**Florida Institute of Technology: OCE4531**

# Instrumentation Design and Measurement Analysis

**A Compendium of Knowledge**

Braidan Duffy [*1] and Wensen Liu [†2]

[1]Department of Ocean Engineering and Marine Sciences, Florida Insitute of Technology
[2]A. James Clark School of Engineering, Department of Mechanical Engineering, University of Maryland

Fall, 2022

Florida Institute of Technology

* bduffy2018@my.fit.edu
† wliu1215@terpmail.umd.edu

OCE4531 Lecture Notes

**Disclaimer**
You can edit this page to suit your needs. For instance, here we have a no copyright statement, a colophon and some other information. This page is based on the corresponding page of Ken Arroyo Ohori's thesis, with minimal changes.

**License**


[1]Department of Ocean Engineering and Marine Sciences, Florida Insitute of Technology
[2]A. James Clark School of Engineering, Department of Mechanical Engineering, University of Maryland



**Colophon**
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.
The source code of this book is available at:
`https://github.com/fmarotta/kaobook`
(You are welcome to contribute!)

**Publisher**
First printed in May 2022 by Florida Institute of Technology

* bduffy2018@my.fit.edu
† wliu1215@terpmail.umd.edu

*Ab experientia, intellectus*

# Preface

The world of electronics and making is one that is constantly evolving. The 3D printer revolution has brought about a distinct 4th age of industrialization and now anyone with the desire and will to learn how can turn ideas into reality. The trivialization of printed circuit board manufacture has also lowered the bar to entry for everyday people to get into their own projects. We are at a point in history where innovation is at an all time high and it is no longer massive well-funded corporations pushing the envelope. Anyone with a computer, an internet connection, and a problem to solve can change a product, or make their own, and have a tangible impact on our world.

I have had a the fortunate opportunity to spend years learning about this world and being deep within the fourth industrial revolution. It is my privilege and honor to be able to put all of my knowledge within this compendium and share it with you, the reader. I wrote this in the context of teaching a class and organizing my lecture notes, but in reality, this is meant for anyone to pickup and start learning. I won't testify that this is the end-all, be-all of electrical engineering information - it's not meant to be! My hope is to use the information I have learned and give you a starting point on your journey and hopefully make the learning curve a little bit easier for you. One day, I hope that you will know more than me and publish your own lecture notes and use that to teach the generations after you.

Yours humbly,
*Braidan Duffy*

# Contents

# List of Figures

# List of Tables

# Learning Materials and Setup 1

This course will require you to become familiar with different software packages and get you hands on with real-world electronics hardware. In this chapter, we will go into the required course material and give you an overview of what it is, how to set it up, and what you will be expected to know.

These are for your reference, so please follow these instructions closely and pay attention to the component overview parts so you can have more time playing and experimenting than troubleshooting.

## 1.1 Autodesk Fusion 360

For this class, you will be required to use Autodesk Fusion 360 to create and work with your projects. This is a fantastic tool that syncs your data to the cloud and allows you to work on your project on any computer with Fusion 360 installed and keep a running version history. There is also a teams feature that allows the instructor to keep a better track on your progress and provide specific feedback in real-time either through comments or direct review. Students also have a free license to all Autodesk products through Florida Tech using an educational license which is valid as long as you remain a student and renew it every year.

To install Fusion 360 and get started, follow the guide below:

# AUTODESK® FUSION 360®

Single user install process and browser access to Fusion 360

As of August 2021, there are two paths that students may take to get an account with educational access to Autodesk products including Fusion 360. The first path, option 1, is also available to educators and design competition mentors in addition to students.

---

## Option 1

**Individual Access**
A student, educator, or design competition mentor may create an account and individually confirm their eligibility to access the Autodesk Education plan. This plan provides free, educational access to Autodesk products for eligible individuals. Products can be accessed this way at autodesk.com/eligibility.

## Option 2

**Access via an Educator**
A student may create an account after getting assigned to use one or more Autodesk products by their educator. This will provide students with access to Autodesk products at manage.autodesk.com/cep/. These students will not have to individually confirm their eligibility for educational access to Autodesk products.

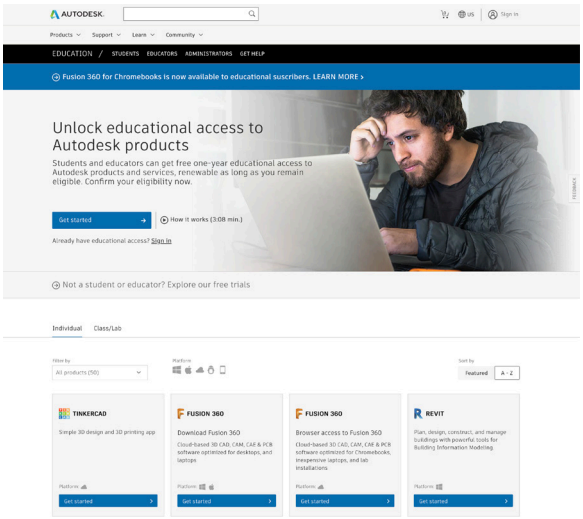
Students may use both option 1 and option 2 at the same time. This document will walk you through how these options work.

# AUTODESK® FUSION 360®

Single user install process and browser access to Fusion 360

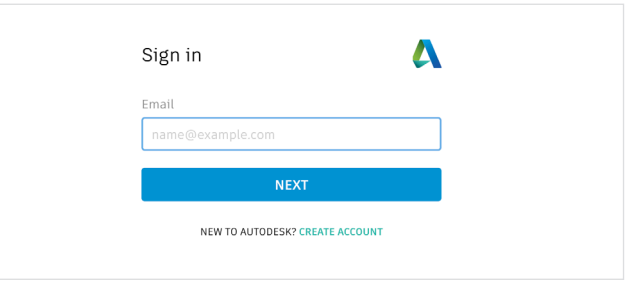
## Option 1: Individual Access

A student, educator, or design competition mentor may create an account and individually confirm their eligibility to access the Autodesk Education plan. This plan provides free, educational access to Autodesk products for eligible individuals.

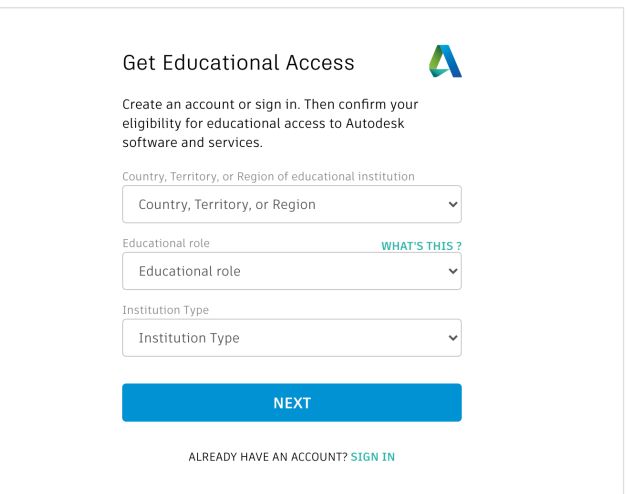



### 1 Create an account

a. **Go to the Get Products page within the Autodesk Education Community at [autodesk.com/eligibility](autodesk.com/eligibility) and click Get Started on the Download Fusion 360 product tile.**

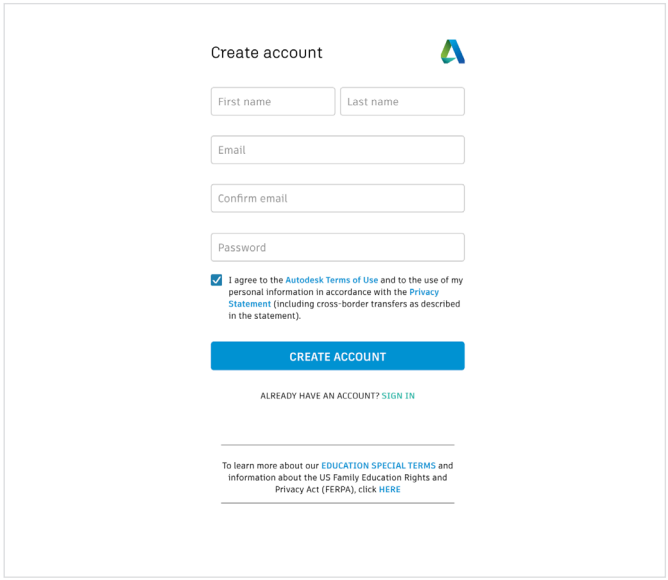



b. **Sign into your account or click Create Account to create a new Autodesk account.**




c. **If you are creating a new account, fill in the required information and click Next.**

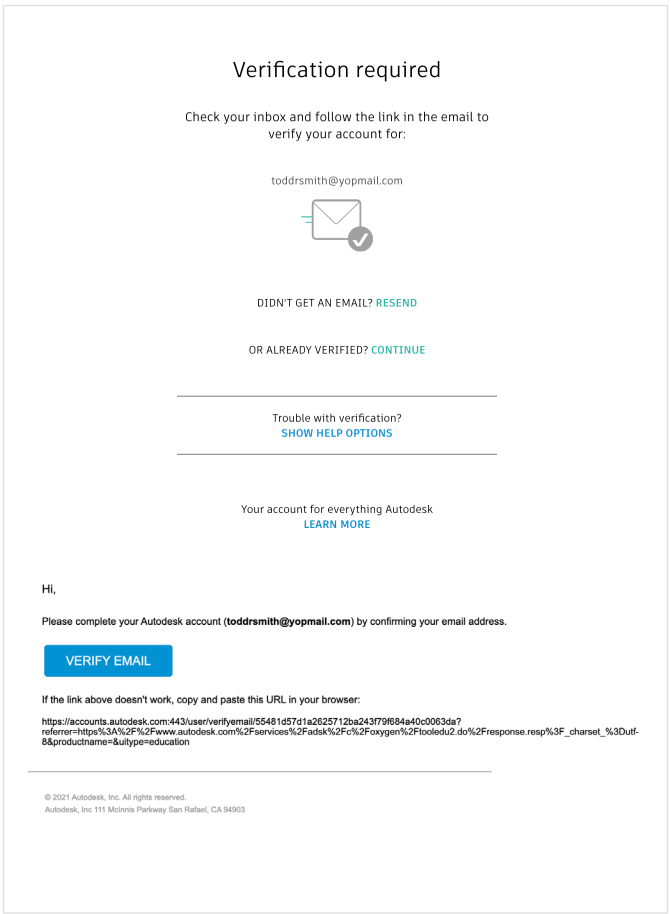
**Note:** It is important to provide the correct information here because your answers will inform the list of schools that you will choose from on the next screen.



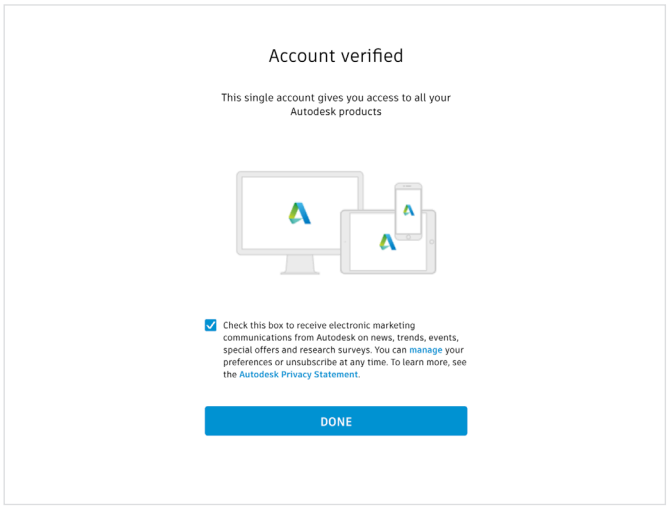

**d. Enter your full name, email address, and a password to create an Autodesk Account.**

**Note**: The first and last name you use in this step needs to match the first and last name on the documents you will submit to confirm your eligibility for free educational access to Autodesk software and services.

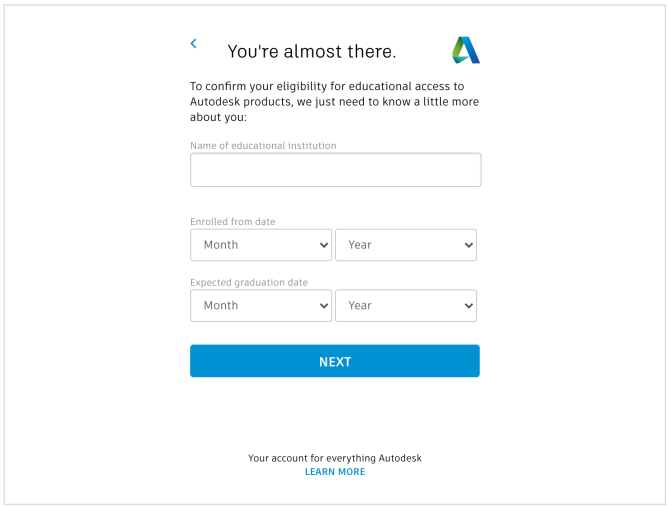
For example, If your school ID says “Susan Smith,” please enter this as your name instead of “Sue Smith.”



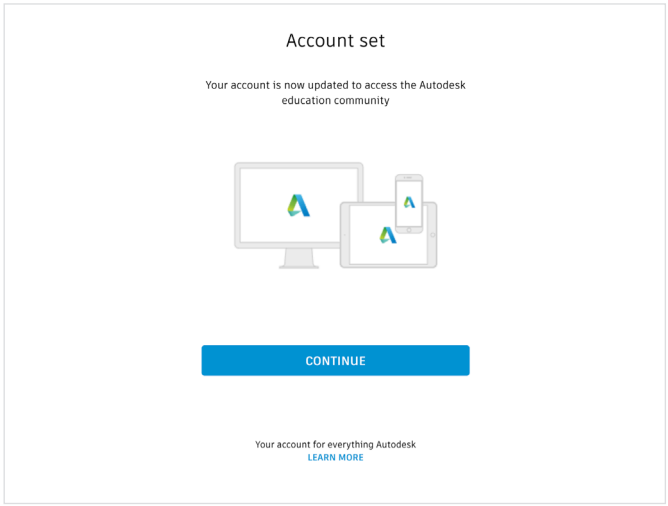

**e. Check your email account for a message from Autodesk, open it, and click the Verify Email button within the message to verify your email address.**




f. A message will display in your browser stating that your account has been verified. Check or uncheck the box on this page to set your email preferences and click Done.



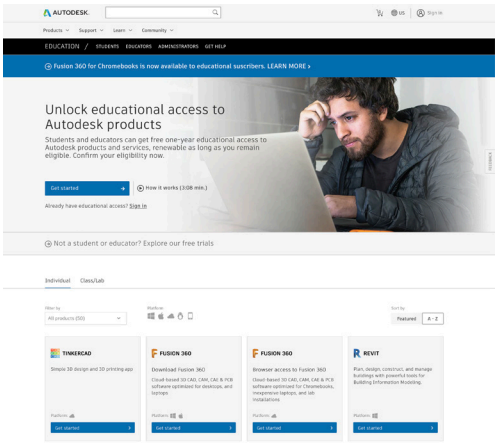

g. A form will display that asks you for a few more details related to your educational eligibility. Fill out this form and click Next.




h. A message will display about your account status. Click Continue and return to the Get Products page within the Autodesk Education Community at autodesk.com/eligibility.

