays and glitches (as depicted in the GG.24) that are prevalent when using TTL Technologies. While our original design used AND / OR and NOR chips, our goal here is to use only NAND gate CMOS chips, specifically the Quad-Input NAND SN7400 Chips. We employ the use of De-Morgan's Laws to do this. Once we figure out the glitches, identified as Static Hazards, we must implement a circuit that avoids the glitch, and method we use to do this is what we explore in the lab. The method used is by ensuring that there is no disconnect between the minterms on the KMap by adding a connective minterm and incorporating it into our circuit.

Written Description of Circuit

Part A: In this section, we created a MUX Circuits with 2 inputs (A, C), one output (Z), and the select line (B). The logic of the MUX is such that if B is 0, it mirrors the input of C, and if B is 1, it mirrors the input of A (As shown in Figure 13, 16). Using this logic, we decided to create a Truth Table, and then develop a Karnaugh map.

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

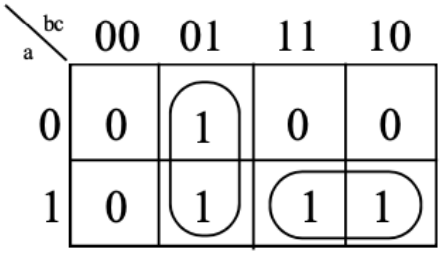


FIGURE 12. Karnaugh map of 2-to-1 Multiplexer Function

Boolean Expression for the KMap:

Z = AB + B'C

Truth Table & KMAP of the MUX Implementation:

Once we created the Boolean Expression, we expressed it in the form of a Circuit using the AND-OR-NOT circuit (shown in Fig 13*). This required 3 chips for its implementation from the TTL components available – 7404 inverter, 7408 quad 2-input AND gate, and a 7432 quad 2-input OR gate.

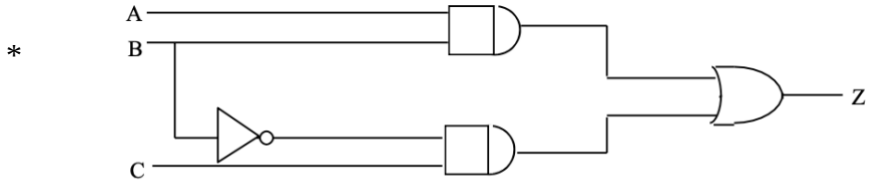


FIGURE 13. Sum of Products implemented with 2-level AND-OR Logic

**Images taken from the General Guide (GG.27)*

Due to the space/chip resources inefficiency and the lack of availability of these chips, we decided to convert this design to a NAND-NAND circuit (shown in Fig 16*). The reason is that NAND is a universal gate, so we would only require one type of gate (NAND Gate) for the implementation of the Boolean Expression, indicating that we would only need a single 7400 quad 2-input NAND chip. Once we decided to use the NAND gates for the system, we redrew the circuit diagram and constructed a new version on our breadboard (shown in the Fritzing diagram).

