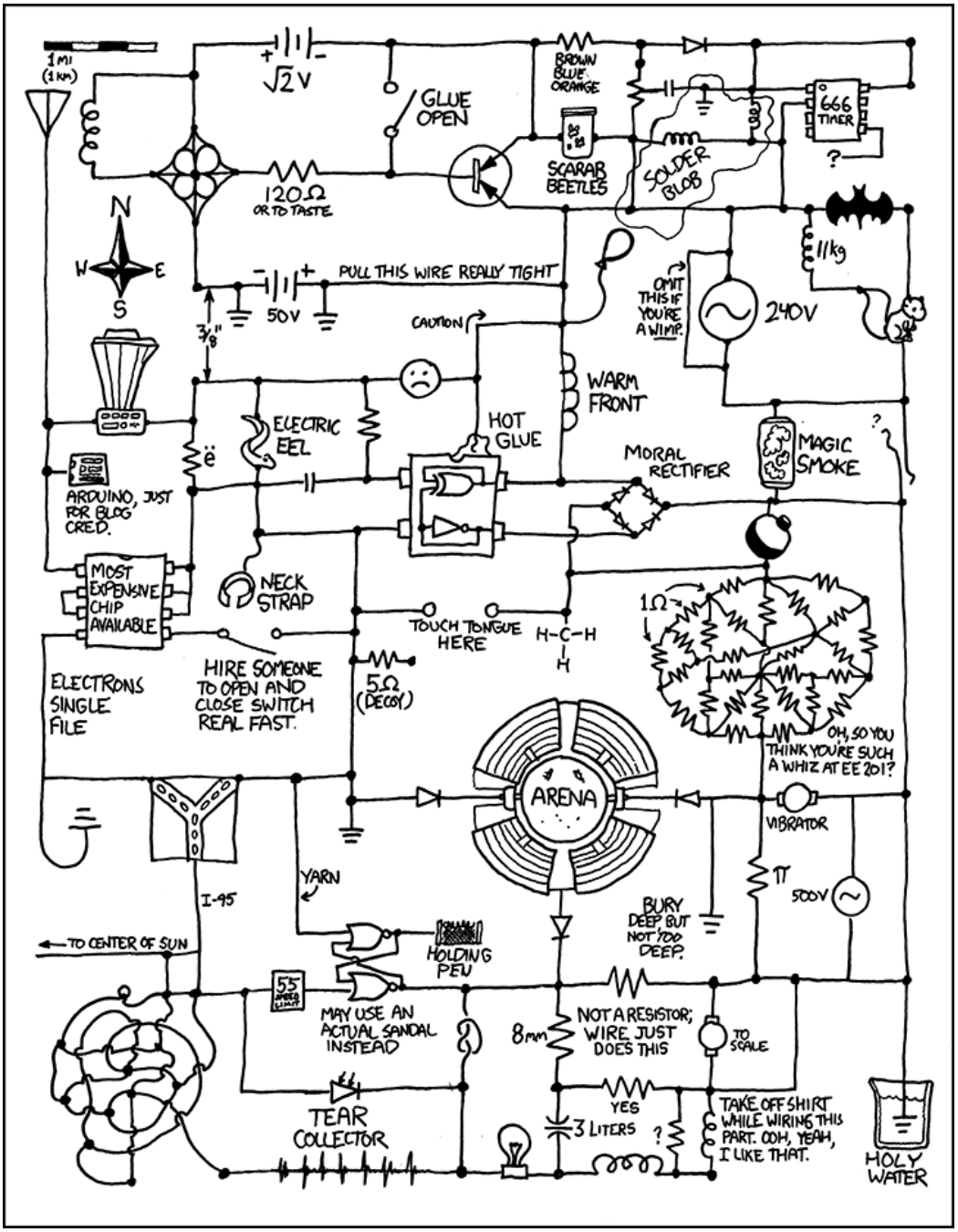
Designing Embedded Hardware

A Guide for Students & New Grads

Alex Ching

March 2025



https://xkcd.com/730/

Contents

1 Introduction

This text is designed to be an addendum to a traditional curriculum for students who plan on working on embedded systems in industry, research, or just for fun. Specifically we will focus on the hardware and electrical engineering found in embedded systems so this content will be most useful to someone working on building hardware or as good background knowledge for someone doing firmware. The content will go over some basics and also give an idea of the workflow in designing a device in industry and more formal research settings. This is not a textbook and shouldn't be read page by page. This is meant to help someone starting a capstone in their final year of undergrad or a new grad just starting their careers at an entry level job. It's a collection of topics that will hopefully lead you to searching for much better explanations and resources yourself.

This text is focused on practical applications and less about the theory. As someone who was a student that transitioned to professional life and then back to academia, I've noticed that the core curriculum in these programs are good at teaching fundamental concepts on paper, but only begin to focus on more complex real world applications at the end of a program. By then, students are already trying to find jobs, but may not have the exposure to how a system might be designed from scratch. This text aims to augment an existing curriculum rather than replace it. I try to explain some of the knowledge that's used to design such systems and some tips and tricks that are commonly found in industry and research settings. Hopefully this helps someone out there to be a strong candidate for an industry job and shortcut some of the incidents I had to go through to pick up this knowledge. A lot of the text is common sense, but I find new engineers tend to want reassurance.

This is not the best or most comprehensive document about these topics, but I hope that having it in one place helps you get a good idea of how modern systems are designed. I list a lot of resources I've found useful in each section that you should visit in order to go in depth into a topic when actually implementing something. I also suck at drawing so I've downloaded a lot of images from the web with source links. I take a lot of comics from xkcd and PhD Comics, please check them out!

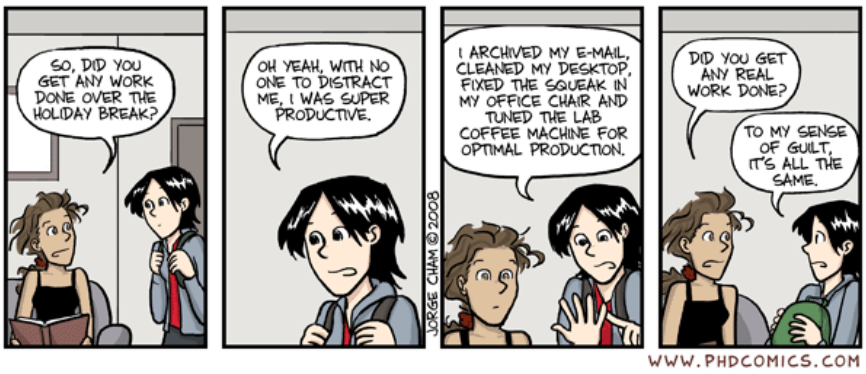


Figure 1: Procrastination: The real reason I'm writing this text... XD

I'm of the opinion that (almost) no one can really call themselves an "expert" in embedded systems because this field is always evolving and the core expertise needed on a project inherently changes between projects. A lot of people who broadly claim to be experts are either trying to sell you something or are full of themselves. I certainly don't see myself as an expert in this field yet. My motivation for preparing this text is to give students more of the practical knowledge they will need to be strong entry level candidates in industry. If you are a software engineer, you can slam leet code for months to prepare for an interview, but when you are after a hardware job there isn't really an analogous preparation solution. When you've done the job long enough you usually pick up techniques and know how that isn't necessarily covered in school.

If anything I've said in here seems wrong, please go and double check with someone knowledgeable or check with online sources that you trust. I referred a lot to Wikipedia and the links I included to refresh my memory while writing this document. Always a chance I messed up somewhere.

1.1 About Me

I did my undergrad in mechanical engineering and electrical engineering and have a masters in computer science from the University of Washington. During my undergrad, I interned at NASA Ames Research Center, Intellectual Ventures Lab, Cisco Systems, and Ball Aerospace. I also did a zero gravity flight with NASA to run an experiment aboard the Vomit Comet as part of the Microgravity Team, and worked on the Human Powered Submarine team. After graduating, I bounced around jobs at Valve, Microsoft Research, Magic Leap, Microsoft, and Meta Reality Labs before coming back to the UW to work on my PhD in computer science in the Ubiquitous Computing Lab. All in all, I have about a decade of post-undergrad experience in industrial R&D labs and occasionally on product. I have been a teaching assistant for the Embedded Systems Capstone for four years. By no means am I an expert, I still feel like a newbie for a ton of this, but hopefully some of this will still be useful.

1.2 What are Embedded Systems?

So what is an embedded system? It's kind of hard to give a definitive answer on that. There's the Wikipedia definition but a lot of the time embedded systems are really referring to most of modern electronics that we interact with in our daily lives. They can range from something as simple as a desk lamp to something as complex as a phone. Something that's not really an embedded system might be something like a server or large scale power systems. While it's a bit hard to classify, most of these devices you would consider as embedded systems have similar design criteria like low power, some kind of smarts, sensors, some user interface etc. This text will hopefully help you apply some of the theoretical engineering you've covered in your undergraduate coursework to making an device from scratch.

There are many engineering disciplines that go into building an embedded device but typically when you hear someone say they are an embedded engineer the person is a firmware engineer. Those who focus mostly on the hardware, which is the focus for this text, typically find jobs as electrical engineers. Most senior engineers will find that they need to have some experience in both firmware and hardware to be successful and you will likely pick those skills up as you work on the job.

Figure 2: The Sony PS5 DualSense Wireless Controller is a beautiful example of an embedded systems device.

2 The Design Process

The design process will vary greatly depending on the project and organization. If you are building a consumer electronic device at a large company, you will have a much different experience than if you're say working in a medical equipment company. Both these places design embedded devices, but with drastically different purpose and requirements. That said, there are some common steps in the design process that I discuss below. Hopefully you will find some of the terminology useful.

2.1 Ideation

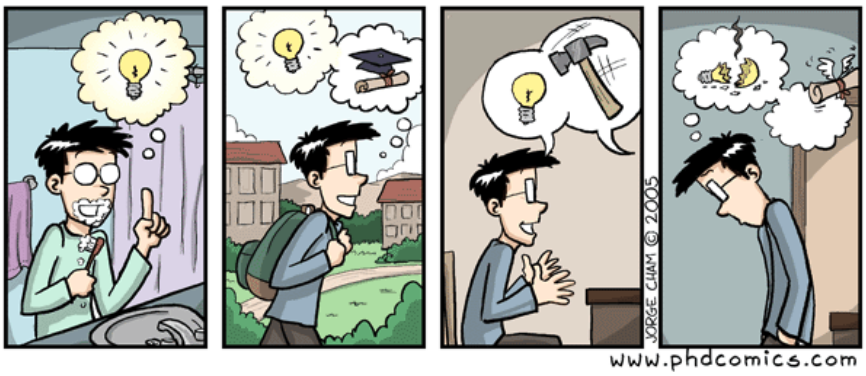


Figure 3: Ideas are hard. If they were easy, I'd be done with my PhD already.

New ideas are hard, but good ideas are what make your product stand out. When you work in industry, new ideas cost a lot of money to try out and are usually the responsibility of the folks at the top or researchers because it is a major investment. There is high risk with hardware because it takes a lot of up front costs to develop something and then build in large quantities.

As a new grad you will likely work on more established projects that don't have as many unknowns. As you get more skilled at your job you will be asked to work more on things that don't have a clear path to completion. You should also keep in mind that not all new ideas from the top are good ones and they can range from a very simple thing like adding an extra button, or a ground up redesign.

As you advance in your career, you should try to keep up to date on the latest technologies because that is how you make new innovations in your product that give you an edge. Talking with field application engineers from companies will give you an idea of the latest and greatest chips as well as their future product offerings which may open up new possibilities for implementing your ideas. Academia is a good place to go when you want to see what the bleeding edge is. In academic labs we often explore ideas that don't have any immediate financial benefit, but often create concepts that can lead to significant advantage once developed.

2.2 Product Requirements Document & Timeline

You may hear the words Product Requirements Document or PRD when you work on a project. As the name suggests, this is usually a document listing the various requirements for the product. This could be something as straightforward as saying it needs to have an indicator light to show that the device is on. Or it could be something more ambiguous like the device must survive a 5m fall.

The PRD is not a technical spec that determines the implementation of the device. It is a collection of requirements that the team has determined is critical for what you want to create. These requirements could be gathered from market research or insights from industry experts. This is also where you will likely interact with a Program Manager who works with many different teams and can craft requirements that are relevant to each team. Engineers generally will participate in these discussions because they need to ensure requirements are not pie in the sky ideas that have no feasible path.

Locking a PRD is usually up to directors and executives as it's essentially an internal contract with the team saying this is what we're going to spend the next few months designing towards.

It is also important at this stage to be mindful of feature creep. You want to lock in a set of requirements for your final product that you stick to. If the team thinks a feature is important to have or easy to implement, that should be written in and locked down and not added after resources have already been allocated. This might be easy to do in a class capstone with a team of 4 and one final device, but when multiple groups are involved adding a simple feature may mess up a lot of schedules and cost lots of money at scale.

You will likely establish a preliminary timeline based on all the things you want to implement and the resources you have to do so. More senior engineers will be able to provide more accurate estimates and you will slowly pick up that intuition as you get more experience. The timeline can be in the form of a Gantt chart or some other formal tool like Asana. Timelines change, and delays are common as people promise more than they can deliver. It's always good to under promise and over deliver.

2.3 Early Prototype

At this stage, your team may be tasked with developing a proof of concept prototype. This is where you prove out a concept or build a single device. At this stage, the project is likely in a pseudo limbo state as new devices can be very costly and you want some assurance of success before going all in. The cost will be high because you are only building a quantity of one but it will allow you the flexibility to test whether or not something is feasible. This stage looks very different depending on if you are in product or research. If you are in research you will likely never move past this stage. You build enough of something to prove your idea, write your papers, and patent or tech transfer the system over to a product group. A product team will be focusing on things like how well this new concept translates to experience or if it's a cheaper way of doing something that's already out there. If the company you are working for has been around for at least one cycle of hardware you'll likely have parts of a system already that you'll build upon to make prototyping a bit easier. This is also one of the most fun stages as you are not fully committed to designing something that needs to be at cost and in high quantities; you are just exploring new designs.

2.4 Engineering Prototypes

An engineering prototype is still a prototype, but your organization has committed to taking it to launch, and you will likely build somewhere between a few dozen to low thousands so as to be representative of the final product. You will hear terms like EVT (Engineering Validation Test), DVT (Design Validation Test), and PVT (Production Validation Test) or maybe something like P0, P1, etc. These are terms for the different prototyping stages in a product's development. After you have a proof of concept you typically go through several iterations of prototypes that progressively test more features that will show up in the final product.

Each stage tests different things for example, an EVT will be a prototype that starts to get close to what the product will be, you'll be using parts and housing that will be analogous to the final product so that you can make sure this design is valid. You can also start doing studies on comfort and use cases to see if there are any changes you'll want to make. You also do this to weed out any issues that could come up in the manufacturing process.

Once you've past that phase you'll do a DVT where you start to validate the final configuration that will go to production. In your EVT you may design in hooks to test different features that are still being tested. In DVT you'll lock down some of the features that you'll be selling as SKUs or Stock Keeping Units. These will also be used to do testing like environmental tests, durability, etc as it will be fairly representative of the final product.

At the end of the prototyping process is the PVT where you are going to test your production pipeline. At this point the design is like 99% locked. You're mostly testing out that ability of the manufacturer to produce the high quantity device and at a quality that you will be ok with shipping

to customers. This step you will start to build some relatively large quantities as you're basically done with most of the hardware engineering at this point and much of the work will be software and firmware features.

By that point you will likely have packaging designs and marketing materials in the final works to prepare for the launch. You will often see some products release with an outdated version of the firmware because the hardware is done, but the team may need extra time to prepare final versions of code that utilizes all the hardware capability or fix minor bugs. In cases like this you want to make sure your system has a seamless way of updating.

2.5 Scaling to Volume

More established companies will have typically shipped in large quantities before so they will have hooks built in to make the scaling of a device to mass quantities easier. If you are working at a startup or are trying to scale up a project yourself, this is where you might find the learning experience a bit expensive. When you are building prototypes, whether in quantity of 1 or 100, the costs per unit will typically be relatively high. You will probably be buying off of places like Digikey or Mouser if you are in the US where prices will be high because they have to source the parts themselves and then need to sell it to you at a profit.

Part of your design requirements or analysis will be some sort of Bill of Materials (BOM) cost. This is the cost of all the raw parts of the device. Very early prototypes where you are testing proof of concepts won't usually be as constrained but once you start designing something that will eventually make it to the shelf you will have to think about costs. If you are choosing between two microcontrollers and one is significantly cheaper, you may need to use the cheaper one even if it's harder to use. You'll also find that using discrete components to build a circuit rather than an integrated circuit will usually cost a lot less, but with the tradeoff of complexity and board space. If you ever take apart a larger appliance like maybe a TV, you'll notice that the power supplies for those still use through hole components, and are single layer. This is so they can build cheaper circuit boards that are single sided and can use things like wave soldering to save on cost even though you could get similar performance using integrated power ICs like something you might find on a GPU which would use just as much power, but needs to be much more compact.

When you start to need 10k microcontrollers every quarter because you expect to sell 40k units in a year you will need to talk with a sales rep for the manufacturer. These orders typically need to have contracts associated with them and agreements at VP or Director level because they are a commitment for you to buy a certain number of parts from the manufacturer at a much lower price than you'll find online. A $5 microcontroller you find on Digikey may only cost you $1 when you commit to buying a hundred thousand units over a certain number of years. As the engineer, you will be the one responsible for convincing the higher ups this is the right part. After that it is up to the lawyers to write something up for people to sign.

You'll find that the more you buy the cheaper things are, and the more support you'll typically receive from that company's engineers partly because a lot of sales reps will earn a commission on those contracts.

2.6 Certification & Testing

Before your device can ship you need to have it go through different certifications. You'll likely start this process when you start building your first engineering prototypes though it will be the final version that needs to be certified. Certification means you go through some very formal checks that certifies your device is essentially safe to sell to the public. When you look at the back of your electronic device you might notice a bunch of logos like UL, or FCC. These are essentially stamps of approval from these organizations.

For example if you are selling any device that has an RF component, meaning it's emitting RF signals like bluetooth, you're going to need to go through FCC approval. Check out this (somewhat outdated) article from Digikey that describes a bit of it.

As you complete testing of the final prototypes and go through certification it is also important to make sure you indicate how your device is meant to be used. For example, a hair dryer should not be submerged in water, and legos should not be ingested. While a use case may be obvious, if you do not put a warning or indication in your product material you may open yourself to lawsuits or misunderstandings when accidents happen even if through no fault of the design. It is not possible to certify a device as safe against all conceivable scenarios so you have to ensure you indicate what the device is certified to do and under what conditions.

2.7 Shipping

Once you have everything good to go, you start getting your product manufactured, packaged, and shipped to distribution. Much of that will not be your responsibility if you are at a larger organization. Marketing teams will work to spread the word and most of the time the places you are selling to will handle the last distribution. If you are a smaller business in which case you may be handling that at your local warehouse or lab.

Your job as the engineer doesn't end when the product is shipped though. As I mentioned before, the firmware could still need work before launch date even if the device is making it's way through the final distribution path. Even after a final stable version is released you will typically need to triage and support as edge cases of the use start to come in. Someone may find vulnerabilities and find an edge case that can lead to serious harm to customers. Think about recalls that car manufacturers issue as their product is put through use cases that can't be easily tested in a lab.

You also need to make sure you support the devices that are out there over the warranty period which is basically the time you promise the device will work if used properly and you will take responsibility if for some reason it doesn't. As products get older support will naturally decline as newer versions get released.

2.8 Next Version

Depending on the success of your first product, or market research you may already be working on the next version with improved specifications that you couldn't fit into the first design. This is often the case in larger companies so that they can meet the demand for annual product releases. Designing an updated product will be easier since you might just be updating a few things to add additional features and won't be doing a ground up redesign which could mean completely new architectures. This is also an opportunity to fix issues in the first product that you couldn't get to before release.

3 Systems Architecture

In my opinion, the most important part of getting an embedded system working well is a good architecture. Most of the time this will be designed by someone much more senior than you or someone with the title architect who looks at the big picture and makes design choices that allow for seamless integration. This is one of those times where sitting down with the team and talking things through ahead of time makes all the difference so things don't clash once resources and time have been spent. In my experience every company does it a bit different, but there are some common things you should consider which designing the architecture your system. A system architecture can take many forms such as a simple block diagram, or a large document that describes all the interconnected systems. When you enter a new company it is best to familiarize yourself with how they do things so you are able to converse with the team in a standardized way. If you go into the Case Studies section of this text you can see some simple examples of an architecture.

3.1 Block Diagram

You'll want to start simple. Once you have an idea you can do a back of the envelope block diagram to start with the high level sections and then start moving into more detail. Depending on what group you are in the block diagrams will look a different. A lot of the time the style is set by the more senior engineers who lead the project. The electrical and embedded diagrams will basically show all the major systems and how they connect to each other. Let's take a look at a block diagram for a development kit from Cypress. A lot of projects in industry or research will start with a reference design around a particular development kit, and you just modify it according to your needs.

Figure 2-2. Block Diagram of Prototyping Kit

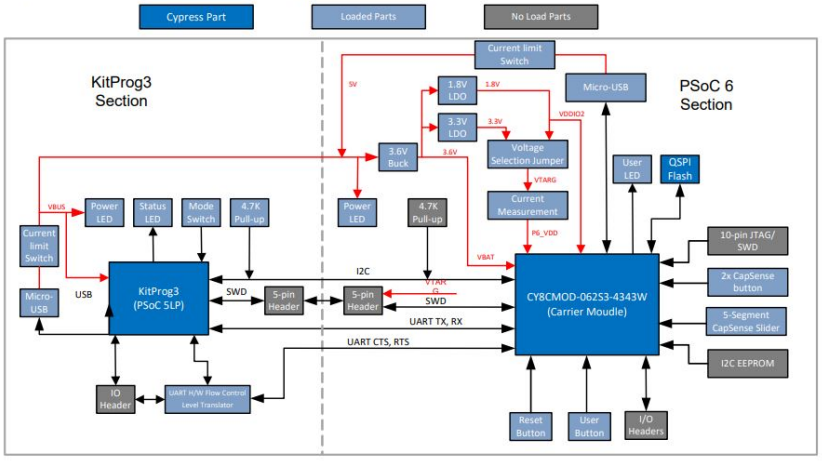


Figure 2-3. Block Diagram of CY8CMOD-062S3-4343W (Carrier Module)

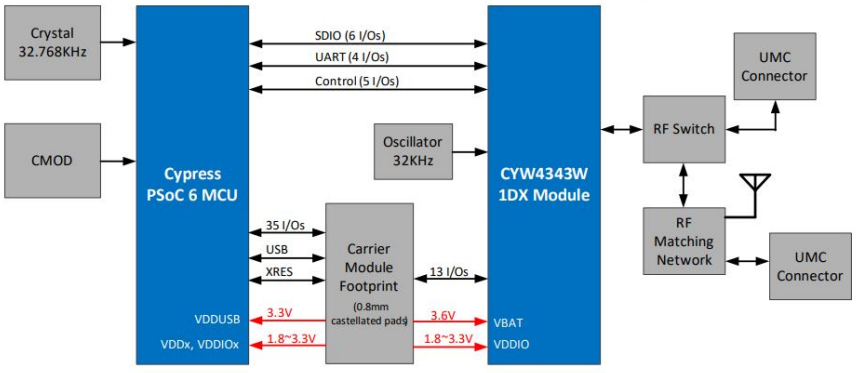


Figure 4: Block Diagram of a PSoC 62S3 Prototyping Kit from Cypress (Infineon)

The block diagram will change over time as you confirm and test parts. The diagram should allow most people without in depth knowledge of the electrical systems to know how things are connected. In general here are some of the things you should include in a embedded systems block diagram:

- **Parts**: You should have all of your major components listed. This could be a general description or an exact part number. The important thing is to show how they are connected to each other. If you have multiple different processors, you want to make sure you know which peripherals are connected to which controllers.
- **Communications**: You should have descriptions of the communications between parts and what ports or pins they connect to. If you look at the block diagram above, you can see for example that the UART communications uses 4 I/O pins. This can be important if you have a limited number of GPIOs and need to keep track of pin count.
- **Power**: Showing the voltage/ current of each device is also useful so you can visually see how the system's power system is connected. It's also a good way to visually see your power tree. There are a lot of things you can do like highlight parts based on their power domain to help visualize what parts need what power.

On top of the core diagram you can add any little details you think might be useful. A block diagram is really just a tool to help with organization and communication. As you work with your peers to finalize a design the block diagram can act as a consensus document so everyone knows how the system is put together.

3.2 Cross Functional Design

While you develop a block diagram, you will have to have close collaborations with all relevant parties in order to have a successful design. In many of your school projects this will be relatively straightforward because teams are small and likely one person will be in charge of implementation. Once you begin to work on larger teams, communication becomes more important. Many teams will have program managers that will assist in organizing the requirements and capabilities of each domain, but as engineers you should make sure you are communicating properly with your counterparts within the project in other disciplines.

Let's take a simple example of this. Say you are designing a security camera. As the electronics hardware engineer you may have to work with an optical engineer to determine what resolution sensor makes sense in order to capture the field of view that marketing has determined is necessary to do well against competitors. This will inform the camera interface to your processor and may push you to one that supports that sensors interface more easily. Now you'll have to work with the MechE team to determine how the camera sensor will be mounted and waterproofed. Maybe there isn't enough room to put it all on one board so now you'll need to design two that's connected with a flat flex cable, but then you realize that's going to be difficult to assemble so you start to consider designing the whole thing as a rigid flex which will increase the BOM cost but only if you make less than 10k units. As you begin doing this more in your career you'll begin to understand the dependencies these embedded systems have within each discipline and catch problems that might come up ahead of time.

I've worked on many projects in research where I design the entire system. While this does make design choices somewhat easier if you have multiple technical backgrounds on smaller projects, it is not feasible in complex one that need to scale and deliver in set timelines. Working with others is a skill that you will pick up as you navigate politics and the workplace during your career and the give and take is something you'll learn to manage.

4 Engineering Tools & Resources

I don't want to rehash the fundamentals you learned in your curriculum so most of this section will be focused on some of the industry standard tools and resources that you might encounter. Every organization has a different preference for tools so I'll just try to list some of the popular ones.

4.1 Electrical Software Tools

There are a lot of software packages used by hardware engineers for different things. If you are building circuit boards you will be using some sort of computer aided design tool to do schematic capture and layout. If you need to design any kind of analog circuit then it is highly advised you do a simulation.

Here is a list of some of the more common electrical software packages you might run into:

- **LTSpice**: The industry standard electrical simulation tool. There are many other flashy ones or those built into your PCB tools, but this one has been around forever and free. You most likely have encountered this at school.
- **Altium Designer**: A PCB design package that is an industry standard. This is the one I personally like the most so I am biased. You will find Altium to be well featured for pretty much any design complexity. It is one of the cheaper professional design suites and is more common in smaller companies like startups.
- **Cadence**: Another PCB design package that is an industry standard. You will find Cadence tools at companies that do very high volume designs or need good integration with a supply chain team. Cadence the company also designs tools used to build ASICs and VLSI chips so is more common at larger companies that can afford the license.
- **KiCAD**: The standard open source PCB design package. They have all of the standard features you would expect to need for a PCB but since it's open source, it doesn't have as nice a GUI or support as a company. This is a completely free package designed by engineers at CERN and is popular with hobbyists or companies that do not have the volume to justify paying for Altium.
- **Saturn PCB Toolkit**: A very useful freeware that encapsulates a lot of PCB calculations into a single software tool. It's not the most accurate thing, but is great if you just need some back of the envelope kind of calculations for quick things.
- **Ansys HFSS**: Ansys makes an industry standard simulation tool for 3D electromagentic designs so anything that requires simulation of antennas or RF signals.
- **Polar Instruments**: They make simulation tools to design PCB stackups. Typically you will use this tool when you are determining the parameters of your PCB stackup such as when you have differential impedance traces. If your internal department doesn't have an engineer specializing in this you will usually just contact the PCB fabhouse for a stackup and they will use this tool.

4.2 Electrical Lab Tools

There are dozens of different debugging tools for electronics work. You will encounter some during your studies, and will eventually learn to master them once you have worked on more projects in industry. Here are just a few of the ones you'll encounter.