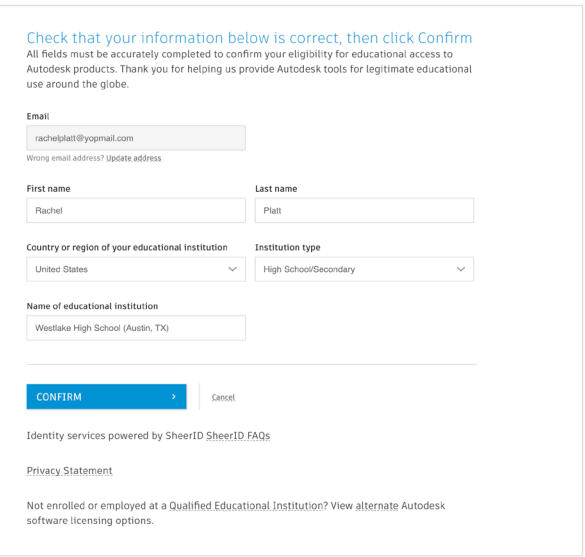
# AUTODESK® FUSION 360®

Single user install process and browser access to Fusion 360

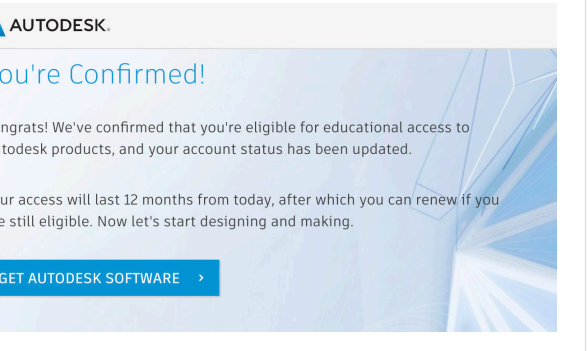



## 2 Confirm your eligibility

a. **Go to [autodesk.com/eligibility](autodesk.com/eligibility). If you see a message stating that we still need to confirm your eligibility for educational access to Autodesk products, then click the Get Started button to begin this confirmation process.**

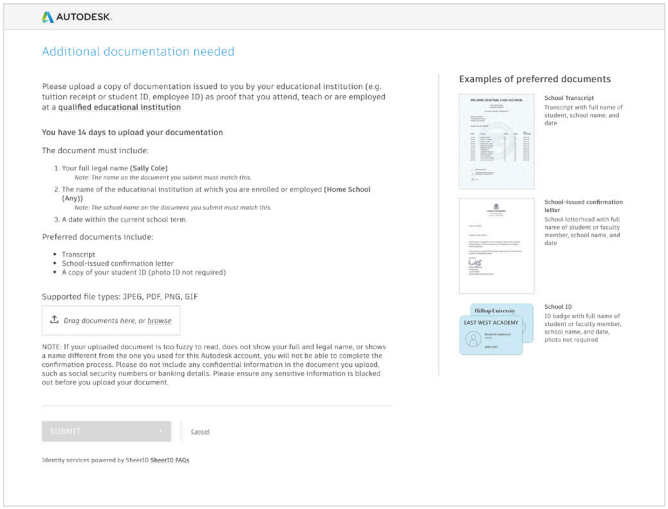



b. **Check that your information is correct and click the Confirm button.**




c. **Autodesk uses SheerID's verification services to help confirm your educational eligibility. If SheerID can immediately confirm your eligibility, you will see a message that says, "You're Confirmed." If you see this message, click the Get Autodesk Software button.**

If you do not see this message, please follow the additional steps below to provide documentation that will allow SheerID to confirm your educational eligibility.

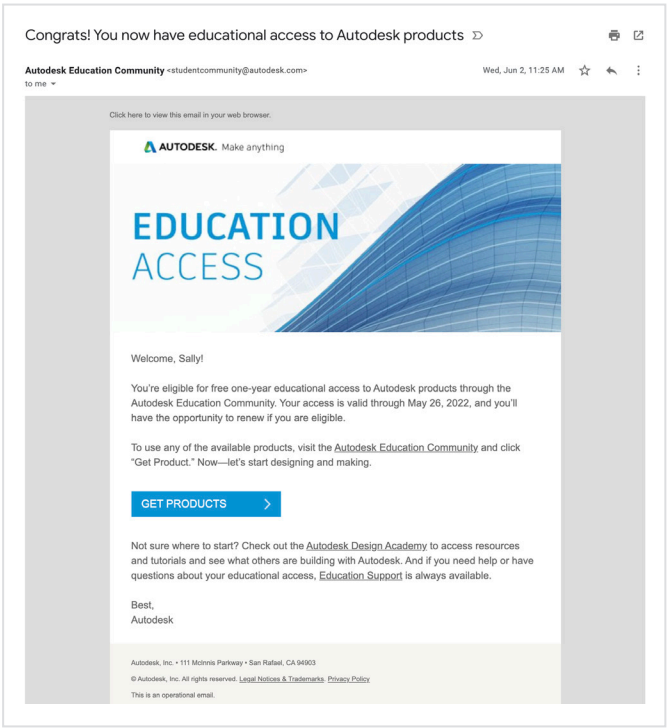



**d. If you reach a page with the heading "Additional documentation needed" you will need to upload at least one document confirming your enrollment or employment status with an eligible educational institution. To improve your chances of approval, please provide more than one document. For example, students can upload a student ID card and a class schedule. Educators can upload an employee ID card and an official employment letter from their educational institution.**

The following letter templates can be downloaded and used to create documentation on school letterhead:

- Student template
- Faculty template

Once your documentation has been uploaded, click Submit. It may take up to 48 hours for your educational eligibility status to be determined by SheerID.

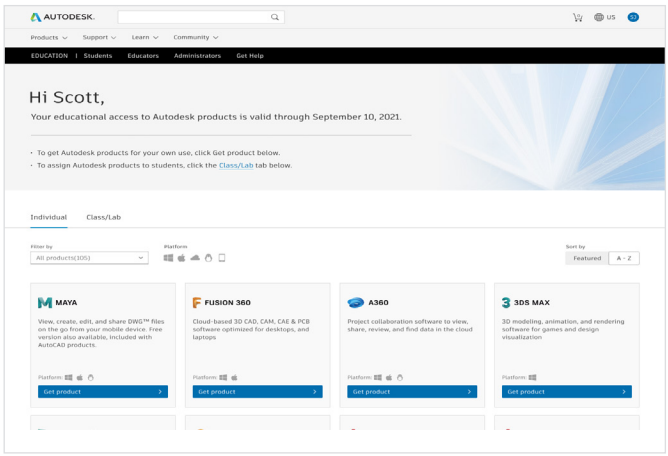



**e. If SheerID approves your educational eligibility, you will receive a confirmation email.**

If you do not receive a confirmation email and would like to follow up with SheerID, please contact them at customerservice@sheerid.com.

# AUTODESK® FUSION 360®

Single user install process and browser access to Fusion 360




## 3 Get Fusion 360 for educational use

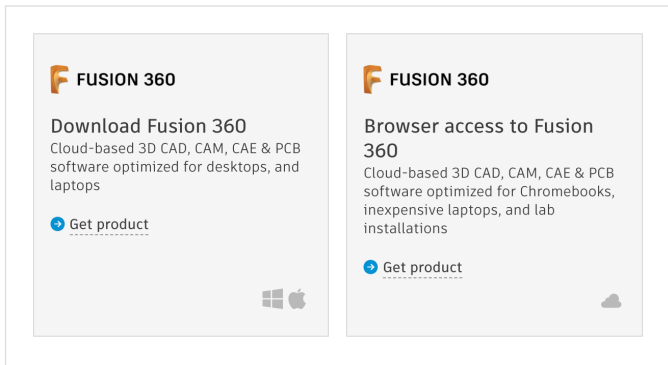
**a. Once your educational access has been approved, return to the Get Products page within the Autodesk Education Community at autodesk.com/eligibility and make sure you are signed in.**

| System requirements for Autodesk Fusion 360 | |
|---|---|
| Operating System | Apple® macOS™ Big Sur 11.x*; Catalina 10.15; Mojave v10.14(learn more about Apple Security Updates)<br>Microsoft® Windows® 8.1 (64 bit) (until January 2023)**<br>Microsoft Windows Windows 10 (64-bit) Windows Release Information |
| CPU Type | x86-based 64-bit processor (e.g. Intel Core i, AMD Ryzen series), 4 cores, 1.7 GHz or greater; 32-bit not supported<br>ARM-based processors partially supported via Rosetta 2 only - see this post for more information. |
| Memory | 4 GB of RAM (integrated graphics recommend 6 GB or more) |
| Graphics Card | DirectX11 (Direct3D 10.1 or greater)<br>Dedicated GPU with 1 GB or more of VRAM<br>Integrated graphics with 6 GB or more of RAM |
| Disk Space | 3 GB of storage |
| Display Resolution | 1366 x 768 (1920 x 1080 or greater at 100% scale strongly recommended) |
| Pointing Device | HID-compliant mouse or trackpad, optional Wacom® tablet and 3Dconnexion SpaceMouse® support |
| Internet | 2.5 Mbps or faster download; 500 Kbps or faster upload |
| Dependencies | SSL 3.0, TLS 1.2+ |

| Recommended specs for complex modelling and processing | |
|---|---|
| CPU Type | 3 GHz or greater, 6 or more cores |
| Memory | 8 GB RAM or greater |
| Graphics | Dedicated GPU with 4 GB or more VRAM, DirectX 11 (Direct3D 11 or greater) |

**b. Check the system requirements for Fusion 360 software. If your computer meets these requirements, proceed to the next step and continue with the installation process.**

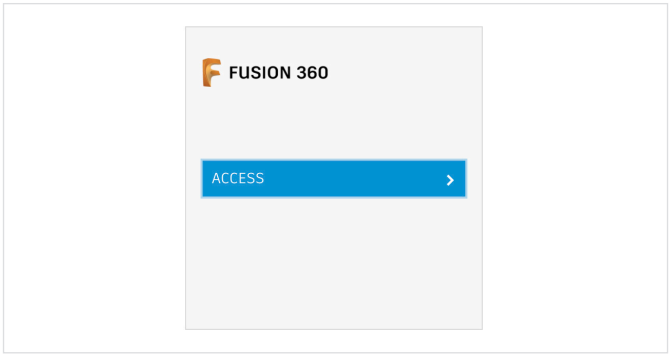
If you are using a Chromebook or a computer that does not meet these requirements, click Get Product on the Browser access to Fusion 360 tile. This will take you to fusion.online.autodesk.com where you can use Fusion 360 in a browser without installing it. This browser access option is only available in English.



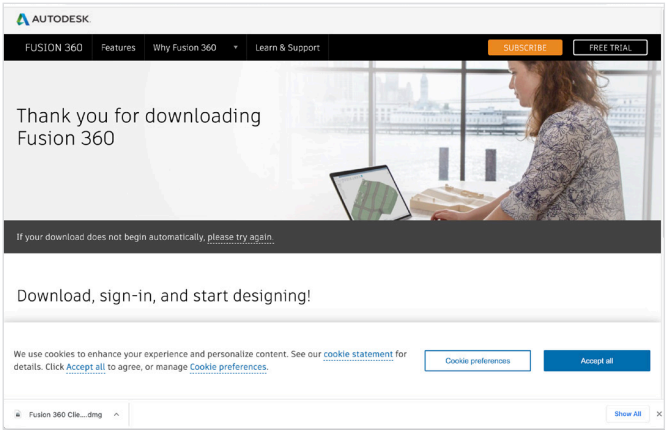

**c. If you are a student or design competition mentor, click Get Product on the Download Fusion 360 tile.**

If you are an educator, make sure you are on the Individual tab and then click Get Product on the Download Fusion 360 tile.

The Download Fusion 360 tile is for downloading the Fusion 360 software.

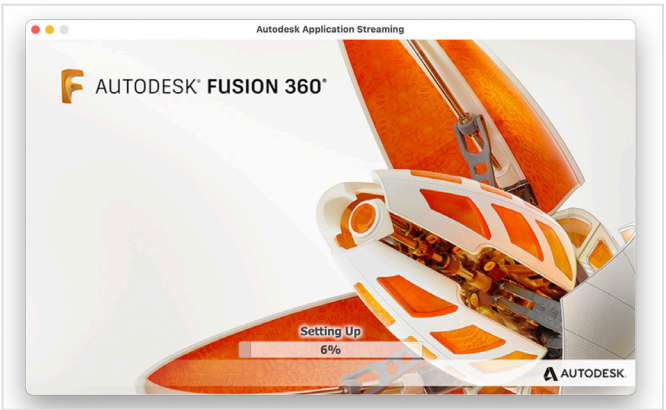



d. **To initiate the desktop client installation, click Access after clicking Get Product on the Download Fusion 360 tile.**

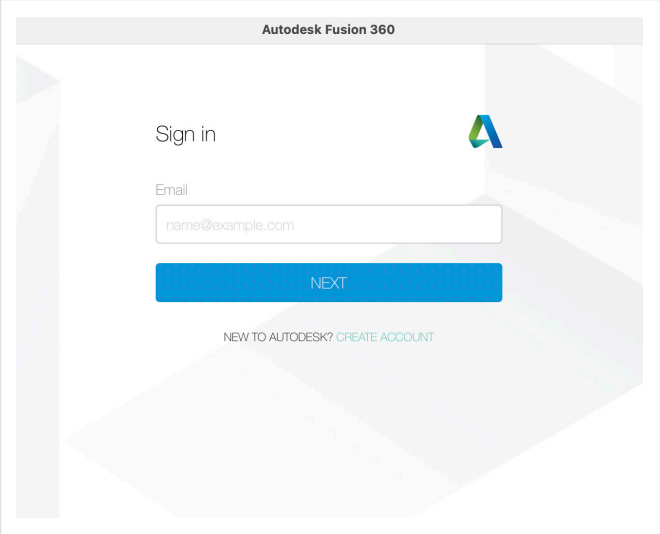



e. **A new browser window will open and the Fusion 360 desktop client will download automatically.**

If your download does not begin automatically, click Please Try Again to try again.

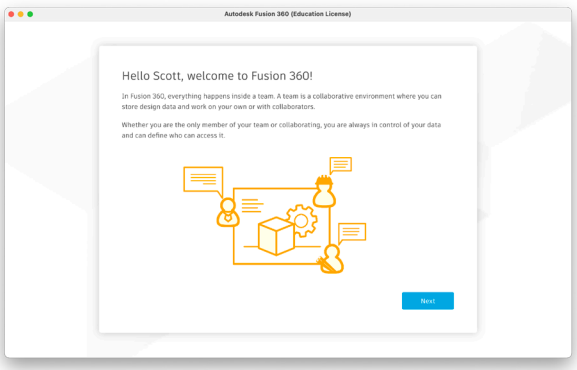



f. **Open the .dmg or .exe file and go through the setup process. During the installation, you will see a progress bar. When the installation is complete, Fusion 360 will launch automatically. This may take a few minutes.**



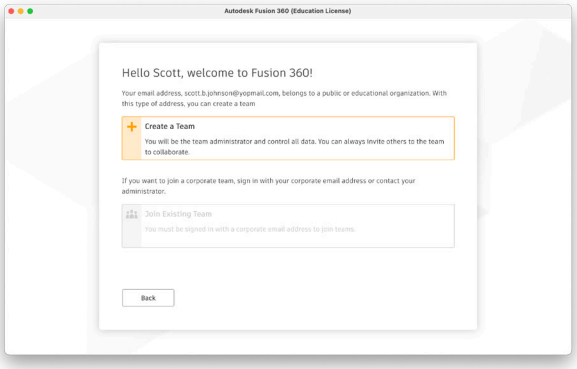

g. **Sign into Fusion 360 using your Autodesk account with education profile credentials.**

**Note:** Be sure to sign into Fusion 360 using the same email address and password that you used to confirm your eligibility for free educational access.



