I think this document will benefit from a little bit of narrative. Lecture slides alone are a bit insufficient for this IMO and I didn't want to record myself speaking for like 3 hours...

This week is a little difficult for me. The material is all easy and straightforward and I've been doing it personally and professionally for a long time but your backgrounds vary widely so toeing the line between covering basic concepts in sufficient detail while not wasting the time of the more experienced is a difficult balance to strike. Additionally we're jamming a ton of concepts into as little time as possible. The opamp that Neil mentioned for ~1 minute is itself a full hour lecture in 6.002. I think the best balance is something of an encyclopedic nature. It'll be designed such that you can reference sections individually or read through it from start to finish if this is your first time experiencing these concepts. It'll probably be overly long and contain not-great images so sorry for that. I should've had this idea a week ago.

2 quick disclaimers:
I apologize in advance for my formatting. I probably should've taken a class that forced you to make pretty slides and papers that are easy to read. Instead it's information dense and long-winded…
To borrow a phrase from a colleague **I reserve the right to lie to you.** There will be some over simplifications that will hold true the vast majority of the time but fail on the fringe cases and things that are true but for the wrong reasons. Someone said being a good engineer is about knowing what to ignore. If we can all ignore the fact that the entirety of electronics is actually based on electrons that have a positive charge and therefore all our flows are in the wrong direction then I hope you won't mind the little bits that I lie about to make it easier to understand.

Now let's get started….

EDA tools:

Why do I need these?

Designing circuits without an EDA tool is kinda like flying a plane without electronic assists. Sure it totally can be done however there are countless safety measures built in that will help you through the process. It will stop you from merging traces that are different signals, yell at you if that does happen, ping areas that are too small for you to mill, remind you things that you meant to connect but then didn't etc etc. This shouldn't be seen as the training wheels on a bike but more akin to the anti-lock brakes on a car. Almost mandatory and a huge risk if they aren't there.

Which one is right for me?

You know my personal opinion. You'll get the same output from any of these tools so my advice usually boils down to "which software works like my brain works". Standalone eagle is a mess of buttons everywhere but nice clear pictures and names for most things and a solid back end which roughly matches me. Fusion eagle has a polished UI that hides a lot of the stronger functionality but it looks quite modern and pretty so maybe that is for you. KiCad seems to have a high activation energy or a rough learning curve. Eventually I'll likely switch to this out of spite given the forced Eagle annexation but I'll need more time until that happens.

How do I use them?

Totally beyond the scope of this document. Refer to Jake/Zach's recitations, online guides, my/Zach's recorded recitations last year, or come by EDS and we'll talk through it in person.

Should I auto-route?

No! This sucks in basically every software and operates under a fundamentally flawed set of assumptions. You can look at a board and see that a single 0 ohm jumper resistor will solve a complex problem. Auto routers cannot add components (and many cannot even move components) so they give you terrible spaghetti results unless you know exactly how to constrain your problems. If you auto route a whole board you're really doing yourself a disservice for the later weeks where it will fail entirely. I'll be able to tell and you'll make me awkwardly scold you.

Practical Electronics:
Blindly copying Neil's schematics is sufficient for a while but we'll need to start building an understanding as to what parts are on the board, what job they're doing, and how to problem solve when creating our own boards. My goal is for you to be able to see something that is wrong and be able to figure out a logical approach to make it better (within reason obviously, there isn't enough time to become an expert everywhere).

Power!
Normally we'll have some AC or DC voltage (essentially sinusoidal or steady state. Really that's a whole subject in and of itself) and we need to turn it into something else. The most common situation is 5V from your USB into 3.3V for a SAMD processor. The easiest and lowest part count solution is to throw it through a linear regulator. These are both wonderful and terrible. The former because they are so cheap and easy and the latter because they are so inefficient. Operating efficiency is roughly input voltage/output voltage and they can only go down. This means if you use a 9V battery to get your project wireless then you're running close to 30% efficient and the other 70% is just turning into waste heat. Obviously terrible but the typical solution in most places as long as your power draw is low. We won't really touch on other options in this course but they do exist and they're relatively simple. Look for buck, boost, buck/boost converts as a good starting point.
Now back to my point… a 3.3V LDO (low drop-out linear regulator) isn't quite enough to turn 5V into 3.3V. It'll kinda work but we want some capacitance to help stabilize both the function of the chip and also reduce the noise on the output voltage.