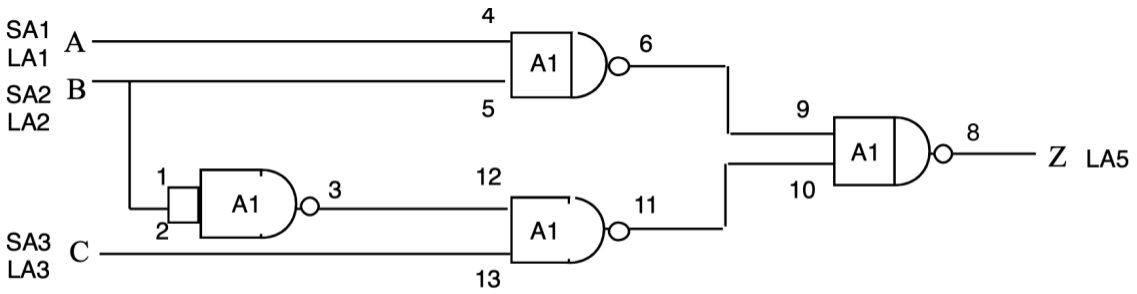


FIGURE 16. Single chip implementation using one 7400 Quad 2-input NANDs

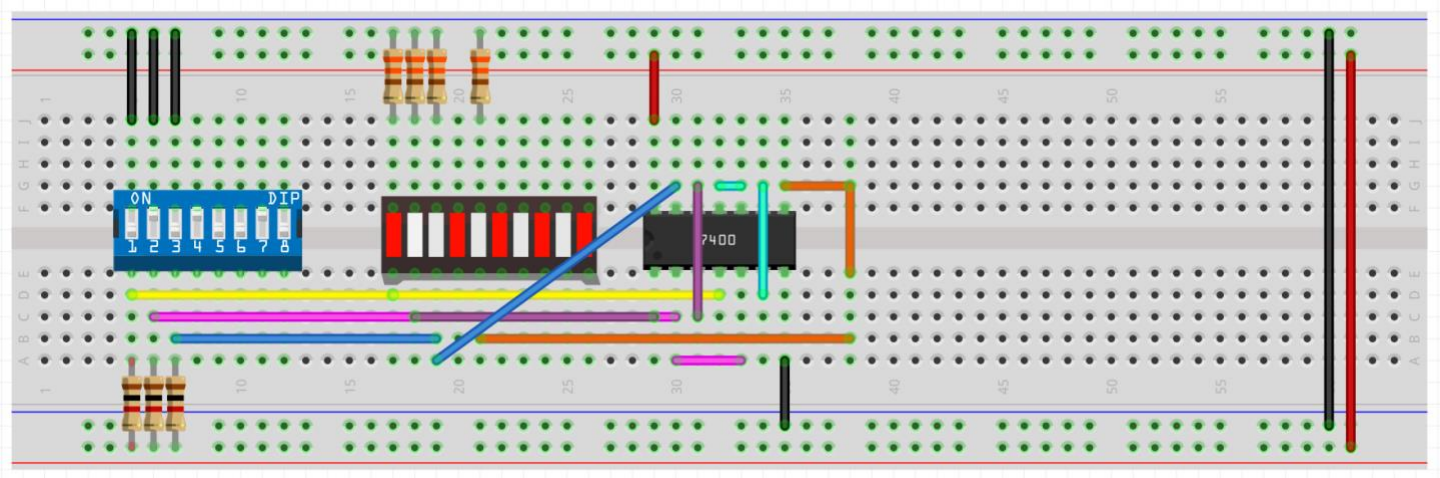


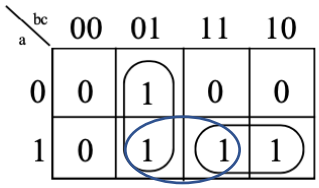
Figure 1.1: Circuit drawn via Fritzing system.

Part B: In the circuit designed above, we discover that there is a glitch when the B input (selector line) switches from 0 to 1 and vice versa. This is because when we create our Boolean expression, we use the assumption of DC behavior, that is the output Z will *eventually* turn into the required output. However, when the inputs A, B, C are changed as required, there is no guarantee that they would switch their values at exactly the same time (there could be possible gate delay that occurs as the logic propagates through the circuit) and there is left a possible scenario where they have a 0 value for an infinitesimally small period of time. This unexpected 0 could lead to a 0 in output, which would differ from the required state of high (1) output, which is known as a glitch.

The example of this happens when we take all inputs as 1 (ABC = 111) where B mirrors the A value (A=1, Z=1). If we switch the B input to 0 (in order to mirror C) we expect to see 1 (C=1, Z=1). However, because of this difference in time where B switches to 0 (the gate delay) we would reach a state where Z would go to 0. Even though it switches back to 1 (high input) two gate delays later, this temporary state of 0 (static 1 hazard) causes an unexpected output (glitch) and hence is called the naïve solution.

**Images taken from the General Guide (GG.27)*

In our scenario, in order to avoid this, we modify the creation of our Boolean expression from our KMap and cover adjacent minterms to ensure this does not happen. We have done this below* where we have added an extra minterm to eliminate the possibility of the static 1 hazard:



Boolean Expression for the KMap:

Z = AB + B'C + AC

Note: AC is an extra term to overcome Static 1 Glitch.