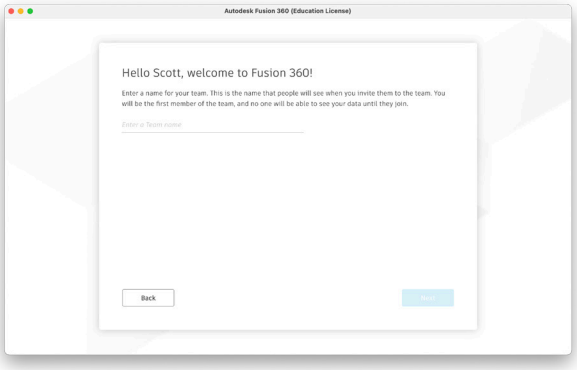

## 4 Additional steps for new Fusion 360 users

a. **If you are a new Fusion 360 user, the first time you sign into Fusion 360 you will see a welcome message. Click Next.**



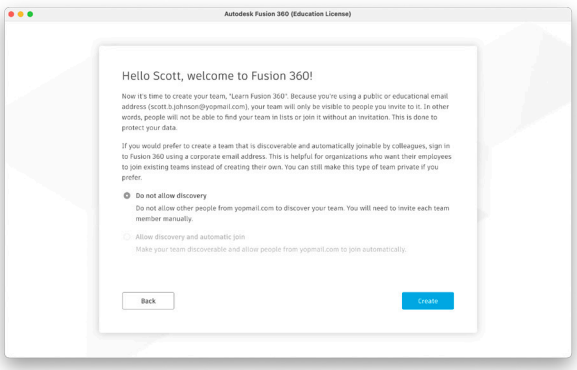

b. **You will be prompted to create a team or join a team. You must be associated with a team to use Fusion 360. Once you have a team, you can use it to create projects, add people to your projects, share and manage your project data, and more. To create a team, click Create a Team.**

For more on this topic, visit the Autodesk Knowledge Network article titled Getting Started with Fusion Team for Education.

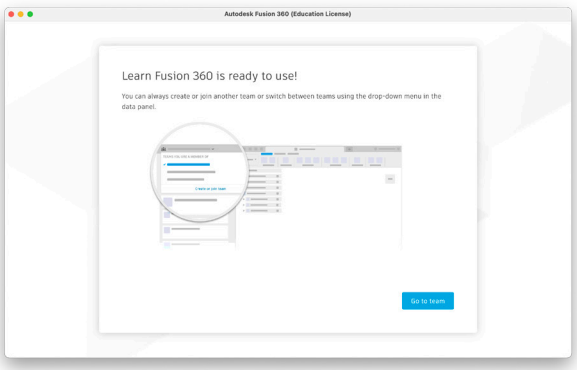



c. **Enter a name for your team. It could include your name or nickname. The team name you use will be visible to anyone that you invite to your team.**

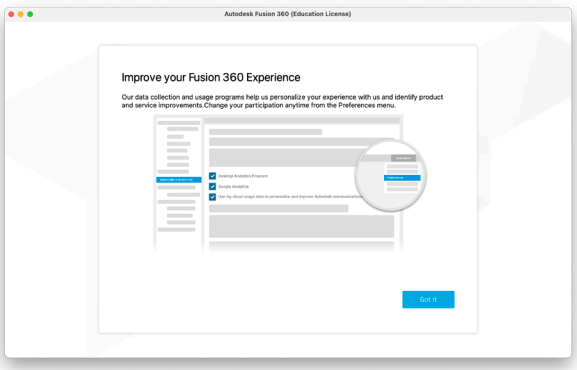
**Note:** Team names cannot contain emojis or any of the following characters: \ / : * ? " < > |



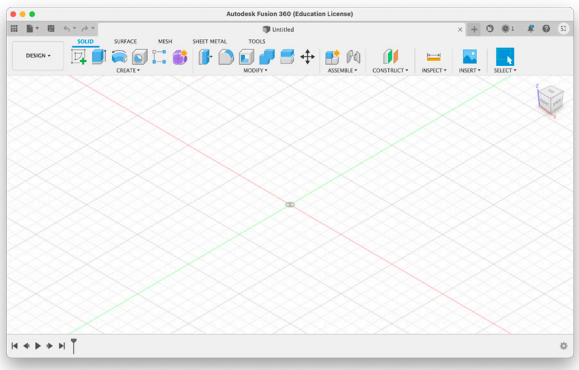

d. Check or uncheck the “Do not allow discovery” option and click Create.



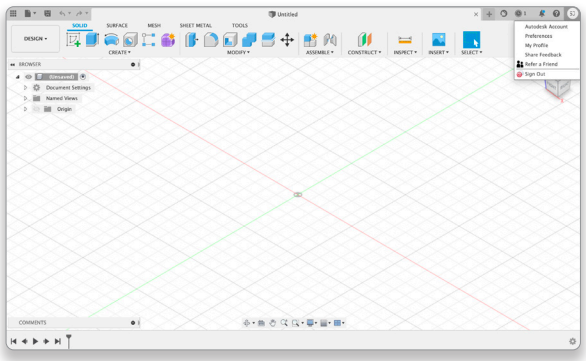

e. Your team is ready to use. Click Go To Team.



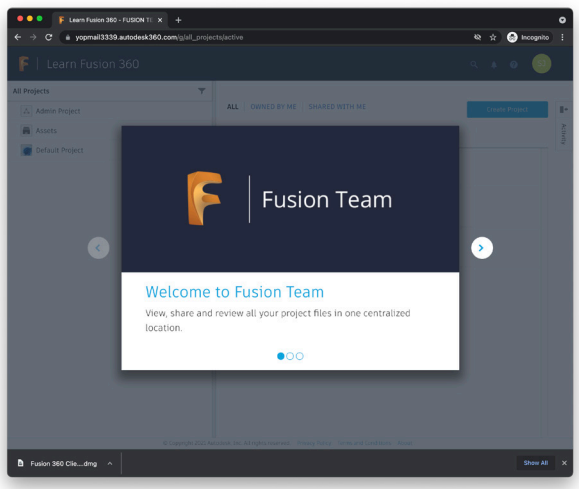

f. A message may display about improving the Fusion 360 experience. Click Got It.



g. Fusion 360 is now ready.



h. To launch Fusion Team in a web browser, click your initials on the upper right-hand side of Fusion 360 and select My Profile.




i. You will arrive at [myhub.autodesk360.com](myhub.autodesk360.com) where you can view, share, and review all your project files in one centralized location.

### Joining the Class

Now that you have Fusion 360 installed on your machine set up,

## 1.2 Arduino IDE

Include Arduino image near section header or in corner

This entire course hinges on you being able to program microcontroller boards with an Integrated Development Environment (IDE). Technically, the Arduino Megas can be programmed using Atmel Studio, but that application has a very steep learning curve and the Arduino foundation has done a fantastic job simplifying the whole process so anyone - young or old - can quickly begin programming. The Arduino IDE has the ability to download and import custom boards and third party libraries, enabling you to program a wide range of microcontrollers for almost any application imaginable. It also has the whole compiling toolchain built into the backend, so all you need to worry about is ensuring the proper configs are in place and press a single button. This makes it a very useful tool for learning, but this IDE is limited if you want to expand your scope beyond a single test or project file and want to do something more complicated. We will discuss an alternative later.

### Installation

Note: as of July 2022, the current Arduino IDE version is 1.8.19

For now, navigate to the Arduino IDE download page (`https://www.arduino.cc/en/software`) and download the latest version for you machine's operating system. You may elect to donate to the Arduino Foundation (it's tax-deductible!) or just download the IDE without donating. Once the installer is downloaded, execute it and follow the steps to install the IDE.

First, accept the software end usage license agreement.

On the next page you will see some additional options for the install. Select all of the additional options as this will pre-install some board drivers and shortcuts, making the IDE easier to use initially.

Finally, you will need to specify the install location. For 99.9% of users, the default install location will suffice. Be sure to keep this location in mind if you elect to install Visual Studio Code with the Arduino extension.

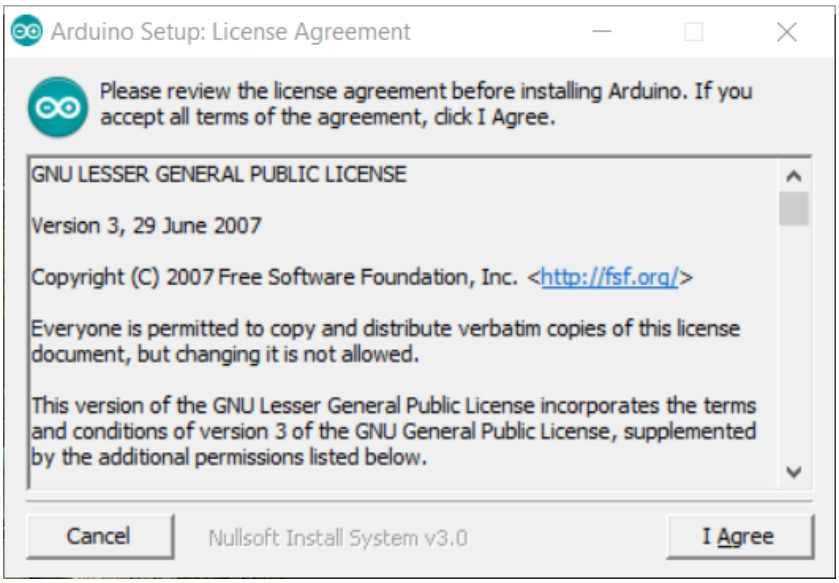



**Figure 1.1:** Accept the license agreement for installing the Arduino IDE

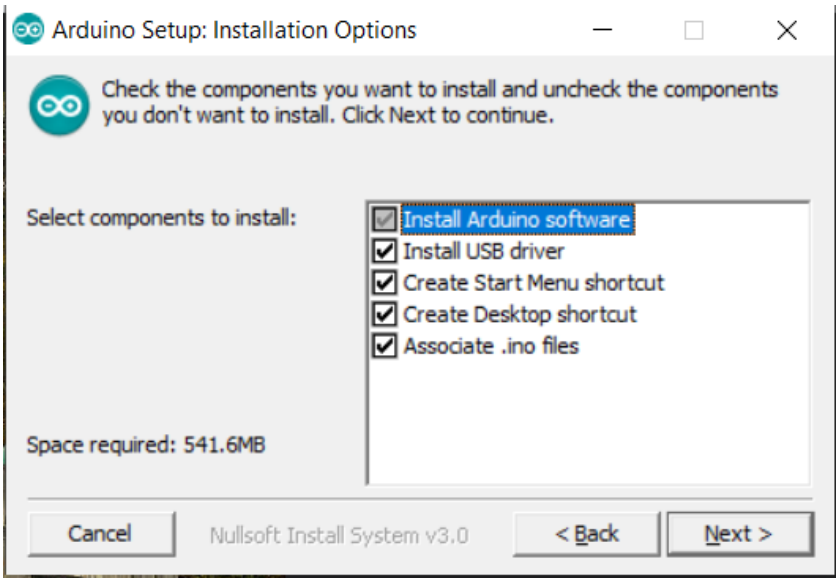



**Figure 1.2:** Select all of the additional options for installing the Arduino IDE

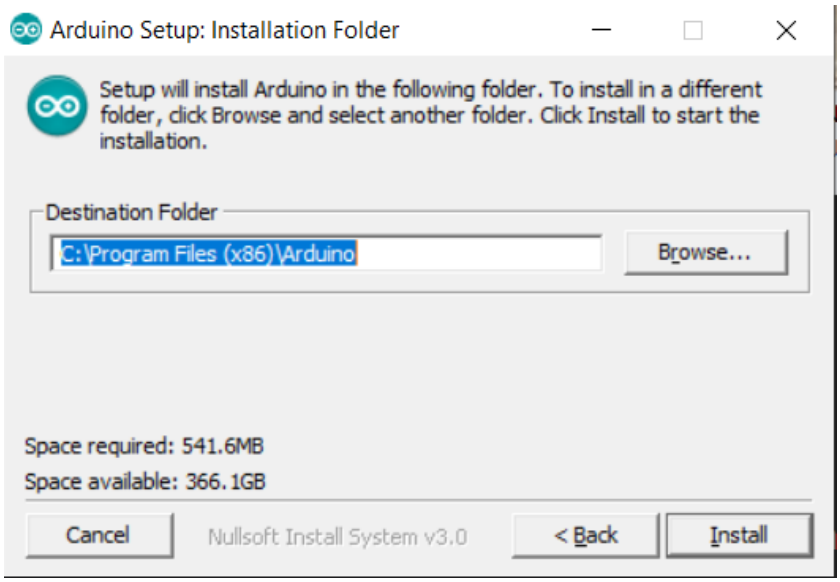



**Figure 1.3:** Specify the install location for the Arduino IDE

## 1.3 Visual Studio Code

Include VS code symbol near section in corner

A superior alternative to the Arduino IDE is Visual Studio Code. This is a fully-fledge IDE with support for a multitude of languages, version control software, filetypes, and editing experiences. Basically, whatever you want to do with VS Code, you should be able to with the right combination of extensions and workspace layout. Speaking of which, one of the most useful features of VS Code is the workspace. It conveniently locates all of your working directories and files into a central location with easy access to edit and move as you please - irregardless of the files' and directories' locations on your filesystem. This can make it really easy to do large projects, like organize source code folders for a project-based course, or write lecture notes that have tons of PDFs, build files, data files, source codes, and images...

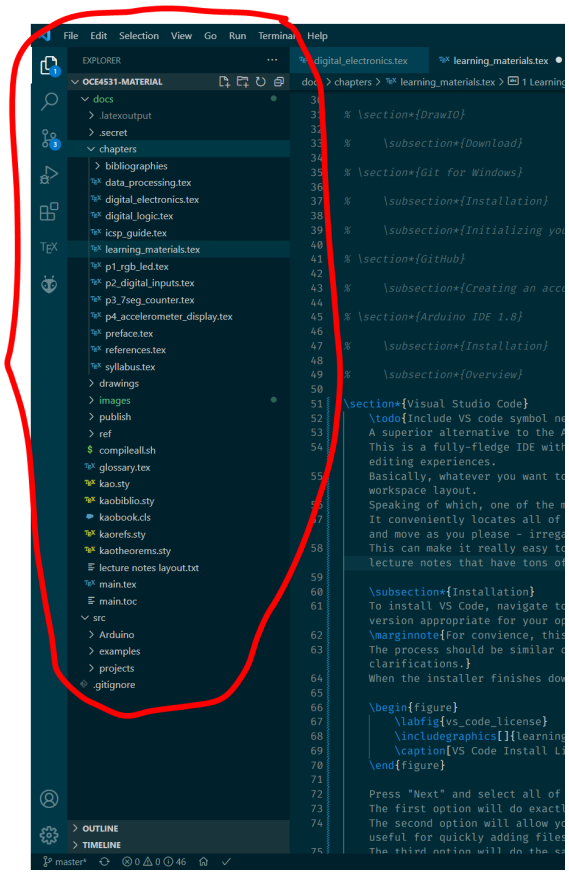



**Figure 1.4:** Workspace within the VS Code application, circled in red. The window shown is configured to use the Fira Code Font, and Midnight Blue color scheme.