- **Oscilloscope**: When it comes to debugging electronics and firmware, nothing beats an oscilloscope, if you learn just one tool this is the one to learn. You should be intimately familiar with how one works as it is critical for you to bring your systems up and debug any issues. Usually you start to get a feel for it in your school labs, but a lot of the time those labs are very guided and you don't really master it until you actually need to debug a circuit. It's really hard to describe what you need to learn so I would recommend you watch some videos on YouTube or read the manual for your scope. To be honest though you will likely learn the functions when

you need to use it to debug. I do recommend when designing your initial prototypes that you put probe points or test clips for the important signals for your scope to hook on to.

- **Multimeter**: The second best tool is a multimeter. It is great for quick measurements of voltage, current, resistance, etc. Keep in mind though that a multimeter is only good for things that don't really change value very quickly. It's not going to pick up fast changing noise or spikes because the display isn't going to be able to keep up. But for any spot checks and
- **Logic Analyzers**: If you want to analyze a lot of digital signals a logic analyzer is critical. These devices operate on only digital signals which allow you to decode and analyze protocols. If you want to make sure the bits you are transmitting are correct and in a correct timing, this is the way to go. This is not the tool to use to make sure the actual digital signal sent is stable, for that you need the analog signal from an oscilloscope.
- **Spectrum Analyzer**: This gives you a real time plot of the frequency components of the input signal. This tool is useful for analyzing signals that rely on stable frequencies like for RF applications.
- **Power Supplies**: You want to have a good variable power supply that lets you do things like limit current and voltage so you can bring up new boards slowly without risking current overloads that can burn your device if there is a short.
- **Function Generator**: You will often want to inject a known signal into your system to see what the response is. An arbitrary function generator can be very useful for injecting a signal you can control. Keep in mind these do not often drive a lot of power so you may need an amplifier.

4.3 Mechanical Tools

Even if you are not going to design a lot of mechanical systems you should still have some mechanical tools with you. Some of the basic ones you should have around are:

- **Calipers**: You are going to need to measure the dimensions of a lot of your electronics while designing. Mitutoyo makes the best out there, but they are super expensive. One day I'd love to buy one for myself, but for now the generic ones off Amazon work fine.
- **Screwdriver set**: You should have a set to assemble or disassemble things. I really like iFixit's sets. Wiha and Wera make good tools too, but they cost a lot.
- **Side cutters**: When you soldering you will need clippers to cut leads and just snip things in general.
- **Tweezers**: Buy extra tweezers as you will need to be able to manipulate very small parts and they will often break.
- **Scalpel & Scissors**: Often you will need to cut things very accurately.
- **Microscope**: A microscope is very useful for inspection and soldering. If you can't afford to get a optical one, there are a lot of digital ones these days with a screen that aren't too bad.
- **Thermal Camera**: A thermal camera can be a very useful debug tool as it lets you see what components are heating up which can help you isolate things like a short where there is high current. This also lets you figure out how hot something is getting so you can design proper thermal management.

4.4 Online Stores

These are some of the online stores that are great for buying parts.

For buying electronics components and related hardware these are some of the places to visit:

- **Digikey**: One of the largest electronics component suppliers in the US. I also feel their search functions to be the best for narrowing down to the part with all the features you want.

- **Mouser**: An excellent competitor to Digikey. I find their search to not be as good but they have a lot of parts that Digikey does not stock or have exclusives you can only find on Mouser.
- **Sparkfun**: When you are just starting out as a hardware engineer, this is a fantastic place to start. They make breakout boards for a variety of sensors and have many of the basic components you would need. What makes them stand out is that they have tutorials, excellent explanations, and example code for pretty much every part they sell. The only downside is the prices aren't the cheapest because they offer mostly designed products and not raw parts.
- **Adafruit**: Echoing everything for them as Sparkfun. They offer great products and tutorials as well.
- **Alibaba/AliExpress**: There are a lot of parts that you might not have access to if you don't buy in large quantities. They would also stock some more niche products like displays or camera systems. Prices can be dirt cheap, but quality may not be the highest. Shipping will also take a very long time.
- **Seeedstudio**: They are kind of like Sparkfun and Adafruit based in China. They also provide some services like PCB manufacturing.
- **Amazon**: While they don't have the best selection of stuff, if you need something quick you may be able to find it on Amazon. They will also sell stuff like dev boards from some generic manufacturer.

For mechanical components I find the best places online are:

- **McMaster-Carr**: This is basically Digikey/Mouser for mechanical parts. High quality, relatively low cost and very importantly pretty much every component comes with a high quality CAD model that can be downloaded so that you can integrate it into your design.
- **Amazon**: You can buy a lot of mechanical parts on Amazon like screws, nuts, etc.

4.5 Home Setup

Most people can't afford to setup a industry level lab at home, but tools have gotten a lot cheaper in recent years. You can find a lot of videos online that can help you with choosing equipment. I found this video from the EEVBlog on setting up a home electronics lab pretty useful back in the day though it's a bit outdated now.

- My recommendation is that getting a cheap multimeter is a bare minimum. Personally I would get something that's in the $50+ range so you know the construction is at least decent, especially if you ever plan on checking high voltage. It's also good around the house if you're ever checking if an outlet is broken.
- An good oscilloscope is going to be a bit out of range for most hobbyists but there are cheaper ones online that are USB based. Digilent's Analog Discovery units are good name brand USB scopes with a very power software suite. If you are a student, make use of that student discount.
- A NanoVNA might also be something good to pick up as a sub $100 vector network analyzer if you plan on doing any RF work.
- You can get a cheap logic analyzer online. Saleae makes great usb logic analyzers and software, but they are expensive. There are ways to use their software with the cheap knockoffs, but it is not the same quality or performance.

4.6 General Resources

I always forget things and need a refresher whether it's for a circuit or the best way to structure part of my code. There are tons of resources out there these days including the many LLMs like ChatGPT that can help answer questions. I've found these websites to have good explanations and tutorials on embedded hardware related things:

- **Youtube**: This is one of the best resources for engineers in general. You want to find videos with decent view counts from reputable people that explains things clearly.
- **Wikipedia**: The standard starting point for a lot of topics. You can be sure that the content on wikipedia is correct, but it can be very technical and may not help you understand new topics.
- Electronics-Tutorial WS: A great compendium of electrical engineering topics.
- All About Circuits: Another great compendium of electrical engineering topics.
- Stack Overflow: Not just for programming questions, stack overflow will often have circuit questions answered.
- **Texas Instruments**: One of the oldest electronics companies in the US. They provide excellent design guides and white papers. Their Handbook of Operational Amplifier Applications and Op Amps For Everyone are I would consider the definitive tomes on the topic.
- **Sparkfun & Adafruit**: Both of these parts stores offer excellent tutorials on multiple subjects tied to the parts that they sell. If you are just starting out, these are the best places to start.
- **ChatGPT, Perplexity, etc**: LLMs are trained on vast amounts of data. While I wouldn't trust them fully it can be a good way to get started. I would recommend asking it to direct you to credible sources.
- **Books** Embedded design references are abundant online and easier to understand than a traditional text book, but there is still a place for them. Books are good in that someone has compiled a lot of the topics you'll want to know about together into one place like what I'm doing with this document. You should use books and similar resources to learn what you need to learn.

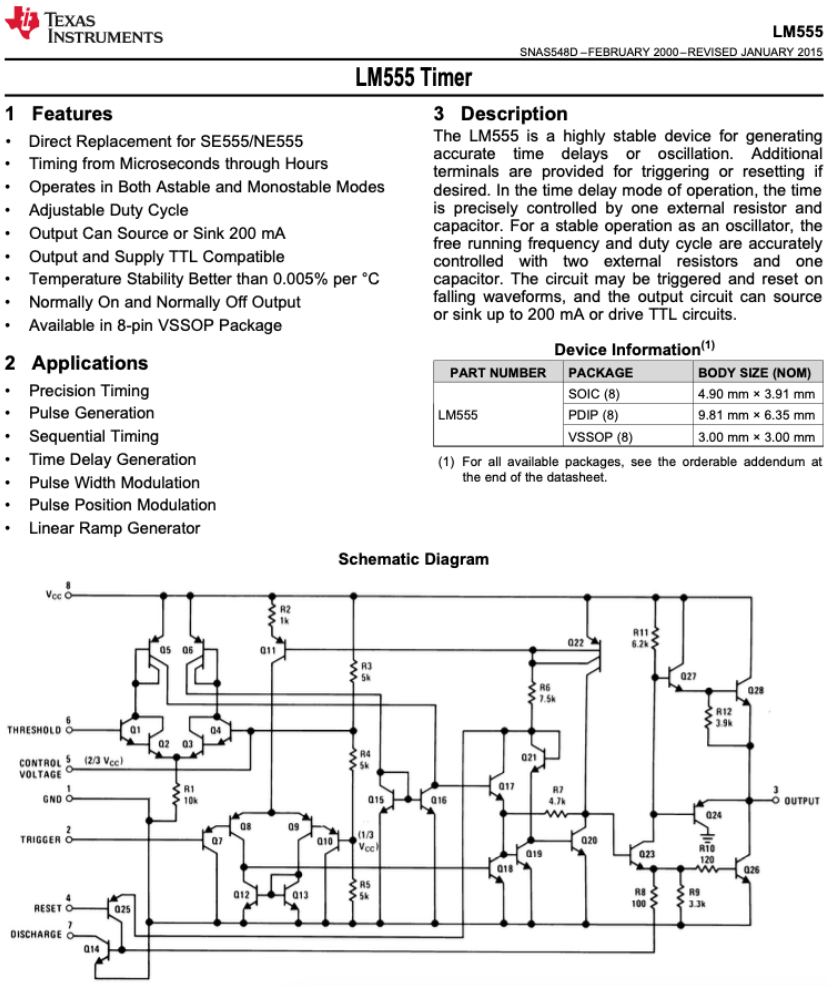
5 Reading a Datasheet

Reading a datasheet is one of the most important skills to pick up. Before I start going over the different systems in an embedded system I think it's important to go over the kinds of things you want to look for in a datasheet and how to read one so you know where to find the information needed to make an informed decision when designing something. Here I'll go over some of the more common sections you'll find in most integrated circuits. In the sensors section we see some of the more application specific things you should look for.

As you get more experience, you'll start to notice what a good and bad datasheet looks like. The larger companies like Texas Instruments or Analog Devices will have standardized very detailed datasheets while smaller companies may not. Sometimes you will also find that there are more detailed datasheets that you have to contact the manufacturer to get. Let's start with looking over a simple but high quality datasheet for a LM555 Timer from Texas Instruments.

5.1 Summary

The first page of most datasheets is usually some kind of summary page that gives an overview of the device. When you are looking for a part to use in your design you'll usually look at this section to tell you whether or not this device is potentially useful and whether you should spend more time looking into the specifics.



TEXAS INSTRUMENTS

LM555

SNAS548D – FEBRUARY 2000 – REVISED JANUARY 2015

LM555 Timer

1 Features

- Direct Replacement for SE555/NE555
- Timing from Microseconds through Hours
- Operates in Both Astable and Monostable Modes
- Adjustable Duty Cycle
- Output Can Source or Sink 200 mA
- Output and Supply TTL Compatible
- Temperature Stability Better than 0.005% per °C
- Normally On and Normally Off Output
- Available in 8-pin VSSOP Package

2 Applications

- Precision Timing
- Pulse Generation
- Sequential Timing
- Time Delay Generation
- Pulse Width Modulation
- Pulse Position Modulation
- Linear Ramp Generator

3 Description

The LM555 is a highly stable device for generating accurate time delays or oscillation. Additional terminals are provided for triggering or resetting if desired. In the time delay mode of operation, the time is precisely controlled by one external resistor and capacitor. For a stable operation as an oscillator, the free running frequency and duty cycle are accurately controlled with two external resistors and one capacitor. The circuit may be triggered and reset on falling waveforms, and the output circuit can source or sink up to 200 mA or drive TTL circuits.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM555	SOIC (8)	4.90 mm × 3.91 mm
	PDIP (8)	9.81 mm × 6.35 mm
	VSSOP (8)	3.00 mm × 3.00 mm

(1) For all available packages, see the orderable addendum at the end of the datasheet.

Schematic Diagram

Figure 5: The summary page for a LM555.

The Features section is basically a marketing summary that tells you why you should buy this part and highlights all of the important specs. For example here it tells you that the output can source or sink 200mA which is important when you want to do something like drive a clock input of another device. TTL compatibility here lets you know what kind of logic level this device supports. The Applications section is another marketing section that lists generic things that this part can be integrated into.

One important thing that might be here is a schematic diagram or a applications diagram where it will show you how the device can and/or should be integrated into your design. A lot of the times, you will just copy the design shown in the datasheet as that is what the manufacturer has tested the part against. In this case, we see the internals of the LM555. We'll see an example of an applications diagram in the sensors section of this text.

Another useful bit of information is the Device Information section which will tell you what packages this device is available in. This is important if you are size constrained in your PCB layout or want something that can be put onto a breadboard for prototyping.

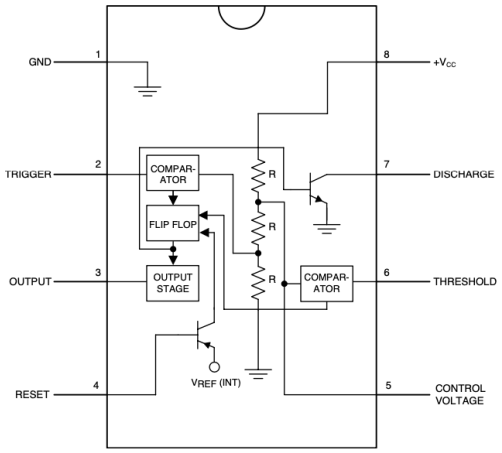
5.2 Pin Configuration

The pin configuration is a great place to get started when reviewing a part because it details what each pin on the device does and typically reviews the different packages the device comes in. You will usually be able to get a good sense of the function from this page as well. For example in this LM555 timer, I know that this chip has the ability to have a reset function which I can tie to a control pin if I don't want the timer output to be active. This part is simple enough that it also details how the internals work relative to the pins, not all datasheets can do this because some parts contain dozens if not thousands of pins like in the case of large FPGAs.

You will also want to pay attention to the different packages as the pin configuration may be different between different packages. In this case, we can see that it comes in different 8 pin packages like PDIP, SOIC, and VSSOP. These all have the same internals, but just have different physical geometries that work better for different constructions. For example DIP packages are larger and work better for single sided constructed PCBs and are easier to solder using through hole wave soldering.

5 Pin Configuration and Functions

D, P, and DGK Packages
8-Pin PDIP, SOIC, and VSSOP
Top View



Pin Functions

PIN		I/O	DESCRIPTION
NO.	NAME		
5	Control Voltage	I	Controls the threshold and trigger levels. It determines the pulse width of the output waveform. An external voltage applied to this pin can also be used to modulate the output waveform
7	Discharge	I	Open collector output which discharges a capacitor between intervals (in phase with output). It toggles the output from high to low when voltage reaches 2/3 of the supply voltage
1	GND	O	Ground reference voltage
3	Output	O	Output driven waveform
4	Reset	I	Negative pulse applied to this pin to disable or reset the timer. When not used for reset purposes, it should be connected to VCC to avoid false triggering
6	Threshold	I	Compares the voltage applied to the terminal with a reference voltage of 2/3 Vcc. The amplitude of voltage applied to this terminal is responsible for the set state of the flip-flop
2	Trigger	I	Responsible for transition of the flip-flop from set to reset. The output of the timer depends on the amplitude of the external trigger pulse applied to this pin
8	V^+	I	Supply voltage with respect to GND

Figure 6: Pin Configuration

Here are some of the things you'll also want to consider while you review the pin functions:

- **Pin numbers and names**: This is more important if you need to make a footprint for your device for layout on a pcb. If you do this wrong it will cause major issues later on when you go to test your system.
- **I/O**: the I/O column will typically let you know if the pin itself is meant for input or output of a signal. Usually it's obvious from the function of the pin but this provides clarity.
- **Description**: As the name suggests this tells you about the overall function of the pin. This can help you determine if this device is right for you before needing to read through more of the documentation.

5.3 Specifications

The specifications section is where the meat of the electrical specs are. There are a couple common things here to look at.

The Absolute Max Ratings is where you'll check for the limits of the device. If you go over any of these specs, there is no guarantee this device survives. This will be different for each device. Here we how a power dissipation and some thermal max ratings. In a more complex part, we may have some voltage values that your pins cannot exceed. In parts that have GPIO pins like microcontrollers or logic gates you may see something like Vdd + 0.3V or something similar which means the voltage applied to that pin can't be 0.3V over the supplied power to the part.

6 Specifications

6.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)(2)

			MIN	MAX	UNIT
Power Dissipation(3)		LM555CM, LM555CN(4)		1180	mW
		LM555CMM		613	mW
Soldering Information	PDIP Package	Soldering (10 Seconds)		260	°C
	Small Outline Packages (SOIC and VSSOP)	Vapor Phase (60 Seconds)		215	°C
		Infrared (15 Seconds)		220	°C
Storage temperature, T_{stg}			–65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) If Military/Aerospace specified devices are required, please contact the TI Sales Office/Distributors for availability and specifications.
(3) For operating at elevated temperatures the device must be derated above 25°C based on a 150°C maximum junction temperature and a thermal resistance of 106°C/W (PDIP), 170°C/W (S0IC-8), and 204°C/W (VSSOP) junction to ambient.
(4) Refer to RETS555X drawing of military LM555H and LM555J versions for specifications.

Figure 7: Absolute Max Ratings

6.5 Electrical Characteristics

(T_A = 25°C, V_{CC} = 5 V to 15 V, unless otherwise specified)(1)(2)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Supply Voltage		4.5		16	V
Supply Current	V_{CC} = 5 V, R_L = ∞		3	6	mA
	V_{CC} = 15 V, R_L = ∞ (Low State) (3)		10	15	
Timing Error, Monostable					
Initial Accuracy			1 %		
Drift with Temperature	R_A = 1 k to 100 kΩ,		50		ppm/°C
	C = 0.1 µF, (4)				
Accuracy over Temperature			1.5 %		
Drift with Supply			0.1 %		V
Timing Error, Astable					
Initial Accuracy			2.25		
Drift with Temperature	R_A, R_B =1 k to 100 kΩ,		150		ppm/°C
	C = 0.1 µF, (4)				
Accuracy over Temperature			3.0%		
Drift with Supply			0.30 %		/V
Threshold Voltage			0.667		x V_{CC}
Trigger Voltage	V_{CC} = 15 V		5		V
	V_{CC} = 5 V		1.67		V
Trigger Current			0.5	0.9	µA
Reset Voltage		0.4	0.5	1	V
Reset Current			0.1	0.4	mA
Threshold Current	(5)		0.1	0.25	µA
Control Voltage Level	V_{CC} = 15 V	9	10	11	V
	V_{CC} = 5 V	2.6	3.33	4	
Pin 7 Leakage Output High			1	100	nA
Pin 7 Sat (6)					
Output Low	V_{CC} = 15 V, I_7 = 15 mA		180		mV
Output Low	V_{CC} = 4.5 V, I_7 = 4.5 mA		80	200	mV
Output Voltage Drop (Low)	V_{CC} = 15 V				
	I_{SINK} = 10 mA		0.1	0.25	V
	I_{SINK} = 50 mA		0.4	0.75	V
	I_{SINK} = 100 mA		2	2.5	V
	I_{SINK} = 200 mA		2.5		V
	V_{CC} = 5 V				
	I_{SINK} = 8 mA				V
	I_{SINK} = 5 mA		0.25	0.35	V

Figure 8: Electrical Characteristics is one of the first sections you look at when deciding if a part meets your needs.

The electrical characteristics detail all of the parameters that would be important while you are comparing different parts. For example the voltages that the part needs, what voltages correspond to logic levels, the current requirements, drift, and a lot of other parameters depending on your specific device.

- Test Conditions column which tells you in what conditions the numbers apply to. For example, some devices can operate on various voltage supplies so the measurements may list several different voltages and the different measurements for that parameter in those various conditions.
- There is typically a range for some values like Supply Voltage. You will need to stay within this range for optimal operation of the device. Going outside this range but still within the Absolute Max Ratings may not necessarily destroy your device, but the performance may no longer fall within acceptable thresholds like for accuracy or timing. If there is a typical value listed you should try to hit that number if possible.
- Common sense, make sure you check the units on something like this. If you read something is millivolts instead of microvolts, it can make a big difference.

There will usually be some section in a datasheet with a bunch of plots. What the plots show will usually depend on the part but it's a good reference measurement of how the device operates under various conditions. For example, Figure 2 in the picture below shows how the supply current increases as you increase the supply voltage for the part. These graphs aren't usually super high resolution so it's like you will extract exact numbers from them, but they are meant more to show trends and for ball park numbers so you know if you are close to an edge condition you can take proper measurements yourself.

For something like a sensor, it might show you how the output voltage curve looks as the sensed characteristic changes. For example a photodiode's response to different light levels may not be linear and you'll need to run your own tests or create your own calibration curve.

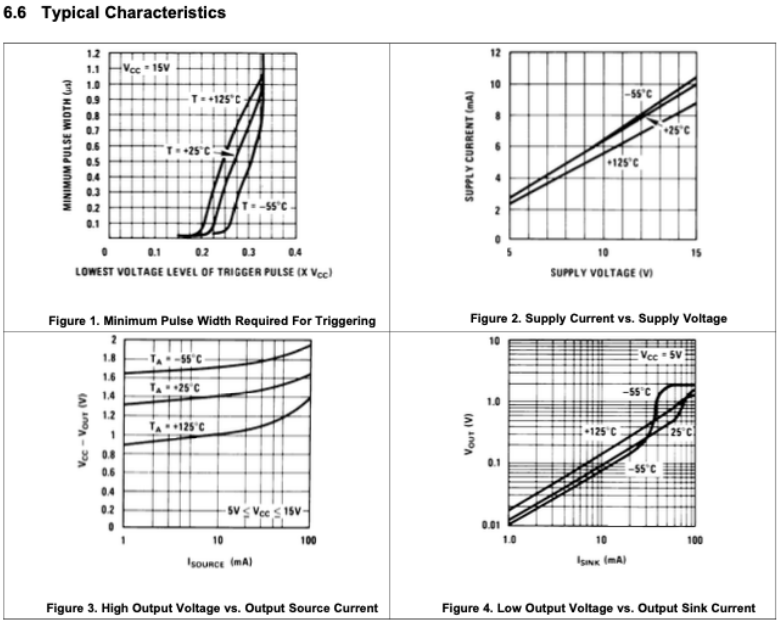
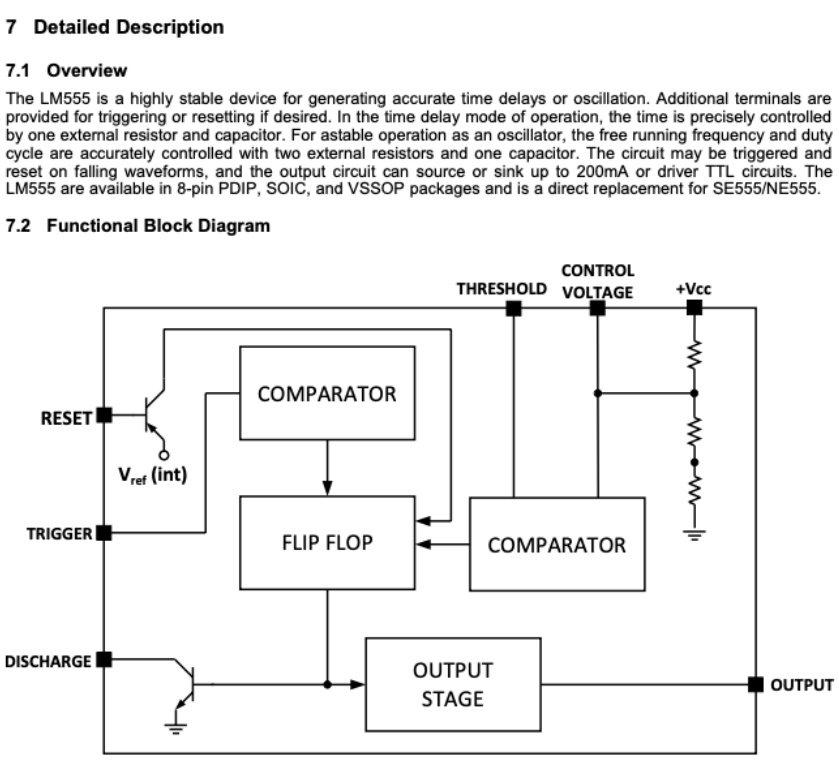


Figure 9: Characteristics plots under various testing conditions.

5.4 Description

The description section goes into details of the operation of the device. For example in this LM555, you are given a block diagram to show the internal make up of the device so you have an idea of how your control signals will change the operation of the device.

This section will likely also detail things like the communications interface, register maps, and other low level details of the engineering that needs to be implemented in order to use the device. Once you've decided to use this device, you will spend a lot of time in this section implementing the details in your firmware or surrounding circuitry.



7 Detailed Description

7.1 Overview

The LM555 is a highly stable device for generating accurate time delays or oscillation. Additional terminals are provided for triggering or resetting if desired. In the time delay mode of operation, the time is precisely controlled by one external resistor and capacitor. For astable operation as an oscillator, the free running frequency and duty cycle are accurately controlled with two external resistors and one capacitor. The circuit may be triggered and reset on falling waveforms, and the output circuit can source or sink up to 200mA or driver TTL circuits. The LM555 are available in 8-pin PDIP, SOIC, and VSSOP packages and is a direct replacement for SE555/NE555.

7.2 Functional Block Diagram

Figure 10: The Description section holds a lot of the low level engineering details.

One of the things you want to pay attention to are any timing diagrams that detail the interface. If you browse this section of the LM555 datasheet you'll notice descriptions of the different modes of operation as well as some diagrams for what the output would look like. For devices that may have a digital control interface like I2C or SPI, this section may detail the timing diagram needed for how data is received by the device. Those protocols are very well established and most libraries will already have functions written that allow for you to interface, but you should check these details if you ever run into bugs and errors.

5.5 Applications

The applications section will usually go up a level and provide you will the details needed to actually make sense of the device. You will sometimes see a Typical Application block diagram or circuit like in the picture below. This section details how this particular device is integrated into larger systems. It might detail how changing certain parameters affects the performance or output of the device. Whereas the descriptions section focuses more on the how, this section is more on the why or what. You will go into this section to learn what you need to modify or configure in order to get the desired outcome like the design procedures for how to make an LED blink in the figure below. If it's a sensor, it may provide an algorithm or conversion math so you can make sense of the numbers the sensor is sending back. You may also find pseudo-code that details how the firmware should bring the sensor up and configure it.

8 Application and Implementation

NOTE

Information in the following applications sections is not part of the TI component specification, and TI does not warrant its accuracy or completeness. TI's customers are responsible for determining suitability of components for their purposes. Customers should validate and test their design implementation to confirm system functionality.

8.1 Application Information

The LM555 timer can be used a various configurations, but the most commonly used configuration is in monostable mode. A typical application for the LM555 timer in monostable mode is to turn on an LED for a specific time duration. A pushbutton is used as the trigger to output a high pulse when trigger pin is pulsed low. This simple application can be modified to fit any application requirement.

8.2 Typical Application

Figure 17 shows the schematic of the LM555 that flashes an LED in monostable mode.

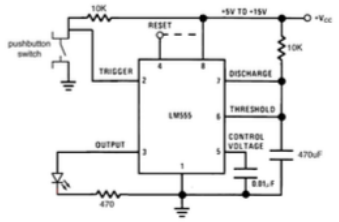


Figure 17. Schematic of Monostable Mode to Flash an LED

8.2.1 Design Requirements

The main design requirement for this application requires calculating the duration of time for which the output stays high. The duration of time is dependent on the R and C values (as shown in Figure 17) and can be calculated by:

$$t = 1.1 \times R \times C \text{ seconds} \quad (6)$$

8.2.2 Detailed Design Procedure

To allow the LED to flash on for a noticeable amount of time, a 5 second time delay was chosen for this application. By using Equation 6, RC equals 4.545. If R is selected as 100 kΩ, C = 45.4 µF. The values of R = 100 kΩ and C = 47 µF was selected based on standard values of resistors and capacitors. A momentary push button switch connected to ground is connected to the trigger input with a 10-K current limiting resistor pullup to the supply voltage. When the push button is pressed, the trigger pin goes to GND. An LED is connected to the output pin with a current limiting resistor in series from the output of the LM555 to GND. The reset pin is not used and was connected to the supply voltage.

Figure 11: Always look for a Typical Applications schematic as it is a excellent reference starting point.