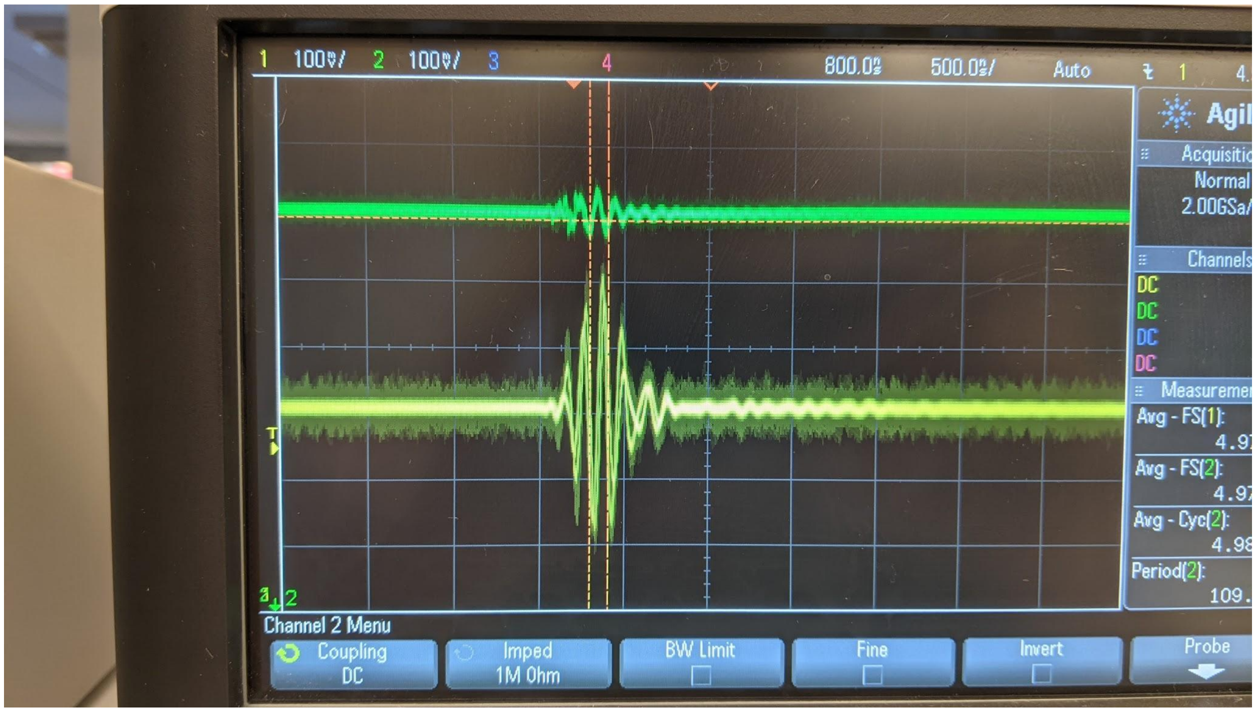

Figure 1
Here I'm using an oscilloscope to show 2 waveforms. They have the same output voltage, I just moved one up a bit so they aren't overlapping. The lower yellow trace is the output from a regulator under no load but also without any current draw. Even under this best case scenario

we can see that the output voltage is swinging by +/- 200mV which is quite a lot. (You can see the scale of the boxes on the top left corner of the scope.) Under load this would increase even more and can easily drop far enough that it would cause your microcontroller to "brown-out" and reboot in an unpredictable manner. The green trace is the same regulator with no current draw but with a 0.1uF capacitor on the output. The exact value that you choose here isn't super important in this case. In the real world we'll do 0.1uF, 1.0uF, 10.0uF caps of a few different sizes and chemistries to get the best frequency response. You however can pretty blindly throw in just 1 of any of those sizes and get perfectly usable results.