### Installation

To install VS Code, navigate to the download site at `https://code.visualstudio.com/download` and select the version appropriate for your operating system. When the installer finishes downloading, open it and accept the license on the first screen.

For convience, this guide will only cover the Windows install. The process should be similar on other operating systems, but please consult other install guides for clarifications.

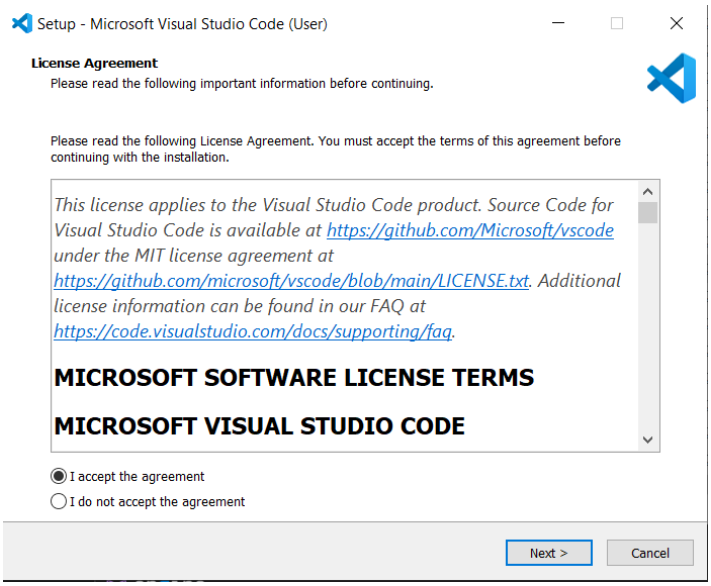



**Figure 1.5:** Accept the license agreement for installing VS Code

Press "Next" and select all of the options on the next page. The first option will do exactly as advertised, there will be a new icon on your computer desktop. The second option will allow you to open files directly into VS Code from the file explorer - a feature that is useful for quickly adding files to your workspace. The third option will do the same as the previous, but for directories - this again is convenient for quickly adding folders to your workspace. The fourth option will tell the filesystem to automatically open supported code files (.cpp, .h, .py, .jar, etc.) in VS Code for easy access. And the fifth and final option adds VS Code to the system path. This last option can make executing custom scripts and running some VS Code extensions a little easier.

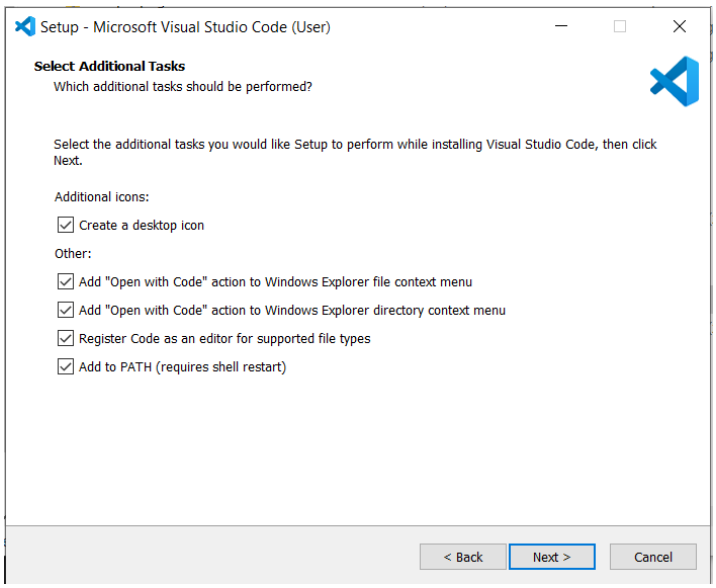



**Figure 1.6:** Select all of the additional tasks on this page

Finally, you will be ready to install VS Code! Press "Install" on the next page and let the rocks and electrons do their work. Be sure to restart your

system once the application is installed!

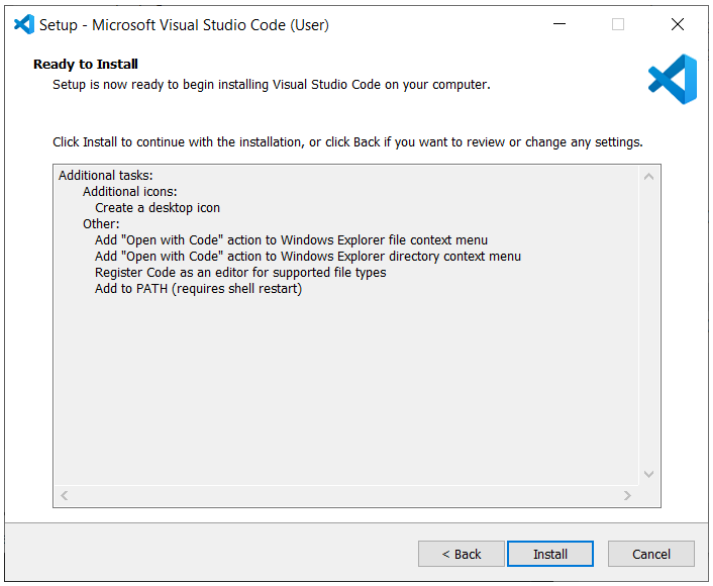



**Figure 1.7:** Press button. Install VS Code

## Arduino and C++ Extensions

To get started with your new instance of Visual Studio code, we will install the Arduino and C++ code extensions. These extensions will allow you to compile and upload Arduino code, just like you would in the Arduino IDE, but simultaneously have greater flexibility with the code workspace, version control, etc.

To begin, navigate to the extensions tab of the VS Code window, shown below.

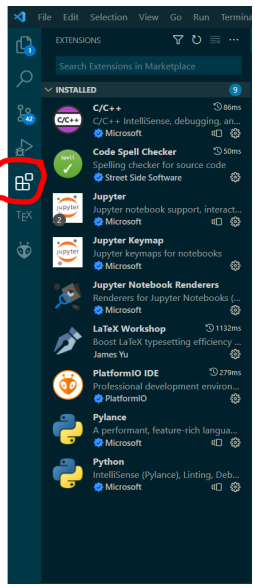



**Figure 1.8:** Open the extensions tab using the menu bar to the left of the screen (circled in red)

In the search bar near the top of the menu, type "Arduino" and find the Arduino Extension published by Microsoft. [1] Click on the little "Install"

1: Should be the first result

box and wait for VS Code perform its magic. After it successfully installs, you will notice the "Install" button has been replaced by a gear symbol. This is a shortcut to the extension preferences. Click on the icon and select "Extension Settings" - this will open a new tab in your VS Code window with a bunch of configurations.

Navigate to the option called "Arduino: Path" towards the bottom of the page. Here, paste the location to the Arduino installation folder on your machine. If you did the default installation path on windows it will be `C:\Program Files (x86)\Arduino`.

For convience, I also suggest unchecking the two options: "Arduino: Analyze on Open" and "Arduino: Analyze on Change". This will improve performance by not compiling the script every time you make a change and save it, something that can take a very long time with complicated programs.

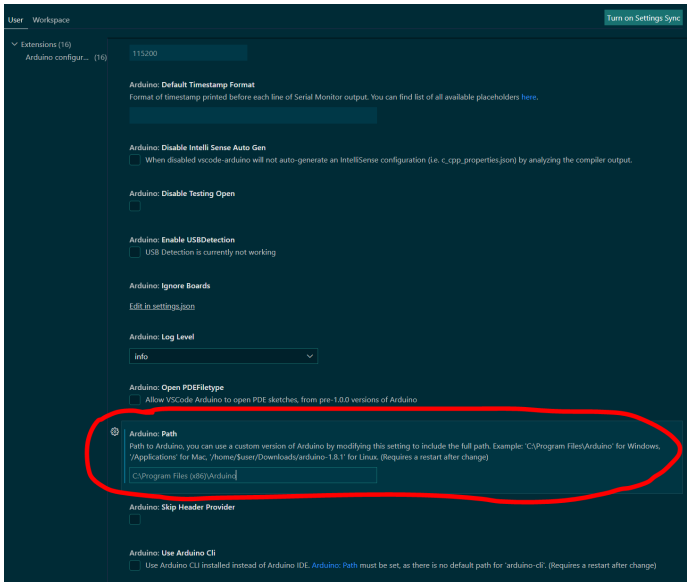



**Figure 1.9:** Specify the location of the Arduino install folder on your machine in the "Arduino: Path" option (circled in red).

Next, as before, open the extensions tab and search for "C/C++". Install the C/C++ Extension published by Microsoft (again, it should be the first result) and reload the VS Code window, if prompted. This will enable VS Code to edit and interact with the C++ files Arduino uses!

With this completed, you may now use VS Code to program your Arduino boards!

## PlatformIO Extension

A far superior embedded programming toolchain is provided by PlatformIO. This is a cross-platform high performance compiler and IDE for a wide variety of embedded platforms. It has superior compilation performance compared to the stock Arduino toolchain and has a project-based structure. This makes it the better platform to use for large coding projects.

Better yet, the core of PlatformIO is written in Python and can be executed through the terminal. this gives it incredible flexibility in allowing users to use the dedicated PlatformIO IDE application or VS Code. Here, we will use the VS Code PlatformIO extension so that we can better integrate it with our code workspaces and version control.

As before, open the extensions tab shown in Figure **??** and type "PlatformIO" in the search box. Click on the "Install" button in the bottom right corner of the PlatformIO IDE extension and allow the gerbils within VS Code to spin their wheels. When it has finished installing, you may need to reload the VS Code window - do so, if prompted.

This install make take longer than the others as it needs to install the entire PlatformIO Core backend in addition to the extension. This may be expedited by having a high-speed internet connection, but your mileage may vary.

With this extension installed, you can now begin to do some really cool things, programmatically speaking. I implore you to explore the PlatformIO IDE on your own and use it to program your projects through this course!

## 1.4 Arduino Kit Introduction

There is a really cool simulator made in the browser that you may want to consider using to test code before deploying it to hardware, or if you don't have access to the Arduino kit. https://wokwi.com/

You will all be required to purchase an Arduino learning kit. It is highly recommended you purchase the Arduino Mega complete starter kit from ELEGOO as it has all of the components necessary for the course, as well as a multitude of projects for your own education. Additionally, the expansive IO of the Arduino Mega will make certain projects easier for you through the course and give you larger amounts of program storage and memory for more complex projects.

Most of the simpler discreet parts will be discussed ad nauseam in later sections of these notes. Therefore, they will be skipped in this section. Below are some of the more unique parts of the Arduino Mega kit and what they can do.

More information on the Arduino Mega can be found the Arduino website - https://www.arduino.cc/en/Guide/ArduinoMega2560/

### Microcontroller

The Arduino Mega uses an ATmega2560 microcontroller to execute programs and interface with a wide variety of inputs and outputs. This chip has:

- 86 programmable General Purpose Input/Output (GPIO) pins that operate at a 5V TTL logic.
- 16 10-bit analog inputs
- 2 8-bit counters; 2 16-bit counters
- 1 Serial Peripheral Interface (SPI) bus
- 4 Universal Synchronous/Asynchronous Receive/Transmit (USART) busses
- 256 KB of flash memory for program storage
- 8 KB of SRAM for program memory
- 8 KB of EEPROM for long term, fast access storage

The ATmega2560 has been widely adopted by the Arduino and maker community so there is rich and mature development support for the chip. In this course, you will be using the Arduino IDE (or Visual Studio Code) to program the microcontroller. Normally, flashing code to a microcontroller is not a streamlined process, but thanks to the hard work on the Arduino Foundation and a globe-spanning community, all that is required to go from code to a blinking light is a USB cable and the push of a button.

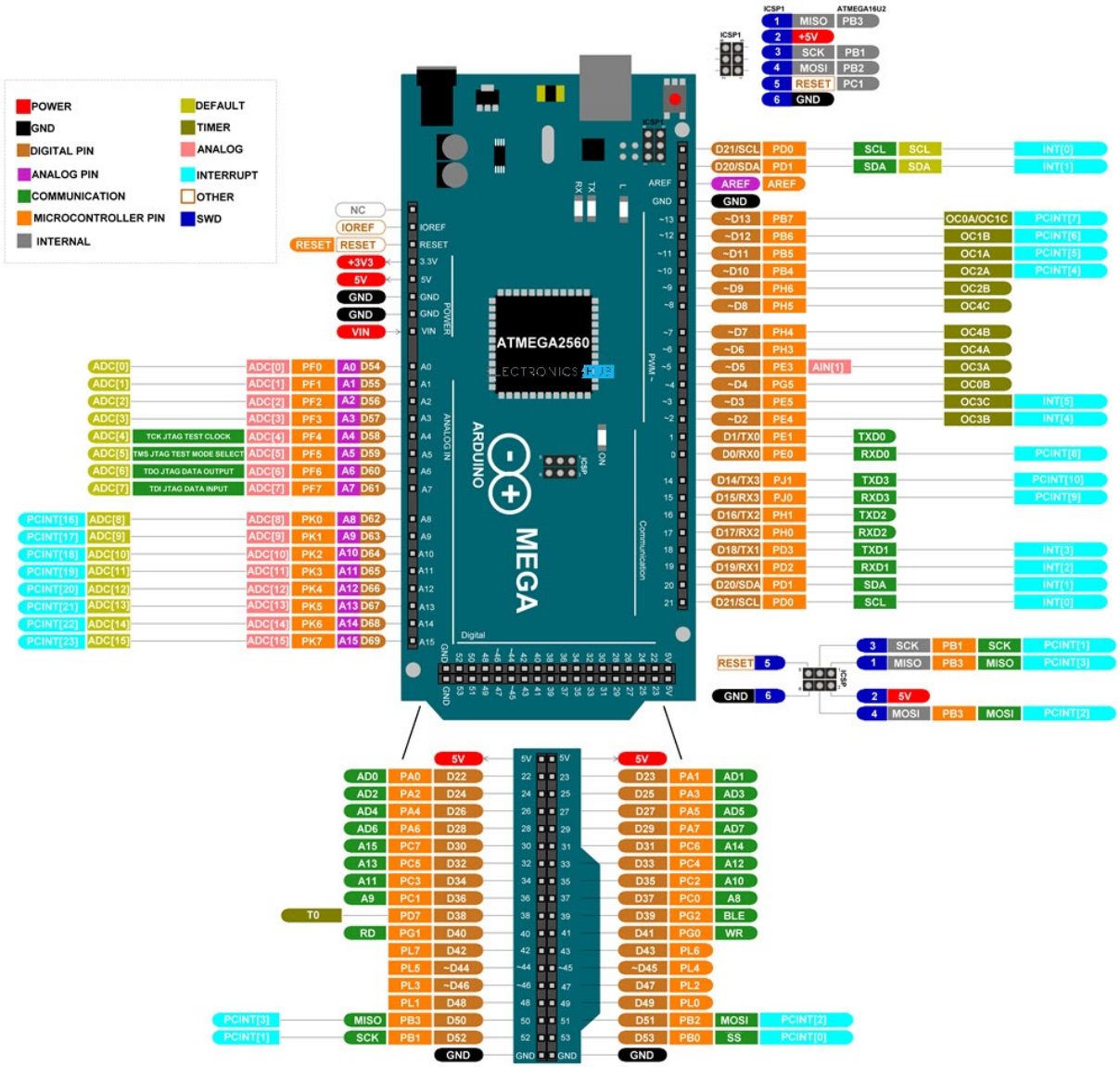



**Figure 1.10:** The pinout for the Arduino Mega. Retrieved from Electronics Hub

## MAX7219 LED Matrix Module

The MAX7219 is a common cathode LED driver IC that has a four wire serial interface compatible with all microcontrollers. The IC is capable of driving up to 64 LEDs making it perfect for driving dot matrix displays and 7-segment displays. It also has a built-in BCD decoder and 64-byte static RAM which makes it very easy to display numbers, letters, or symbols on a dot matrix or 7-segment display.