5.6 Layout & Packaging

You will usually find the mechanical details of a electronic device at the end of the datasheet. These sections are important for the layout process. You may find a layout example which can be a good starting point for how you want to position your components. These are not hard restrictions, but if you follow them, it increases the likelihood of your design working. For example, you always want to place decoupling capacitors as close to power pins as possible. For more complex analog parts, these layout examples can be very helpful to avoid noisy signals. You may also find recommendations like using polygon pours for high power nets instead of just traces to reduce impedance.

10 Layout

10.1 Layout Guidelines

Standard PCB rules apply to routing the LM555. The 0.1-µF capacitor in parallel with a 1-µF electrolytic capacitor should be as close as possible to the LM555. The capacitor used for the time delay should also be placed as close to the discharge pin. A ground plane on the bottom layer can be used to provide better noise immunity and signal integrity.

Figure 20 is the basic layout for various applications.

- C1 – based on time delay calculations
- C2 – 0.01-µF bypass capacitor for control voltage pin
- C3 – 0.1-µF bypass ceramic capacitor
- C4 – 1-µF electrolytic bypass capacitor
- R1 – based on time delay calculations
- U1 – LMC555

10.2 Layout Example

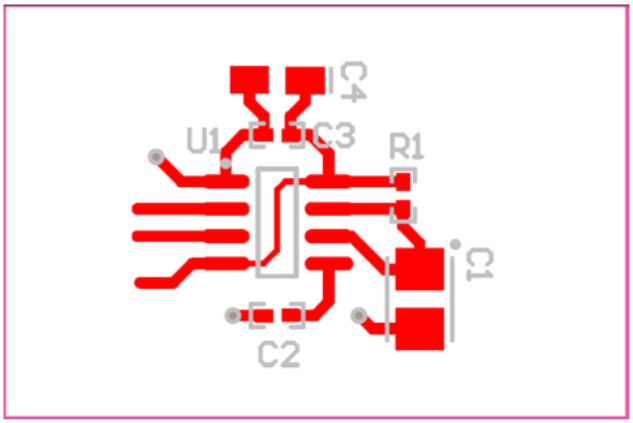


Figure 20. Layout Example

Figure 12: Layout guidelines are always good to follow.

If you need to make a footprint and symbol for the device you will need to use the measurements for the specific device package you are using along with the pin descriptions to create a virtual representation of the device that you can use in your layout. At larger companies, you may have a dedicated PCB engineer or librarian who you can give the datasheet to and they will make the parts to spec.

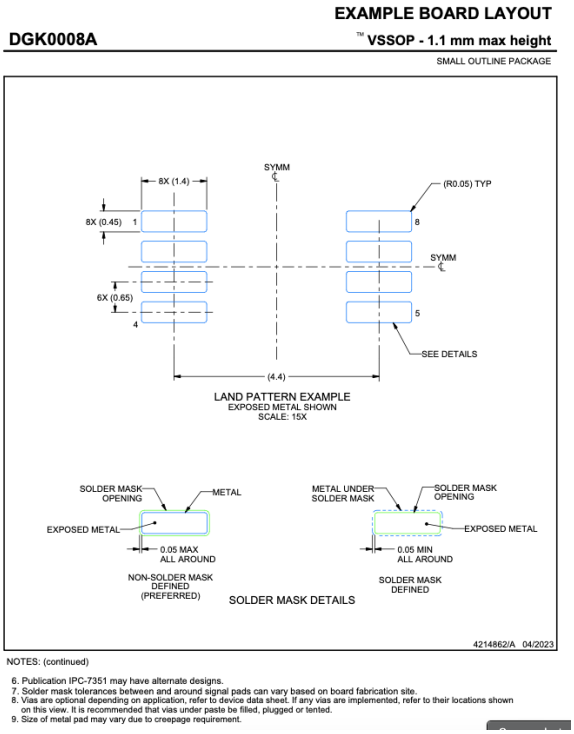
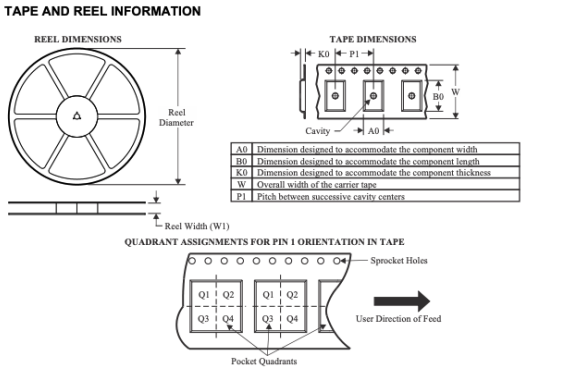


Figure 13: An example of a footprint for the device to sit on top of.

The packaging section is also important when you are ordering your parts. You need to make sure you are ordering the correct SKU. This part is relatively simple, but for some families of parts you could have dozens and you want to refer to this section to make sure you are buying the right one. A part number can refer to specific parameters within a part family like crystals for example can be at various frequencies within a product family and the part labeling with be described in this section. It will often also describe letters and labeling you might find physically printed on the part or how the part is oriented within the tape or reel. This is highly important to make sure you know which pin is pin 1 as many devices are symmetric.



*All dimensions are nominal

Device	Package Type	Package Drawing	Pins	SPQ	Reel Diameter (mm)	Reel Width W1 (mm)	A0 (mm)	B0 (mm)	K0 (mm)	P1 (mm)	W (mm)	Pin1 Quadrant
LM555CMM/NOPB	VSSOP	DGK	8	1000	178.0	12.4	5.3	3.4	1.4	8.0	12.0	Q1
LM555CMMX/NOPB	VSSOP	DGK	8	3500	330.0	12.4	5.3	3.4	1.4	8.0	12.0	Q1
LM555CMX/NOPB	SOIC	D	8	2500	330.0	12.4	6.5	5.4	2.0	8.0	12.0	Q1

Figure 14: A description of the part labeling and ordering information.

6 The Processor

The processor(s) is probably the most important part of an embedded system so we're going to dedicate a whole section on it. Everything runs off of the processor; all of the sensors, communications, even control of the power can be tied back to it. It is also often the most expensive part of a system and careful thought has to go into deciding what to use. Below I've outlined some of the common types as well as what some of the modern options are as of 2024 and the considerations that go into the selection of a part.

6.1 Microcontrollers

For most of the Internet of Things or relatively basic devices you might need to design, a Marvel Cinematic Universe, or MCU, will be your best bet for the main processor. (just kidding) Micro Controller Units are usually the default processor for basically all of our embedded systems designs so let's take a look at some things you should know. This text is not meant to teach you much embedded firmware as that is a whole subject on it's own but as a hardware engineer you should be familiar with embedded firmware/software and should be able to code basic programs in order to test your hardware. I'm not going to go over what a microcontroller architecture (like RISC) is since I am not qualified to do so, and that is more theoretical in nature unless you are doing ASIC or highly constrained design work.

There are dozens of companies that make microcontrollers, and hundreds if not thousands of available controllers to choose from. Here are some of the major companies that you should look at and some of my personal thoughts on them. They all offer microcontrollers that are high quality, it's really just up to you on a case by case basis to choose the right part and company to work with. ARM Cortex architectures are quite common these days with the M series reserved for lower power applications and the A series for more demanding ones. Also keep in mind that there have been major acquisitions in this industry in the last decade so you may encounter some devices that go by their former family name.

- **STMicroelectronics**: An industry standard, they have a huge line of products ranging from small 8 bit MCUs to very performant ARM M7 based ones.
- **Nordic Semiconductor**: They offer a variety of BLE enabled SoCs that have been very popular in consumer devices like VR controllers. The products are well supported with documentation and devkits and are generally low power.
- **Expressif Systems**: The ESP line of SoCs have been incredibly popular with hobbyists, students, and academics in recent years. This is because they provide support for bluetooth and wifi along with their Xtensa core. Many devkits available with a multitude of features.
- **NXP**: They offer the powerful i.MX line of processors that you will see in things like the Teensy as well as lower power ones.
- **Texas Instruments**: TI makes a range of microcontrollers but tend to be more on the ends of the spectrum. They make the low power MSP line as well as the more powerful Sitara line. I haven't seen TI microcontrollers a lot, but I think they are used more in industrial applications. TI as a company is very well seasoned, but have found their support less than helpful unless you have ordered parts in high quantity.
- **Infineon**: They acquired the PSoC line of processors from Cypress a few years ago which is the industry standard for built in CapSense capability. PSoCs are also nice in that they offer very good flexibility for internal configuration. Their internal routing matrix allows for you to switch functionality among pins as needed which can be helpful if you routed something incorrectly. They also offer modules that are already FCC certified for BLE that allow you to just place into your design.
- **Atmel**: You have probably worked with their AVR family of microcontrollers which were popularized by the Arduino.

- **Microchip**: The makers of PIC microcontrollers, they have been around for a while. I used PICs when I was in undergrad before the ARM Cortex microcontrollers became popular. They also have a line of BLE enabled modules.
- **Raspberry Pi**: They are better known for their stand alone linux devices, but they have also started making microcontrollers.

The datasheets for these parts range in the hundreds to thousands of pages because it describes all of the features, registers, memory addresses, etc in exhausting detail. That said, when you are choosing a part, you typically want to focus on some of the higher level details to help narrow down to a few parts before delving too far into the details. These are some of the general features you should review when picking a microcontroller:

- **Development Environment**: Absolutely critical if you do not already have a team of seasoned firmware engineers working on your project. A well supported and documented SDK could save you months of headache. Having a well featured IDE can also help. Most manufacturers will have a dedicated environment you can download or examples to setup on something like Eclipse or VSCode. Some features that might be included are things like power estimation, or the ability to use the GUI to configure some parameters like clock speed.
- **Online Support**: You will always run into bugs or edge cases no matter what part you choose. If you do not have the direct support of an Field Applications Engineer (FAE) you will want to have a good community. FAEs are typically only assigned to large account holders who will purchase millions of dollars of parts. For everyone else you typically rely on the discussion boards and forums which will have engineers monitor and attempt to answer questions, but it isn't a guarantee. For hobbyist popular devices like the ESP32 line, you will often find forums like StackExchange very useful as other senior engineers may have found bugs that they've fixed.
- **Clock Frequency**: The clock frequency is basically how fast your mcu operates. It is the absolute fastest you will be able to execute instructions. The higher the frequency, the more you will be able to do, but also the higher the amount of power needed to operate. These days it is not rare to see chipsets that run well into the hundreds of MHz and even low GHz for some SoCs. If you are looking for low power design though, reducing the clock frequency to something like 8MHz for something with low compute is a good way to drop power requirements.
- **Voltage Level**: You will want to check that the microcontroller logic level meets the needs of peripherals. If you look at the datasheet you may see that a mcu has multiple power domains. This means you might have the option to power the analog circuits in a micro with different voltage than the digital logic. The lower the voltage the lower the overall power you will end up burning so for many power saving electronics, you may want to power your controller off of the lowest voltage it will function at.
- **Pin Current**: If you have any peripherals that are drawing power directly from a pin you need to keep in mind that most microcontroller pins can only source or sink a limited amount of current. Sourcing means it is driving a "high" on a pin with a current going out, and sinking means it is acting as a "low" or grounding with the current going into the pin. Typically pins can sink more current than it can source. Unless the micro pins are tied to the rails, most pins can only tolerate something in the low 10s of mA at most. If you are trying to control something that needs higher currents it's advised to have the mcu pin control something like a FET that can help source higher current from the rails.
- **Peripheral Support**: A critical thing to consider is if the mcu supports a type of peripheral. For example if you want the most performant I2C communication, you need to ensure you have enough I2C buses. While you could implement the I2C protocol on a standard pin, you will not necessarily get the max speeds and performance unless it is on a hardware based I2C bus. Most controllers these days will have a minimum of I2C, but other features like an ADC, CAN, MIPI, Ethernet, Flash are not as certain. For higher end SoCs you might even start looking for support for protocols like PCIe.

- **Memory**: You need memory to store your program and variables as the code runs. Internal memory for mcus is usually fairly limited so you need to make sure your compiled code will fit. You may also need to add additional memory like Quad SPI flash to your system if you are going to need large amounts of stored data like perhaps for images. If you have a device that might have firmware updates over the air (OTA) using bluetooth, you also need to make sure you have enough memory to store the new firmware image that will replace the currently operating one.
- **Bootloader Support**: Something that is important to check is the bootloader support for your device. A bootloader is basically a piece of code that helps load your microcontroller's program. You usually program a microcontroller using a programmer which looks like a circuit with a bunch of wires coming out of it that you attach to your micro. What the bootloader does is lets you load programs more conveniently like over USB because the bootloader helps to load the program into your micro's program memory. An example of this is any of the Arduino boards you are using. If you buy one of those chips raw from a manufacturer unprogrammed, you aren't going to be able to just run Arduino sketches on it. You need to first burn a bootloader on the chip so that it can receive sketches from Arduino and load them into the program memory.

6.2 FPGA & CPLD

You will also likely run into logic devices like Field Programmable Gate Arrays (FPGA), Programmable Array Logics (PAL), and Complex Programmable Logic Devices (CPLD) at some point. A FPGA is basically a very large collection of logic gates that you can program. Think AND, OR, XOR kind of operations which all add up to do somethng more complex. Microcontrollers are nice because you can write very readable programs and there is a lot of support but they execute their operations sequentially or at most on a few truly parallel threads if you have a multicore micro. FPGAs can be programmed so that tasks can work in parallel so instead of say polling 20 sensors one after another, you can do all 20 at the same time. This allows for you to handle massive amounts of data all at once. It is also common for FPGAs to be used when prototyping ASICs. If you have a logic circuit you want to test out, it is easy to reconfigure in an FPGA before locking in the design and taping out to an ASIC. Here's a good Youtube video from EEVBlog on FPGAs.

To program an FPGA you need to use a hardware description language like Verilog. Unless you really like low level design, Verilog can be very frustrating. You are essentially synthesizing the circuits in the FPGA. When creating a design you have to worry about hardware level issues. Let's use timing as an example, in order to physically synthesize the circuitry in the FPGA you need to compile a design, but if the routing path internal to the FPGA ends up being too long, then the time it takes for a high speed signal to propagate through say a gate may not line up with the rest of your system's clock. Many engineers who do FPGA code will become very specialized in it. Personally I think FPGAs are incredibly powerful and useful, but will always defer to a dedicated FPGA engineer to do the programming.

CPLDs and PALs are similar but are lower complexity devices and are what you go to if you need some more basic logic operations done in your system without the need for complex operation. For example if you are using it as a gating mechanism to enable an indicator light to turn on when two other input signals are positive. Things that are more one dimensional and do not require complex logic.

There aren't that many players in the FPGA market. Xilinix (now AMD) and Altera (now Intel) are the two major players when it comes to the larger FPGAs that you might find in server AI accelerators or applications that require very high speed throughput. They also offer SoCs that combine FPGA and more standard processor architectures. Lattice and MicroSemi (now Microchip) offer smaller FPGAs that are more suited for smaller devices for applications like image aggregation.

6.3 System on Chip (SoC) & Application Specific Integrated Circuit (ASIC)

Once you start working on higher end and more complex designs that scale into the millions of units, it is likely that you will start using custom designed ASICs and/or SoCs.

System of Chips are relatively common these days and most of the time you can just use the term interchangeably with microcontrollers, but they do tend to refer to ones that are higher capability or add on additional peripheral capabilities that you won't find in a basic microcontroller. A simple example of a SoC are the numerous chips out there that include some sort of wireless capability like bluetooth or wifi, such as the ESP32 line. They have that functionality that can be a step up from basic control and allows you to have a more integrated system for your application.

ASICs are highly custom pieces of silicon that will usually cost a lot of upfront cost to develop, but will be much cheaper when produced in mass quantities. An example of this is the chips that are used in your oscilloscope. For example Rigol designs their own chips to optimize the signal path of the signal being sensed.

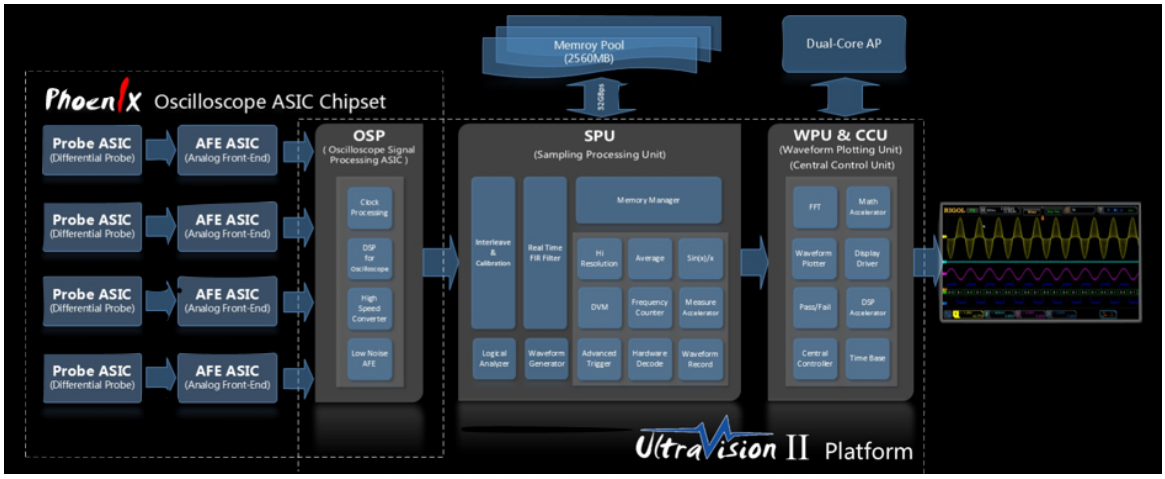


Figure 15: Rigol's Ultravision II Platform

7 Power Systems

Designing the power system may not be the most sexy task, but is critical for your system. These systems can be relatively simple to extremely complex. A lot of embedded devices will have a focus on low power since they could be battery powered and want their device to run as long as possible.

7.1 Power Tree

A power tree is basically exactly what it sounds like. It's a diagram in the shape of a tree that describes your power subsystem. For simple systems you probably won't see a power tree, but you often will for more complex ones. This is to ensure that you meet systems requirements for power such as having a battery with sufficient voltage to power all your systems or choose power architectures that can supply the current for all of your subsystems.

There isn't a standard way to draw out a power tree, I've seen it done in different ways across different teams and companies. Let's take a look at one off the internet from Arduino. If you go to the link, they also do a good job of explaining things.

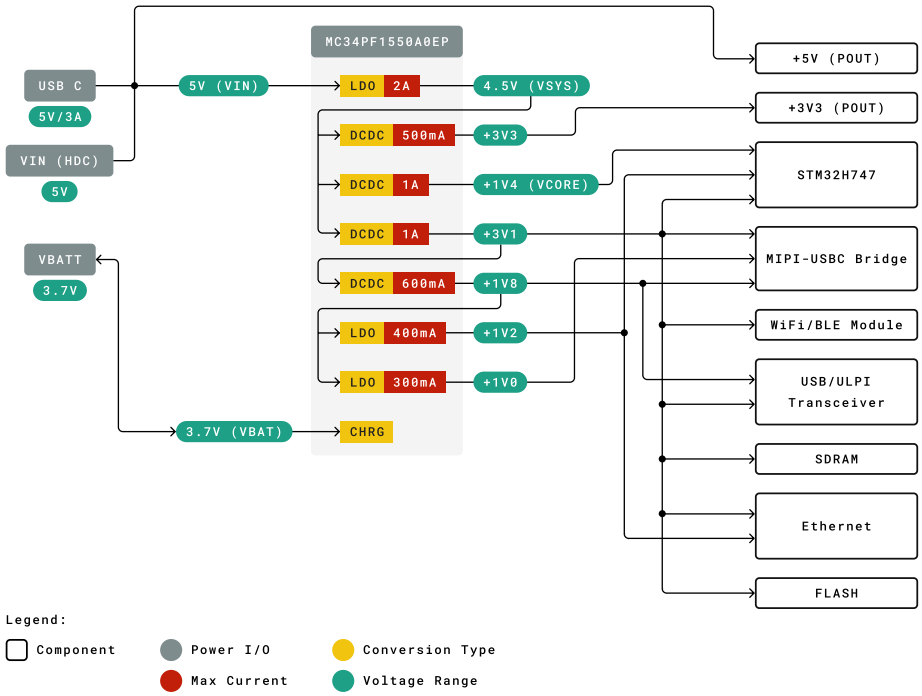


Figure 16: The power tree for the Portenta H7 Dev Kit

Whether you go for a side tree like this or a top down, you typically start with your power input and work your way down to the different subsystems that require power. In this example you can see that they put the inputs to the left.

7.2 DC/DC Converters

DC to DC converters take one DC voltage and regulate it to another set or programmed DC voltage. You may be familiar with them as Buck or Boost Converters from your text books. DC/DC converters have a multitude of different topologies that allow for different behaviors. You can refer to your textbooks or an online search to learn more about how each specifically work. Buck, Boost, and

Buck-Boost topologies all basically adhere to the concepts you learned in class which uses diodes, inductors, and some kind of fast switch in order to manipulate input voltage to the correct output. I am not an expert on these power topologies but there are tons of resources online.

Some of the general things you'll want to consider when choosing a part are:

- **Voltage Range**: You will want to make sure it supports the expected input voltage and output voltage ranges you have in your system. Going outside of these will likely fry your part.
- **Current Range**: Equally important, you need to make sure the part supports the current you expect for your device. If you choose a part that only supports 200mA but you have a 1A requirement, the part will likely not be able to supply the power before burning out. You will also need to consider inrush currents for when a device boots up. Typically you have a current spike that normalizes out but you have to make sure the absolute maximum ratings for your part can support that.
- **Ripple Voltage**: This is the amount of ripple noise you will see on the output voltage. Because this is not a linear regulator, you are relying on the output capacitor and filter to essentially smooth out the signal. This is never absolute so you will always have some sort of ripple. It's usually something pretty small like 100uV, but if you have very sensitive components or amplifiers this could become an issue. You don't want to use a 100uV ripple as the source for your analog potentiometer if you have say a 100000x amplifier for that signal because the ripple can show up as a much bigger signal.
- **Switching Frequency**: Very importantly you must ensure the switching frequency of, and subsequent harmonics of, the DC/DC is not close to the frequency of other parts in your system. For example if your IMU part has a gyroscope has a resonant mems frequency of 200kHz and your DC/DC system powering it is at 100kHz there may be harmonics that get transferred to your system as signal noise. This is also important to consider when you use a DC/DC to power something like an op-amp that does not have sufficient Power Supply Rejection Ration (PSRR) which can cause the minor ripple of the voltage rail to be amplified into your signal as noise.
- **External Parts**: DC/DC circuits typically require an external inductor at a minimum plus capacitors. You have to follow the design instructions on the datasheet to make sure you choose the proper value inductance and current rating. One thing to note is that there are modules that you can buy that essentially have the inductors and capacitors built into them so you just need to solder the modules down which can be helpful if you are size constrained.
- **Layout**: Some layout considerations need to be made when using these as they will radiate noise at the switching frequency. Most of the time you will want to isolate this system to one section of your board if you have sensitive components like a bluetooth antenna or analog circuits.

7.3 LDO Regulators

Low Dropout (LDO) regulators are commonly used to supply very clean regulated DC voltages. They are an ideal choice when you want a very clean power rail like when you are powering an amplifier circuit or something that wouldn't do well with high amounts of noise. The downside is that they are not power efficient because they will drop the voltage difference essentially as wasted power or heat. You will find that many of these components get very hot especially if you are dropping large amounts of voltage like from 12 to 5V at high current. That 3.2V of drop at 1A is 3.2W of power coming out of a relatively small area! Some parts will have a hole for you to mount a heat sink onto to help dissipate heat. In general you should use LDOs for rails that need very clean power like for an analog sensor or an ADC. In order to maximize efficiency it is generally recommended that you use a DC/DC to drop the voltage to something close to your target rail voltage and then use an LDO to take you the rest of the way there.

Some things you'll want to look out for for LDOs:

- **Voltage and Current Ranges**: Pretty much the same reasoning as for the DC/DC above.
- **Dropout Voltage**: The dropout voltage is the voltage that gets dropped at a minimum when regulating the input to the desired output. If the dropout voltage for example is 0.3V, and you want an output of 3.3V, it means the input voltage must be at least 3.6V or above. The higher the dropout voltage the more power you're going to lose as heat.
- **Thermal Concerns**: LDOs tend to get very hot because of the way they regulate power. Make sure you properly design your thermal dissipation path whether it's passive with a heatsink or active with cooling fans.

7.4 PMIC

When your system starts to get really complicated and you start using either very complex controllers or have a lot of components then you will likely use a Power Management Integrated Circuit or PMIC. These devices are meant to be a one stop IC for your power needs and can have dozens of LDOs or/and DC/DC channels with built in control of bring up.

One example of when you might need a PMIC is if you use a SoC like the ones in the Qualcomm Snapdragon line or an FPGA like ones from Xilinx (Owned by AMD) and Altera (Owned by Intel). These chips have multiple power rails and require a very specific bring up of each rail for optimal performance and to prevent damage. With designs like this you of course can choose individual components to regulate power for each rail but that adds up in cost and pcb real estate as your chips get more complex. Having one chip do all of that for you can be a big benefit though as we see below it does add additional engineering complexity.

Let's take a quick look at an example of a PMIC, the TPS6521905 from Texas Instruments. You will see that it has several LDOs and DC/DC converters along with a complex control system. Choosing a PMIC can be very complicated so I recommend that for sufficiently complex power systems that you contact the Field Applications Engineer (FAE) for the company that manufactures it. They will have internal tools and better knowledge of the product base to help you choose the right part. You will also find that for complex SoCs or FPGAs there are recommended PMICs already chosen that already have all of the proper power rails for those chips. I highly recommend you follow the datasheet and recommendations for those designs.

You can see from this example that the TI chip has multiple LDOs, and Buck converters. It can take input from multiple sources to regulate and has a digital control section that lets you configure settings. All of this lives inside one package which allows you to centrally place your power systems. This of course also has the downside of localizing your power system in one place which could make routing difficult if the loads are in separate locations. A more complex PMIC like this is probably designed for a very specific SoC so you can turn on power rails in the appropriate order at the appropriate voltages.

6.2 Functional Block Diagram

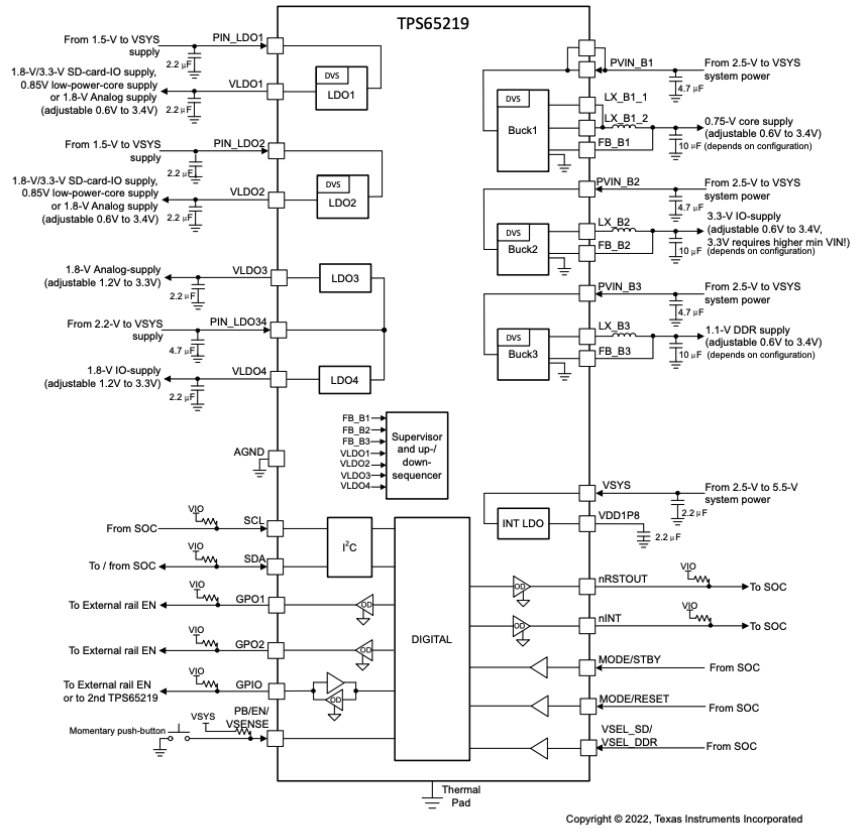


Figure 6-1. Functional Block Diagram

Figure 17: Texas Instruments TPS6521905 Functional Map

7.5 Safety

Safety plays a very important role in a design. In the context of power, we want to make sure a system won't accidentally electrocute someone or start a fire. This isn't an exhaustive summary of the things that could go wrong, but are things to add to your list of things to look out for.