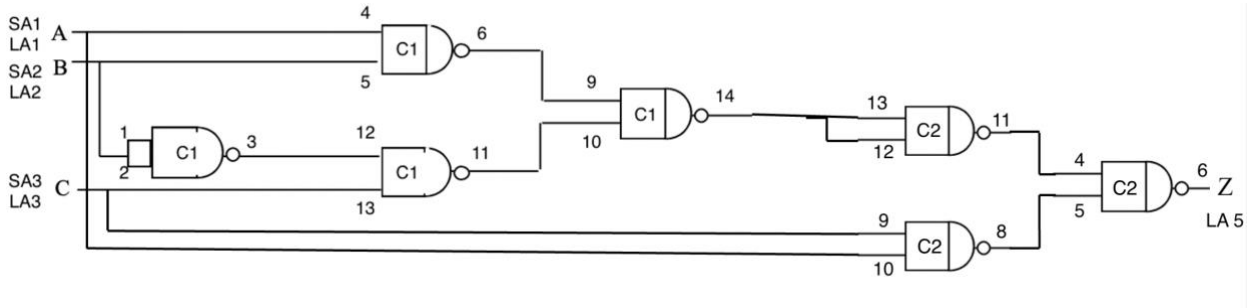


Figure 1.2: Circuit edited from general guide.

Now, we represent this circuit via the circuit layout above (Fig 1.2), where we execute Output = (Z.AC)'. Below (Fig 1.3) is the circuit executed on the Fritzing system.

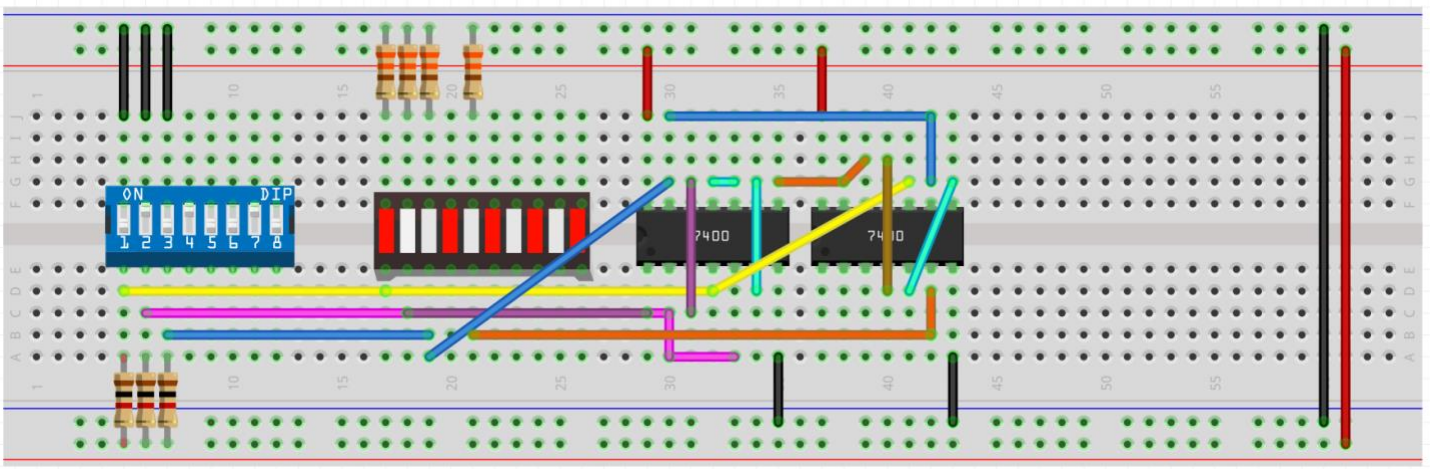


Figure 1.3: Circuit drawn via Fritzing system without the static hazard.

**Images taken from the General Guide (GG.27)*

Geitanksha Tandon
ECE 385
Spring 2022

Documentation:

Pre-Lab Questions:

a. *Not all groups may observe static hazards (why?) If you do not observe a static hazard, chain an odd number of inverters together in place of the single inverter from Figure 16 or add a small capacitor to the output of the inverter until you observe a glitch. Why does the hazard appear when you do this?*

I believe that not every group may observe static hazards. This is because the gate delay in relatively simpler, smaller circuits is almost negligible. In addition, if the circuit has balanced propagation delay (that is, the gate delay happens simultaneously with all leveled gates) then the static hazard may not be visible, it gets masked. In the lab, the smaller level of gate delay meant that we were unable to view the glitch appropriately, hence we added a 1uF capacitor in increase the amount of gate delay observed on the inverter for line B, leading to our view of the glitch on the oscilloscope.