Additional information on the MAX7219 can be found here: `https://microcontrollerslab.com/max7219-8-digit-led-display-driver/`

For this particular kit, the MAX7219 drives an LED dot matrix with 8 rows and 8 columns (64 LEDs total). Each LED is addressable by its particular row and column number. As seen in Figure 1.12, the positive terminal of the LED "dot" is connected to a row pin, and the negative terminal of the dot is connected to a column pin. By setting a particular row to a positive

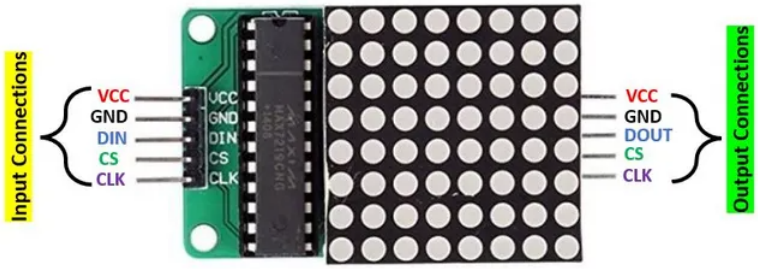



**Figure 1.11:** The pinout for the 8x8 LED dot matrix present in the Arduino Mega Starter Kit. Retrieved from Microcontrollers Lab

voltage and a particular column to ground, we can turn on a dot. This matrix layout is a form of multiplexing and allows a microcontroller to drive 64 LEDs with only 16 pins!

However, 16 pins is still a substantial amount for an embedded platform and computing the logic for which LEDs to turn on/off for a specific symbol wastes valuable compute cycles. It is much more efficient for us to use a dedicated IC like the MAX7219 to handle driving the matrix for us. The IC can use its built-in serial decoder to drive all 64 LEDs from 4 pins on the microcontroller. It also handles all of the logic for displaying specific symbols, relieving the microcontroller from additional calculations. Since the MAX7219 sole purpose is to drive LED displays, it can also run much faster allowing programs to display text that smoothly scrolls across the matrix with minimal interruption.

The MAX7219 serial bus also supports daisy chaining so multiple LED matrices can be connected in series to make a larger overall display.[2]

2: This does come at the cost of increasing input delay with every module. Too many modules chained together can have undesirable performance.

## DS3231 Real-time Clock Module

The DS3231 is a sophisticated real-time clock module that maintains a high time keeping accuracy over a long period of time and a range

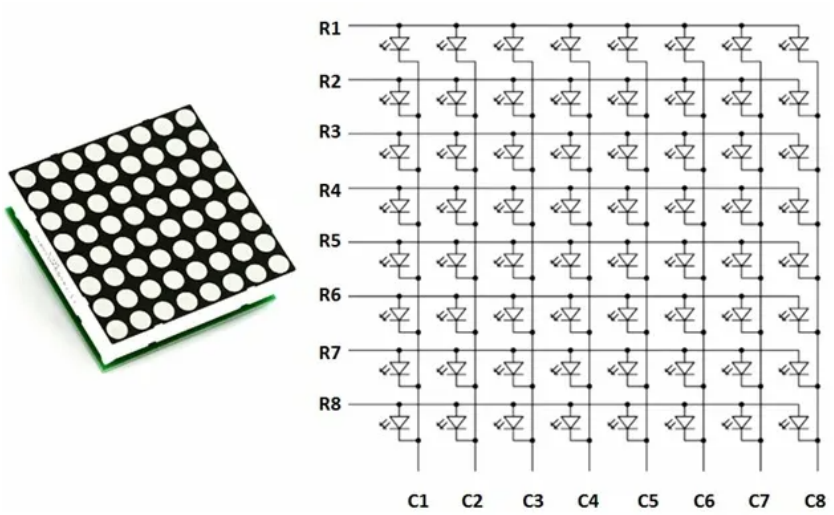



**Figure 1.12:** The schematic for the 8x8 LED dot matrix present in the Arduino Mega Starter Kit. Retrieved from Microcontrollers Lab

of temperatures. You can communicate with the DS3231 over the I2C bus which makes it extremely simple to get up and running with most microcontrollers. It also has a low power draw so it can be powered by a dedicated battery to ensure that projects have an accurate time keeper for data-logging, wake up interrupts, etc.

Once you initialize the date and time on the module, you will be able to routinely get the day, month, year, day of the week, hour, minute, and second from the IC over I2C. So long as it has power, it will provide you the time.

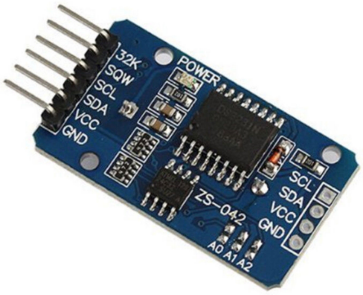



**Figure 1.13:** The DS3231 RTC module from the Arduino kit. Retreived from All Top Notch

## Matrix Keypad

The ELEGOO kit comes with a 4x4 matrix keypad with 16 individual keys. Much like the LED dot matrix, the keypad multiplexes the 16 buttons into four row pins, and four column pins, as shown in Figure 1.15. In order to use the keypad, you will "scan" the matrix by connecting each column pin to an input with a pull-up resistor, and each row to an output pin. By incrementally setting each row pin to low, and checking the input of each column pin, you can determine which button has been pressed. For example, if you are checking the second row of buttons by setting that pin to low and the third column pin is also low, than the button in position (2,3) (normally '6') is pressed.

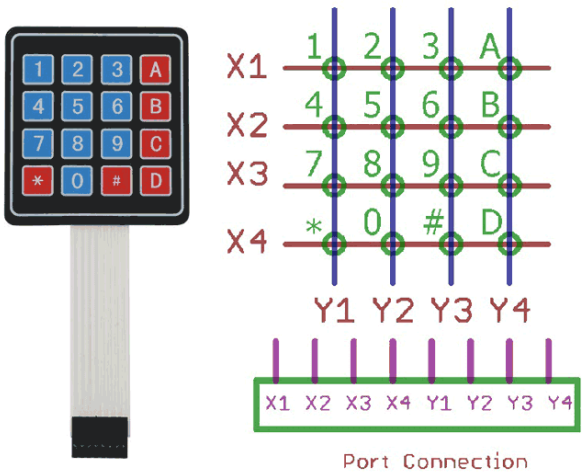



**Figure 1.14:** The pinout and schematic for the 4x4 matrix keypad included in the Arduino starter kit. Retrieved from Dhanesha Blogs

## PIR Motion Sensor

Passive Infrared (PIR) motion sensors detect levels of infrared radiation in their field of view. Inside a plastic bubble housing is a crystal that is sensitive to IR radiation and is split into two halves. If one half of the crystal detects a different level of IR radiation than the other, the PIR driver IC will trigger a change on the interrupt line. This allows a microcontroller to do basic human sensing applications for monitoring foot traffic in an area, or if someone has entered/left a room. Additionally, the PIR sensor has some

potentiometers that allow users to tweak the sensitivity of the module and the time delay between triggers.

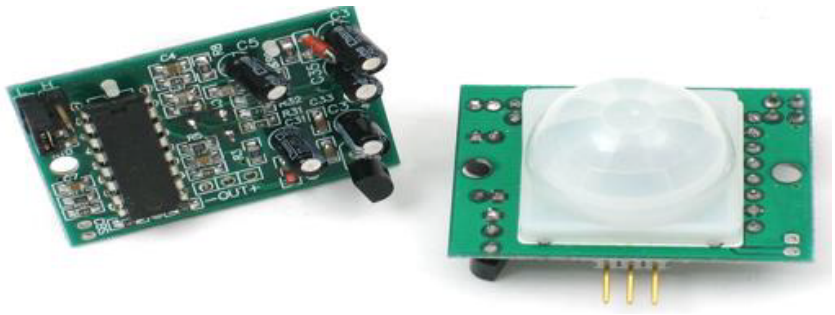


**Figure 1.15:** The PIR sensor module included in the Arduino kit. Retreived from Tutorials Point

## IR Receiver

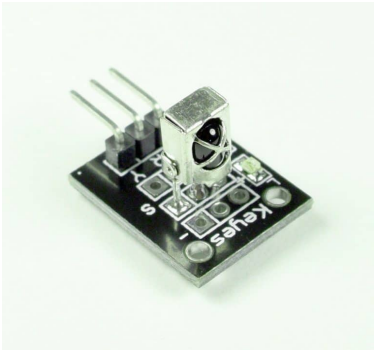


**Figure 1.16:** The IR receiver module included in the Arduino kit. Retreived from Circuit Basics

The IR receiver operates with the same physical principles as the PIR motion sensor, but has much simpler circuitry. Inside a small housing is a crystal that is sensitive to IR radiation. When the crystal is excited, it creates an electrical signal that can be read by a microcontroller on the output pin. All IR remotes (including the one in your Arduino kit) encode your button presses as flashes of IR light. When the IR receiver "sees" these flashes, it will convert them to an electrical signal with roughly the same timing as the remote output. This allows you to communicate with your microcontroller over light waves and signal it for different actions or functions.

This is the same exact way older TVs and AV receivers and DVD players function. Pressing a button on your remote would send an encoded signal to turn up or down the volume, change the channel, or fast forward a movie. Universal remotes simply were able to switch between different encoding schema for different devices. Can you decode the messages your TV receives?

You will become very familiar with this module in Project 4!

## LCD Screen

You kit comes with a neat module called the LCD 1602 Display. It can display text, images or icons in a 16-character by 2-column matrix. LCD stands for Liquid Crystal Display which is a technology that most screens in the world use. Each character in the display is comprised of a 5x8 matrix of pixels. A driver IC applies an electric field to each of those specific pixels, polarizing the liquid crystal masking filter above it. This allows light from the backlight to be filtered through so you can see the message written on the display. The data used to drive the LCD is sent from the Arduino over the parallel bus which sends multiple bytes over multiple lines *in parallel* to increase overall throughput.


**Figure 1.17:** The LCD1602 module included with the Arduino kit. Retrieved from Orient Display

## Joystick

The joystick is a ubiquitous human input device (HID). If you have every played with a game controller, you have used a joystick. It consists of two potentiometers: one on the x-axis and one on the y-axis, and a push button. These potentiometers are spring loaded to return to the center (neutral) position when no force is acting on the stick.[3] When you push the stick forwards or backwards, the y-axis potentiometer will change the voltage on the output pin, allowing the Arduino to detect movement. Same goes for left and right motion on the x-axis. Pushing the stick in will press the built-in button, adding another input method to the Arduino.

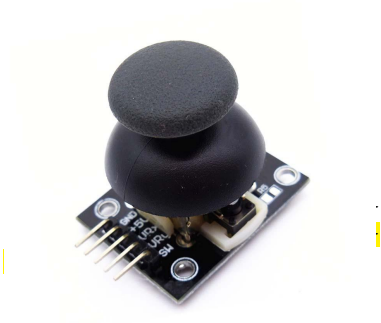


**Figure 1.18:** The PS2-style dual axis joystick included with the Arduino Kit. Retrieved from DH Gate

## Ultrasonic Distance Sensor

The ultrasonic distance sensor included with your kit is a time-of-flight based tool for detecting the proximity of objects. Much like how a bat or dolphin uses echolocation, this sensor emits a pulse of sound for a certain duration at a certain frequency. The sound is emitted from the transmitter, reflects off an object, and is heard by the receiver. By measuring the time between the pulse being sent out and the reflected pulse being received, and knowing the speed of sound through air, we can determine the distance to an object.

**Figure 1.19:** The ultrasonic distance sensor included with the Arduino Kit. Retrieved from Alexnld

## Sound Detection Module

The KY-038 microphone module is a capacitive microphone that is sensitive to frequencies from 50-10k Hz and has an amplification circuit on-board. This microphone converts sound pressure waves into an electrical signal

that the Arduino can read for various tasks. By fiddling with the sound level potentiometer on the module, a threshold sound amplitude can be controlled. When the microphone detects a sound above the set threshold, it will output a digital signal on an output pin. Additionally, there is an analog output pin which the Arduino can use to detect the amplified sound waveforms captured by the microphone. The pinout for this module is shown below.

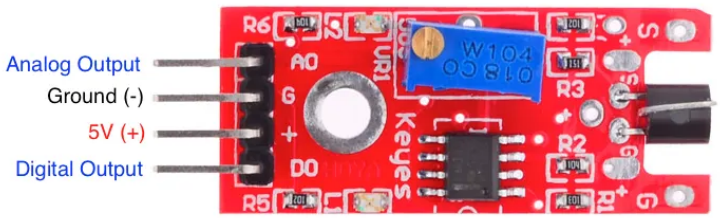



**Figure 1.20:** The KY-038 sound detection module included with the Arduino Kit. Retrieved from Microcontrollers Lab

## ULN2003 Motor Driver Board

The ULN2003 stepper driver IC is a common motor driver module for stepper motors. [4] This driver uses 4-inputs to determine what sequence to drive the stepper motor poles. By connecting the Arduino to these inputs and setting them high or low in a certain sequence, the ULN2003 module will "step" the motor, rotating it a precise and known amount. This can give you very precise control over rotational and linear motion. There are also a multitude of libraries available to help simplify working with this module and making precise movements.

4: You can learn more about stepper motors and how to drive them in the ACTUATORS SECTION

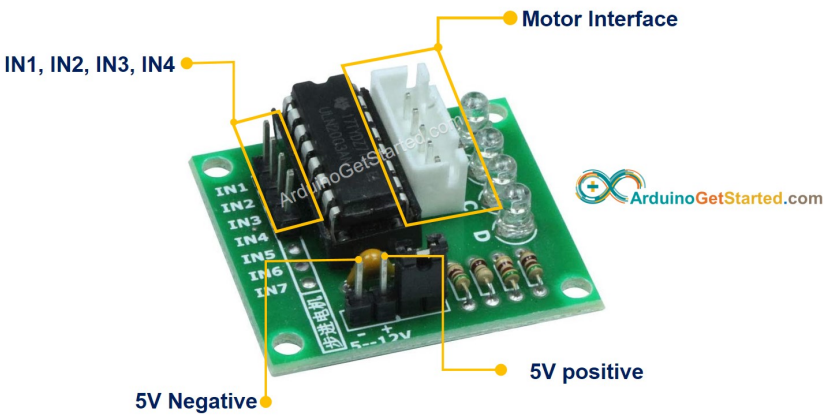



**Figure 1.21:** The ULN2003 motor driver board included with the Arduino Kit. Retrieved from Arduino Get Started

## DHT-11 Sensor

The DHT-11 sensor module features a humidity and temperature sensor combined into a signal package with a calibrated digital output. The

temperature sensor is an Negative Temperature Coefficient (NTC)-type that reduces its resistance as temperature increases. The humidity sensor is a resistive-type component that changes its resistance based on the ambient humidity. These sensors are connected to a small dedicated microcontroller through a amplifier network. The microcontroller reads in the sensor values and outputs the data over a single wire digital interface. Your Arduino can connect to that interface and get the ambient temperature and humidity using a multitude of libraries.