What are capacitors again?
We can think of them as tiny batteries that respond to changes in voltage incredibly quickly. One of their many purposes is to take a messy input voltage and kill some of the noise until it approaches a nice clean flat line. For the mathy folks capacitors help you minimize dV/dT and they are used everywhere. Spotting one is a circuit is usually easy but sometimes a bit challenging because they can take on so many different shapes and sizes. I don't think it's super useful for me to break down all the different types so I'll skip that for now.

Resistors

These are the most boring but totally wonderful little things. They essentially are just little chunks of carbon and ceramic that are great at reducing the flow of electrons in a nice repeatable manner. Their usage however varies widely and can help you in so many different scenarios. Need a reference voltage cheaply with no real tolerance on it? Resistors got your back. Need to clamp the current to particular devices? Resistors are there for you. Need to drop charge times or set some RC timings? Man it would be so convenient if the R in and RC oscillating circuit stood for resistor right? Did you run out of layers and need to get a trace over something else? Well may I present the 0 ohm resistor? It's so bad at it's job that it doesn't drop the current at all*** but boy will it help your routing troubles. Don't even get me started on how awesome Wheatstone bridges are for measurement amplification! I won't waste your time covering all the fun little uses for these bits so here are the most common uses in how to make.

I totally forgot but let's start with how to **read resistor codes!**
Through hole resistors have a funky color code that can be hard to read, changes based on the number of bands, and is entirely useless to the colorblind. It does exist and it's roughly like scientific notation where the colors go like black, brown, then follow the rainbow spectrum in increasing value and have a final color that indicates your tolerance. We however won't be using these guys so I won't bother explaining it any further. When in doubt google it or use a multimeter!
Surface mount resistors (at least in the sizes we are using) should all have numbers on one side and follow a modified scientific notation. First if you see an R that will stand for a decimal place. Then things can get a little funky because different brands include a different number of sig figs. The general rule is that the last one is the power of ten and the rest of the

numbers are multiplied by that power. It's a little ambiguous then because a typical 10K resistor could say 103 (10 times 10^3) or 1002 and both are correct. I'd assume that the more digits the better the tolerance but I'm not 100% positive there.

Resistors follow something called the E12 series which means they are only available at 12 values per decade (meaning the power of 10, not that it changes every 10 years). So if you want a 126.47 ohm resistor your shop will not have it and you'll have to special order it (though you should never ever need that). Additionally if you do happen to need a value your shop doesn't have you can 1 add things together in series/parallel or if it is an E12 value just find me (Anthony) and I'll definitely have it. I buy these in a couple different sizes (that link is for resistors that are too small for you) which is a huge lifesaver if you want to do this kind of stuff yourself regularly.

I'm conflicted about whether to mention this. If you find it confusing, ignore the next few sentences and chances are you'll be fine. We do have to remember that our resistors are small meaning they don't have a ton of surface area and are not great at dispersing excess heat. Power lost in a resistor (as heat) is (I^2)R with the units of Watts. Ours are typically 0.25 watt resistors so anything above that and you risk it burning up.

0 ohm jumpers

Neil covered this well in lecture on 10/06. We use 1206 resistors (120 mil/thou/ 0.12" long by 0.06" wide). These are big bulky guys that don't get a lot of professional use but they're great for hobbyist level and prototype level boards. They don't stop current at all* but we use them as an overpass so that one signal can hop over another one without shorting or making us mill another layer. This is one of those great tools to have in your pocket when you route yourself into a corner.

Pullup resistors

So the idea behind these guys is that we're adding some reference to a voltage. Why do we need this? Let's take the scenario of a button input. We'll generally hook one end of it up to Ground and the other end right to an input pin of your microcontroller. Now when we press the button that pin gets shorted to ground and we read it as low. However once we let go of that button what value would we read? I'd argue that it's undefined. In reality the other end of that button is gonna act like an antenna and we'll pick up noise from the surrounding circuit, 60Hz from the wall, and whatever voltage from random people coming and poking the circuit. It sucks and is totally unpredictable. If this is just triggering an LED maybe this doesn't matter but if you're controlling motors or doing things in your final project then the unintended operation is much more critical. Now let's look at this circuit that should help solve that problem.