Lab Questions:

a. *Complete a truth table of the output. Does it respond like the circuit of part A?*

The static hazard-free circuit we created in Part B had the same truth table as shown in part A (Fig 1.3), and it responds almost exactly like the circuit of Part A, except without the glitch (but it still retained noise).

b. *Next, disconnect the switch from input B and hook the pulse generator to input B. With inputs A and C high, observe input B and the circuit output on the oscilloscope. Describe and save the output and explain any differences between it and the results obtained in part 2 (of the Lab Experiment). Consider the following question and explain: for the circuit of part A of the pre-lab, at which edge (rising/falling) of the input B are we more likely to observe a glitch at the output?*

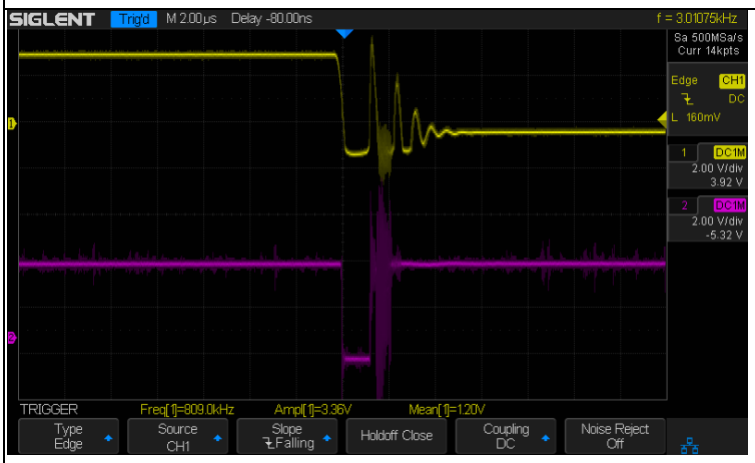
We noticed that when A, C were on high, we know that Z should display 1 because it was selecting between two high inputs. However, as the pulse generator fluctuated from 0 to 1 and vice versa, that the falling edge of input b faced a larger number of glitches (periods when it would show a false 0, static hazard 1). This was because the B pulse had an inverter which caused an additional gate delay. In our diagram of the oscilloscope Fig 1.5), this short period of unexpected 0 is visible. However, in the case B goes from 1 to 0, there is always the presence of a 0 being input into the final NAND gate, and hence this means that the output Z will always be 1, regardless of the gate delay when B has a rising edge.

Geitanksha Tandon
ECE 385
Spring 2022

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

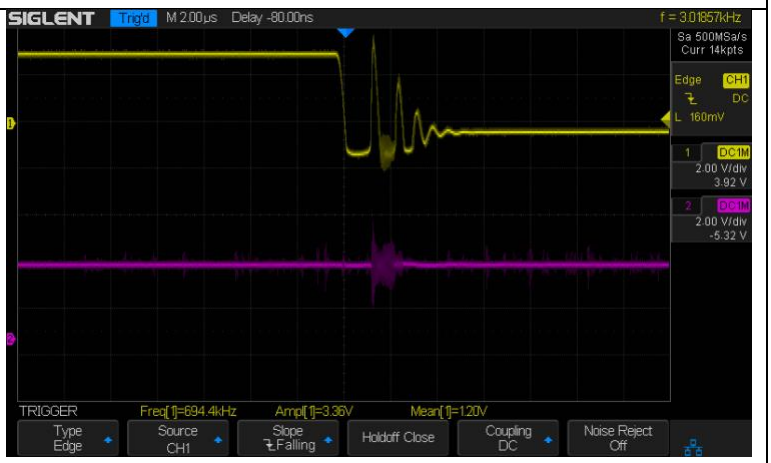
Figure 1.4: Truth table of Part B.

Figure 1.5: Oscilloscope reading for falling edge of B in Part A.



In this, we notice that there is a glitch due to the static 1 hazard, as demonstrated in the lab while executing the Part A circuit.

Figure 1.6: Oscilloscope reading for falling edge of B in Part B.