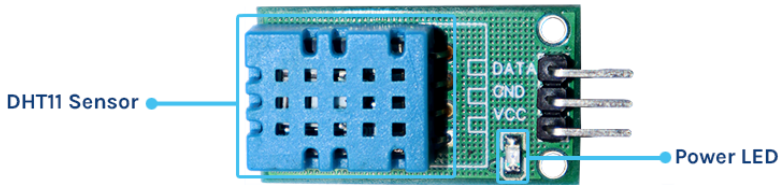



**Figure 1.22:** The DHT-11 temperature and humidity sensor included with the Arduino Kit. Retrieved from Circuit Digest

## Water Level Sensor

The water level sensor has a series of ten traces exposed on its PCB; five are power traces, five are sense traces. These traces are interlaced such that they are not connected until they are submerged in water and bridged together. As the water level increases, more sense traces are connected to more power traces and thus, the overall resistance of the circuit is decreased. This resistance is inversely proportional to the water level (i.e. as the water level increases, the resistance wll decrease). By measuring the voltage across the sensing pin and ground, we can correlate the water level to the voltage reading, much like we would with a potentiometer. One caveat though is that constantly powering the sensor while it is submerged will dramatically increase the galvanic corrosion and reduce the sensor's lifespan. Therefore the positive voltage pin should only be driven high when a reading is actively being taken.


**Figure 1.23:** The water level sensor included with the Arduino Kit. Retrieved from Arduino Getting Started

## 4-Digit 7-Segment Display

The 4-digit 7-segment display is a series of 7-segment displays connected together in parallel with a common cathode. Much like the keypad and LED dot matrices, this module can display different numbers and values based off which lines are turned on or off. There are four "digit" lines that

You will become very familiar with this module in Project 3!

are connected to the anode of their respective digit. Driving these lines high, while pulling one of the display LEDs low will cause that LED to illuminate. For example, to display a "0" in the first digit, you would drive the first digit pin high, and pull the LEDs pins A-F low, illuminating a "0".

On the Arduino, it can be inconvenient to connect so many data lines, so often, a shift register like the 74HC595 is used to drive the display LEDs, while the Arduino controls which digits are activated.

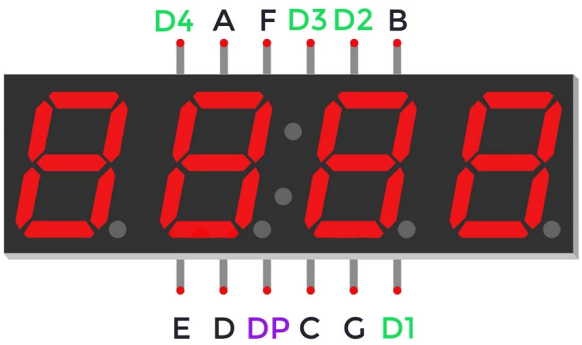



**Figure 1.24:** The 4-digit 7-segment display included with the Arduino Kit with pinout. Retreived from Make Crate

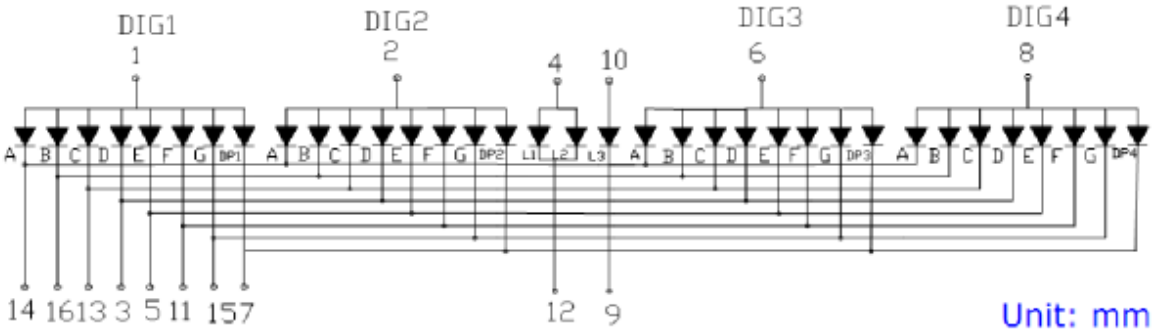



**Figure 1.25:** The inside schematic of the 4-digit 7-segment display. Retreived from All About EE

# Digital Electronics 2



Computers have existed in some fashion or another for millennia. The ancient antikythera mechanism was a hand-powered computer that predicted the position of astronomical bodies throughout the year. Mathematicians in the 18th and 19th centuries used complicated machines and mechanisms to run calculus and trigonometric calculations at speeds way faster than possible by humans by hand. Then, in the mid-1940's nations used analog computers to compute the trajectories of ballistic projectiles and free falling bombs. Today, analog computers are still used in niche applications, but since the transistor revolution of the 1960's and 1970's, almost every computer in the world has transitioned to digital logic.

## 2.1 Introduction to Binary

Digital logic is electrically simple to implement: either current flows (a logical '1') or it does not (a logical '0'). Even better, the circuit required to calculate digital logic can be generalized or reconfigured, depending on the architecture used. This makes digital logic a far more versatile tool compared to analog computers which can only be made for specific applications.

The fundamentals of computing are above the scope of these notes, but it is still important for engineers to understand the basic logic behind digital computations and know about the 1's and 0's that dictate our modern lifestyles

**What is Binary?** Binary is the representation of the state of an object. As the latin prefix implies, there are two states represented by binary: on (logical '1') or off (logical '0'). A lightbulb is a good example of a binary object - it is either on, lighting up a room, or off, leaving the room in darkness.

Binary is fundamentally a numbering system, much like the arabic numerals we use in our day-to-day lives. Arabic numerals are in Base10, meaning we comprehend counting in multiples of 10 using the digits 0-9. For example:

$$\begin{aligned} 20 &= 10 * 2 \\ 450 &= 4 * 10 * 10 + 5 * 10 \\ 245683 &= 2 * 10^5 + 4 * 10^4 + 5 * 10^3 + 6 * 10^2 + 8 * 10^1 + 3 * 10^0 \end{aligned}$$

Binary is counted in Base2, meaning everything is counted in multiples of

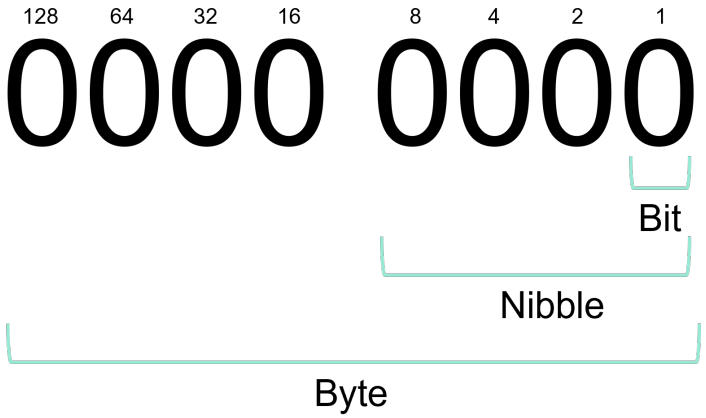



**Figure 2.1:** Breakdown of a byte from the largest unit, to the smallest, with indexing.

2 using the digits 0 and 1. Using the same example as before:

$$20 = 2^4 + 2^2$$
$$450 = 2^8 + 2^7 + 2^6 + 0^5 + 0^4 + 0^3 + 0^2 + 2^1 + 0^0$$
$$245683 = 2^{17} + 2^{16} + 2^{15} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^5 + 2^4 + 2^1 + 2^0$$

While this is inefficient for a human to understand its meaning, a computer can quickly transform this on/off pattern of electrons into numbers for calculations.

Binary numbers are broken down into three groupings: bits (one number), nibbles (four bits), and bytes (two nibbles or eight bits). The groupings are extremely important as the order and number of the bits in a binary representation dictate the number or logical input used in the calculation. Each bit is indexed starting from 0, as shown in FIGURE and goes from the Least Significant Bit (LSB) at index 0, to the Most Significant Bit (MSB) at the furthest index. [1]

1: These indecies get their names because of how they affect the binary number when changed. The LSB only changes the numerical value by 1. The MSB, on the other hand, can change the value by upto an infinite value, depending on how many bits are present in the binary number

**Example 2.1.1** Let's take a look at a practical demonstration of binary using your Arduino kit. You will need:

- 1 x Red LED
- 1 x button
- 1 x 220 Ω resistor
- 2 x Male-to-Male jumper wires

Wire these components to a breadboard like so:

The dfault state for this system should be active high, meaning the LED will be off when the button is not pressed - a logical 0. When you press the button, you push the system to the high state and turn on the LED - a logical 1!

### 2.1.1 Binary Arithmetic

Much like our beloved arabic numerals, binary numbers can be added, subtracted, multiplied, and divided.

**Addition** Adding binary numbers is straight-forward. Simply stack the numbers you wish to add on top of each other, ensuring the indices align. Note, binary addition is commutative, meaning the number order does not matter, so long as the indices are aligned. Then, for each index, add the numbers together. If the result is greater than 1, then set the result of the index addition to 0 and carry a '1' over to the next index. Repeat until you have calculated the last index!

**Example 2.1.2** (Adding binary numbers) Let's add $7_{10}$ ($0111_2$) and $11_{10}$ ($1011_2$):

$$\begin{array}{ccccc} {\scriptstyle 1} & 0^{1} & 1^{1} & 1^{1} & 1 \\ + & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 \end{array}$$

**Subtraction** Subtracting binary numbers is simply the same algorithm for Base10 numbers. Starting at the LSB, subtract the two numbers index-by-index. If a case arises, when the subtrahend [2] has a larger number than the same index in the minuend [3], borrow a '1' from the next index. If you are are unable to borrow a '1', grab one from the next index until you can get a '1' to bring back. When you bring a '1' back, then perform the subtraction. Repeat this process until you have calculated the last index.

2: bottom number

3: top number

**Example 2.1.3** (Subtracting binary numbers) Let's subtract $24_{10}$ ($11000_2$) and $7_{10}$ ($111_2$):

$$\begin{array}{ccccccc} & 1 & 1^{0} & 0^{10^{1}} & 0^{10^{1}} & 0^{10^{1}} \\ - & 0 & 0 & 1 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 & 1 \end{array}$$

**Aside: 2's Complement Method**

The above method is easy for humans to understand, but computationally expensive. For operations that require borrowing, many repetitive borrow operations are performed, wasting CPU cycles. Instead, computers use a complement method to simplify the subtraction operation and increase performance.

Here, we are going to perform the same calculation as before. Begin by aligning the two numbers as previously, "padding" [4] both numbers so they are the same length.

4: adding leading zeroes, as necessary

| | | | | | |
|---|---|---|---|---|---|
| | 1 | 1 | 0 | 0 | 0 |
| - | 0 | 0 | 1 | 1 | 1 |

Then, switch all of the bits in the subtrahend. In this case, 00111 becomes 11000. [5] Then, we are going to add 1 to the switched term:

$$11000 + 1 = 11001$$

We are then going to change the math problem from subtraction to addition and add the original minuend and the new, switched subtrahend.

| | | | | | |
|---|---|---|---|---|---|
| | 1 | 1 | 0 | 0 | 0 |
| + | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |

To finish the algorithm, remove the MSB from the resultant. You can compare this number to the previous calculation and notice that the final result is the same, depite the radically different method. Even though the complement method is unintuitive for humans to understand, it is vastly superior for a computer to execute as the number of required operations is dramatically smaller than the original method, even with larger values.

5: What is occuring here is known as the 1's complement as we are subtracting '1' from each bit in the number. This only works in binary

**Multiplication** Binary multiplication occurs in the same order as decimal long multiplication. It is a commutative process, like addition, but in order to reduce the number of steps required, it is recommended that the number with the least amount of bits is the multiplicand [6] and the other factor is the multiplier [7]. For the operation, you will multiply the multiplier by each bit of the multiplicand to calculate an intermediate value. With every new index of the multiplicand, shift the resulting product over one bit until you have calculated the intermediate product for the last index of the multiplicand. Simply add all the intermediate products together to calculate the final result.

6: bottom number

7: top number

**Example 2.1.4** (Multiplying binary numbers) Let's multiply $14_{10}$ ($1110_2$) and $3_{10}$ ($0011_2$):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | 1 | 1 | 1 | 0 | |
| | | * | 0 | 0 | 1 | 1 | |
| | | | 1 | 1 | 1 | 0 | |
| | | 1 | 1 | 1 | 0 | | |
| | 0 | 0 | 0 | 0 | | | |
| 0 | 0 | 0 | 0 | | | | |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | |

This leaves us with the answer to the life, the universe, and everything: $0101010_2$ ($42_{10}$).

**Division** Binary division works exactly like decimal long division. You start with a dividend within the division operator and a divisor operating on it. Compare the divisor to the first digit of the dividend; if the divisor is larger, than a '0' is placed above the first index of the dividend, if it is smaller, than a one goes there. If the latter case occurs, subtract the section of the dividend from the dividend and continue the dividing operation with this new value.

**Example 2.1.5** (Dividing binary numbers) Let's divide $69_{10}$ ($01000101_2$) by $3_{10}$ ($0011_2$):

$$
\begin{array}{r}
00010111 \\
0011\overline{)\,01000101} \\
-11\phantom{000} \\
\hline
101 \\
-11 \\
\hline
0100 \\
-11 \\
\hline
11 \\
-11 \\
\hline
0
\end{array}
$$

This is the exact same operation performed in Base10 long division, just everything is in Base2. As you can see, the division leaves us with the correct answer of $10111_2$ ($23_{10}$).

## 2.2 Introduction to Logic Gates

Logic gates are the building blocks of digital circuits. An arrangement of gates grants you a specific output for corresponding sets of inputs. This allows you to "construct" logical equations with visual representations or even physical devices.

### 2.2.1 Truth Tables

It is sometimes inefficient to display a logic flow with a diagram of gates and wires, as shown in FIGURE. . So, these flows can be summarized by a truth table as show in in TABLE. This shows the input parameters and the corresponding outputs. These tables can also be helpful in designing the logic diagram when given a problem.

Create a figure a bunch of logic gates wired together

To setup this table, dedicate a column to each input into your system and have a column for the outputs. The LSB will be the column before the output column and the columns will increase in significance to the left. Then, for every row in the input columns, start at 0 and alternate between 0 and 1 at a specific interval. This interval is dictated by the index (column) of the input bit. The LSB will switch every row (0,1,0,1, etc.), the next bit will switch every other row (0,0,1,1, etc.), the next bit will alternate every

| Sensor 4 | Sensor 3 | Sensor 2 | Sensor 1 | Output |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

4 rows (0,0,0,0,1,1,1,1, etc.), and so on and so forth. Then, in your output column, place a 1 where you want the gates to output a logical 1 given a set of inputs.

For example, lets design an alarm system for a fire place. This specific system has four alarms of various types that output a logical 1 when they detect excessive smoke from the fireplace. However, these sensors are known to be a little too sensitive and give false positives. To prevent the fire department from unnecessarily coming to the rescue, we will not sound the general alarm until two or more sensors detect excessive smoke. Therefore, we will construct a truth table like so:

As you can see in Table **??**, for every row where two or more sensors are sending an alarm signal, the output (general alarm) is sending out a signal as well.