The most common highly dangerous thing you have to design around is probably mains power. If you have a device that plugs into the wall, it means that there is a path between the user and a massive amount of power. Most outlets in the US are going to be 120V and likely 10A with outlets that support things like your washer and oven even higher. Whenever you have the ability to, you should design your device to run off of DC voltage and rely on a dedicated power supply to perform AC to DC conversion either by using an off the shelf module like the one pictured above or by designing your system to run off a barrel jack input from an wall adapter.

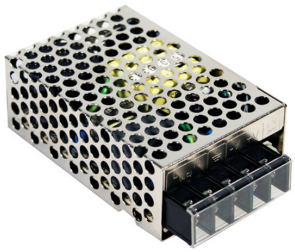


Figure 18: RS-25-12 AC to DC Converter

When designing for something that takes a barrel jack input, make sure you design for the appropriate polarity. CUI has a great tutorial on different kinds of connectors. Sparkfun has a good tutorial too. The positive polarity means that the center pin of the barrel has the positive voltage, and the reverse for the negative polarity.

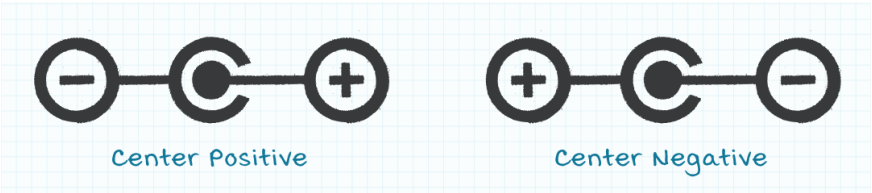


Figure 19: Polarity diagram for barrel jacks.

If you are going to work with any kind of dangerous voltages make sure you have precautions built in. Do not have designs where it is easy to accidentally short something to dangerous power rails while someone is debugging and poking around. Make sure the connector to those rails are clearly labeled and have connectors that are polarized and unique. I remember an incident at work once where someone did not properly label things, and very high voltage was plugged into something it shouldn't have because the connectors were similar. When working with mains power always double check before powering, it could very well save your life.

7.6 Power Measurement

Measuring power is a core skill as an electrical engineer and these days, low power design is crucial for a lot of embedded systems. There are a couple ways to measure power, and depending on your application you may want to measure it at multiple points in your design in order to gauge how much power individual subsystems are using. You may also want to turn off power to systems you find are overloading or trigger warnings when things like battery level drop too low. The most common way of measuring power is to use a shunt resistor and measuring the current and voltage at that point. A current shunt resistor is usually just a very high accuracy, very low resistance resistor. This will allow you to have accurate measurement of current by measuring the voltage drop across the resistor and dividing by the resistance. You do this by measuring the voltage on both sides of the resistor and taking the difference. You can also use two probes on a scope (or a probe designed to do current measurements) and then use the Math function to take the difference between the two. A low resistance value will allow the system to have a relatively low power drop so as not to affect your downstream systems and to reduce power loss. For a few measurements you can just use an multimeter but for more complex systems you may consider designing in analog to digital converters to measure your power and store and sync that data for multiple points across your system. An ADC is also more ideal if you need high accuracy for very low currents as most oscilloscopes are usually just 10-12 bits and a multimeter is usually fairly low too.

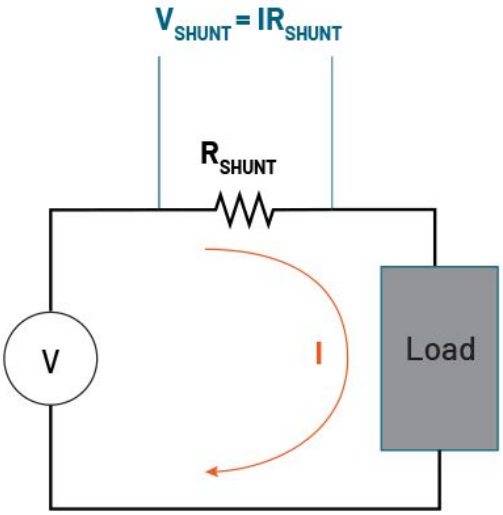


Figure 20: Simple way of measuring power.

For some complex designs your team may want to design a debug board that measures power at different points in your design. This could involve a bed of nails setup where you have test points in your design that a separate daughter card can probe using spring loaded pins. This allows you to design a separate board with things like ADCs that let you monitor a system's status in real time and under normal operational loads which can help a lot with bring up. Keep in mind properly designed systems that do this won't be cheap as it's a whole design in itself. This is more relevant in systems where the board under test is very complex like a board with a large FPGA where you cannot easily probe or systems that have very tight tolerances for operation that you want to confirm.

7.7 Low Power Design

Many embedded devices are designed to be low power. These can be devices that are run off a battery or something you want to be power efficient. An example of this could be a battery powered air quality monitor, or a security camera that uses a motion activated flood lamp. There are also things that are designed to be ultra low power but I'm not going to go into that as much.

There are a few low hanging design choices you can make when doing low power design. The first is to utilize dc to dc converters instead of low drop out regulators. This can introduce more noise into your system, but they are far more efficient. How your power system is designed will play a role in how much extraneous power loss you have in your system. For example, not using a power plane may cause your power to be loss over very long traces that act as extra resistance along the power path.

Most microcontrollers have some sort of sleep or deep sleep mode where the power draw is in the micro over nano amps. This basically suspends the system until something triggers it to wake back up and resume operation. This is something you have to include in the firmware in conjunction with your hardware design. You will also want to make sure you are able to disconnect any power hungry peripherals from their power source. This is where having load switches or a PMIC would be useful in your design. Some peripherals will also have built in deep sleep or reset states. Once a system is in deep sleep, you will need to make sure you have something that wakes it up. This can be any number of things such as using a timer interrupt, or an external interrupt from something like an IMU. Many IMUs have the ability to output a interrupt signal when the is movement above a certain threshold.

This is a good article from Rohm explaining how load switches work. You are essentially using a mosfet to cut off current until a signal turns it on. You can either add the circuitry yourself or buy an

IC that encapsulates all of the discrete components. These are useful if you are designing something that doesn't have an explicit on/off switch like say a game controller. You may want to separate the battery from the rest of the power system when the system goes into sleep mode to reduce the amount of leakage current when the controller is not being used. Then when someone hits a switch, it wakes the system up long enough for the system to hold the load switch open and all for active use. This is also how many battery powered devices ship. They have a load switch separate the main battery from the circuitry until the box is opened so the system has enough battery to work out of the box even if it sits on the store shelf for a few months.

While there are plenty of techniques for lowering power including the ones stated above, some of the highest power savings will come if you create an ASIC. Creating circuits that are 5nm across is just going to be inherently cheaper powerwise than using a transistor you put on a breadboard.

7.8 Power Harvesting

For some devices you may have to design something like power harvesting. The first law of thermodynamics is the conservation of energy. You cannot create or destroy energy, it just moves from one form to the other. Electronics run on electricity, and there are many ways to source that electricity. When we talk about power harvesting we don't mean from a outlet or battery. Typically we are referring to some sort of transfer of energy to our system like from solar panels, or using NFC. Here are some of the ways of power harvesting you might encounter.

- **Solar**: Solar panels are everywhere these days and you can get small ones to power your system. Just keep in mind solar is variable depending on how much sun hits it at specific angles. You will likely need a regulator if your panel doesn't provide it already and use this to charge a battery which can then provide the energy needed to power your device.
- **Magnetic**: Hydroelectric and wind turbines generate our electricity by rotating magnets. You can take advantage of this if you have those renewals to harvest for your system. This also applies to things like those generators you find in wind up flashlights and regenerative breaking in your car.
- **NFC**: NFC is relatively low power and a very specific protocol, but there are chips like the NXP NTAG5 allow you to harvest power from the NFC signal to power small electronics.
- **Qi Wireless Charging**: Wireless charging on phones can provide a fairly sizeable amount of power, sometimes in the 10W+ range, however you need to be right up next to it.
- **Ambient RF**: In our modern world, we have RF signals bounding around us everywhere. There is a WiFi or telecom signal in every home. You can harvest some of this energy to power some low power devices.

8 Sensors & Peripherals

This is probably my favorite section because this is where you find endless possibilities. I'm also biased because a lot of my PhD work focuses on this. In this section I'll go over a lot of the different kinds of sensors out there and some of the considerations you need when working with them. I've not worked with all of these before so some of these are just from my general knowledge or things I've learned over the years. If you are working with a specific sensor, I highly recommend you read the datasheet to make sure it is sufficient for your needs.

8.1 Analog

Analog sensors are ones that output a varying voltage or sometimes current depending on whatever they are sensing. These sensors can vary a lot like a analog temperature sensor or an analog accelerometer. In order to get useful data, you will need to get an Analog to Digital Converter (ADC). Some of the parameters you'll want to consider when choosing an ADC are:

- **Voltage Range**: You need to make sure the ADC is capable of sensing voltages that your analog sensor will output. For example if your sensor is a potentiometer that's powered by 5V and can output 0-5V, but your ADC only supports up to 3.3V, then you will lose the remaining 1.7V of data as it will hit the upper rail or potentially damage your ADC by overvoltage on the pin.
- **Bit Depth/ Resolution**: The bit depth of your ADC gives you the resolution that you can measure your analog signal. This plus your voltage range will determine your dynamic range. An analog signal is continuous so you need to determine what's the lowest voltage step you will need. A 10 bit ADC will give you 1024 steps within the voltage range. We'll look at an example below.
- **Sampling Rate**: The bandwidth of your ADC matters as the analog signal has infinite bandwidth and you are choosing to sample it at something you can process. If you are trying to measure a joystick you probably don't need anything more than a few hundred Hz, but if you want to measure something like audio, you might need something that can measure in the tens of kHz. This also requires that your communications protocol can handle that data so you can store or process it before the buffer fills. The capacitive nature of I2C limits it's bandwidth, and SPI will have it's limitations as well though these protocols covers most of the situations you are likely to run into.

8.2 Optical

Optical sensors are usually sensors that generate a voltage or current based on the photoelectric effect when electromagnetic radiation like light hits a material like doped silicon. The most common ones are probably a variation of a photodiode or phototransistor. I'll talk about cameras separately below.

There are digital optical sensors like those integrated into PPG sensors but most of the time if you are working with a photodiode you are going to need to design a circuit and read the value as an analog input to an ADC like described above. Because photodiodes are usually current based, you need something called a transimpedance amplifier in order to convert that current into a voltage you can measure. There are a few ways to do this, but the most common way is to use an op amp. The All About Circuits page on this configuration gives a good intro to how it works. Essentially you take advantage of an opamps characteristics in order to convert the current to a voltage across a resistor. It's common in interviews to see how you would measure an optical signal so good to read up on it even if you don't use this sensor in a project. Keep in mind that a photodiode's current is usually going to be relatively low so you will likely need some additional gain amplifiers to get a signal that is of reasonable range.

While a photodiode may seem simple, it is also the foundation of our modern communications infrastructure. The high speed optical fiber networks that make up the bulk of the world's inter-

continental data connections utilize high speed lasers and photodiodes to send and receive data respectively.

A phototransistor is a little different in that the silicon that collects the light is tied to the base of a biopolar junction transistor. Because of this, you already have high gain of the current and you can simply apply a resistor to it with a voltage source. I also recommend the All About Circuits page on this. As they point out on the page, while a phototransistor has it's advantages, it does have drawbacks like a worse linearity and slower response time.

Some of the things you need to consider when picking photodiodes or phototransistors:

- **Wavelength**: You need to make sure that the spectral range of the part you choose lies within the the range of the emitter or signal you want to measure. Most of these datasheets will come with a spectral response curve and you will want to pick one that has a peak around the wavelength you are measuring.
- **Dark Current**: This is the current you should expect to see when the diode or transistor is in the dark and is essentially a measure of the noise. This isn't really important in most applications unless you need very high sensitivity.
- **Viewing Angle**: Another parameter that isn't super important for most things but you need to make sure that the signal you are trying to measure sits within the physical input angle of the lense that is put on the sensor. If the viewing angle is say 90 degrees (meaning 45 degrees from center), and you have signal that could be coming in from closer to 60 deg off center, you probably aren't going to pick it up.
- **Response Time**: This is a very important parameter to consider. Since photodiodes are used often for data transmission, you need to make sure the diode you choose responds fast enough. First if you are say sending a laser pulsed data stream that operates in the GHz, a diode that can only respond in the microsecond range will probably not be fast enough to pick up all the data. Similarly if you are trying to detect say the edge of an event and it is critical that you respond within a few microseconds or nanoseconds and the diode you choose does not have a response time that registers the signal edge in time you could have an issue.

8.3 Camera

A camera is basically an array of photodiodes that allow you to take a digital image. There is a very wide range of possible cameras sensors out there so you need to figure out what kind of properties you need. When you look at a datasheet for a sensor it is typically describing the properties of the silicon sensor itself and not a digital camera you can order online from Sony to take pictures with. There are some camera modules you can buy that have the optics and packaging around it already, but most of the time when you are designing a camera sensor into something you are the one who will need to create the mechanical housing, packaging, and optics to make the device work.

These days we are used to cameras on our phones with resolutions like 12MP. For a lot of computer vision embedded applications you will probably not need something that high in resolution, but even a 1MP camera will produce a lot of data. Because of that, most of these sensors utilize MIPI CSI-2 which is a high speed differential protocol. The actual protocol document for the Camera Serial Interface is hundreds of pages long so I won't go over it, but it essentially consists of a clock and data line. The more bandwidth you have, the more data lines you might have. Some of the much larger camera sensors may require the use of dozens of non MIPI differential lines. At that point it necessitates the use of an FPGA.

There are a lot of camera sensors out there, but Sony and Omnivision are two of the biggest ones. Let's take a quick look at a product brief for a Omnivision OG01A1B and some of the features it has. Some of the key numbers you want to look at when considering a camera sensor are:

- **Resolution (array size)**: This tells you how many pixels the sensor is. If you have resolution requirements like you need to be able to resolve certain details this will be an important number.

- **Frame Rate (image transfer rate)**: This is how fast your camera sensor can resolve an image. Some sensors allow you to capture faster, but you have to reduce the resolution because the bandwidth of the communications prototocol is limited and can only support so much data.
- **Shutter Type**: There are two primary kinds of shutter, global and rolling. Global shutter cameras resolve all the pixels in the sensor at once so you can a whole snapshot. Rolling shutter cameras on the other hand resolve line by line and roll down the frame. Because of this, you might have rolling shutter artifacts when objects move quickly as the object in the frame will have moved between the top row and bottom row of pixels.
- **Power**: you have to consider power for any sensor you are using but cameras can use a large amount of power depending on the resolution and frame rate you are capturing data at.
- **Color**: Some sensors are monochrome only meaning they are grayscale, and others can support RGB-IR. Make sure you choose the right kind for your application. If you are using a color camera keep in mind that you will have to demosaic your image once it is received by your application processor. If you zoom into an RGB sensor you won't find one magical sensor that tells you how much R, G, or B light is coming in. Instead you'll find 4 pixels grouped together that each supports a different color, usually via a mask or Bayer Filter. The data stream doesn't label this so you have to parse the incoming data and perform demosaicing or color reconstruction.

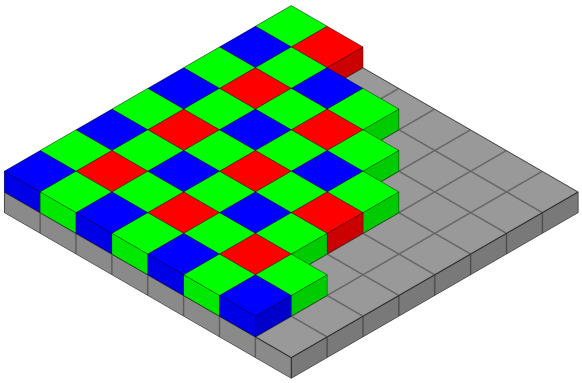


Figure 21: Bayer Pattern

- **Pixel Size**: This is more of an optical parameter and relates to how much light each individual pixel can capture. You might see that some commodity digital cameras work better in low light situations than others even if they are the same resolution, and that is because the size of each pixel is larger.
- **Lens Parameters**: The lens size, chief ray angle, image area, that kind of stuff is important for your optical engineer who is designing the lens array that is utilized with your camera. If these parameters don't fit the optical performance they deem necessary for your application then you can't really use this sensor.
- **Output Interface**: Most cameras these days will use MIPI or sometimes LVDS which are high speed differential protocols. There are some that will work over some serial protocols or parallel, or have the option for both but those are not as common for the ones seen in high volume electronics.
- **Other Features**: There are other features that you might want to look for like the ability to control an external strobe light so you can sync the taking of an image with an external light going off, or things like binning where you can artificially reduce the resolution by averaging pixels together.

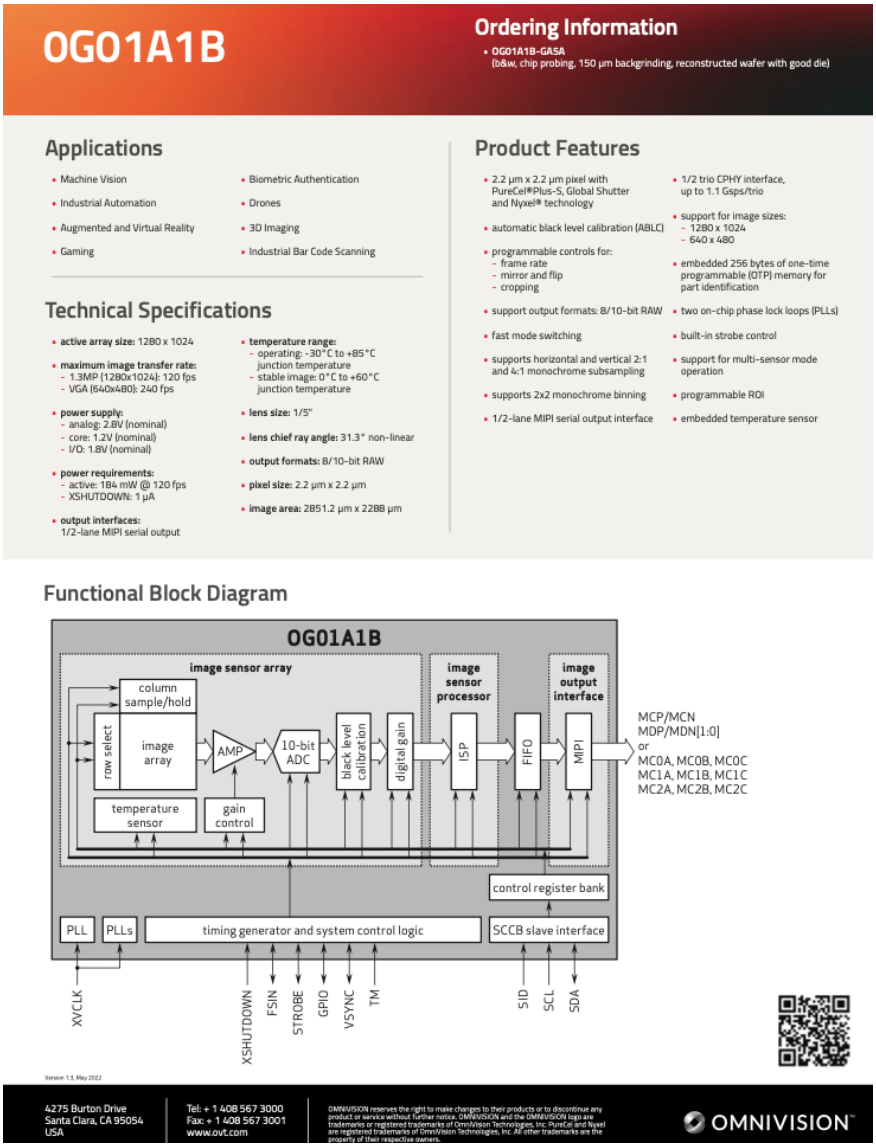


Figure 22: Omnivision OG01A1B Product Brief

Apart from these kinds of cameras there is also a less common newer sensor called an event based camera. Instead of capturing the light in a frame, these cameras output the changes that it sees so instead of capturing what's necessarily in the frame, they capture what is changing in the frame. These are much less common and are used for very niche applications. While the overall resolution is low, these are able to capture things at much higher frame rates because it is not sending all the data from the full resolution of your sensor.

8.4 Thermal

Thermal sensors can exist in a variety of packages like a thermistor that changes resistance based on temperature all the way up to thermal cameras that sense thermal infrared radiation, and output an image. Some motion sensors also work on the basis of sensing infrared light. The interface to these sensors are going to be similar to the analog and camera interfaces we have talked about before. One thing to note with thermal sensors is they are typically much slower than traditional light or camera sensors partly because thermal readings don't really change that quickly relatively speaking. It is common to include a thermistor or digital version on circuit boards and designs to prevent overheating.

8.5 Audio

Audio sensors are basically just microphones of some sort. There are MEMs based microphones which have become very popular in modern electronics and operate off of microscopic cavities that sense minute changes in the air pressure due to sound. There are piezoelectric based microphones that are meant to sense vibrations on contact surfaces like those you might glue to an instrument to amplify the sound. Some will output audio directly as a voltage, and you will need to design an amplifier. You may hear audiophiles hear about analog amplifiers that use old school transistors which allow for the full range of sound to be amplified. Most modern digital microphones will have an ADC that digitizes the sound. Regardless of the configuration or type of audio sensor you get there are certain parameters you need to consider:

- **Frequency Range**: Each type of microphone will perform optimally within certain ranges of frequency. You need to make sure the sensor you choose will pick up the signal you want. You can also take advantage of this to omit certain frequencies like the 60Hz hum that is present in a lot of rooms with electronics or fluorescent lights.
- **Frequency Response Profile**: Each microphone setup will respond to the frequency spectrum differently. It might pick up a certain frequency better because of the resonance of it's construction, or just the audio path inherent to the devices geometry allows for stronger responses in certain frequencies. This isn't usually an issue unless you need very high quality audio recordings. Think about this like a reverse EQ profile that you might find with speakers or headphones.
- **Voltage**: Like any sensor you need to make sure the sensor is in the logic voltage range of your controller or you can damage it.
- **Bit Depth**: A higher bit depth will allow you to resolve details better. These days most digital MEMs microphones have pretty good resolutions.
- **Communications**: Digital microphones can have a couple ways of getting the data out such as SPI, I2C, I2S, and PDM. Make sure your microcontroller supports the protocol your microphone outputs in at the speed you need.
- **Mounting**: Some microphones will perform better depending on the audio path of your mechanical design. This might be adding in gaskets to isolate the audio path to just one input direction, or you might want to sound proof or vibration proof the microphone from external noise sources such as with a digital stethoscope.

8.6 Biometric

Biometric sensors have become much more widespread in recent years. You can find many well designed sensors for heart rate, ppg, bioimpedance, galvanic skin response, eeg, ekg, emg, the list goes. The parameters you have to look for will be very application specific so I recommend you read the datasheet in depth.

Whenever I plan on using one of these sensors, I always check if there is a dev board or breakout board from the manufacturer, or places like Sparkfun and Adafruit. They typically have a circuit that is tested, opensource, and well supported with firmware. You can start there to do initial testing and can then make a custom integration later once you've proven the signal can be collected.

8.7 Environmental

These days you can basically build a weather station at home because of the availability of environmental sensors like humidity, barometric pressure, temperature, pH, air quality, gas, etc. You will have to read the datasheet for the ones you want to use, but something that you want to keep in mind is that most of these sensors require calibration of some sort. They might have been calibrated in the factory which means you'll have to read those calibration numbers off the internal registers and use that in your calculations to reach a final number. Some sensors like the gas sensor may also need boot up time to heat up in order to be able to read accurately.

8.8 Haptic and Audio Feedback

You may want to include things that provide feedback in your system. Common ways of doing this are adding vibration motors, that are basically motors with off balance weights, linear resonant actuators where you apply a waveform at a frequency that mechanically resonates with the device, and voice coils for playing audio. Something to keep in mind with motors is there can be a current kickback since motors are basically windings of wire which is basically a inductor or sorts. It is best that you drive these with a driver chip and not directly off your power supply or GPIO pin as that can cause feedback and damage your circuits.

8.9 Capacitive Sensing

Capacitive sensing is used a lot of our modern world. Your phone's touch screen is based off of capacitive touch. If you recall from your basic EE courses, an RC circuit has a time constant that dictates how quickly a signal degrades and essentially acts as a variable filter. Capacitive sensing works by having a transmitter and receiver pad and taking advantage of the fact that we are basically big bags of water that we are electrically grounded to the earth. As our fingers approach the TX/RX pair, we alter the capacitance with our proximity, and the receiver pad will basically see a change in the TX signal proportional to your finger's effect. Sensing capacitance change is also how a lot of MEMs sensors work. If you ever look at the internals of an accelerometer under an electron microscope, you will see that it has a structure like a comb and as you accelerate, it moves these combs closer together or further apart, changing the capacitance that you measure.

9 Communications

You have the brains and you have the sensors, now you have to get them to talk together. Communication protocols allow you to send data between different systems and have them understand each other. The common scenarios are you are trying to read something off of a peripheral or you are trying to send it a command from your main processor. Companies who design these things have decided on some specific protocols to make everyone's lives easier so there are common ways of interfacing with a lot of different things.

Below are some of the more common protocols you'll encounter and some of the things to look out for. Some of these protocols have documentation that number in the hundreds or thousands of pages so I won't really go into too much detail, but I will link to resources that you should refer to for more in depth explanations.

9.1 I2C

One of the most common serial protocols for interfacing with sensors is I2C. Designed in 1982 by NXP, this protocol involves two signals, clock (SCL) and data (SDA). You have one device that's considered the Controller (Master) which is typically something like a microcontroller and a second device called a Peripheral (Slave) that is usually a sensor or other device under control. You can have multiple devices connected to the same two physical I2C lines.

There are a lot of really good resources on the web that describe I2C. I recommend you do a search, but this is a decent one from Sparkfun. This is a more in depth guide from Texas Instruments. I highly recommend you look at the TI one.

Here are some of the things you need to consider when using I2C (and that are often asked in interviews):

- I2C is open drain. That means the pins for SCL and SDA are physically connected to the drain of a mosfet. Because of this, those pins cannot drive any signal which means you MUST connect the pins to a resistor that is pulled up to a voltage supply.
- Because of the open drain nature of this protocol, a RC circuit is created with the pull up resistor and the inherent capacitance of the pins. The more devices you connect to the channel, the higher the capacitance. Remember parallel capacitors ADD their capacitance. This means that if you choose too high a R value, you create a natural low pass filter that will cause the edges of your clock and data to degrade. You should always do a back of the envelope calculation or find an online calculator to check that your pull up value is properly chosen.
- I2C has a limited number of addresses. That means that you may on occasion have two devices that have the same address or have a need to have multiple of the same device on the same I2C bus. When this happens you will usually need to use some sort of I2C expander. The addresses will typically be 7 bits with the 8th bit determining if you are writing to the device or reading from it.

Let's look at an example of I2C below. In this case we have the MAX30102 which you can commute with over I2C. The address is 8 bits 1010111X with B0 being 1 for read and 0 for write. So if you want to write to the device you send it 0xAE, and if you want to read you send it 0xAF. When the microcontroller wants to start communication with a device on the bus, you initiate a start condition which is a falling edge of the SDA line while you have the SCL line high.

After that, you "clock" in the data at the rising edge of the SCL line. The value on the SDA line must stay the same until the falling edge of the SCL line or the protocol may be interpreted as a start or stop condition. The first data that you write will be the address of the device you want to communicate with. This way it lets the other devices that might be sharing that bus know they are not being talked to and to ignore the incoming data. After the first byte is sent from the controller, the peripheral will send an acknowledgement bit on the next SCL pulse. If the peripheral pulls the SDA line down to a 0, then it means an ACK and it got the data, if it is a 1 then it means a NACK

and the data was not properly received. This could be for many reasons like the peripheral was doing internal functions that prevent it from accepting communications at the time.