*In this, we notice that there is **NO** glitch due to the static 1 hazard due to addition of an extra term, displayed in the lab while executing the Part B circuit.*

**Images taken from the General Guide (GG.27)*

Geitanksha Tandon
ECE 385
Spring 2022

Post-Lab Questions:

a. *Given that the guaranteed minimum propagation delay of a 7400 is 0ns and that its guaranteed maximum delay time is 20ns, complete the timing diagram below for the circuit of part A.*
The timing diagram is shown below: (Fig 1.7)

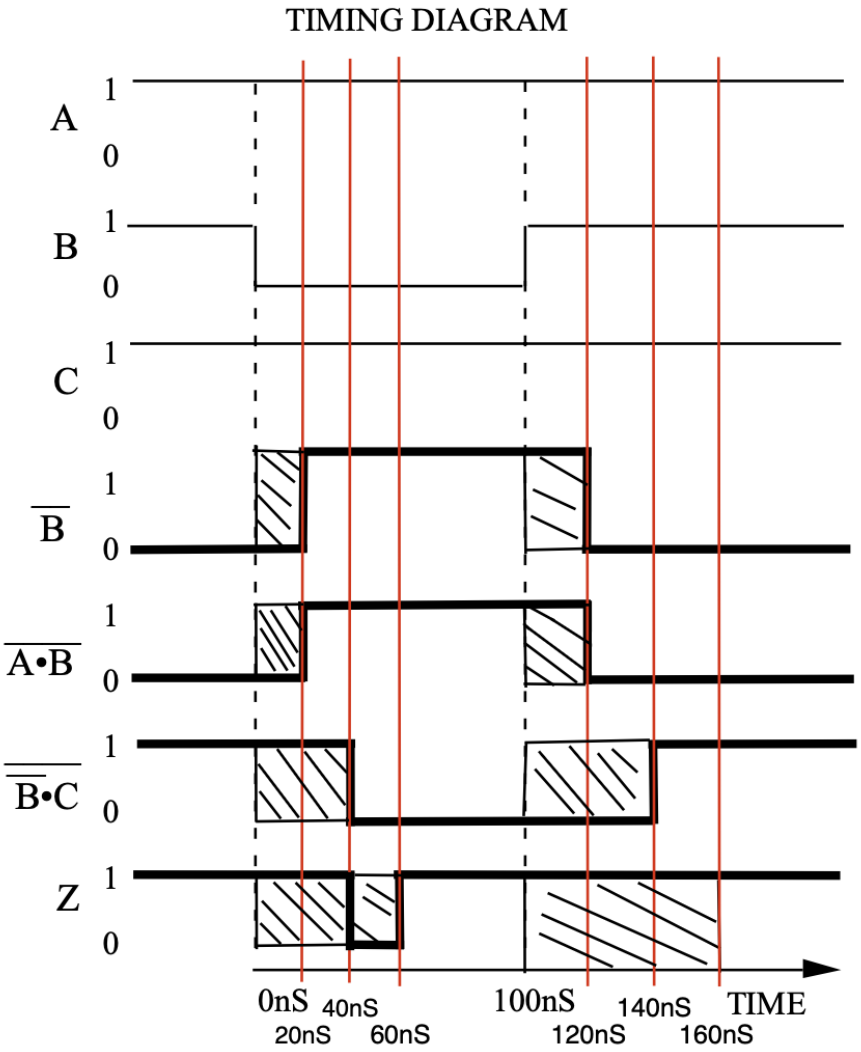


Fig 1.7: Timing Diagram for Part A, executed in Lab

b. *How long does it take the output Z to stabilize on the falling edge of B (in ns)? How long does it take on the rising edge (in ns)? Are there any potential glitches in the output, Z? If so, explain what makes these glitches occur.*
It takes Z **60nS** to stabilize on the **falling edge of B**. Between 40 and 60 nS, we can expect a **static-1 hazard glitch** to occur (Z=0 even though the correct output is Z=1). This is due to the additional gate delay for the B input selector line. This makes the Z value temporarily 0 when it should be 1. On the **rising edge of B**, it takes **60 additional nS** (stabilizes at 160nS) to stabilize. The other potential glitches occur between the times when the regions are shaded, when there could be an output, we do not expect due to the maximum gate delay one can expect from the 7400 chips.