### 2.2.2 Basic Logical Operators

In logic, we can write a flow like an equation with a series of operators and notations that determine what outputs will result from given inputs. The basic operators are **AND**, **OR**, and **NOT**.

**AND** is a basic logical operation. If the input 1 *and* input 2 are both true (logical '1'), then the output is also true. Otherwise, the output is false. This can be represented by the equation below where $A$ is input 1 and $B$ is input 2.

$$A \cdot B$$

And the truth table:

| $A$ | $B$ | $A \cdot B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Graphically, AND gates are represented by symbols like Figure **??**. Wires protrude from the three pins on the gate and "carry" information to and from other gates on the same diagram.

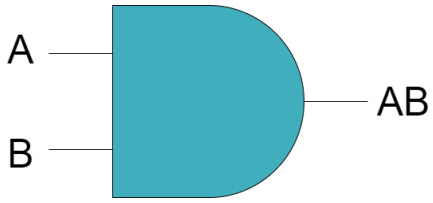



**Figure 2.2:** Visual representation of the AND logical operation with two inputs and one output.

**OR** is another basic logical operation. It is the inverse of the AND operation as the logical output will be true when any input is true i.e. input 1 *or* input 2 is true. It can be represented by the expression:

$$A + B$$

With the truth table:

| $A$ | $B$ | $A + B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

And the visual symbol:

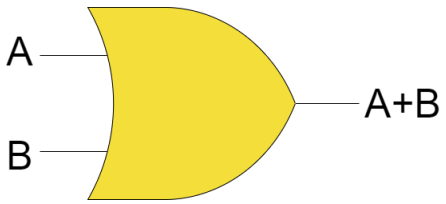



**Figure 2.3:** Visual representation of the OR logical operation with two inputs and one output.

**NOT** is the simplest logical operation of all. Simply, whatever input is passed into the NOT gate, the output will be the opposite. For example, a logical true input will result in a logical false output, and vice versa. This can be represented by the truth table:

And the visual symbol:

The NOT operation can be represented by either $\bar{A}$ or $!A$ where ! is called the "bang" operator. The latter operator is widely used within computer programming for the NOT operation, whereas $\bar{A}$ is more used in mathematical and handwritten expressions.

| $A$ | $\bar{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

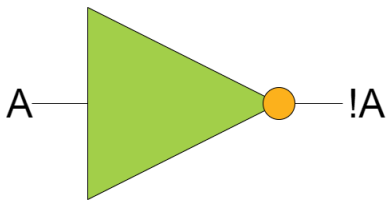



**Figure 2.4:** Visual representation of the NOT logical operation with one input and one output.

**Example 2.2.1** (Basic Logic Gates 1) Let's practice the basic logic gates. We will consider a scenario with four inputs and we only want the logical output to be true when inputs A and B and (C or D) are true. We can set this up as the expression:

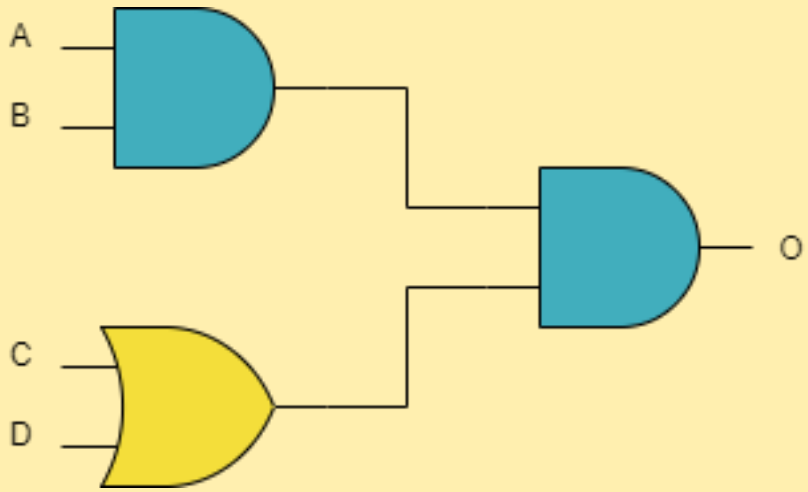
$$(A \cdot B) + (C + D)$$

We will then establish the truth table as:

| Input D | Input C | Input B | Input A | Output |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Here, we can easily see that there are 3 input sets that will allow a logical true output in our scenario.

Next, we can create a visual representation of this scenario:




While the required input set to get a logical true output is not obvious, it is a little easier to understand the "logic flow" through our scenario and we can test different input ideas and comprehend what is occurring a little easier.

**Example 2.2.2** (Basic Logic Gates 2) In this scenario, we are going to add a NOT gate and change the output conditions. Here we are looking for A and B or not (C or D). As before, we set up the scenario as the expression:

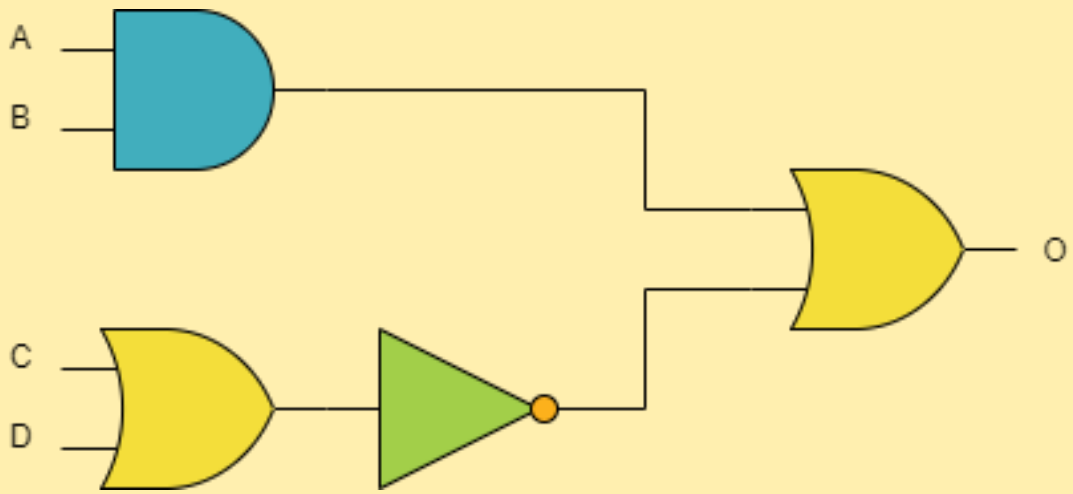
$$A \cdot B + \overline{(C + D)}$$

We can also build the truth table:

| Input D | Input C | Input B | Input A | Output |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

We can see from the table that we have several more conditions that will result in a true output from our expression!

If we create a diagram of this scenario, we get the following:



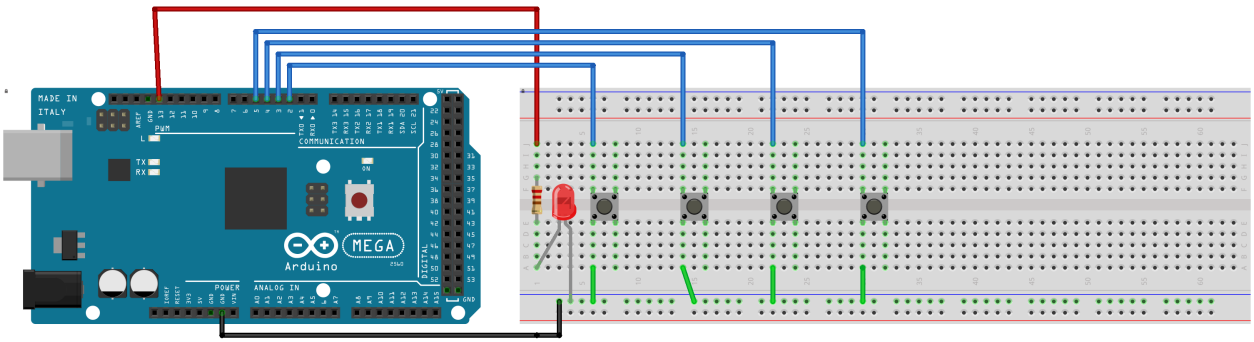
**Arduino Example: Basic Logic Gates**

Let's practically demonstrate the circuit found in Example 2.2.2 using your Arduino Kit!

You will need:

- 4 Buttons
- 10 Male-to-Male jumper wires
- 1 220-Ω resistor
- 1 Red LED

Connect these components together in the following circuit:




Then, we will use the following code to turn on or off the LED according to our button inputs.

```
#include <Arduino.h>

#define A 2
#define B 3
#define C 4
#define D 5

uint8_t inputPins[4] = {A, B, C, D};

void setup() {
    for (uint8_t i=0; i<4; i++) {
        pinMode(inputPins[i], INPUT_PULLUP);
    }

    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    if (!digitalRead(A) & !digitalRead(B) | !(!digitalRead(C) | !digitalRead(D))) // Check logic
    expression
        digitalWrite(LED_BUILTIN, HIGH); // Turn on LED
    else
        digitalWrite(LED_BUILTIN, LOW); // Turn off LED
    delay(125);
}
```

Let's break down this code to understand what is occurring. First, we define the input pins for our four input buttons: A, B, C, and D. The C++ pre-processor command `#define` essentially substitutes all calls of the variable

name with the value we define in the second parameter. Ergo, whenever we have `A` passed as a variable, the code will automatically substitute it with the value 2. We also create an array of all these pin numbers so we can more easily initialize them later.

```
#include <Arduino.h>

#define A 2
#define B 3
#define C 4
#define D 5

uint8_t inputPins[4] = {A, B, C, D};
```

Then, we can define `setup()` which is the first function the Arduino will call when it boots up. Any code that needs to be initialized first needs to be included within this function! Here, we have a simple for-loop that iterates through the pin array we defined earlier to initialize them all as `INPUT_PULLUP`. This particular mode means the Arduino is able to take digital input signals on this pins and pull up these inputs to 5V using internal resistors. When we press a button, the pins are pulled low, giving a clear distinction between not pressed and pressed. We also set the builtin LED pin to an output so we can turn it on or off.

```
void setup() {
    for (uint8_t i=0; i<4; i++) {
        pinMode(inputPins[i], INPUT_PULLUP);
    }

    pinMode(LED_BUILTIN, OUTPUT);
}
```

This finally brings us to `loop()` where we implement the actual digital logic. We first encounter the if-statement that is our logical expression for this example. It is important to note that here, `&` is the bitwise AND operator, `|` is the bitwise OR operator, and the `!` is the bang or bitwise INVERT operator. Since this circuit is an active-low configuration, we must invert the readings from the input pins hence `!digitalRead(A)`. If our logical expression is found to be true, we turn the LED on; otherwise, we turn it off using `digitalWrite(LED_BUILTIN, HIGH/LOW)`. We then implement a delay of 125 milliseconds which is useful for debouncing the button inputs.

```
void loop() {
    if (!digitalRead(A) & !digitalRead(B) | !(!digitalRead(C) | !digitalRead(D))) // Check logic
    expression
        digitalWrite(LED_BUILTIN, HIGH); // Turn on LED
    else
        digitalWrite(LED_BUILTIN, LOW); // Turn off LED
    delay(125);
}
```

### 2.2.3 Combination Logical Operators

As we can see from the previous example, we can combine gates together to change their logical outputs accordingly. Adding a NOT gate to the first OR gate gave several more conditions that would make the system output a logical true value. However, it can be unnecessarily clumsy to draw a gate immediately followed by a NOT gate, so we can combine them together into a single gate.

**The NAND operator** is a combination NOT and AND (NOT-AND, NAND) operation. This means the operation outputs a logical true value only when both input 1 *and* input 2 are *not* true. This can be represented by the expression and truth table below:

For you computer storage afficionados out there, NAND flash memory is one of the predominant architectures for high speed, solid state drives.

$$\overline{(A \cdot B)}$$

| $A$ | $B$ | $\overline{(A \cdot B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

This gate is graphically represented by the symbol. As you can see, the NAND gate is distinguished from its former identity by the circle at the front of the gate. You may also notice that this is the same circle present at the tip of the NOT gate, clearly relating the two.

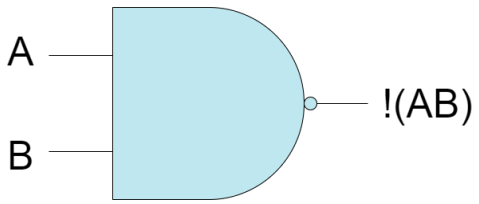



**Figure 2.5:** Visual representation of the NAND logical operation with two inputs and one inverted output.

**The NOR gate** is a combination NOT and OR (NOT-OR, NOR) operation. The bottom symbols in Example 2.2.2 can be combined into this single operation to achieve the same effect in a more space-efficient manner. This gate has a similar equation to its predecessor, looking like:

$$\overline{(A + B)}$$

It is represented by the truth table:

And the graphical representation below. Again, notice how the only distinguishing factor between a NOR and OR gate is the circle at the tip of the gate.

| $A$ | $B$ | $\overline{(A+B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

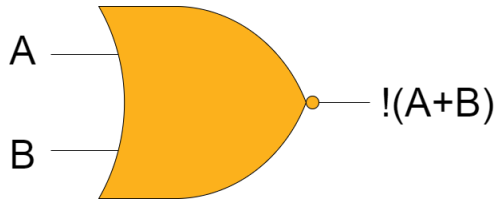



**Figure 2.6:** Visual representation of the NOR logical operation with two inputs and one inverted output.

### 2.2.4 Advanced Logical Operators

Sometimes it is desirable to have some more limited conditions in which an operation will output a logical true.

**The exclusive OR operation** is an operation that only outputs a logical true value when both inputs are different and not the same. This gate is represented by the expression:

$$A \oplus B$$

This operation can also be shown by the truth table and gate visualization below:

| $A$ | $B$ | $(A \oplus B)$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Table 2.1:** The truth table for the XOR gate.

As we can see from the truth table, this operation can be best explained as "*either or, but never both*"

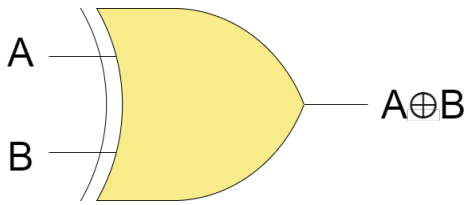



**Figure 2.7:** Visual representation of the NOR logical operation with two inputs and one output.

Note that the visual gate for the exclusive OR is differentiated from its basic counterpart by the additional arc at the back of the gate.