Table 17. Slave ID Description

B7	B6	B5	B4	B3	B2	B1	B0	WRITE ADDRESS	READ ADDRESS
1	0	1	0	1	1	1	R/$\overline{W}$	0xAE	0xAF

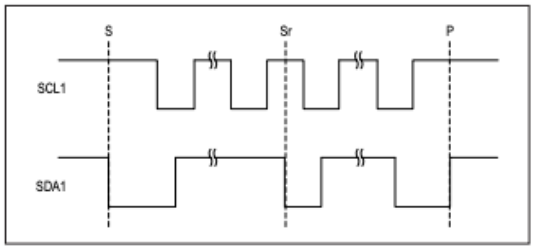


Figure 7. START, STOP, and REPEATED START Conditions

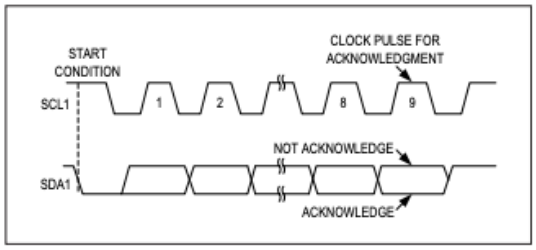


Figure 8. Acknowledge

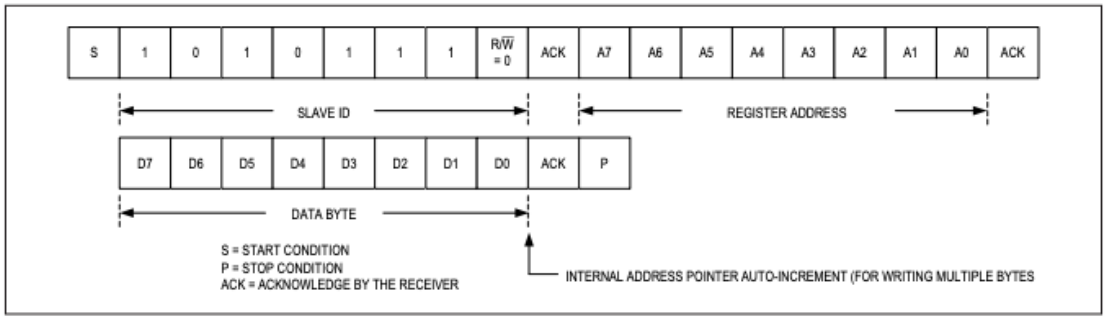


Figure 9. Writing One Data Byte to the MAX30102

Figure 23: I2C Write to a MAX30102

After that first byte and acknowledgement, you send the address of the register you want to read or write to. This is then followed by the data you want to write or read. In the example below you see that we write 0xAE, followed by the register address, then the data byte with ACK/NACK bits in between. If you have more than one data byte to write (or read) the internal pointer on the peripheral will usually increment to the next byte on it's register map so if you started at address 0x00, the next byte it would write to is 0x01 and so on if you are writing multiple bytes. Once you are done you will send a STOP condition which will be transitioning the SDA line from low to high while the SCL line is high. This tells the system that the communications are complete and the line can be released. There is also a repeated start condition which is used when there are multiple controllers on the same I2C bus, but I will let you read about that yourself.

9.2 SPI

Serial Peripheral Interface (SPI) is one of the most common protocols you will encounter when interfacing with peripherals. This protocol was designed in the early 1980s by Motorola and has 4 signals. There are a lot of guides online that do a much better job of explaining SPI like this guide from Sparkfun and this one from Analog Devices.

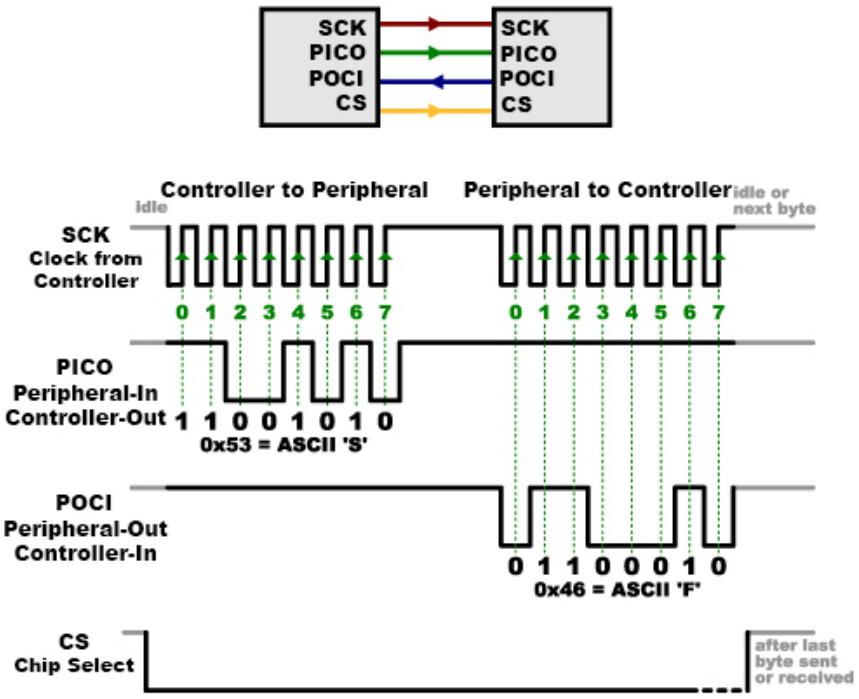


Figure 24: SPI diagram from Sparkfun.

You have a chip select (CS) pin that basically enables you to turn on communications with a device by pulling it low. The Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins act as the data communications between the controller and peripheral device. The modern terminology for these are Peripheral Out Controller In (POCI), and Peripheral In Controller Out (PICO) respectively though they are still used interchangeably for the most part. The MOSI/MISO pin on an MCU connects to the same MOSI/MISO pin on the peripheral. SPI is full duplex which means you can have bidirectional data transfer on the clock unlike I2C where it is single direction. The serial clock (SCLK) acts as the clock line to clock in and out the data bits. You can have different configuration of SPI devices where you have every device on the same bus, or you can have them daisy chained where the serial data out (SDO) of one device feeds into the serial data in (SDI) of another device. You will often find SDO and SDI labels on peripherals so just make sure you connect the SDO to the MISO and SDI to MOSI if you are connecting directly to the MCU.

Unlike I2C there is no initial transmission of the address because you are selecting the device using the CS. This can be a problem if you have many devices because you need a CS pin for every single device. You can get around this by daisy chaining the devices together, but then you won't get the relevant data bits until you've clocked the data through all of the devices. Another difference is SPI pins are all driven so you do not have the same RC low pass filter issues as you find in I2C which means no pull ups and you get much faster bandwidth with many devices operating well into the MHz.

9.3 Logic Level Shifting

When working with communications between different devices, you have to consider logic levels. If a controller runs off 1.2V logic on the pins but you connect a peripheral that operates on 5V logic, that's not going to work out well. You could damage the microcontroller or the signals may not be high enough for the peripheral to register a logic high. In situations like this you need to use a logic level shifter. This can come in many different shapes. You can use a dedicated IC or something as simple as a resistive divider if the data is one way. Most of the time this can be implemented with mosfets.

9.4 UART

Universal Asynchronous Receiver Transmitter (UART) was one of the earlier communications protocols. I haven't run into many devices that use UART for primary control. The exception is devices that send communication in the form of text like when attaching a dedicated bluetooth communication device, or gps which I've commonly seen UART on. UART is also commonly used for debugging when you send messages back to your PC over USB. You might see this referred to as a Virtual COM port that you connect to to receive messages or send messages to your device over USB. You will often find the UART associated with the RX/TX pins on a controller. When connected two devices, make sure that you are connecting the RX on one device to the TX on the other and vice versa. You may also run into something caused the baud rate which is basically how fast the UART is running at.

9.5 Wireless

These days you have a ton of well supported wireless communication protocols. For most embedded designs, you would choose a System on Chip (SoC) part like the ESP32 line of devices, they will come with well documented SDKs that allow you to use these protocols easily. It is pretty rare for you to need to implement a standard protocol from scratch, they are typically built into the device's hardware or in the SDK that comes with the part. Here are a few common ones:

- Bluetooth is a very common one because of it's ubiquity and relative low power. It's everywhere, easy to implement, and well supported.
- WiFi is used for things that require an internet connection or high bandwidth in general, but can use a lot of power.
- Cellular LTE is readily accessible these days using a number of chipsets or dev boards. There are services that allow you to pay for mobile connectivity for embedded devices at relatively low rates. You can also use the lower speed cellular networks like EDGE.
- ZigBee has been around for about two decades and is another alternative to bluetooth though not as common.
- NFC is designed for very close proximity wireless communications for things like reading a key fob or low power sensor.

9.6 SerDes

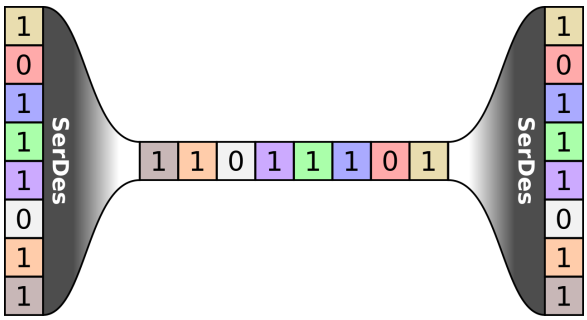


Figure 25: SerDes

At some point you may run into a situation where you need a Serializer Deserializer, or SerDes as it's usually called. This isn't a protocol in that there isn't a standardized way of doing it, but it's more of a concept. When you have a ton of signals that you want to send over a single physical link, you need a way to serialize all of those signals into a sequential series that you can then send over your medium, and then deserialize it so that you can reconstruct the signals from each of the individual sources at the destination. There are chips that do this, or you could implement it yourself using an

FPGA. You could use a microcontroller, but typically that isn't fast enough as you want to serialize a lot of parallel signals at once.

9.7 MIPI

The MIPI Alliance is a organization of companies that writes the spec for the mobile space. It was formed by ARM, Intel, Nokia, Samsung, STMicroelectronics, and Texas Instruments back in 2003 when the mobile space was just starting to form and has now grown to over 300 companies. The two protocols that you will most likely run into are the Camera Serial Interface (CSI) and the Display Serial Interface (DSI). These two are used a lot in embedded systems because cameras and displays show up so often in a lot of projects regardless if they have anything to do with mobile phones. These are high speed differential protocols and will have hardware controllers that control the data flow. Only very specific microcontrollers will have support for these protocols, they aren't very common in standard micros. System on Chips are more likely to have this supported because SoCs are used in higher end embedded systems like phones or VR devices and FPGAs will have transceivers that are fast enough to support these protocols.

9.8 CAN

The Controller Area Network or CAN bus is very common if you are working in the automotive space and in some more industrial related electronics. Outside of those industries you probably won't run into CAN much. Many companies have automotive versions of their microcontrollers that support the CAN protocol if you need it.

10 Machine Learning

Machine Learning, AI, Deep Learning, LLMs have been buzzwords that have permeated tech for the last few years. There are still many engineering jobs that do not utilize AI, but that is likely to change over the next few years. Whether an AI The internet of things wave happened when I was an undergrad about a decade ago and at this point is fairly ubiquitous and mainstream. That era saw the explosion of smart devices that would connect to services controlled in the cloud. The next generation of products will likely start seeing AI incorporated onto these embedded systems on the edge. While cloud based services will continue being a thing, many applications require the privacy and immediate response of computation at the edge.

Figure 26: Machine Learning is complicated.

10.1 Using Large Language Models for Work

Absolutely use ChatGPT, Perplexity, Claude, etc to learn about things and help you in your engineering work. Even CEOs like Jensen Huang have stated he uses AI to learn. Not using them is the equivalent of when teachers said not to use a calculator because you might not have one in the future. That said, keep in mind most companies do not allow you to use them directly because you don't want to feed IP sensitive data into these systems owned by another company. As a student, you should take advantage of these tools just don't copy and paste answers, the instructors have tools to check this built into the systems that you submit the files to (trust me).

What you can do is use these as a tool to learn about new topics or help with general engineering things that are not project or company specific. For example, instead of uploading a schematic to it and asking it to improve it, you can ask one of these chat bots to recommend a specific microcontroller based on your specific needs of speed, cores, memory, peripherals, etc. This can help you to save time digging through a dozen options and instead narrow your search parameters more quickly.

I have made use of these chatbots to do literature reviews and to help me with coding. I still need to understand what the code does, but it can recommend libraries and code structure that saves me time from having to read all the jargon heavy documentation. I have also heard of people using these tools to upload the very lengthy datasheets we went over in a previous section of this text and have the LLM distill the content down quickly.

Regardless of your task you should make use of these tools as needed. Just make sure you have an understanding of what you are getting out of the tool, and don't blindly follow it.

10.2 Basics

Even if you are focused on embedded hardware, having a basic understanding of how machine learning or deep learning works can be beneficial. This is especially true if you are working on things that involves sensors and signal processing. Each project is going to be a little different depending on what you are trying to model. At the end of the day you are trying to create a model of a phenomenon. Here is a general outline of the process for trying to model and interpret sensor readings.

I'm not going to go into all the details of developing a model so I highly recommend you watch some videos online or take a course on it. Andrew Ng's online courses are fantastic, and he walks you through the basics in easy to understand terms, great examples. Machine Learning and Deep Learning are slightly different so take a look at what is appropriate for you or take both if you have time. The only downside in my opinion is it's taught in TensorFlow and these days PyTorch is the more popular framework in academia.

1. **Collect Data**: Without sufficient data, any model is going to have a hard time generalizing a pattern. If you are building something with the intention of using machine learning to model a pattern, the data should be representative of the pattern you want to model. For example if you want to know when a dog barks, you will probably be collecting audio data and not say light intensity data. In order to have an accurate model you will want to have lots of instances of each output scenario so that when you train the model it has enough information to accurately predict the outcome. For example you'll want to have a lot of known dog barks as well as a lot of other sounds that you know are not dog barks.

2. **Data Processing**: This can often be the step that takes the most amount of time. In order to train or do analysis you need to have your data processed and organized. Ideally you would have designed your hardware to collect data in an appropriate way. For example if you know the signal you want to model is going to be a bit weak, you might want to design amplifier and filter stages into your sensor input to maximize the model's ability to find patterns. When you prepare your data for training you will need to extract the features that you believe will be relevant to creating a model. Part of this also depends on the model architecture you choose as many have very specific inputs. For example if you say want to recognize a specific gesture using imu input data, you might need at least a few seconds of data, which might require a buffer, and you may find that for this gesture, it is better to do initial sensor fusion from your imu inputs to output a vector rather than feed in three separate raw streams for each axis of motion.

   This is also where you clean up some of the data using signal processing techniques if the hardware wasn't able to optimize the data collection. This could be as simple as performing some digital filtering to focus on specific frequency bands or ignoring sections of your sensor readings that are obviously too low or too high. Some of this analysis and preparation can also help inform how you deploy this model once it has been trained. For example you may see that the data you collect often has inputs that have exceeded the input range and you can apply a simple heuristic in your firmware to ignore the sensor readings until it falls within range. This way you save on power and compute time for a signal that would otherwise need to be run through the model.

3. **Model Training**: Here's the really simple and definitely not thorough explanation. Essentially what model training is is you take your data and you use it to tune a model. A model is basically just a complex mathematical equation that spits out an answer. Think about the y=mx+b equation you learned in middle school. That is a simple model for a straight line. When x, the input, is a certain number the model will spit out a answer y. The models we refer to in machine learning and deep learning are much more complex, but you typically do not have to implement these yourself because much smarter people than us have already created model architectures that work well for certain applications. For example, the famous AlexNet is a convolutional neural network model that worked really well with image recognition. It's an architecture that is just a sequence of mathematical operations that is performed on an input image.

When you "train" the model you are tuning the weights of the model. To use our previous example, you would basically be changing m and b until it fits with your data, but instead of just two parameters you could have hundreds of millions or billions of parameters. What makes these models so powerful and accurate is they have all these parameters to basically shift the representation of a input to output phenomenon and very importantly a ton of data to make sure these parameters turn out accurate. In supervised learning we know what y is when you input x so you write code that checks whether or not the m and b in the current iteration of the model produces the correct y based on an input x. Then it changes m and b in a way that makes the output y closer to the actual y that you know it's suppose to be. You do that with all of the input and output pairs you've collected in your data set and repeat the process over and over again until the accuracy of your model is acceptable. What acceptable means is up to your application. When you scale this up to something like a large language model that has billions of parameters it requires these massive data centers because you are trying to tune billions of weights so that the output is closer to something accurate. For the much more simple models that an embedded application might use, you can probably train on your personal desktop with a decent GPU or use a service that lets you use GPUs online like Google Colab.

Once your accuracy is to an acceptable number, you now have a model which is just a collection of weights (the m's and b's basically). You can then run new data through your model to get predicted values which is called inference. There's a lot of nuances to training a model well like tuning hyperparameters, splitting datasets into training, validation, and test sets, architecture selection, and a whole lot of other things. If you are going to do any model training yourself, then I highly recommend watching some YouTube videos. Using something like Perplexity or ChatGPT to help write your code is something I've personally done a lot and can help you understand the process.

4. **Inference**: Once you have a model, you can run inference on new data. This means you are loading your model and using new input data like from a sensor or user and running it through the giant array equations that's weighted from your m's and b's calculated during the training. The output of your model may not always be right, but if you've trained it well it should be right most of the time. You typically do not deploy a model into production or publicly until you're reasonably sure it will perform well. In embedded devices you are limited by the capabilities of your processor so you have to do a bit more work to get smaller models to be accurate. This is why having an optimized front end, and really full stack, gives you much better results in the end. Inference is usually where a company will need to optimize a lot because once a model is deployed, running new inputs into it from your users can add up very quickly as the number of users scale. When you train a model, you only have to do that once, inference doesn't have a upper limit. This is where embedded AI can come in handy. If your model can be run locally on a user's device then you don't have to pay for the server costs, the caveat being your model needs to be small enough to run on a user's device.

There are a few options out there to do this, but the most popular ones are PyTorch and TensorFlow. Models also run better on hardware that is optimized for these mathematical vector and matrix operations so GPUs have done very well in the last few years as they were designed for processing visual graphics which have similar operations. It is also increasingly common to see accelerators in microcontrollers that are designed to run models efficiently and go by a lot of names like neural accelerators or neural processing units.

10.3 Embedded AI

These days there is a huge push for on device and edge AI. There are lots of advantages of this including privacy where you are able to process sensitive data like images without sending it off the device. It also reduces data overhead so don't need to send it across multiple networks which can cause lag. I should probably write more on this section but I'm also still learning this myself so I'll talk about some basics and resources until I update this text with more specifics.

For the most part, TensorFlow and PyTorch are the two dominant frameworks for machine learning.

Tensorflow has an embedded version called LiteRT that allows you to run models on embedded systems like microcontrollers or your phone. PyTorch has a similar offering called PyTorch Edge or ExecuTorch. While these are the standard for industry, if you are just starting out you will probably want to use some of the other systems out there that help to encapsulate some of the more complex coding.

Here are some resources to look at if you just want to start your hand at embedded AI:

- **Teachable Machine (Google)**: This is an online free service that lets you create image or audio based models. You basically make a project and start uploading data for specific classes. For example you might want to train a model that recognizes clapping sounds so you'll use the online tool to input a bunch of clapping sounds. This is designed for kids to use so is very easy to get started. Once you have a trained model you can then export the model to something like TensorFlow lite.
- **Edge Impulse (Qualcomm)**: A similar site to Teachable Machine and free for students. This platform also allows you to upload datasets and train a model, but offers more powerful tools than Teachable Machine and is specifically designed for training and running these models on edge devices.

Most embedded systems are referred to as Edge devices because they run close to the edge of these large IoT systems. They do not have tons of processing usually, but are closest to the real time data. For AI models, you need data to train, the more the better. This might mean you collect a bunch of data from your edge device and then train a model using a ton of GPUs which you then distill down to something that will fit into your smaller device. Running a model is called inference and is one of the reasons why using AI is so costly.

It takes millions of dollars of GPUs to train some of these giant LLM models that have billions of parameters, but a lot of things don't require that. Training a model that recognizes say someone clapping, it can probably be done on your modern laptop. However, if you deploy that model online, and it gets run millions or billions of times, you're going to use up a ton of power. If we instead push that model onto the edge and have someone run it locally on their smart watch, we no longer need a giant power hungry data center or the infrastructure for the data transmission path. But we have to have a model that can run on a compute and memory constrained device.

11 Signal Processing

Embedded hardware often works to collect and react to data which is why signal processing is an integral part of any embedded hardware engineer's skillset. You may not need to be an expert, but you should understand many of the basics. For many electrical engineers, a signal processing class is required in the curriculum. Programs like Matlab can help you do very quick prototyping of signal processing techniques.

In order to design the best hardware, you need to have an understanding of a design's goal. If you're trying to pick up very small signals then you need to choose the appropriate input front end that will allow you to capture a high dynamic range input. Without high resolution data, it can be hard to squeeze out the relevant signal you want to see or have your machine learning algorithm find patterns.

Signal processing is a very mature field with decades of developed techniques. If you take a signal processing class you will probably get introduced to the basic theory, but you will probably learn a lot more applicable techniques once you start trying to analyze signals yourself.

11.1 Time Series Analysis

Time series analysis looks at features of the temporal signal. This is usually the simplest analysis where you'll look for stuff like spikes in your amplitude or set thresholds. For example, if you're growing plants in a greenhouse, you might set a threshold for brightness to see how long your greenhouse receives light throughout the year or use it to trigger growing lights. Time series analysis gives you data on intensity relative to time which makes the analysis relatively low on computation, but may require more memory depending on how much time and data you need. Think about this as the analysis you might do on the fly looking at a raw signal from an oscilloscope. Time series analysis alone can have it's flaws due to noise and other issues which you can address with techniques like averaging. For the best results though you typically want to do a combination of time and frequency analysis dependent on the computational constraints you might have.

11.2 Nyquist-Shannon Theorem

One of the core concepts in digital signal processing is the Nyquist-Shannon Theorem. It basically states that in order to reconstruct a digitally sampled signal (which is pretty much all electronic based data collection) you must have a sample rate at least twice as fast as the fast component frequency. So if you are trying to capture a perfect 500Hz sine wave you must sample it at a frequency of at least 1kHz. In practice you should always do more than double to account for variations and noise in your signal, but this sets the bare minimum. If you don't sample at this rate you get aliasing which is basically false low frequency signals created by sampling too slow. This should help you decide what the floor should be in your data sampling which can help you set your required processor speed, communication protocol, memory requirements, etc.

11.3 Frequency Analysis

You are probably familiar with a Fourier Transform if you've taken differential equations. In signal processing we use the Fast Fourier Transform (FFT) which is a discrete fourier transform algorithm that can run on digital processors. Basically we use this technique to determine what kind of frequencies are present in a time series signal. Unlike the output of an oscilloscope, think about this as the analysis you might do on the output of a spectrum analyzer. This can be immensely useful as a lot of signals are not obvious to us when looking at a time series signal. If you look at the signal below for instance, you can see that the noise has done a good job of obscuring any obvious pattern. Let's say you are trying to track a bird that very specifically chirps at 300Hz but your microphones sampling at 3kHz are being obscured by the 60Hz line noise that is present all around us from wall electricity, and general noise from the environment. That might produce the noisy signal you see below.

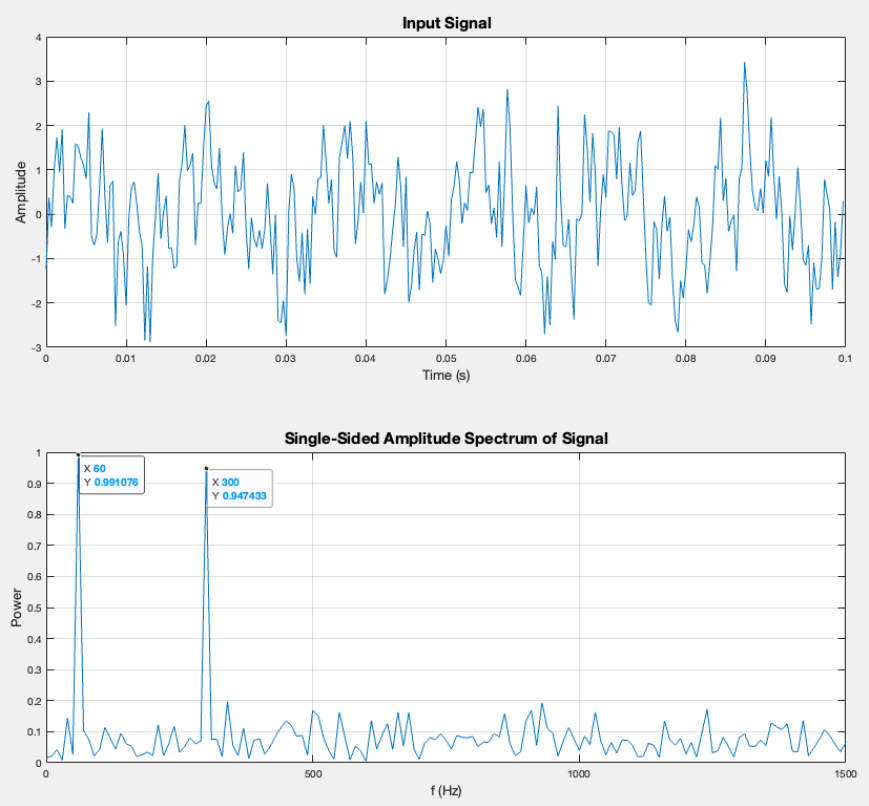


Figure 27: An example of an input signal you might see. In this case it is a 300Hz signal with noise and a 60Hz signal added on.

However if we do a fast fourier transform of this signal, you can see much more clearly what there are spikes in the frequency domain of this signal at 60Hz and 300Hz which would make analyzing this signal for presence of the bird much easier. Now FFTs have their limitations as well. With discrete fourier transforms you are limited by the samples you have. When you do an FFT you basically use buckets for frequency because you cannot have continuous frequencies given you only have discrete temporal measurements. This means the resolution of your frequency analysis is limited. You are also limited in the fact that the FFT is performed over a set number of samples meaning you get the frequency components within this snapshot. You lose the temporal information of when something happens. In this example the 300Hz is used throughout, so it doesn't matter, but say you have a whole day's recording. You would need to break up the signal into smaller chunks to get higher temporal resolution (which could lower the power of the frequency you want to find). A spectrogram is one way of visualizing frequency over time. You basically take an FFT window and slide it across the input time series and then plot it. There are limitations to this as well since your fixed window size still prevents good time frequency resolution.

Another part of frequency analysis is applying filters like low pass and high pass filters. A common thing to do to clean up your signal is the apply these frequency filters to eliminate noisy parts and focus on the frequencies you are looking for. An example would be eliminating 60Hz electrical noise, or focusing on the frequencies of human speech. This is a lot better than just averaging because we are targeting specific parts of the noise that we know aren't wanted.

11.4 Time-Frequency Analysis

While time series and frequency analysis are both powerful tools, they both have their limitations. An FFT gives you a power spectrum of a fixed window but you don't have any temporal data like when these frequencies appear in the signal. If your time window is like 0.1 sec then maybe it

doesn't matter, but now you have to process a FFT every 0.1 sec which is going to eat up a lot of compute.

Now let's just extend our example a bit. I'm going to remove the noise just so the result is a bit more obvious, but let's add a 500Hz signal to the middle third of our window. I've plotted that in green as well to show you where I added it. From the FFT you can see that we now have a 500Hz peak as well, but we have no idea where in this 0.1sec window that happens.

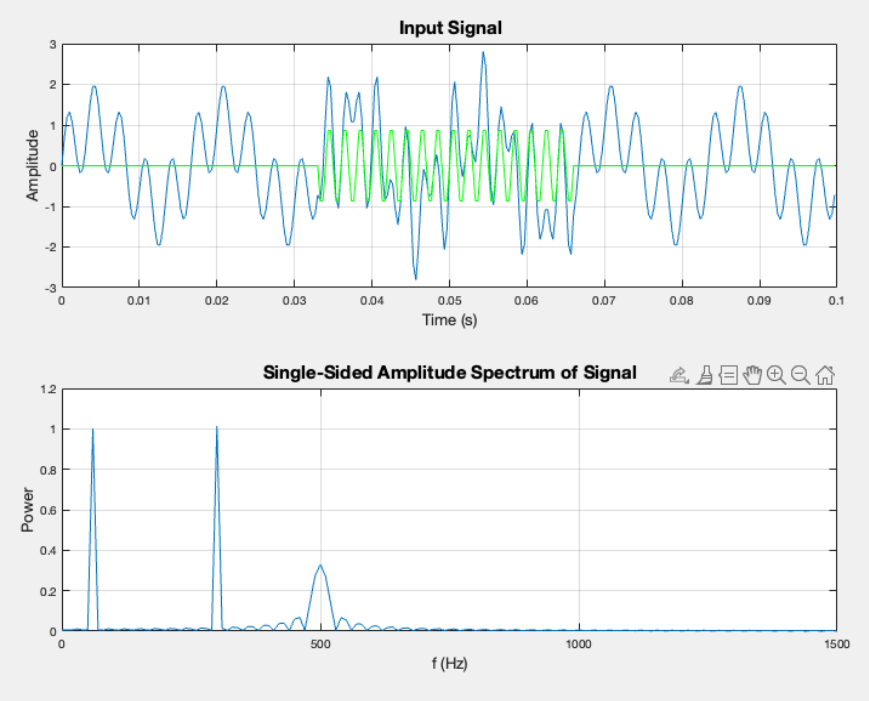


Figure 28: Adding a 500Hz chirp in the middle of the signal.