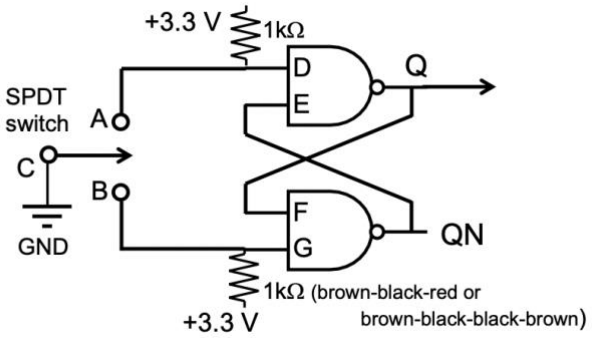
**Images taken from the General Guide (GG.27)*

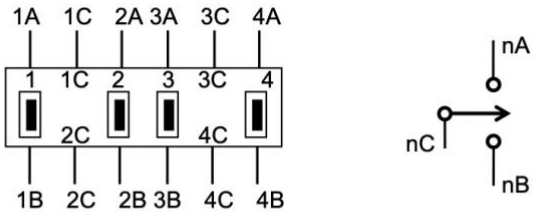
c. Explain how and why the debouncer circuit given in General Guide Figure 17 (GG.32) works. Specifically, what makes it behave like a switch and how the ill effect of mechanical contact bounces is eliminated?

The normal DIP switches we use may have **contact bounce**, which is the phenomenon where the force with which a switch is closed may cause a few extra consecutive open-close-open switch connections, which means that the contact takes a momentary separation before joining again. This means that the contacts will only stabilize after a few milliseconds of contact. However, this proves to be an issue because if the switch is used for the clock of a counter circuit, the counter will have advanced a few times each time the switch is flipped. This is fixed using the **de-bouncer** circuit as described below.

How this works: In this circuit shown (Figure 17), we use a Single-Pole Double-Throw switch, an SR Flip-Flop and two NAND gates to simulate a de-bounced switch. The inputs D, G (connected via A) are connected to pull up resistors, and the 2 NAND gates form an SR latch. We will assume the top NAND gate is N1 and the bottom NAND gate is N2. If we begin in the state where C touches B, then G is connected to ground (0V) and the N2 gives an output of 1, making QN hold the value of 1 (and Q hold the value of 0), but the value itself is independent of Q. Now, whilst toggling the switch to position A, there is a chance that the switch may get suspended in the middle, which in a normal DIP switch could cause fluctuations due to contact bounce. In our debounced switch, however, this suspension makes D, G go to 1 (as they are connected to pull up resistors). Thus, the value of QN stays 1 and Q stays 0. When the switch finally touches A, it makes D go to 0, and then makes Q have an output of 1, regardless of what the value of QN is.

*





QUAD SPDT Switch

Figure 17. A Debouncing Circuit for an SPDT Switch

This nature of specific non-fluctuating values of Q during the effect of a contact bounce is what makes this a switch – Q has an output of 0 when it is connected to B, when the switch is moved from this position it assumes a suspended state, which makes Q stay at a value of 0 due to the nature of the SR latch. Finally, when the switch finally lands on A, only then does it change the value of Q to 1.

**Images taken from the General Guide (GG.27)*

Geitanksha Tandon
ECE 385
Spring 2022

General Guide Questions:

a. *What is the advantage of a larger noise immunity? Why is the last inverter observed rather than simply the first? Given a graph of output voltage (VOUT) vs. input voltage (VIN) for an inverter, how would you calculate the noise immunity for the inverter? See the following figure.*

The advantage of a larger noise immunity is that the chip itself is more likely to tolerate fluctuating values of the external noise (stray electric / magnetic fields) without disrupting the logical level. It is less likely to chance the logic circuit to go below the higher voltage or go above low voltage when this noise is perceived and thus is less likely to cause an error or glitch in our circuit.

The last inverter is observed instead of the first because in the first inverter, the voltage drop across it is almost negligible – hence, the interrupting noise is also insignificant enough that the natural noise immunity of the chip can handle the noise with no disruptions. With each inverter that is used, we notice that the voltage drop across each gate decreases, and so eventually the chip itself has lower capacity of noise immunity. Hence, it is more likely to have fluctuating values for the voltage output displayed which depends on the external noise.

The way that we calculate the noise immunity is by getting the difference between the least input value and the largest input value at which a logical value is correctly interpreted by the chip.

When we look at the diagram, the inverter correctly recognizes the input voltage as **low** when the values range between 0.35 and 1.15 volts (and in the graph provides an output of high).