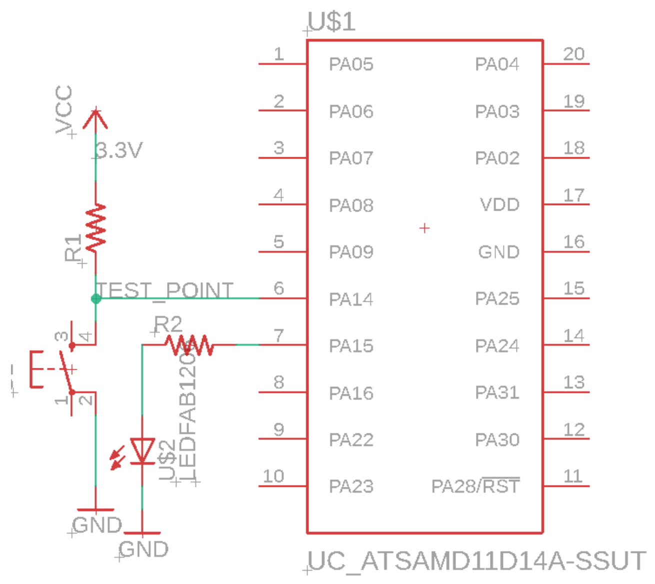

Figure 2

We'll ignore R2 and the LED for now and just focus on R1 and the switch. Additionally don't copy this guy directly, it's clearly missing most of the connections that make that microcontroller work. Just pay attention to the stuff on the left.
We'll analyze this simple circuit in 2 situations:

1) button is **pressed**
Pressing the button is going to create a ~0 ohm short between the place I labeled TEST_POINT and ground. Since the voltage drop is equal to the current times the resistance and the resistance is 0 then the voltage drop must be 0. Ground is our designation for what we are calling 0V (again it's own whole topic but the important part is that we think this is 0 and anything connected to it must agree on the same 0) Anyway that will make the voltage at TEST_POINT 0V and we don't even have to consider what R1 is or our overall current as long as we assume it is substantially larger than 0 ohms. Really we'll pick an R between say 5K and 100K just to keep our waste power down. We don't want to be dumping an amp through that button just to read it when we could do the same job with fractions of a milliamp and save on the energy lost as heat.

2) button is **not pressed**
So now we aren't pressing the button, what does that mean? An unpressed button won't allow any current to flow through it so I becomes 0. Again if delta V = current times resistance and now I is 0 then deltaV must also become 0 right? Now the only difference is which end we're looking at. Since R1 is connected to Vcc then it becomes 3.3V-0=3.3V. Now we have a nice robust regime where a pressed button is 0V and an unpressed button is 3.3V.

This gives us an easily predictable input as we're either high when unpressed or low when pressed. First in the diagram above I used a physical resistor that I would put on my board.

Since the pullup resistor is so ubiquitous many microcontrollers actually have pullup resistors built into them so we could ignore R1 and the connection to 3.3V completely. There is the occasional funkiness where some pins might not have an internal pullup resistor so I usually just use the external one anyway but the datasheet will tell you if it exists or not.
Now let's test the button and look at it on a scope.