Now let's see what happens if we create a scalogram of this signal. FYI, I'm using Matlab's Signal Analyzer app cause I'm too lazy, and don't want to ask Perplexity to write the code for me, but you'll get the idea. You can see that the 60Hz and 300Hz signals are present throughout this time snippet, but that the 500Hz signal only appears near the middle. I should note that creating these images are going to be a lot easier on a much more performant system and low power microcontrollers may not be the best choice to do this analysis. FFTs can be done on microcontrollers, there are many libraries for that, but creating an image will usually require a lot more memory.

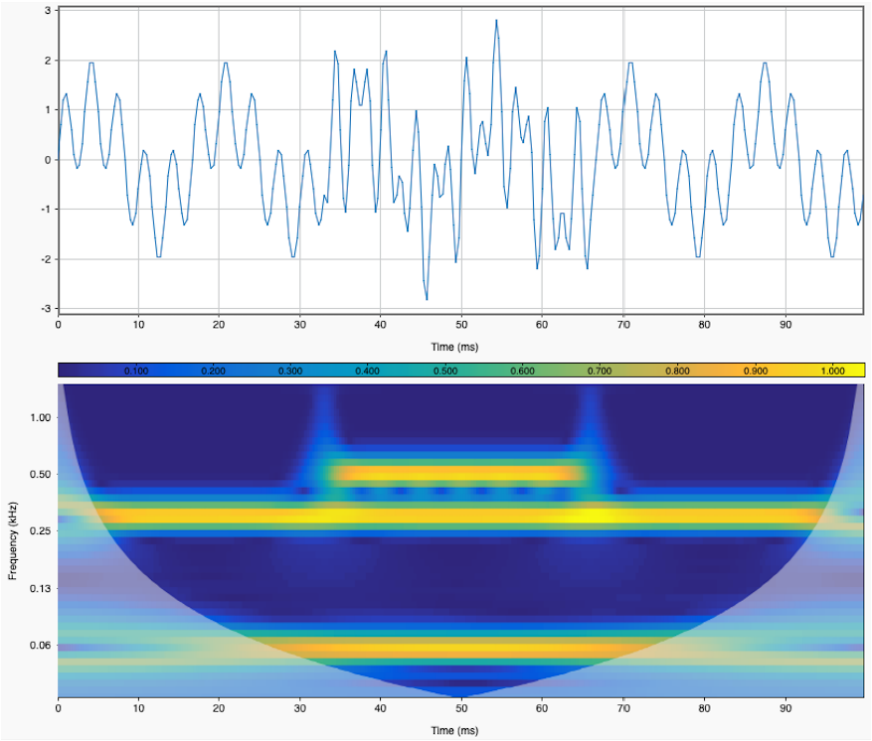


Figure 29: Scalogram of the signal. We can clearly see when the 500Hz signal is introduced in the time domain.

I'm not qualified to explain scalograms so I **highly** recommend Dr. Nathan Kutz's series on time frequency analysis, most of his lectures are pretty awesome. The important thing to know is that scalograms offer you much better resolutions than a spectrogram because it has the ability to vary it's window rather than a fixed window size. You pretty much never implement any of these yourselves, there's libraries out there so the important thing practically is to know how to use the library appropriately. Once you become an expert on it, please write a guide and send it to me so I can understand it myself.

12 Printed Circuit Boards

Circuits can be incredibly complex, but are crucial to all of our modern electronics. As undergrads, we are taught the basic building blocks of those circuits so we have a strong foundation. You should be familiar with breadboards that you've used in your labs. It is not as common to be taught how to make a printed circuit board, at least not until the last years of your education. I can't go over all of the specifics of making a pcb in this document, but there are dozens of articles and very useful YouTube videos. If you are a UW student, I do a PCB tutorial a few times a year as well, feel free to reach out to me if you are interested.

12.1 Schematic Capture

The process of making a circuit board begins with the schematic. This is where you connect all of your components together into the program. As an electrical engineer designing circuits you will spend most of your time in this portion of the PCB process. You will decide what components and what configuration to use. Because this represents how the final device will function you'll want to do some due diligence for making sure it works. This could be putting your design into a circuit simulator like LTSpice or having colleagues do a design review to make sure it will likely work.

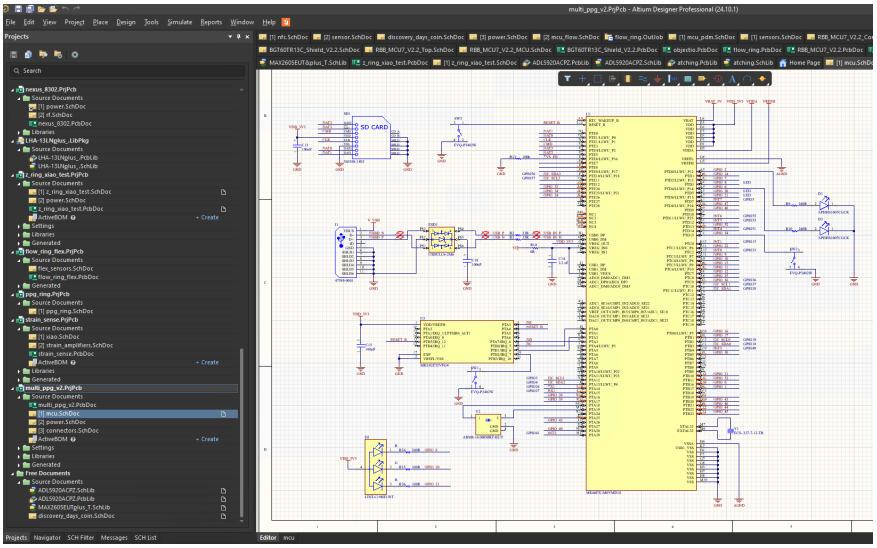


Figure 30: An example of a schematic made in Altium.

As you do your design, you want to choose the best part to do the job, but you may also notice that you need footprints and schematic symbols to represent a part you have chosen. Packages like Altium will often have thousands of parts already loaded into their online library under Manufacturer Parts Search, though these are not all encompassing. If you're on a big team you might have a pcb librarian who can create the part for you based on the datasheet. Otherwise you will have to create the part yourself. I recommend watching some YouTube videos on how to do that. Also make sure you check online resources like SnapEDA or Ultra Librarian where there are often thousands of free parts that have already been made. You just need to import it into the package of your choice. Digikey and Mouser also have many parts that have footprints linked on the product page, there is even a filter for parts that have EDA/CAD models available. Always double check a downloaded symbol or footprint with the datasheet as there can sometimes be errors which can seriously mess up your design if a pin is swapped. With pcb librarians, there is typically a review process, but many of these online ones are made by enthusiasts or companies and aren't always guaranteed accurate.

12.2 PCB Layout

After you have a schematic more or less finalized you will start to focus on the layout. If you are working on a larger team or company they will likely have dedicated PCB layout teams where all they do is use the software to create the PCB. Those are professionals who know the tools and process in and out, but may or may not have a strong engineering background. I learned how to do PCB layout before having the benefit of a team of pros so I'm proficient at it, but once a design becomes too complex, it's not feasible for you to spend the time on it.

Layout is an art and unless you've done it for a few years you'll like have a bit of a learning curve. That said if you are doing some layout, here is a rough guide on some steps to take and things to consider. Having a good plan for doing layout will help the process go much smoother. After a few years of doing this you will be able to jump around and make changes more fluidly but when you are new it can help to be a bit more structured.

When you are starting your layout it helps to have a rough outline of your board and any mechanical constraints like placement of specific sensors, buttons, etc. This will help you begin to constrain your placement.

Autorouters exist and let you route a board with the click of a button, but you pretty much never want to use it. In order for autorouting to work you need to constrain the autorouter so that it can calculate optimal paths within the constrains you put, but it usually outputs a web of spaghetti that looks horrible.

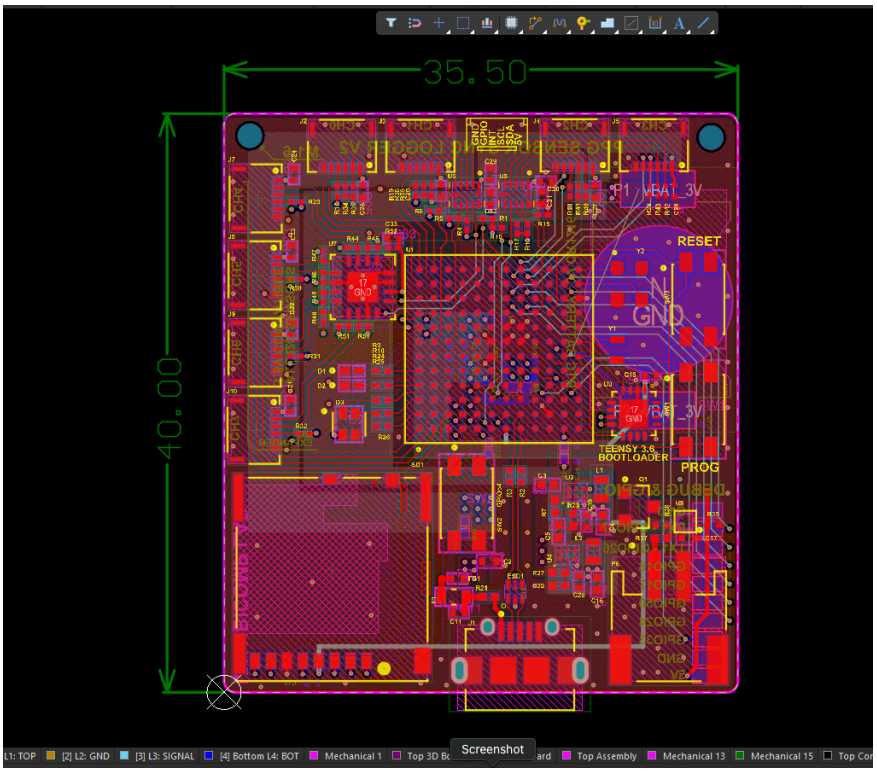


Figure 31: An example of a 4 layer PCB layout in Altium for a multichannel ppg data logger.

Here are some steps you can take when you start layout:

1. **Placement**: Group your parts together by subcircuit. This means you group all the parts needed for each individual system like your microcontroller, buck converter, sensor, etc. Your ECAD program should allow you to select parts on your schematic to make grouping of systems easier.

2. **Layout Individual Systems**: Do the layout for your individual systems based on optimal placement according to best practices and perhaps the datasheet. This means placing all the decoupling caps near the power pins, or laying out the regulators as recommended in the datasheet. Keep in mind things that require specific mechanical orientations, and things like that.

3. **Subsystem Placement**: Once you have the major subsystems laid out, you can then place those within your mechanical outline in the optimal place. Make use of the ratsnest to see approximately what traces need to be routed where so you can place it for optimal routing. You'll get an idea of what that means when you do this more.

4. **Full Route**: Now you can start to connect all the subsections together. This means routing all the signals from the processor to individual peripherals. Once you have more experience you will be able to visualize this a lot easier so that clean up later is easier. You will also want to start connecting the power supplies to all your peripherals with a polygon or traces.

5. **Clean Up**: There's always going to be some cleanup. Perhaps you realize that you don't have enough space to route out all the signals without adding an additional layer, or you realize it's more optimal to move one section elsewhere. Once you've cleaned it up you'll be mostly done.

6. **Final Polish**: Now you can go in and polish up the board layout. This includes things like adding in a final GND pour, cleaning up all the overlay layer so the labelling is lined up with your parts, adding in additional labels for debugging or instruction, adding stitching vias to connect your ground layers together, things like that. You will also want to export a 3D model of the board to confirm integration tolerances with your mechanical design.

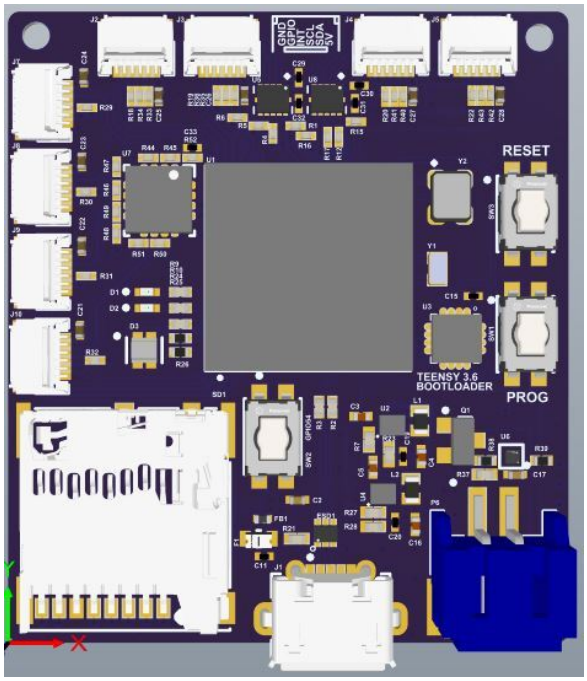


Figure 32: Export your 3D model and integrate it into your mechanical CAD. This is often how electro-mechanical co-design is done.

12.3 Fabrication

The fabrication of a circuit board is incredibly complex. Luckily, there are a ton of easy to use factories who can fabricate them for you in high quality and at reasonable prices. Back when I was an undergrad we didn't have nearly as many options. Today there are dozens of factories in China that have easy to use web portals where you can order a dozen custom circuit boards for the cost of a coffee, though shipping isn't always cheap. PCBWay has a really good webpage on the process of making a PCB if you are interested in the details. If you are working on anything ITAR, military, or other sensitive designs you are legally not allowed to fabricate it overseas so please keep that in mind or you may find yourself on a list you don't want to be on. That said the fabs overseas have the most advanced PCB fabrication technology at the best price, and is pretty much your only option for fabrication in large quantities. If you need something done overnight then you will have to go through a domestic fab house.

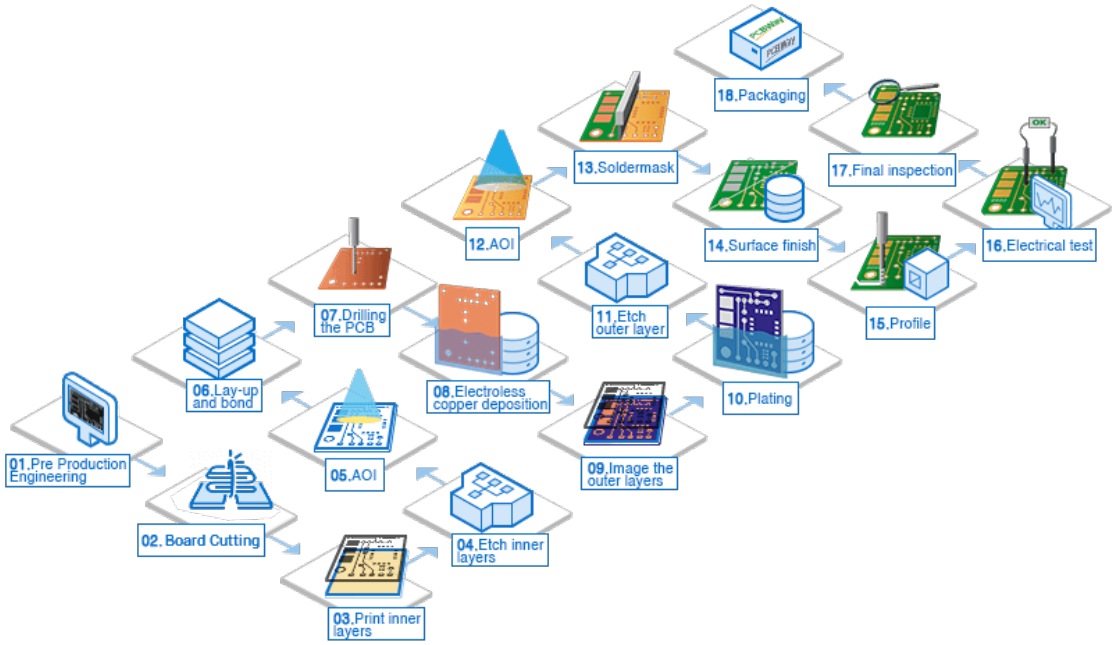


Figure 33: PCBWay pcb fabrication process.

When you send a board out to fabrication, you will typically upload Gerber or OBD++ files. Some places will also accept the raw PCB layout file though you typically don't want to do that because it is a design file and you only want to release the least amount of IP sensitive files possible. If you are doing any work that is ITAR or military related you will likely not be allowed to have your boards made by an overseas fab.

I also recommend you order a stainless steel stencil. This is usually fairly cheap (low tens of dollars) and is a way for you to assemble your PCB more quickly if you need to by hand. If you order through PCBWay they have options for a framed one which is meant for a machine that can help tension the stencil over your board. Personally I've only ordered the unframed ones as they are smaller and a piece of tape is usually good enough for most boards to hold it down. If you have a very complex design with dense small pads like on a BGA package, then you might want to opt for a framed one, but at that point I wouldn't attempt to assemble it yourself.

You will submit a request for a quote with some parameters set so they know how you want to manufacture your boards. I'm going to use PCBWay's quote process as an example below, but most of the online fabs will have a similar system. Here are some of the parameters you'll have to enter in:

- **Size**: These are the dimensions of your board. Make sure you check the units.

- **Quantity**: How many are you going to order? I think for the super cheap $5 option you can do up to 10. I usually just increase the number until the price starts going up.
- Layers: The number of layers in your board. The more layers, the longer and more expensive it's gonna get.
- **Material**: Most of the time it's going to be standard FR4 unless you have a specific need to support high speed design or thermals.
- **Thickness**: How thick do you need the PCB to be. 1.8mm is a standard thickness though I will usually go 0.8mm for something that's a little more compact. You may want a thicker board if you are using it for mechanical strength or a thinner board if you are size constrained.
- **Min Track/Space**: This is high thin of a trace and how far apart are those traces on your design. If you have super thin traces and they are super close together, you're going to have to pay more because it's harder to fabricate to those tolerances.
- **Min Hole Size**: The smallest sized hole in your design. If the vias get too small, they cannot physically drill those and need to start using lasers and you're going to pay more.
- **Soldermask**: This is the color you will make the PCB. Green is usually the default color, some of the more exotic ones will cost more. My personal preference is matte black, when someone else is paying for it.
- **Silkscreen**: The color of the text and labels printed on the board on top of the soldermask. If you choose a dark soldermask, you'll want a light silkscreen so there is contrast.
- **Surface Finish**: HASL is the default and cheapest. They basically just coat the copper pads with a layer of solder so it'll look silver. This process is not exact and you will have no guarantee of even pads. If you require RoHS, then choose the lead free option. RoHS is the European Union regulation on Restriction of Hazardous Substances and basically restricts the use of lead and other hazardous materials in electronics. The majority of electronics these days use lead free solder. ENIG is gold plating of the copper with a very thin layer of gold atoms. This is necessary if you have very large BGA parts as you need extremely level surfaces for the reflow to be successful. You should choose some finish because copper on it's own will oxidize.
- **Via process**: This is how you want the vias to be handled. I usually just tent the vias which means they are hidden under the soldermask. They would otherwise be exposed which can be helpful for debugging but look bad.
- **Finished Copper**: The "weight" or thickness of the copper used. 1oz is usually fine for low power electronics but if you have traces that carry high power you may want to use thicker copper to reduce impedance. If you have controlled impedance traces you will need to make sure you got your stackup from a simulation which should tell you the copper weight.
- **Remove Product No**: Most services give you the option to not have the fab's numbers stamped onto your design for a nominal fee. It's usually worth the two bucks to not have some weird logo on your design.
- **Stencil**: Stainless steel stencils are pretty cheap. If you have the extra $10 and can wait a few extra days it can be good to have around even if you do plan on soldering by hand.

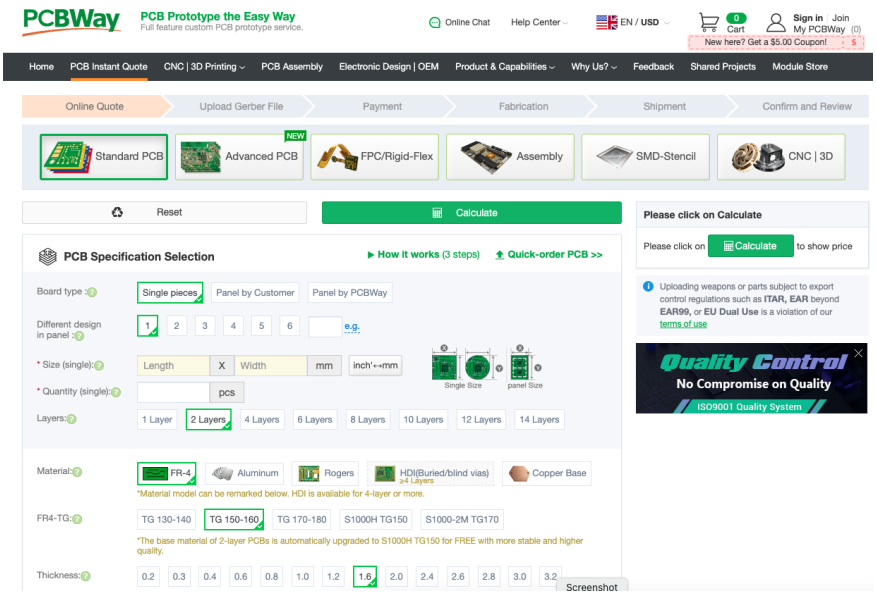


Figure 34: PCBWay instant quote page.

12.4 Assembly

PCB Assembly can be tedious and when you have hundreds or thousands of components, it is going to be inconvenient to do by hand. Luckily there are companies that do pcb assembly for you.

If you are in the Seattle region, I recommend going to Printed Circuits Assembly (PCA) in Bellevue. I have done hundreds of designs with them and they are a reliable local option, though not the cheapest. For best pricing of a professional service, a lot of the PCB fabs in China will also do assembly for you. The downside is you need to ship them the parts, which might cause delays with customs going into China. In the past I have heard of issues where less than genuine parts get sourced, but I personally have never used those assembly services so can't say for sure. There are similarly "turn key" vendors who will take your design and do all the fabrication and assembly steps for you, returning a fully assembled device, but you will be paying for those services.

That being said, if you are limited by your funds or time you will often need to solder the PCB yourself. To do this more quickly you can use a stencil. A stencil is just a thin piece of usually steel that has been etched with holes where the pads of your layout usually are. This way you can line it up with your raw pcb and spread a thin layer of solder paste onto it. You will then place your parts on the board by hand and then reflow it on a hotplate or oven to melt the paste to form a solid connection.

Figure 35: Hakko FX-951 Soldering Iron

Soldering is a skill but there are a couple things you can do if you are soldering by hand with a soldering iron.

1. First, you should put your part down on the pad lined up. Depending on the part you should add the flux to begin with.
2. You should solder down one pin even if messily to lock the part down to the footprint.
3. Add your flux if you didn't do so at the beginning, then begin to solder down the rest of the pads. Use the appropriate solder tip. For pins that have high thermal capacity like GND you will want a larger surface area tip like a chisel tip which is my personal preference for general soldering.
4. Go inspect the soldering with a microscope if you can to check for solder bridges which you can clear with a solder braid. Or you may need to touch up areas with extra solder and flux.
5. To finish up, you should clean up the soldering by taking a short haired brush, and scrubbing the area with isopropyl alcohol and a lint free wipe to clean off the extra flux.

You should be familiar with some of the other kinds of soldering techniques even if you don't do them yourself.

- **Wave Soldering**: This is done more commonly with through hole circuit boards where all the components are on one side. You essentially dip one side of the
- **Bar Soldering**: A lot of designs need to connect flat flex cables between multiple boards. Think something like a wireless mouse where the inherent geometry requires boards be mounted in various locations which might have a few signals and power running between them. To save on parts cost what a lot of designers do is put a footprint on a board that a flex cable can solder directly onto. You essentially use a solder bar to solder all pads of the cable onto a footprint on the pcb.
- **Wire Bonding**: Wire bonding isn't really soldering but you are placing a raw die onto your pcb and attaching the electrical connections using very thin wires. This requires a special wire bonding machine or an assembly house equipped to do so. The raw die will have pads that are fractions of a mm wide, and the bond is basically a cold welded joint that is then attached to a larger pad on your pcb. After bonding the die, you typically protect it with some sort of epoxy compound. This is basically what goes on inside a part you might buy off digikey. The bonding is done to a small board that then has the larger pads that you are familiar with in standard footprints. If you ever dissolve the plastic on one of these ICs you buy you will see

something similar. The reason you might want to wirebond is to reduce size, or for cost as you are not paying for the packaging of the device. If you see a blob of epoxy on pcbs of cheap electronics, it is likely there is a wire bonded part under it.

12.5 Bring Up & Debugging

Once you have a newly assembled design in hand you will need to bring up your design. You may find it helpful to formalize your bring up process until you have more experience and know what is more important to watch out for in your specific design. Here are some basic steps you can follow to get started:

Figure 36: Debugging can be hard.

1. Typically what you want to start on is making sure your power systems are brought up correctly. I would recommend that a first power up be done using a bench power supply with current limits set so that if your board has a short or design flaw somewhere that the inrush of current does not destroy your prototype.
2. The exact set of steps is project dependent, but I will typically start with testing the power supplies next. This means checking that they are all up to the correct voltages. If there is one that seems to be much lower than spec it could mean it's drawing too much current due to a short or you configured it wrong. For more complex designs you may also need to check that the supplies all come up at the appropriate times, for example some rails on FPGAs cannot come up until others have or it could damage it.
3. Once power is good, I will move on to the main processor unit. You should have some basic code written up already to perform tests on your subsystems. A good way to ensure fast bring up of your systems is to have some testbench code to connect to your peripherals and print statuses on a debug terminal or light up a debug LED. You may also need to burn a bootloader image to your microcontroller the first time in order for you to be able to load code onto it over USB or Over the Air (OTA).
4. If you are able to program your processor, the next step is to ensure the peripherals communicate properly and are sending back data that you expect. For example, a barometer sensor should registers around 1 atms of pressure at sea level or thereabouts.
5. Once you are confident the sensors are working you can start testing out your systems level code. It is often at this step that you find bugs in your sensors or peripherals that you may not have found with the first cursory test. Edge cases can happen when say a system first boots up, if there is a spike and one of your sensors registers something wrong, it could cause something in your code base to trigger when it shouldn't. If you are using push buttons and you didn't implement some sort of hardware smoothing or software debounce, you might see phantom clicks.
6. At this point your bring up should be more or less complete and you move on to testing and debugging of your actual system. You'll likely have found some bugs that might require some

rework or additional wiring. You might need to "dead bug" components onto your design which means freeform wiring it onto your PCB. Once you have a critical list of changes, you can integrate those into the next version of your design.

13 Mechanical & Industrial Design

If you are reading this document you are likely a electrical engineering (maybe a computer science) student. However, it can be extremely useful to have a good sense of mechanical design when designing your systems. I am fortunate in that I did my undergrad in both EE and ME so these principles have stayed with me throughout my career. Most electronics will be designed into some sort of mechanical structure. We as consumers rarely interface directly with a circuit board in a polished system. During the design process you will work closely with the mechanical engineers and industrial designers who will help to define the physical limitations that you will have to fit your circuitry into. Here are some of the concepts that you might find important.

13.1 Hardware Integration

Properly integrating your PCB into your mechanical design is fairly crucial for embedded systems. Many embedded systems have to interact with the physical world. Whether it's a camera that needs an optical stack, or just a circuit board that needs to be mounted in a housing, you will need to interact with some mechanical team. You may even have electro-mechanical systems like haptics feedback systems that require you to integrate electronics directly with a mechanism. Most hardware work pipelines will utilize some sort of software that allows you to visualizse the mechanical and electronics together in a 3D space. Make sure you take advantage of these to ensure things fit together with additional tolerances. I personally use Altium and Solidworks which work very well together.