Hence, the noise immunity for logic input "0" at the input is:

$X_0 = 1.15\ V - 0.35\ V$

$X_0 = 0.8\ V$

When we look at the diagram, the inverter correctly recognizes the input voltage as **high** when the values range between 1.35 and 3.5 volts (and in the graph provides an output of low).

Hence, the noise immunity for logic input "1" at the input is:

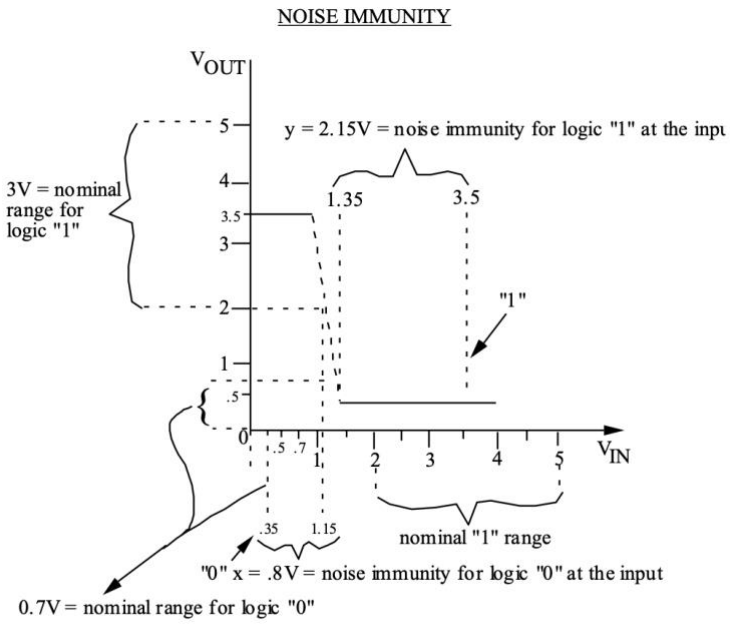
$Y_1 = 3.5\ V - 1.35\ V$

$Y_1 = 2.15\ V$

As noted below the diagram, the actual noise immunity for the gate is the lowest value between (X_0, Y_0):

Noise immunity = Min(0.8, 2.15)V

Hence, noise immunity = 0.8 V for the inverter.



THE OVERALL NOISE IMMUNITY OF THE GATE IS THE SMALLEST OF THE RANGES X AND Y.

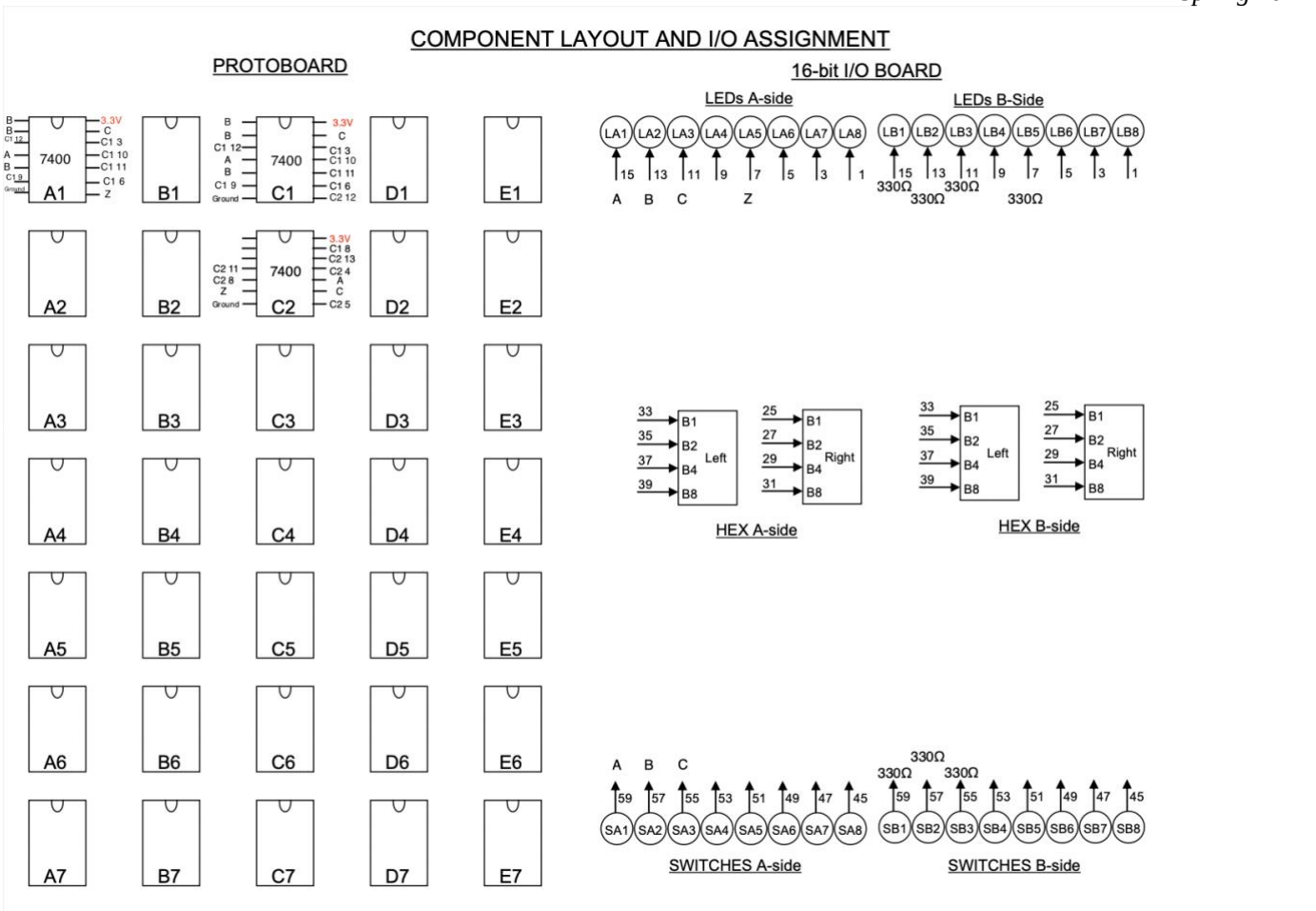
**Images taken from the General Guide (GG.27)*