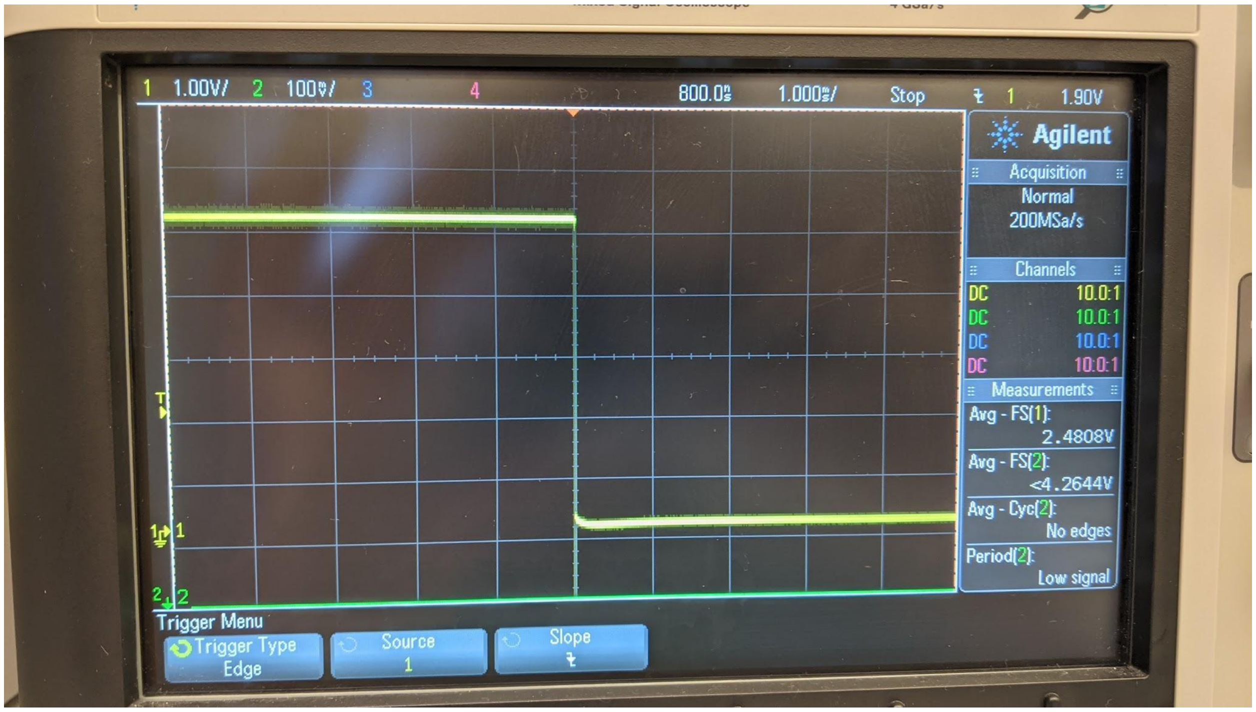

Figure 3.
Alright, that looks pretty good right? It has a clean transition from High to Low but let's dig a little deeper. Each of those boxes covers ~1 millisecond in the X axis which isn't much for humans but at 20MHz each box becomes **~20 Thousand clock cycles** so we'll have to enhance!

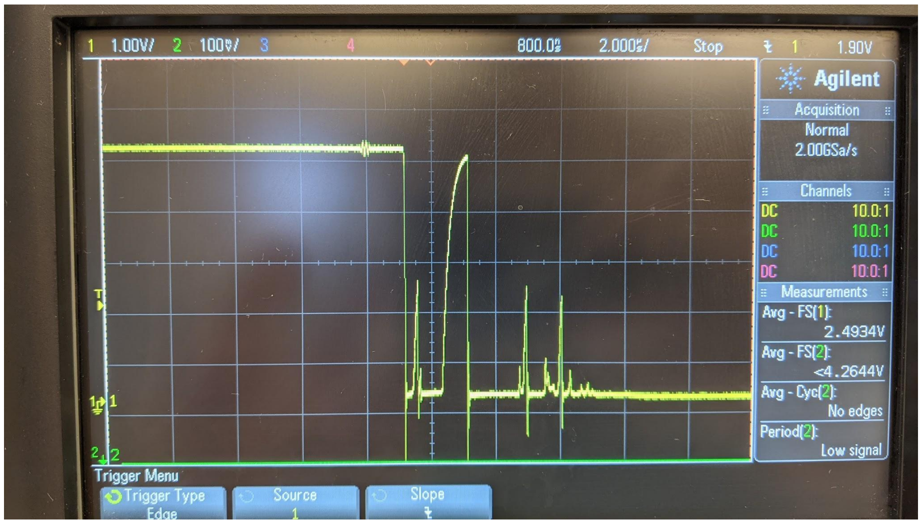

Figure 4.
Whoa what's that garbage? This is called button bouncing. Our buttons are not perfect (none are) so as you press it down it will actually make and break contact a couple times. We could

spend loads more money and get better performance or we can take care of the problem in software. On this timescale each box is 2 microseconds so ~40 clock cycles which makes it pretty likely that we will be able to read those peaks if our code is running quickly. The three smaller ones *might* not be quite high enough to get detected as a push/release but the big one certainly is. We'll have to do something to smooth this guy out otherwise we'll be erroneously triggering things in our code. I won't go through all the approaches but I'll give you a couple. The bad ideas that are technically functional are adding delays in your code and reading it more slowly or adding a capacitor around the switch so it switches more slowly. Both of these are the lazy way out that will sorta work for some situations. More robust solutions tend to take sliding averages of your switch value or detect the edge then read the switch again after a settling time has passed.