13.2 Industrial Design

Figure 37: The Dyson Hair Dryer is a great example of a beautiful industrial design paired with excellent engineering. Wish it was within my budget while I still have some hair on my head.

Often many of the electronics will be constrained by how the industrial designers want to make the device look. As consumers we are always drawn to things that look cool and sleek which is why a good industrial design is so important. As you gain experience you will know what is possible and what is not so you can best inform those making these designs how their choices affect the electronics systems and add or subtract from final complexity and costs.

13.3 Thermal Considerations

Electronics generate heat, that is unavoidable. Most of the time, your low power embedded designs will not require extensive thermal considerations but for more complex systems that have higher compute or power requirements, thermal desgin becomes critical. For example, a VR headset uses multiple sensors to drive a HD image to your face. That uses a ton of power and if you don't dissipate it properly, your PCB may start to get uncomfortably hot so close to your eyes. You will want to work with a mechanical engineer on ways of dissipating heat, you will need to provide estimates of power loss from parts so they can plug it into their simulations. A thermal camera will also help in this situation to help isolate regions of high heat. There are a lot of ways to siphon heat away from your system. A heatsink is usually the simplest thing to do but you can also use things like a fan, peltier coolers, and even vapor chambers to help with heat dissipation. Remember that electronics operate within a temperature range and too hot or cold can cause devices to behave in unpredictable ways.

13.4 3D Printing

When I was an undergrad 3D printing was just starting to become popular in the consumer space. These days there are a myriad of offerings at reasonable prices and stellar performance. There are also hundreds of different colors and materials to choose from. 3D printing is a valuable tool when doing your prototypes because it allows you to iterate very quickly. However it's not as likely you will use it for the final product because it's hard to scale and you don't have as good a price point. You need to be careful as some 3D printed geometries are not possible to injection mold. It will however allow you to test things like mechanical structures, fit, and overall feel of a prototype before you commit the money and time to a design.

Figure 38: The Bambu Lab X1C. One of the modern gold standard prosumer 3D printers.

14 Economies of Scale

You will likely not encounter anything that requires you to scale while you are a student, but it is something that you will learn is critical in industry. In fact, a lot of companies are dominant in their fields because of the fact that they can scale and do it better and cheaper than others. Amazon is one of the largest retailers in the world because they are able to reach millions of users. Many companies spend millions hiring the best engineers who can optimize algorithms that take less compute cycles and server space to run inference on modern ML models. In this section we'll go over some of the things you will want to consider when you are building an embedded device that you expect to ship in the thousands or more.

When you first design your device you should have some idea of how many units you will end up shipping. This is something that the marketing team will have figured out early in the project and you should have this number in your Product Requirements Document along with targets for pricing.

14.1 Electronics

As I mentioned in previous sections, once you start ordering thousands or millions of units of a component, the price goes way down. Scaling electronics usually involves going to China for fabrication where the most advanced techniques are deployed for fabrication and at low costs and ordering large quanities of parts directly for manufacturers.

14.2 Mechanical Parts

Mechanical parts tend to be very costly in the prototyping stages. If you get a part machined at a prototyping shop like ProtoLabs, it could cost you thousands of dollars depending on the complexity. Once you have a design that needs thousands of units though, what you typically do is machine a injection mold or a negative of your part and then use that to pop out parts for pennies worth of plastic. The mold itself may be somewhat costly, in the thousands of dollars, but the parts themselves will usually cost just a little more than what the plastic you are using costs.

An injection molding machine works by melting down plastic pellets and then injecting it into that mold at high pressure. When you open the two halves of the mold, the part pops out ready for finishing. As an embedded or electrical engineer you likely will never need to design a mold, but it can be useful to understand the limitations. For example, injection molding is limited to certain geometries because you have to physically be able to inject plastics and super thin parts may require a lot more pressure.

14.3 Unboxing Experience

A lot of modern electronics have to work out of the box if they are run on batteries. That means that your system needs to retain enough battery while on the shelf that the user can power it up when they finally buy it. If your device is battery powered or there is a physical power switch that makes things more simple. If your device has a built in battery and no way to physically disconnect, you will have to design in a deep sleep mode that prevents the battery from draining completely.

In the past, companies have included accessories like charging cables, wall adapters, ear phones, etc, but have slowly reduced those. It is up to you how much you want to provide as far as accessories and what makes sense financially especially if you are a smaller company. Larger companies already have scale so their customers will likely still buy a product if they have to buy their own charger. A smaller niche company may benefit from having accessories and nice packaging as they build up their loyal customer base.

15 Case Studies

As I mentioned before, when you're in school they do a decent job of teaching you the fundamentals, but don't really get into the complex systems. It can be really helpful to have an example of how all these things go together. Below I've outlined how some common systems could be designed and some of the considerations that go into it. This section may also be useful for those interviewing as it is a common interview question to ask how someone would design an arbitrary system. These designs aren't going to be comprehensive, but will give you a general sense of what might go into designing something like this. I'm only focusing on the electrical hardware components and not so much the firmware or mechanics which would be critical for overall system design as well.

What can be very helpful in understanding a system is looking at tear downs online. You will find many such tear downs ranging from basic ones to very in depth technical analysis. Learning about these systems from a reverse engineering perspective can help you better understand how complex systems work as you see how others have done it. There are even companies that do this analysis for a fee and is very common in industry. You will pay these companies to basically tear down a device like a competitor's phone to see what components they used and reverse engineer the cost of making the device so you can decide what constraints on the BOM you need to be competitive. They will even cut through components like the PCB to analyze how many layers it is, and perform things like an X-ray to see layout.

There are many free teardowns such as those from iFixit that have high resolution images you can look at. They aren't super technical but often will give you a good idea of parts used. If you need actual analysis, companies like Tech Insights will offer subscriptions for reports of tech teardowns that include much more in depth analysis and reverse engineering of products.

YouTube is a fantastic place for teardowns. Channels like EEVBlog, The Signal Path, Big Clive will open up electronics and attempt to reverse engineer and explain how things work from a technical perspective.

16 Case Study: Bird House Monitoring System

Let's start with a fun one and design a birdhouse that has a camera that video streams over your network and can take some time lapses when you aren't monitoring. We're going to target bird watching enthusiasts, perhaps retired individuals or nature lovers who want to hang a bird house in their yard and can watch what's going on inside the house remotely. Also you should check out the Big Bear Bald Eagle Live Webcam.

16.1 Product Requirements

- 1080p at 30fps live image stream of inside the birdhouse.
- Wireless streaming capability over a network.
- Time lapse photo capture capability up to 1 min resolution.
- Waterproof enclosure and works from 0C - 25C.
- Must weigh less than 2lbs.
- Cost less than $100 to manufacture.
- Battery powered.

16.2 Design

When you first start a design, it helps to start drawing a block diagram to help organize the design. Let's think about what we need to design something like this.

As a baseline, we know we're going to need some sort of System on Chip that supports WiFi, probably supports MIPI because most cameras use the MIPI protocol. This narrows things down a lot so we might want to start looking at chips like the ESP32-P4 which supports both those things. We'll also need some other peripherals like a real time clock (RTC) since we probably want to be able to trigger the photo capture at once a minute or longer. This will require the micro to support something like I2C or SPI which is pretty common. We also need to be able to measure the battery voltage of the battery so we know if the charge is too low so we'll need an analog input.

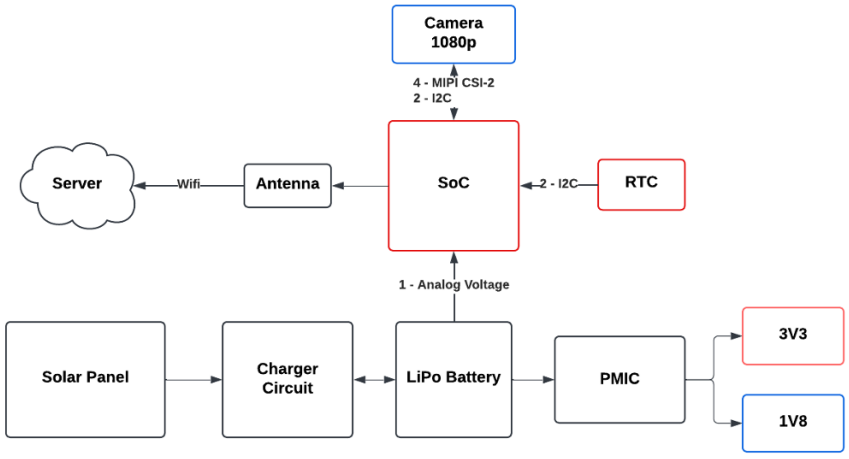


Figure 39: Bird house architecture block diagram.

Now that we have some basic requirements down on paper, let's do some systems design considerations. This is going to be an outdoor system presumably, or at the minimum mounted somewhere on your porch. This means the system will need to be waterproof. If we think about this, we can probably put everything onto one circuit board which will make waterproofing this easier. It can

all fit inside one compartment that we seal. The rest of the enclosure will be up to the mechanical engineering to make usable for the birds and yourself for maintenance. This might mean a removable roof, easy mounting points for things like a hook, and lightweight enough that you can hang it somewhere. We probably want materials that look good, but don't break the bank which probably means some sort of injection molded plastic, but with a pattern that looks inviting to birds. We'll also need a mounting point on the outside for the solar panel, and a way to wire it into the waterproofed area.

Alright, let's look at the block diagram. Remember that the first iteration doesn't need to have all of the specifics and is more of a way to get your architecture down and see how the system fits together. We'll start with a SoC in the middle. We know we'll need an antenna that allows us to do WiFi, though to save on design time we could buy a SoC Module that already has that built in and is already FCC certified which will save a lot of time getting our device to market but at a higher BOM cost. Next, we'll need an RTC so we can do timelapse photos without needing to connect to the web for power saving. That will likely be connected to I2C which will take up two pins. The camera will likely be MIPI based. Since we don't need that high frame rate, one channel of MIPI should be sufficient as it supports 2.5Gb/s per lane. This means 4 pins, 2 for the data, 2 for the clock.

With a system like this, you want to focus on the things that your customers will care about. The device needs to have nice images so you should choose a high quality camera. If you're taking photos in the dark you may want to design some sort of lighting inside the house. The design should be rugged, and be able to run off the solar charging which means you will do a lot of low power design. A nice website and interface will also go a long way to package your system up into something that feels high quality.

17 Case Study: Virtual Reality System

VR/AR systems have had a lot of hype the last few years. Let's look at what a simple VR design would look like. Let's say this is a VR system targeted for people who don't need something as precise as a HTC Vive, but better than using your phone as a VR display. Maybe someone who wants to play home videos that they've captured on a 360 Camera.

17.1 Requirements

- Stand alone, wireless, 2 hour battery life.
- 90Hz 1440p Display.
- 6 Degrees of Freedom Tracking.
- Non-Tracked controller with joystick and button input.
- Eye Tracking.

17.2 Design

This one is pretty complex even if you're going for a lower featured device. Let's start off with considering what a VR system is. A system like this has a primary goal of rendering a virtual scene to a user that follows their head movement in order to create a sense of immersion for the user. In order to do this, it has to utilize sensors such as an IMU, and cameras to perform head tracking in real time and send that information to a GPU that can render a scene with those movements to displays in front of a user's eye. We also want to have some eye tracking cameras to detect where the user is looking on scene and a way to measure adjustment of the intra-pupilary distance since everyone's eyes are a different distance apart. A system like this is going to need some sort of controller, in our case lets just make it very simple with a non-tracked controller that lets you select things manually. Ok, now that we have some idea of what the system is, we can start to do some engineering thought behind subsystem design and selection.

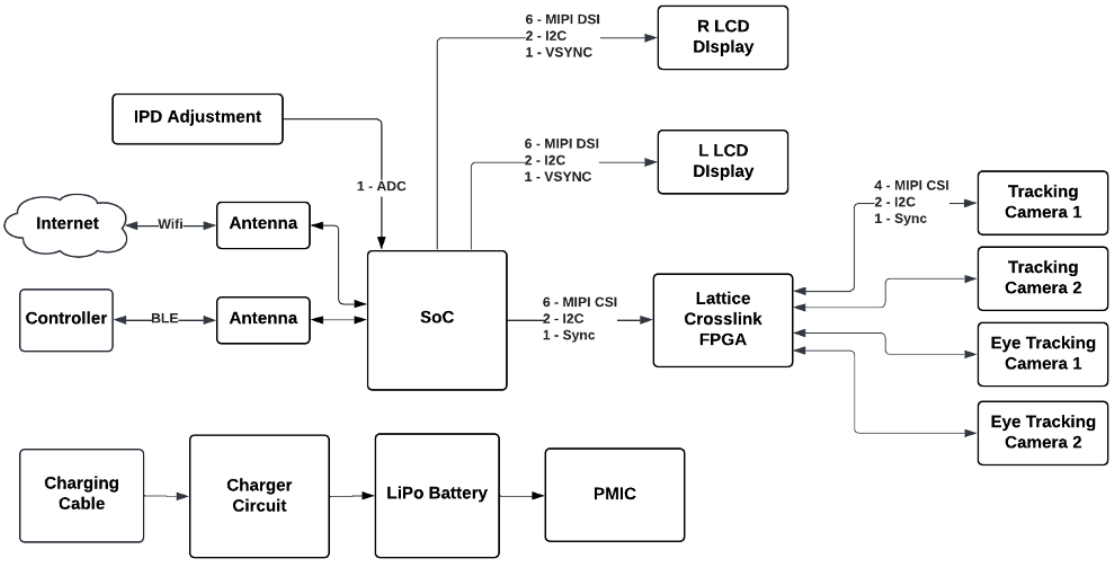


Figure 40: VR architecture block diagram.

Let's look at the block diagram. We'll need a SoC, and a pretty powerful one that can run an operating system since we'll have to display HD content. It will also need to support BLE and WiFi. Not many devices out there can do all of that, and you are probably looking at ones that

power our modern phones like Qualcomm's Snapdragon XR chipset which was designed specifically for VR/AR/XR. The displays we will probably need to run off MIPI DSI and with our bandwidth likely needs at least 2 channels of data along with the clock. We also need I2C to do controls like brightness and a sync pin to ensure both L/R displays update together. We'll have 4 cameras, 2 for updating the head tracking, and 2 to track the eyes. That's a lot of MIPI CSI lanes so we probably want something like a Lattice Crosslink FPGA which can aggregate multiple MIPI streams into one, acting like a serializer in a SerDes. The SoC can then interpret the larger image in the relevant sections. Of course we'll need I2C and sync lines as well.

The power architecture for something like this is going to be complex so I didn't try to detail it all out here. A SoC of this caliber will have very specific power domains and bring up so you would need to follow the recommendations of the the datasheet. You also have power requirements for the FPGA, cameras, and displays all of which have high power requirements. This could mean each have dedicated regulators and decoupling caps to prevent current spikes from affecting other systems.

I didn't make a block diagram of the controller, but it would take a bit of engineering on it's own. You need a BLE enabled microcontroller and the ability to send joystick and button signals from the user's interaction quickly. You don't want a user to click a button and need to wait a second for it to register. This means the data stream needs to be performant, but also done in a way that conserves power which means you might need to add an IMU or grip detection in there that turns things off when the user is not actively using the controller.

VR systems are really complicated and for any complex system you want to make sure all of the team is onboard and communicating well. For example, all of these chips will generate a massive amount of heat that sits very close to your eyes. You will probably need to do some thermal simulations with your mechanical team to see how to design in active cooling and where to place temperature sensors. Your eye tracking systems might be affected by how the optical team needs to make the lenses in order to get the best quality image. And of course the operating systems and firmware teams need to make sure you have chosen a sufficiently powerful enough processor to generate the content needed for the user experience. You are going to be the individual responsible for your deliverable but you have to make sure it integrates into everybody else's parts for the design to be successful.

And yes, I got some of my arrow directions missing, but I'm too lazy to fix the diagram and reload the image.

18 Case Study: Fitness Tracker Watch

Let's look at how you might design a watch that tracks fitness and wellness like what you might find in a Garmin, Fitbit, or Apple smart watch.

18.1 Requirements

- 3 day battery life.
- Water and dust resistant.
- Heart Rate, SpO2 sensing, Step Counting.
- Basic menu system.
- Ability to sync data to the phone.

A smart watch might seem like a simple design, but there is actually quite a bit to consider. Let's first think about all of the devices we'll need. There is an accelerometer to do step counting, a biometrics sensor like the MAX30102 for doing heart rate and SpO2 sensing, a touch screen enabled display for interaction, and a bluetooth enabled microcontroller. Let's take a look at what the block diagram for that might look like.

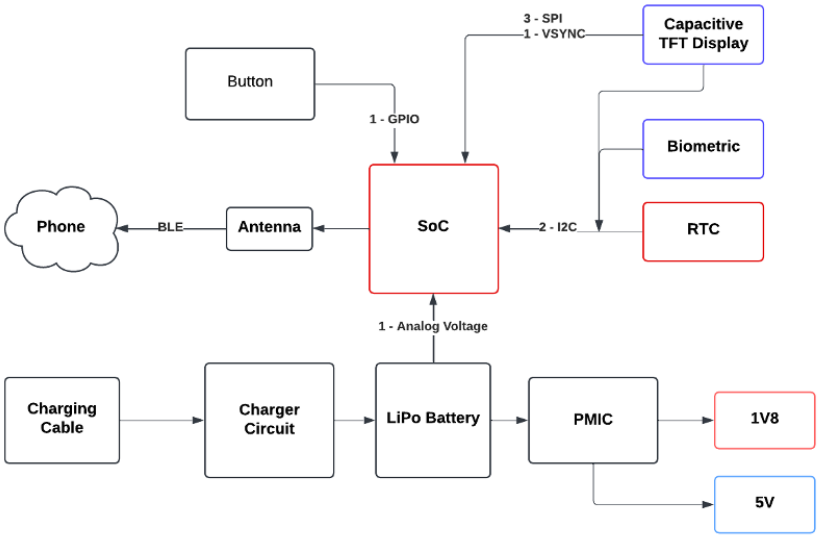


Figure 41: Fitness watch architecture block diagram.

We're going to need another SoC that has BLE capabilities and since this is a wearable, we probably want low power. The NRF52 line from Nordic Semiconductor would be a good choice for this. The display will need to connect to the microcontroller over SPI most likely. One thing to note is that a lot of these smaller displays will have a dedicated controller chip as part of the display module that actually drives it, and that is what you interface with over SPI. You might also have a pin to sync the display's update.Since you are controlling the display using touch, you'll need to interface with a capacitive touch controller which is likely also built into the module from the manufacturer which you'll interface with over I2C. We'll need a biometric sensor of some sort like the MAX30102 which would communicate of I2C (SPI would also probably work). And of course we'll need a real time clock of some sort over I2C to keep time, it is a watch after all. A button would be nice which we can put on a GPIO pin and knowing how much battery is left is useful so we'll put that on an ADC capable pin.

The power system would mainly be a charger circuit for the lipo battery and then a PMIC. Let's just assume most of the logic runs off 1.8V, but for things that need to emit light like the display and

the LEDs for the biometrics sensing, you'll need higher voltage like 5V. Because most LiPo batteries are 3.7V nominal, you would need to find a PMIC that can do buck and boost.

As you progress in your engineering, the block diagram will change and you will create more detailed documents. You'll also make design changes as you run experiments in your prototyping phase. Maybe you find that you cannot fit all of the circuitry into the form factor you want and you need to choose a more compact part. Or you might find that the biometrics data you collect doesn't fit on the onboard memory of the microcontroller you chose before the data is synced over BLE and you need to add a small amount of additional SPI memory as a buffer.

19 Ethics

Let's talk a little bit about ethics. You don't typically take an ethics course as undergrads, but engineers in any discipline have a responsibility to design and do things ethically. To philosophy students that could be the topic of a thesis. To the rest of us generally speaking it means don't do shitty things when you're being an engineer. Don't cut corners when it comes to safety, don't cheat, and don't try to hide malicious code, etc. That sort of thing.

Figure 42: https://xkcd.com/1390/

While you are a student you should abide by the honor code. Do not copy things off the internet without citation and pass it off as your own work. You may or may not get caught, but you do deprive yourself of learning. Your grades will have very little bearing on your future, it's the skills you learn and character you build in your education that will carry you in your career. Even for applying to graduate school your projects and letters of recommendation from your professors will carry far more weight than your GPA.

Engineering is a wide field and almost everything we interact with these days had an engineer work on it at some point. Whether it is an embedded system or your favorite insulated water bottle, someone had to design it or design the thing that made it. The point is, if engineers did not have some code of ethics society would be much worse. What happens when you don't do things ethically? I'm sure you've heard of all the alleged (don't sue me) issues with some of Boeing's planes in 2024.

Let's narrow it down a bit to what ethical choices an embedded hardware engineer might run into. Some of the more important topics you'll encounter will be focused around safety, privacy, and product life. Safety is obviously important for anything, but when you deal with electronics especially those that are plugged into mains you need to be absolutely certain that the user cannot get electrocuted. This means you use a high quality AC to DC converter, and don't buy cheap parts that can fail and cause harm. You design systems that do not infringe on the privacy of the user. This means any device that holds sensitive identifiable information like a person's name or date of birth is encrypted. If you're using a camera, you make sure the user knows it's on or you make sure that data doesn't make it onto the cloud. You don't design things in a way that easily breaks to make more profit or ends up being toxic when it ends up in a landfill.

There is a balance between making the best and worse design choices, and it's up to you to decide what that balance is. The best privacy is no camera, but a lot of things can't work without one so you need to find what most people will find acceptable. Doing the right thing can and usually is costly, but it can be even more damaging in the long run if your reputation is tainted. Keep in mind many managers and business executives do not have engineering backgrounds, and treat many of these problems as economics. It is your job to ensure they know the reasoning behind design choices.

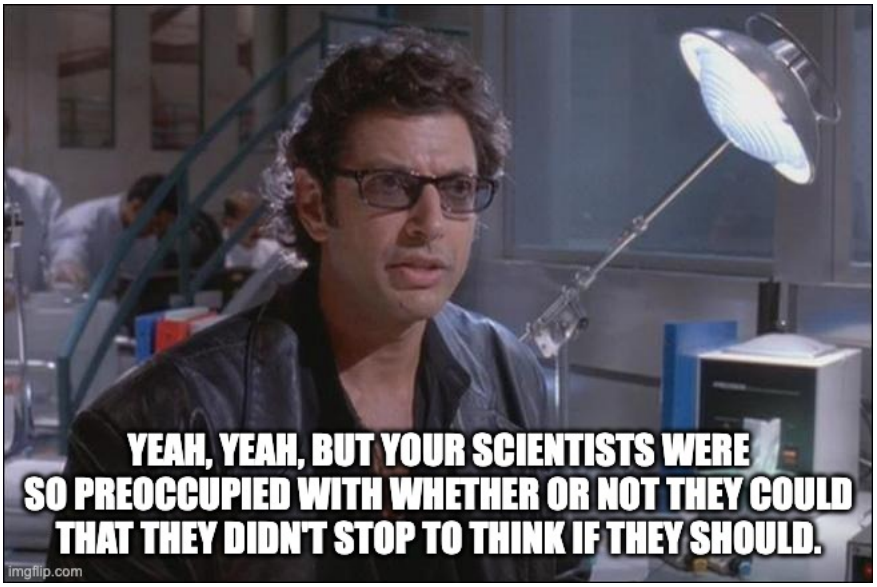


Figure 43: Quote from Ian Malcolm, Mathematician

When research is done, you typically need to go through an Institutional Review Board (IRB) to review your project for ethical and safety issues. They could go by a different name in your organization or company, maybe Ethical Review Board (ERB), or Independent Ethics Committee (IEC). You can read more about the formality of such a committee on Wikipedia but they are essentially a formal group that looks over projects to determine if they should be done. This is especially relevant if you need to test anything out on people or animals. This not only protects your participants by you as the researcher and formalizes the process of making sure whatever you want to do won't cause any harm to anyone and ensures consent is given at all stages of an experiment or test. This process is found throughout academia and research labs in companies and is something you will want to familiarize yourself with in general as you will likely encounter it at some point even if you are not the one performing the research. Based on Ian's quote and what happened at Jurassic Park, InGen probably didn't go through as rigorous an IRB process as they could have before they decided to clone dinosaurs and open a theme park.

There's also the philosophical discussion of designing weapons which I'm not gonna get into here. Personally I decided not to work on weapon systems and I'll leave it at that.

Usually what pulls companies away from doing the right thing is money or fear. A lot of companies start out doing the right thing with mottos like "Don't be evil", then they grow big and start making billions of dollars and the ethical scales get a little more fuzzy. Or they need to juice the earnings before a shareholder meeting so that the executives can reach goals for bonuses and an easy way to do that is cutting corners or layoffs.

At the end of the day it's really up to you to decide what you're comfortable with against your own moral compass. Some people may be comfortable with being one or two levels removed from a controversial project or decision and others may not wish to remain at the company at all. If you work hard to join a prestigious company it can be hard to give up a job, especially if you are supporting a family. There isn't a right or wrong answer in these scenarios, just what you are personally comfortable with.

20 Some Career Advice

No matter how cool I hope you find embedded systems, at the end of the day you gotta eat. Engineering has historically been a stable and sometimes lucrative profession, but these days that's not a guarantee. I hope that the insights you've gained from this text will help some of you start a meaningful career that you enjoy. This section is a little different. Here I'm giving some of the non-technical things I've learned along the way that I think might be useful. I will caveat by saying everyone's career journey is going to be different so what I have here shouldn't be taken as an absolute.

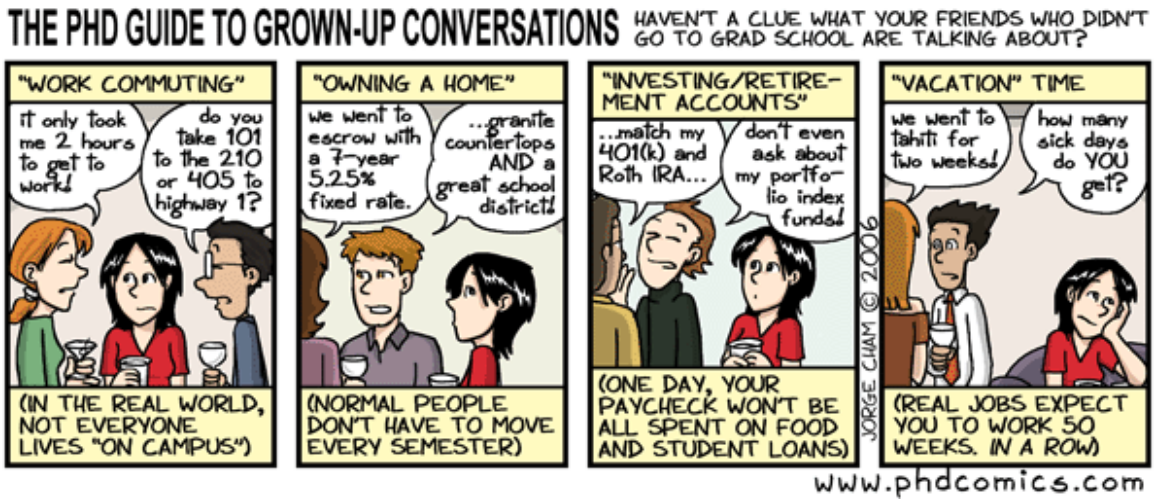


Figure 44: Life after school can be scary.

20.1 Finding a Job

Not gonna lie, when I'm writing this at the start of 2025, the job market is real tough even for senior engineers, and not looking like it's gonna get better any time soon. Folks these days are getting laid off left and right and it's saturating the market with senior engineers who are all competing with each other. Your advantage is you are cheap as a new grad and coming in with a strong background can get you in the door.

When I was an undergrad I applied to, in my recollection, like 500 jobs my senior year and got a grand total of none of them. It was day in day out of applications, going to those company presentations on campus, and career fairs. Eventually one of the jobs I applied to told me I wasn't senior enough but offered me a tour and lunch and that lead to an internship/contract job of sorts that helped kick start my career.

Other than applying, making use of your network can be critical. While referrals aren't always successful, it does help get you at least to the top of the pile of people applying. When there are 100 qualified people applying for the same job, every little bit helps since there is no such thing as a perfect candidate.