b. *Why is a capacitor necessary close to each chip? Hint: what happens inside the CMOS circuit when it switches?*

We use CMOS circuits as they are used in Modern ICs, which combine paths for 1, 0. This means that they have low static power when the value is consistent, and there is no switching going on. However, while switching, there is a momentary short circuit between Vdd (positive supply) and ground. This means that it shows up as a lot of noise propagating between the logic chips when there is a switch going on. The addition of the decoupling capacitors thereby reduces the amount of "switching noise" propagating through the circuit. In cases where you have shift registers, and a lot of combinational CMOS logic, there tends to be more amount of noise which can cause the circuit to malfunction, and so decoupling capacitors are used to prevent this.

c. *Irrespective of which polarity you choose for your LEDs, it is important that each LED has its own resistor. If we have two or more LEDs to monitor several signals, why is it bad practice to share resistors?*

You should not share resistors for LEDs because they are essentially diodes, which cannot be safely connected in parallel. In the situation where we have multiple LEDs in parallel, the situation that arises is that we first assume that all of them have a constant (and equal) voltage drop across them. This is incorrect as these LEDs may not have this constant voltage drop due to real-world limitations. Diodes are based on a forward-biased graph, hence any voltage above a certain value is acceptable to light the voltage – this does not mean that each LED has the same voltage drop across it. If one LED has a smaller value for this forward biasing, then it will limit the voltage across the other LEDs and hence those will not have enough current running through, thereby not lighting up. In addition, the power dissipation through the resistor can become increasingly high in the case that all LEDs share resistors, since P = I^2/R. Hence, due to a single R, power dissipation becomes high and it can cause a large current across the LED.



Conclusions

I learned about glitches and delays – specifically the static 1 hazard. In particular, I understood the real-life consequences of aspects of circuits we usually ignore, and the thinking that is required to fix them. I discovered the way that this specific hazard – static 1 – requires the integration of additional minterm(s) to cover up for the plausible delays that may occur in our circuits, and how the overall way to overcome this particular issue is by using a clock, to ensure that we can account for worst-case delays without making our circuits less optimized with space (due to the additional minterms). In the lab that was demonstrated, this method of adding minterms did work, however in the future one could try to use the clock method in which the gate delays are overcome by the use of systemic clock cycles that account for the worst-case propagation delays of the chips being used.

**Images taken from the General Guide (GG.27)*