Current limiting resistors

This week we'll be running LEDs which are kinda funky devices that have almost no internal resistance of their own. If you hook one directly up to power and ground you'll see a bright light, hear a crack, and need a new LED very quickly. In order to make it live and be useful we throw a resistor in series with it. I tend not to do the math and just throw a ~1K resistor because that's pretty dim and doesn't bother my eyes. However I will do the math so you can see it. Let's assume we are driving with 5V and have an LED with a forward voltage drop of ~1.3V and a maximum current of 20mA. These are ballpark numbers that will be device and color specific. Now we want to find a resistor whose job it is to eat 5-1.3V and set the current to 20mA. deltaV=IR so it follows that 5-1.3=0.02*R so 3.7=0.02*R and therefore R=185 Ohms. Make sure you use the right units when calculating this or your numbers will be wonky. Again we're limited to the E12 series and what the lab has in stock so find the closest value that is larger than this number. We used the max current so we wouldn't want to go under 185 or we'd risk blowing our LED prematurely.

Back to resistors for 1 more thing...

Resistor dividers

So these at the most basic level are ways to take a voltage and drop it to another value however they have some drawbacks. If I just need a reference that won't be used to pull any current then this can be ok. Resistor tolerance varies from +/-10% to tight ppm specs so your voltage reference is only as good as the parts you use to make it. Cheap trim pots will sometimes get added in series to manually adjust out any error but usually tight tolerance stuff won't have resistor dividers involved at all. Let's say I just want to check if a sensor is above or below a specific value. Then I could use this divider to set that specific value. They are falling out of favor with the rise of digital electronics but you can still find them doing jobs like that or doing the cheapest conversion of a slow 5V signal to a slow 3.3V signal.

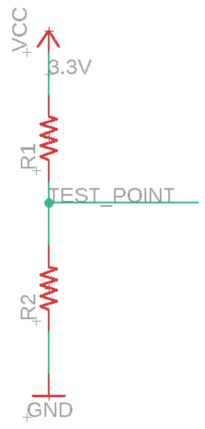

Figure 5.

Here the voltage at TEST_POINT is easy to compute. If we take deltaV=IR and refactor it a bit we can get Vtp=(R2/R1+R2)*Vcc. Intuitively this should follow that the larger R2 is with respect to R1 then the more of the voltage drop occurs across R2. This is a little boring but it does have its uses in the analog world.

Much more interesting (to me)is the wheatstone bridge in which 4 resistors are arranged in a diamond pattern. 3 of them will be relatively tight tolerance but boring pieces and the last one will be a sensor of some kind. Maybe it is a strain gauge (resistor whose value changes as you apply force to it) or maybe a thermistor (same thing with temperature). This is one of those things that is kinda like capacitive sensing. It has tons of awesome uses limited mostly by your creativity.

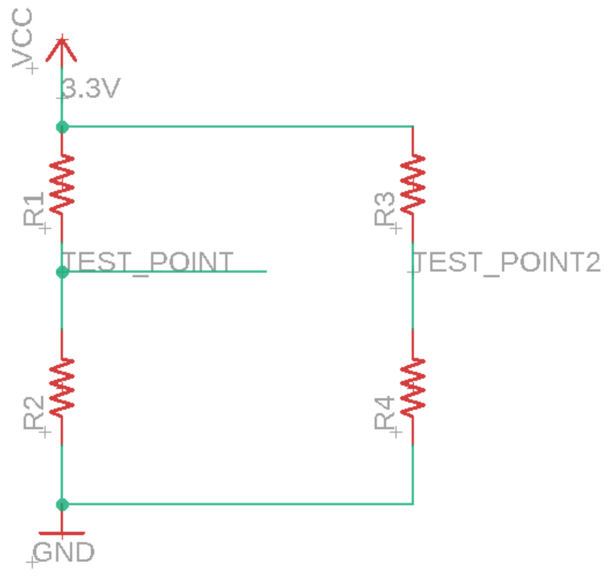

Figure 6.