These days its a lot harder because the field of engineering has gotten very crowded and you are competing with a lot of very strong candidates who are in a similar spot. Many companies are also staffing overseas where the talent is often just as strong, but at a much lower cost. You will be competing for these jobs with your friends and colleagues from school. Don't be sad if you don't get a job. It could be because of a hundred different reasons and recruiters are horrible when it comes to telling you why it didn't work out. Luck is very real when it comes to securing a job.

AI and automation also contribute to a weaker job market. Just a few years ago, software engineers were in high demand, but these days LLMs have made coding basic programs much more accessible to even those with limited CS background. Luckily for embedded hardware engineers like you, these

AI programs haven't yet gotten to the point where they can design, build, and debug hardware systems and there is nowhere near as much parseable schematics and board designs online for them to train off of.

Some things to keep in mind when searching. There are definitely cycles for recruitment, especially for students. Early fall is when a lot of companies start recruiting for the batch of undergrads that will be graduating in the spring. Do not miss the job fairs or online postings for new grads or you may need to wait a whole year. You also want to be mindful of when a company starts their fiscal year as a lot of openings are dependent on budgets for the year. You may find a sudden increase in postings once a team knows how many new hires they can afford.

20.2 Grad School

Another option after your undergrad is to go to grad school either to finish off a Masters or start a PhD. A PhD is NOT something you want to do without serious thinking. It is a very long commitment and will likely not result in higher financial returns in the long run. If you just want any job in your field, a Masters will get you much further financially and career wise. It is very likely if you are entering industry post PhD in a non-research role, you will be reporting to someone who only has a BS or MS who climbed the ladder while you were still working on your thesis. What a PhD does enable is a path into academia and certain research roles in industry that requires this credential to enter.

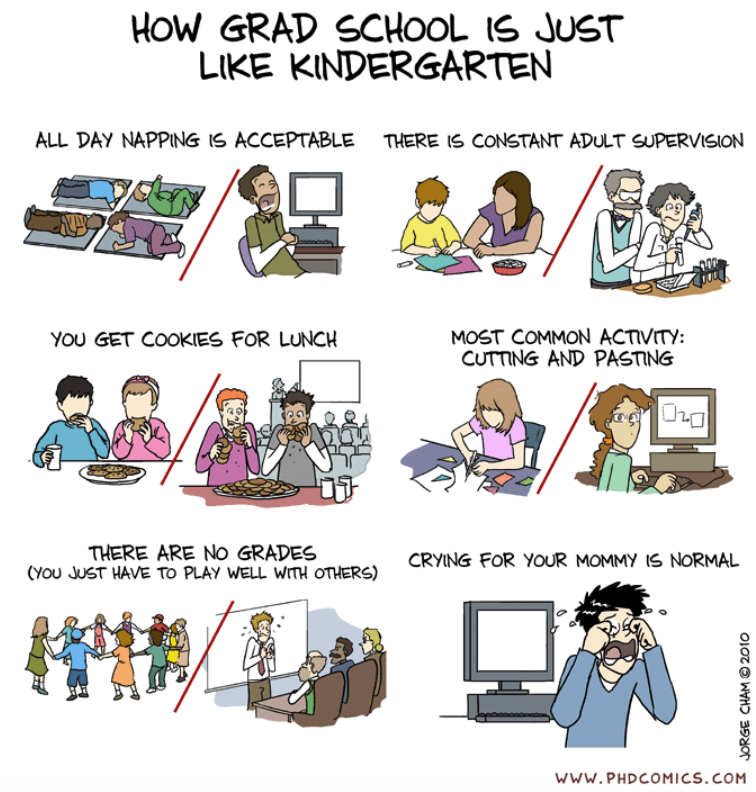


Figure 45: Can confirm. Now... where's my blankie.

You will learn how to do research and ideate which is a lot harder than you might think it is. There is a lot of process that some might find tedious. You will become very knowledgeable about a certain domain and pick up a lot of skills to enable work in that narrow slice. You will wear a lot of hats including ideas person, engineer, writer, marketing, coordinator, etc. Once you start publishing and attending conferences you will start to build a community and get a better sense of what the state of the art is in the field. This can be different from a corporate job where you tend to get a bit siloed

into the company's state of the art which may not be where the industry is heading. A lot of the time you will learn these skills informally to get your research done which is very different from the structured formality you might find in a corporate job where the structure is to enable efficiency and coordination within an organization. Someone who can code really well or make quick circuits in grad school may not directly translate well to a corporate org trying to create products that follow a very specific design process.

For me personally, I'm loving grad school. I get to explore tons of different topics from a blank slate with no concrete expectation of delivering anything other than a paper. I also have the free time to write guides like this. Having a good adviser in a well funded lab will offer far more freedom than any industry job will offer. I get to learn new things and work with some really smart people. I can say without a doubt that many of the PhD student's I've worked with at UW are among the most capable engineers and researchers I've come across.

Since this section is about careers, let's talk a bit about how grad school affects you financially. I worked for many years before coming back full time and was able to earn my Masters while still working a full time job and building my industry experience and savings. Most students go in straight into a program and it is much harder to build finances on a graduate student stipend especially if you do not have a fellowship. I highly recommend you start to applying to grants and fellowships as soon as you know you will be attending grad school. Many of these are reserved for early year students and you won't qualify to apply later. Unlike undergraduate scholarships which tend to skew more towards helping those who have financial need, graduate funding tends to be more oriented towards research potential and will fund those who are working on research projects that the organization is interested in. If you're going to join a STEM related graduate program, you should apply to NSF Fellowships.

Keep in mind most programs are not fully funded. The CSE department at UW is uniquely positioned to guarantee us funding in the form of an RA or TA grant that gives us health insurance and tuition waivers along with a monthly stipend. Many programs are not so lucky and you may have to take out loans which can set you back financially for decades if you don't get a good job after. For most public institutions, our salaries are posted online so you can see about how much you might expect to make somewhere. I'm not trying to scare you away from grad school, but it is something you have to factor in. I quit my job knowing that the next few years my net worth probably wasn't going to grow as much.

Now some nicer things about grad school.

- You get more freedom and control over your day to day since essentially you are the one paying the school, not the other way around like in a job. Your first years will still involve a lot of homework and classes along with research projects that are most likely assigned to you. That said, I don't have any useless update meetings, all of my meetings are called by me or to sync with people who I actively need input with. I also get to show up to the lab on my own schedule.
- You get to work with a lot of folks your own age. Companies tend to be more heavy in mid-level engineers who do most of the productive work so many of your colleagues may be in a more mature life stage than you right out of undergrad.
- There is a much larger sense of community here compared to a company where you are naturally divided up into orgs to focus on your work.
- Collaboration within the department and the university as a whole is encouraged and unrestricted. I am in the CSE department, but work frequently with MDs in the medical school, brainstormed with professors in the ECE department, had professors in the MechE department help me with math equations I didn't understand, and visit the shop master in the machine shop for ideas on fabrication. Lots of opportunities to dip your toes in any number of projects.
- There are also opportunities to pursue entrepreneurship under the blanket of a larger institution. If you work at a company they may restrict you from moonlighting in other jobs, in grad school there are resources for you to develop your entrepreneurial ideas more.

- If you like to teach like I do, mentoring students and teaching undergrads can be very rewarding, though now they will look to you as the expert so you'll have to pretend that you are.
- Unlike industry where the annual performance review can be stressful, our annual reviews are more meant to keep us on track to graduate. You do still need to make progress, but as long as you have a supportive adviser the process usually is pretty smooth.

I will say having a supportive adviser is by far the best indicator of a good graduate school experience. If you get an offer to join a program I would highly recommend you figure out if your adviser is going to be someone you think will be a good mentor to yourself. The support and guidance I get from my adviser is one of the primary reasons I decided to come back to academia full time.

At the end of the day, grad school is not for everyone and I really encourage you to do some thinking if you are serious about pursuing a PhD. I'd say if you are a student who wants just a bit more project experience before going out into the entry level market (and you can afford it) a MS is a great 1-2 year transition into industry. Otherwise good luck in your job search.

20.3 Entrepreneurship

A third option if you don't want to find a job, or go through more school is to start your own company. I've never started a company so good luck to you...

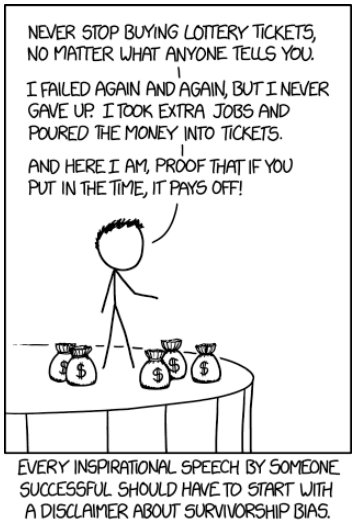


Figure 46: Most startups don't become super successful.

Closest I've come is working at a mid sized startup so I'm not much help. The only advice I can give is to just be careful of putting all your eggs in one bucket. If you go out and ask for advice, I think you'll find a whole range of opinions related to entrepreneurship that you can peruse. Just keep in mind a lot of those visionaries and founders were at the right spot at the right time. It doesn't mean their path is a valid manual for your success.

20.4 The Interview

If you want a job, you're going to need to go through an interview. This is a way for the company to gauge whether or not you are the right person to do the job and for your potential colleagues to see if you're someone they want to spend time around. Make sure you prepare for the interview. Even principal engineers need to prepare because most of us forget things and you don't want to get caught up on something simple.

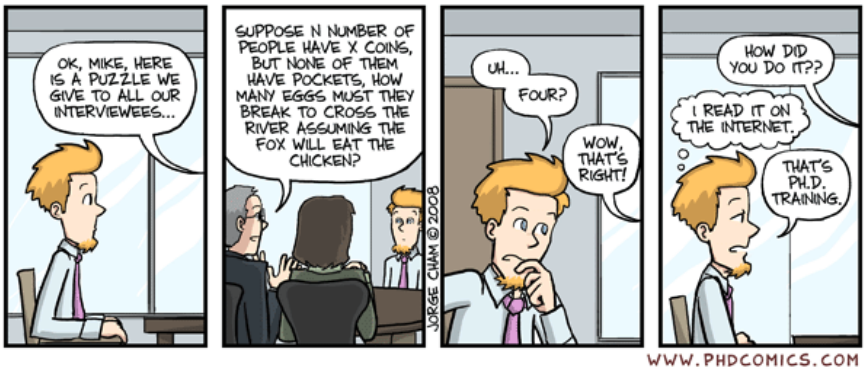


Figure 47: Don't be nervous.

When you are a new grad, you will most likely get asked questions about core competencies. For embedded and hardware jobs that usually means circuits, and firmware questions. Once you are more senior, it's less likely you will be hired based off of whiteboard questions and more likely you get hired based on your previous projects. Here are some things that might be helpful to review before an interview.

- **Core Basics**: You should have decent understanding of core electrical engineering concepts like voltage, current, etc. Engineering is not physics so you will probably not need to know Maxwell's equations, but you may be asked to analyze a circuit so best to understand how components work. Know things like what an ideal opamp is, how a BJT applies gain to a current, etc.
- **Circuits**: Review or memorize some common circuits. Simple buck/boost topologies, opamp circuits, filter circuits (high pass, low pass, band pass). It's best to understand how they work as most interviewers will delve deeper as they recognize that you know something. For example they may ask, what happens if you switch the positive and negative inputs of a opamp.
- **Systems**: You will often be asked to design systems. Look at block diagrams for systems related to the job you are applying to. If you're getting a job at Apple working on phones, you should know the common parts and basic topology of a phone architecture.
- **Company background**: Know what the company does so you can ask knowledgeable questions. It shows you have done your homework if you can show you understand the things the team is working towards.
- **Interpersonal Skills**: You will likely have one interview that focuses on behavioral issues. This could be asking you if you've ever had difficulty working with a team mate and how you responded to it. Or giving them an example of when you've failed and how you dealt with it. It can be very helpful for you to come up with examples of this ahead of time so you can answer right away.
- **Resume**: Make sure you understand and can speak to every part of your resume. If you exaggerated anything on there and they catch you, that's likely to end your chances right there. Don't say you are an expert in pcb design if you've only built one 2 layer board.

As a new grad you might be nervous during your interviews. That's totally normal, but you also want to remember that the interviewer is just another person like you. They may have a few more years experience than you, but they might be just as nervous. Be yourself. Be friendly. Remember that the interviewer may be representing the company's needs, but is also human and probably doesn't want to work with an asshole for 40 hours a week.

20.5 Salary Negotiations

Congratulations, you have an offer! Always try to negotiate your compensation and benefits. It can be hard to do when you are a new grad and are excited to just have an offer.

Do your homework on how much someone in that role makes. There are many places online for you to look like Glassdoor, Levels.fyi, Blind, Reddit, etc. Search for your role at that company and make sure the offer is in the ballpark. Once they have chosen to hire you, they are going to try to hire you for the least amount possible, while you are trying to get as much as possible. Leverage is key. Do not tell them what your salary expectations are first, they will choose the lowest numbers. Try to deflect so that they offer the number first, these days many companies are required to post salary ranges by law which you can use to your advantage. Having other offers will help your position as it means you do not have to take their offer. Once they have spent the time and energy to interview you and choose you as the most qualified candidate it is not in their best interest to lose you as it will mean projects get left in limbo, and engineers have to spend more time interviewing people. That's going to add up and probably worth an extra few thousand on it's own. You can highlight things you've done that make you valuable as a justification for the increase and it doesn't even have to be a lot. If a company has stock offerings, it is usually easier to increase those in an offer or ask for a signing bonus.

If they rescind an offer because you negotiated, it is likely the place has a very toxic culture. Negotiations are common, and all recruiters have had to negotiate, it is literally their job. It's also perfectly ok for them to say no, the budget is what it is and you'll have to decide whether or not that's ok with you. When you're first starting out, the money is honestly secondary to what you'll learn. Knowing what I know now I would still take that $20/hr job I started with because I learned skills that would last the rest of my career there. Once you start getting more senior, it will be more important to get paid what you are worth as you will be bringing more to the table. Every dollar less you end up getting is probably one dollar more someone higher up the food chain is getting in their annual bonus so don't feel bad fighting for your worth.

20.6 Impostor Syndrome & Your First Few Years

When you're new in the career field you might get intimidated by the managers or more senior engineers. Don't be, half of them have no idea what they are doing either, they've just gotten better at hiding it. Trust me, everyone looks stuff up online. The first few years of your career are for you to learn. Learn to ask questions when you don't understand something. It might not be during the speech the VP is making, but you can take a note and follow up with someone on your team later. There's a lot to learn, but here are some things to consider as you grow into your career:

- **Technical Knowledge**: For most people an undergraduate or master's degree is usually not sufficient technical training to become super skilled at something in any industry. You will find that the larger scales seen outside of academia in industry bring technical hurdles that you will not encounter in school. The projects you will do in school and even in a lot of research labs will be very different than the ones you do in a job. It is critical for you to learn new skills on the job and do so in a way that allows you to continue delivering.
- **Engineering**: Spend the time to learn your new company's systems and nuances of how they do engineering. While the core concepts you learn as an engineer are going to be common, the way you approach things will vary company to company.
- **Company Structure**: Apart from normal curiosity, it is helpful to know how your organization is set up and who the important players are. You want to know who on teams are your counterpart and who are the people you may need to reach out to collaborate on projects.
- **Office Politics**: An unavoidable part of corporate life. While it would be great if the world ran as a meritocracy, the reality is office politics plays a large role in your success at a company. People who are well liked or well connected will tend to have more opportunities or be better protected during layoffs. I personally try to avoid as much of this as possible, and just do my job but it will mean your career will likely develop a little slower which is ok.

- **Work Life Balance**: Incredibly important to maintain long term wellness. Do not think for a second that your employer cares about you as a person. As soon as you are a liability or do not produce more for the shareholders than you are paid, you will be let go. In recent years we have seen this with the waves of layoffs in the tech sector. Working a 9-9-6 isn't the badge you might think it is. Your health and relationships are far more important than your success at a job. It's great if you believe in the mission of your company or feel your role is making a difference in the world, but do not let yourself be defined by your job. People often take school for granted where you are encouraged to join student groups, explore hobbies, and make friends. Outside of the academic bubble, you need to actively maintain things outside work so you still have a life. Scary I know. Take your vacations, especially if they have an unlimited policy. Recharge when needed. If a team cannot survive without you there, there is something wrong in the organization and your team will likely be phased out.
- **Money**: I have a subsection below on this, but the first few years out of school you want to start learning about money so that you can have a financial successful future. You're making money now, not paying tuition so make sure you learn how to manage it. Don't suddenly become a big spender just because your income has increased. You would think this is obvious, but you'd be amazed how many people fall prey to lifestyle creep.

Don't feel you are an impostor just because you don't have as much experience as the others. Everyone feels that way at some point in their career, and typically they'll feel like the impostor a couple times as you make your way up. A new director is probably going to feel like an impostor at times if they go from managing a small team of 5 to suddenly being responsible for 5 teams of 10, just like how you go from managing homework to managing deliverables you are paid for on time.

When I first started, I remember asking a senior engineer if I could ask a stupid question and he responded that "there's no such thing." You should always feel free to ask clarifying questions without feeling embarrassed. It's always good to try to figure things out yourself, but you'll often find yourself still stuck after a solid effort. Good mentors will share their knowledge without ego or judgment. Anyone who responds otherwise is not worth asking.

You will also encounter impostor syndrome at every point of your career. When you don't know something and everyone around you does it can feel like you don't belong. But remember your colleagues probably have a lot more years under them and you are there most likely because they think you are good at what you do and have potential. The good mentors will be the ones who jump at the chance to share knowledge or help guide you to learn it yourself.

20.7 Networking

Networking is very important in engineering. You hear a lot that after the first job, you get a lot of subsequent jobs through your network and that is often true. I've personally gotten several jobs because someone I used to work with was an "in" into the company or could vouch for me. You will find that buying someone a cup of coffee to chat can pay off in the future.

There's no handbook on networking. It's easier if you are an extrovert as it's natural for you to engage with strangers. For those like me who are more introverted, you can use platforms like LinkedIn or go to networking events that may be more structured. You can also join organizations on campus or professional organizations where you can meet peers. Conferences are also a great way to network where you talk with peers. Talk with more senior students who may have internship experience. If you are lucky enough to get an internship, make sure you network within the company and talk with engineers there.

While you are in school, you should try to engage with others in your class and lab partners as you will all be entering the job market at roughly the same level. Help each other out with practicing for interviews and finding good opportunities. You are competing for the same jobs, but as new grads you will benefit a lot more from helping each other than going at it alone. Keep in mind that careers are long and you will likely run into your classmates in the future and they may be in positions to help hire you in the future.

20.8 Leaders vs Managers

You will have a lot of managers throughout your career. Managers have important roles in organizations by helping to form cohesive teams that can produce more than the sum of their parts. Successful managers help the team deal with blocks to their work like logistics and keep the team informed of major organizational changes so their jobs are not disrupted. Managers work with other teams to set up resources and necessary collaborations. They also deal with the formalities of your career growth like checking to make sure you grow your responsibilities over time to meet corporate expectations of the next seniority level. You need to work with your manager because for better or worse, they are going to be responsible for how your career progresses. A good manager can act as a mentor in your career's growth, while a bad one may be more focused on their own career than helping you with yours.

Managers are not necessarily leaders and it is important to make that distinction. Leaders in organizations are typically directors, and above who guide direction in your organization and inspire. They make the risky decisions for how they steer what the group works on and are more focused on corporate level issues than individuals. Many managers will rise up the ranks and stay within middle management where they can be content with managing engineers, but few will rise to leaders who the organization relies on for guidance. Leaders are typically more aware of where the industry is and the latest technology used in your application. Their insights will extend outside your company, you may even see them invited to talk in industry conferences. Manager guidance may or may not translate to other organizations if you ever leave that company.

I've seen leaders who aren't the best managers, but have vision that is very hard to see for those who do not have a deep technical knowledge and passion for the space. I've also had managers who have a lot of functional experience which is core to forming a cohesive team, but who I would not trust to do actual engineering themselves. Both roles are important to an organization and it is important for you to distinguish between the two as you learn how to best craft your own career. You should never blindly follow anyone, always interpret guidance within the context of your own career and goals.

20.9 Money & Investing

Figure 48: That compound interest is real.

Just the standard legal caveat of this not being financial advice. These are just some of my opinions having started as a dirt poor undergrad waiting for $5 Chicken Friday's at Safeway (btw it's now Monday's and not $5 anymore, damn inflation) to being "rich" enough to fill my Honda's gas tank to the top whenever I need, and getting the soft toilet paper at Costco. Haven't quite gotten to buy a house in Seattle rich yet.

Engineering won't make you rich the way a job on wall street will, but most of the time (in the USA anyway, non US engineering salaries are quite low by comparison) you'll be compensated above average with decent benefits and stock. If you end up in a big tech company your total compensation

could reach into the millions when you are more principal. While it's tempting to start spending your hard earned cash, it's always a good idea to sock some of it away. This is not something they teach you in school (maybe if you're an Econ major) but will have ramifications for many years.

It's kind of a tangent to the rest of this text but we've had conversations about this in the lab and I've noticed a lot of students aren't taught these things. I didn't even start putting money into a Roth IRA until I was in my late 20s which I look back on regretfully. A good paying job is not a guarantee for your whole life. You could get laid off, get sick, have family, etc which can very quickly eat up your savings. It's always good to have a safety net and something that continues to grow in the background for your future.

Do some reading/ YouTubing into these things:

- **Roth IRA**: A retirement account that you can put in a few thousand into ($7k right now) every year. You can't take money out until you're 60, but it grows tax free and you can invest the money into stocks tax free. You will also pay no taxes when you take the money out. If you can, max this out every year.
- **401k/403b**: Offered by your employer, you can deduct parts of your income and put it into this retirement account pre-tax. You can put in max like $23.5k (this goes up every year) which reduces your taxable income. You also can't touch this until you're 60, but like the Roth IRA you can invest and grow this tax free. You do however pay taxes when you take the money out in your retirement as income. This is still useful because when you retire, your income is likely lower which means you will be in a lower income bracket and you can decide how much to take out. The caveat is that this account is tied to the company and you may have limited investment options that the company offers. If you leave the company you may have to move the 401k or roll it over to a IRA.
- **IRA**: An IRA is an Individual Retirement Account that anyone can open. Some companies will offer a IRA instead of a 401k and they are basically the same when it comes to tax advantage. A major difference though is the IRA is yours and you have more flexibility in what you can invest in. Instead of limited funds, you typically have access to the broader market just like the Roth IRA except like the 401k you will pay taxes when taking money out.
- **Credit Score**: This is relevant if you ever need to borrow money like for buying a car or house or getting a new credit card. You should check your credit report every year with the big 3 credit bureaus (Experian, TransUnion, Equifax) or use an app that can monitor it. This is free by law, and helps make sure your identity hasn't been stolen.
- **Capital Gains**: When you sell stocks in a traditional brokerage account you pay taxes on those earnings but they are often significantly lower taxes than the money you earn in income. This only applies if you've held that asset for more than a year, otherwise those gains are taxed at the normal income rates. Do some research on this to help maximize your gains before Uncle Sam takes his cut.
- **Taxes**: Learn how to file your taxes if you haven't already.
- **Scams**: Once you have money to lose, a loss can feel much more devastating. Keep vigilant about your accounts and remember, if it sounds too good to be true, it probably is. There is no easy way to get rich. Investing is about calculated risk, not gambling.
- **Mortgages**: You may not be ready to buy a house anytime soon, but you should understand how a mortgage works. A house will likely be the most costly purchase of your life and it is best to understand how that would work when planning your financial future. Personally I've given up on buying a house, but if there's a sudden downturn in the market I want to be ready.
- **Budgeting**: You don't need to be an accountant, but have a rudimentary idea of where your money is going. You don't need to be pinching pennies, but you don't want to live outside your means especially when you are just starting out. Just because you can spend the money doesn't mean you should. Find a system that works for you.

- **Bankruptcy**: Understand what bankruptcy is. Hopefully you never have to go through this, but understand that it is a tool to restructure your finances if you ever end up with debts you cannot pay.

Right out of school you will likely have some debt from college loans. My parents didn't pay for my undergrad so I took out loans, but I was able to pay those off by saving during my summer internships. I would recommend you pay off your debts as soon as possible. Compound interest is a double edged sword and can work against you as much as it can help you. Being able to grow my net worth without having debts will make a huge difference.

Most larger companies will offer stock as part of your compensation, and once you are more senior it becomes the majority of your compensation. I see a lot of people sell their stocks immediately so it's like an extra bonus every quarter in order to buy big ticket items like a car or a vacation. While you should enjoy your compensation, keep in mind that if the company you work for is doing well there is a likelihood the stock will rise over time. I remember there was a story at Microsoft of an executive assistant who keep the relatively meager amounts of stock she was granted throughout her many years there and by the time she was ready to retire, she had enough to build a dream retirement home. That said, you should be smart about it. For every success, there are dozens of failures where people keep their start up stock and it becomes worthless when the company fails.

As an undergrad, you may still be on your parent's health insurance, but that will eventually end. If your company offers health insurance you should learn what the options are. Figure out what a Health Savings Account (HSA) is and the differences between a Preferred Provider Organization (PPO) and Health Maintenance Organization (HMO). Learn what a premium, deductible, and co-pay is as it determines how much you pay every month out of your paycheck and how much you pay when you go get medical treatment. In the US, medical expenses can be quite high and if you have any chronic health issues or sudden injuries those bills can add up very quickly. Do the calculation during your annual open enrollment period to see which plan is right for you. That's right, you can only choose once a year unless you have a qualifying event such as a job change, marriage, or new kid.

The stock market broadly speaking has been doing quite well and been in a bull run for the last few years. While this isn't likely to continue indefinitely it is a way to grow your wealth in the long run. Unless there is a major upset to the financial markets it's likely that your portfolio will grow before you retire. If you're in your early twenties as a soon to be new grad, you will have 40 years of compound interest on your side. Investments like S&P500 ETFs tend to rise over decades even if there are years that dip. That's not always true though, take a look at Japan's Nikkei which has taken 30 years to recover to the levels it was in the 90's. Typically when you are young you are able to have riskier investments like individual stocks because you have the time to make up the money should anything happen. However you probably don't want to fall into gambling, if you know you have those tendencies be very careful. I personally am fairly risk averse when it comes to finances so while I do have individual stocks, I balance it out with ETFs and broader market funds. I don't touch any of the riskier investments like options or crypto, but like I said, this isn't financial advice so do what you will with that statement. I also only invest in companies that I understand decently well which is mostly tech since I am an engineer.

There isn't any one way to become financially stable and at the end of the day you have to make the decision that works best for you.

Hope this has been helpful. Wishing you the best in your career!

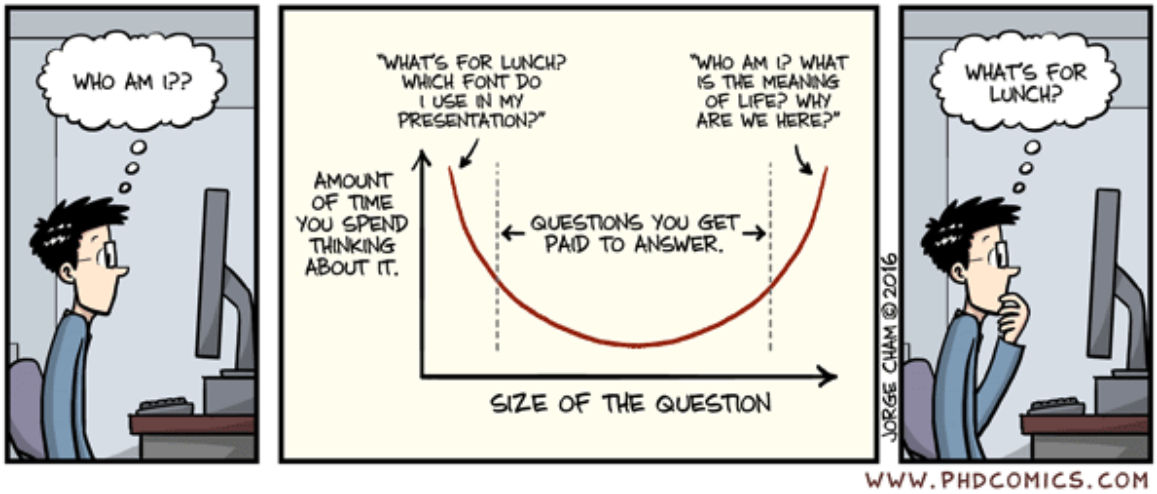


Figure 49: Remember to ask questions.