Now these are normally arranged so they look like a diamond but apparently in eagle parts need to have 90 degree rotations so sorry about that.
Anyway the magic here is that we don't really care about what the voltage at the 2 test points are, we just care about what the difference between them is. It will generally be quite small but we'll amplify it and then use it to learn lots of interesting things about our system. Most of you won't need this so we'll talk about it more in depth if you do (I'm running out of time before office hours so won't waste time on unnecessary things).

MOSFETS:

Neil's drawing had drain and source switched on the N-fet. It's pedantic so I didn't say anything and it's unlikely you'd use it without finding a circuit anyway.

Why do we need a mosfet? Well it would be nice if our microcontroller could source and sink infinite current but it actually maxes out in the ballpark of 20 or 50 milliamps which is enough to light an LED and power some sensors but no wheres near enough for big things. If we want to say send an amp to a motor that would take 20 pins!* (that's a lie this wouldn't work because of global maximums but you get the drift). Instead we want to control a little device that will in turn control a much bigger device.

Generally we'll ignore most of the middle ground of mosfets and treat them as all the way on or all the way off. This is known as "bit banging". You can think of these as electronic switches. They come in 2 flavors (N and P channel) which changes both where they go in the circuit (below or above your load) and how you turn them on (high or low) Annoyingly in an N channel mosfet current flows from Drain to Source and in a P channel mosfet it flows from Source to Drain. If you look at the full schematic symbol for a mosfet you'll see the body diode which is a good indicator of current direction. If you put it in backwards then current just flows through the body diode and you can't turn it off. Many a project has failed for this reason. **With proper heat sinking** these little guys can handle quite a bit of current. If you don't have a nice heatsink they'll get way too hot and stop working well so you can just step up to a bigger package that can handle it more easily.

Unfortunately mosfets are also a phantom killer of projects in a way that technically isn't their fault. One of the common use cases is for driving motors and motors act like large inductors (they want to keep current constant). If we turn them off quickly then we are trying to push the current to 0 faster than is physically possible and as a result the motor will cause a huge voltage spike (tens to hundreds of volts) which can fry the fet and maybe fry your microcontroller. The best defense against this is a flyback diode (maybe also called a snubber diode?) and you can also define soft stops in your code (slowly dropping the gate voltage rather than all at once). If your stuff randomly breaks ESD and inductive kicks are usually the first culprits.

Don't be afraid due to that warning. MOSFETs are fun and incredibly useful little devices.
Some helpful photos are still to come but we likely won't need these this week anyway.

PWM:(not actually a part but we'll talk about it anyway)

This stands for Pulse Width Modulation and is a really strong technique for controlling things that we don't want to be all the way on/off. Think about how we vary the speed of a motor or the brightness of a light etc. They're also used in relatively cheap sound generations and I wouldn't recommend it for that.

The idea here is we have a fixed wave (say 10KHz) and we vary the amount of time that it spends on vs off (known as the duty cycle). Since this is happening so much quicker than our device can respond we'll see a natural averaging of the speed so a motor spinning at 50% duty cycle will be roughly 50% of the max speed. Technically this isn't quite true for LEDs but that's really due to the funky way in which your eyes perceive light.

PWM can be implemented both in hardware (we have modules designed in the microcontrollers to do this really well) and in software. In general my approach is to use a hardware module to do anything if it exists. This means we don't need to waste valuable compute cycles checking the time and setting pin states when we could just pass over the setup parameters and let it go on its own until we stop it. The software implementation however is super useful if we don't have enough pins, made a routing mistake and don't have access to the PWM pin, or things of that nature.

Test equipment:

This is one of the few areas where I get to flex on the CBA a little bit. Their machine collection puts ours to shame but the sheer amount of scopes and other test equipment that we have totally dwarfs them. The machines are way cooler than scopes though :(