**The XNOR operation** is just like previous combination operations where we combine the XOR operation with a NOT operation. This changes the input requirements for a logical true output to both inputs have to be the same, rather than they have to be different. This operation is created by combining multiple operations ike so:

$$\overline{(A+B)} \cdot (A \cdot B)$$

And again, these complex combinations can be simplified by adding the $\oplus$ operator an negating it:

$$\overline{(A \oplus B)}$$

This results in a truth table and graphical representation shown below:

| $A$ | $B$ | $\overline{(A \oplus B)}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

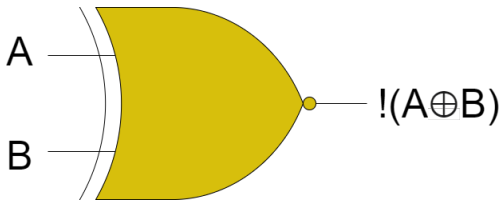



**Figure 2.8:** Visual representation of the XNOR logical operation with two inputs and one inverted output.

## 2.3 Boolean Algebra

Boolean algebra can be sed to formalize expressions for logic states. For these expressions, $A$ and $B$ are binary logical values, meaning they represent either a logical '1' or logical '0'. $Q$ is representative of the logical output, again either a '1' or '0'. Recall the following basic logical operations, multiplication denotes the AND operation, addition denotes the OR operation:

$$A \cdot A = A, A + A = A, A + \overline{A} = 1, A \cdot \overline{A} = 0, \overline{\overline{A}} = A$$

Using these basic definitions, we can define all of the logic gates described above as:

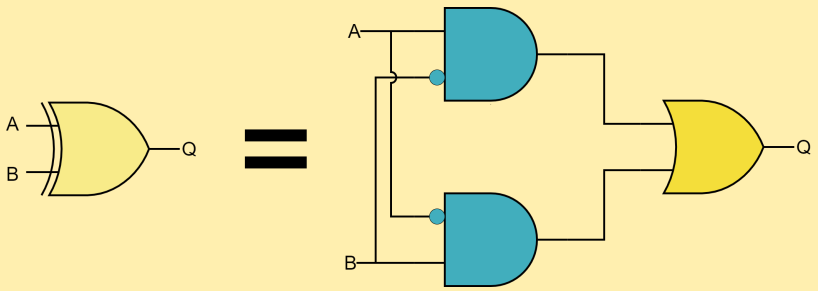
AND: $Q = A \cdot B$
OR: $Q = A + B$
NOT: $Q = \overline{A}$
NAND: $Q = \overline{A \cdot B}$
NOR: $Q = \overline{A + B}$
XOR: $Q = A \oplus B$

XNOR: $Q = \overline{A \oplus B}$

**Example 2.3.1** (Logic Gate Equivalency) In this example, we will be breaking down the XOR gate into it's composite boolean algebra expressions and logic gates. From Table **??** we can see this gate can be represented by the boolean algebra equation:

$$Q = A \cdot \overline{B} + \overline{A} \cdot B$$

This garners the logic gate equality shown below:




### 2.3.1 Logical Optimization

It is possible to represent any logical problem using an expression containing the appropriate order of AND and OR operations. However, for arbitrary truth tables and problems, this can create a lengthy process that is cumbersome, hard to write and follow, and can lead to extensive errors. This expression can also be needlessly inefficient and computationally expensive for calculators. Therefore, we can implement some strategies to reduce the computational load and optimize an arbitrary expression so we can improve computational efficiency and performance.

For this process, two methods can be used: algebraic optimization and Karnaugh mapping. The former can be done by hand for relatively simple problems, or by a computer analysis when there are a large number of binary inputs. The latter is most commonly done by hand and drastically reduces the size of a truth table. This can make the logical problem easier to understand and represent. However, this method is limited to a maximum of four binary inputs.

Let's examine a problem where we are given the arbitrary truth table shown in Table **??**.

We can begin exploring the algebraic optimization by writing the algebraic expression for the true values of the table:

$$Q = \overline{ABCD} + C + A + AC + AB + ABD + ABC + ABCD$$

Using algebraic simplification:

| Input D | Input C | Input B | Input A | Output |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**Table 2.2:** An arbitrary truth table ripe for optimization

$$
\begin{aligned}
Q &= \overline{\cancel{ABCD}} + \cancel{ABCD} + CD + A(1 + C + B + BD + \cancel{BC}) \\
&= C + A(1 + C + B + BD) \\
&= C + A + \cancel{AC} + AB + BD \\
Q &= C + A + AB + ABD
\end{aligned}
$$

Here, you will notice some AND expressions have been cancelled out (re: $BC$ and $AC$). This has occurred because the expressions $B + C + BC$ and $A + C + AC$ simplify down to the first OR operation. Let's explore this by drawing out the truth table for these expressions:

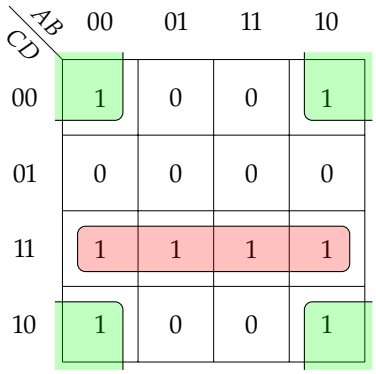
| $B$ | $C$ | $B + C + BC$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

You will notice that this truth table is the exact same for the OR gate (Table **??**) and therefore proves the simplification and cancellation of the AND operation.

This expression is complicated and would require three OR gates, and three AND gates to represent. Let's see if we can simplify this using a Karnaugh map.

**Karnaugh Maps** or K-maps, are condensed representations of truth tables that combine up to four distinct inputs into a two-dimensional table. These not only condense truth tables into a simpler form for easier understanding, but they can simultaneously simplify the algebraic expression for the truth table. K-Maps have a special property where if any logical true outputs that are grouped together in powers of 2 can be combined into a simpler AND operation.

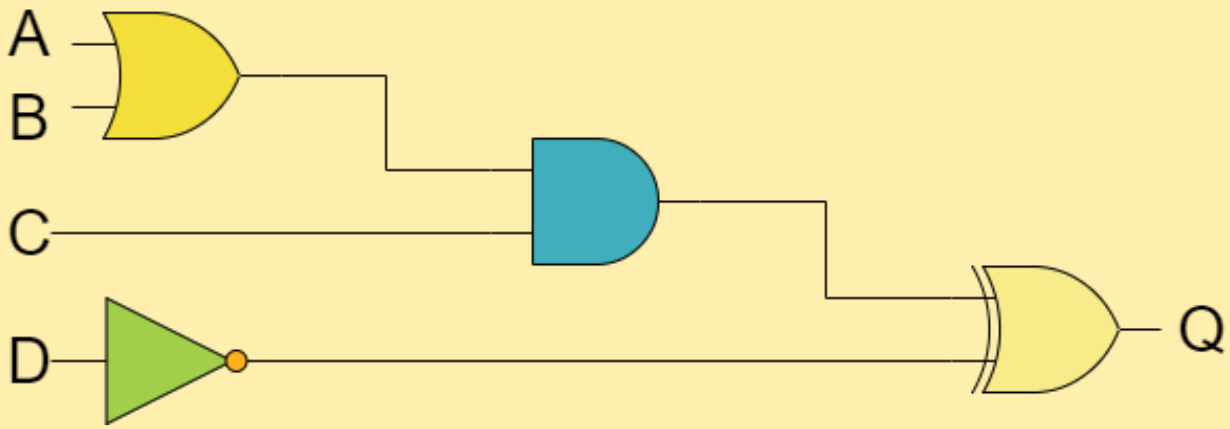
For the example found in Table **??**, the corresponding K-Map would be:




The highlighted groupings mean the truth table from before can be represented by the algebraic expression:

$$Q = AB + \overline{BD}$$

**Example 2.3.2** (Applied Logic Gate Simplification) For this example, we will be combining everything we have learned through this section into a single problem. We will start with the logic gate diagram shown below:




We can represent these gates with the expression:

$$Q = (A + B)C \oplus \overline{D}$$

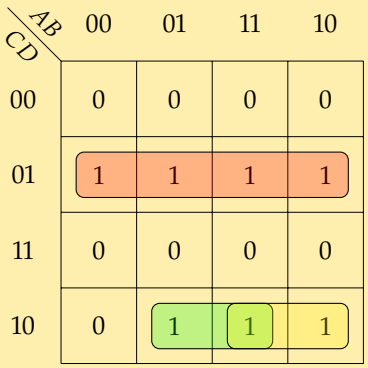
Which expands to:

$$Q = \left((AC + BC)\overline{D}\right) + \left((\overline{AB} + \overline{C})D\right)$$

This expression is a little complicated and is not ideal to determine what inputs will result in our desired outputs. Therefore, we will construct a truth table to represent these logic gates so that we can directly check a given output with an expected output.

| Input D | Input C | Input B | Input A | Output |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Both the representative expression and truth table are a little large and inefficient. If we look at the representative expression, we will notice that it is difficult, if not impossible, to algebraically simplify by hand. So, we will simplify the truth table with a Karnaugh map and determine the simplified algebraic expression from there.




From the K-map, we can simplify the algebraic expression for the logic diagram as:

$$Q = \overline{C}D + BC\overline{D} + AC\overline{D}$$

This expression is the most basic form for the logic diagram and now we can check any arbitrary value of $A$, $B$, $C$, or $D$ without have to directly look it up in the truth table or run it through the logic diagram.

## 2.4 Multiplexing and Demultiplexing

# Electrical Schematics 3

One of the key pieces of instrumentation design is the electrical schematic and Printed Circuit Board (PCB).



Set chapter image to a schematic OR paste a big image of a goo schematic somewhere on this page

## 3.1 Schematic Basics

The electrical schematic is a key piece of documentation that communicates to engineers what the circuit is comprised of and how it will work. All modern electronic devices have schematics of varying complexity and depth that describe how electrons flow from one piece to another and document what behaviors can be expected during operations. To understand these documents is to understand how a product fundamentally works, and to understand how to make these documents well is a skill that needs to practiced and refined by reviewing schematics and having yours reviewed. For now, we will go over the basics and give you a fundamental understanding of what the schematic entails and what most of the symbols you might encounter mean.

## 3.2 Basic Notation

## 3.3 Component Symbols

The core of the schematic are the 2D representations of the components in the design called, *symbols*. There are a variety of symbols that all mean different things and can have slightly different meanings depending on the drawer and application. The symbols covered henceforth are the standard-accepted versions so they should be what you see in most schematics you may encounter or be asked to work with or design.

### 3.3.1 Power Symbols

Power symbols are the set of symbols that indicate where the electrical power is coming from, and where it is going.

#### The Power Source Rail

the power source rail is typically denoted by a vertical "T" or arrow as shown below, and can be labelled with 3V3,[1] 5V, VBAT, VBUS, or any other source name.

1: Sometimes, it is easier to use a letter to represent a decimal point. In this case, 3V3 is 3.3 volts.

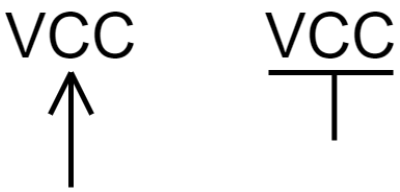



**Figure 3.1:** Schematic symbols for power rails. The value label at the top can be any value like 3V3, 5V, etc.

### The Power Sink Rail

The power sink rail is the "ground" path from electrical energy will flow back to the negative terminal of the power supply. It is typically represented by a vertical "⊥" or upside down tree as shown in FIGURE. It should always be labelled "GND" with sometimes a prefix of "A" or "D" to denote a different analog or digital ground reference.

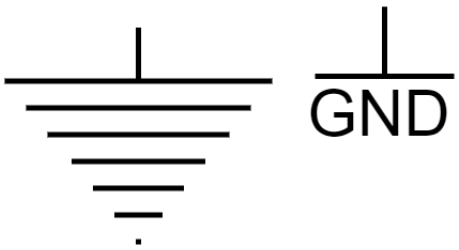



**Figure 3.2:** Schematic symbols for ground rails.

### Power Supplies

are generally represented by a circle with a positive and negative input. In order to differentiate a DC supply and AC supply, we can use either a plus/minus or sine wave, respectively. Both of these symbols are represented in the figure below.

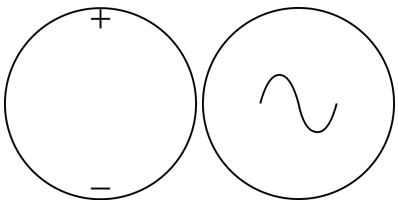



**Figure 3.3:** Schematic symbols for DC and AC power sources, respectively. Notice how the AC power source uses a sinusoidal wave to differentiate it.


**Figure 3.4:** 18650 Li-ion battery. Retrieved from Himax Electronics

**The Battery** is a subset of DC power supply for embedded applications or circuits that will not have access to a general DC power supply. These components are meant to store a large amount of electrical charge for a long duration and be able to discharge it into a circuit to run it. Some batteries, like Lithium Ion batteries, are capable of being recharged using special integrated circuits and power supplies. Others, like alkaline batteries, are only usable once and must be disposed of properly when they are depleted.

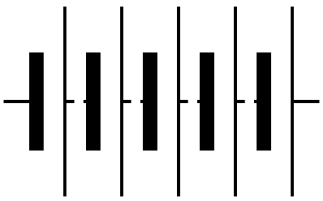


**Figure 3.5:** Schematic symbol for a battery. The different sized lines that comprise the battery symbol represent the different plates that are within modern battery construction.

### 3.3.2 Basic Components

The following basic component symbols represent discreet components within an electrical circuit. Most of these components perform a single service within the circuit and must be used in conjunction with other basic parts to achieve an action.

#### Resistors

Resistors are components that resist the flow of current and can be used to dissipate energy, reduce voltage levels, or limit current going into other components. There are represented with a jagged line or box with "whiskers" as shown in the figure below.

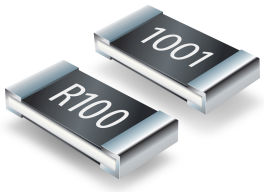



**Figure 3.6:** Surface mount chip resistors. Retrieved from MPDigest

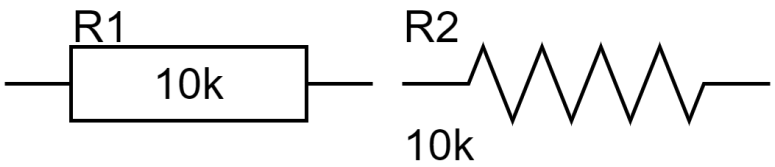



**Figure 3.7:** Schematic symbols of resistors. European standard is the left, US standard to the right.

**The Potentiometer** is a variable resistor that changes resistance depending on the position of "wiper" across a band of resistive material. Typically, these components are turned by human operators and can be used as an input to determine a variety of things, depending on application.

**Figure 3.8:** Panel mount potentiometer. Retrieved from Phipps Electronics


**Figure 3.9:** Schematic symbols for potentiometers. Notice the arrow through the resistor to demark it from standard resistors. The value at the bottom is the maximum resistance.

**Photoresistor** The photoresistor is a variable resistor that changes resistance depending on the intensity of light interacting with it Typically, these components are used as basic light sensors that can be used for presence detection, sun tracking, or other light intensity measurement.

**Figure 3.10:** LDR photoresistor. Retrieved from Pixel Electric


**Figure 3.11:** Schematic symbols for photoresistors. The arrows pointed towards the resistor indicate its light dependency.

### Capacitors

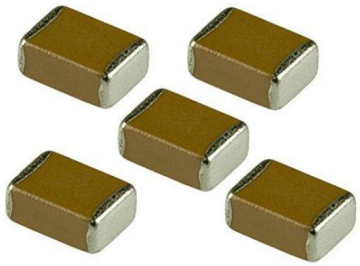


**Figure 3.12:** Chip ceramic capacitors. Retrieved from Universal Solder

Capacitors are components that store and discharge electrical energy. They use parallel conducting plates isolated from each other with a dielectric that allows charges to build up on one face at a certain voltage, then discharge when at a lower voltage. This gives capacitors the ability to smooth ripples in voltage levels (called a decoupling). This is commonly used to decouple components from electrical noise upstream of their power supply, giving them a cleaner electrical input. Additionally, they can provide a high current energy source that has some advantages over traditional batteries. In most circuits, you will encounter ceramic capacitors (pictured to the side) that do not hold a log of charge, but work for most applications. These capacitors are bi-directional or non-polarized so they can be placed in any orientation on the schematic.

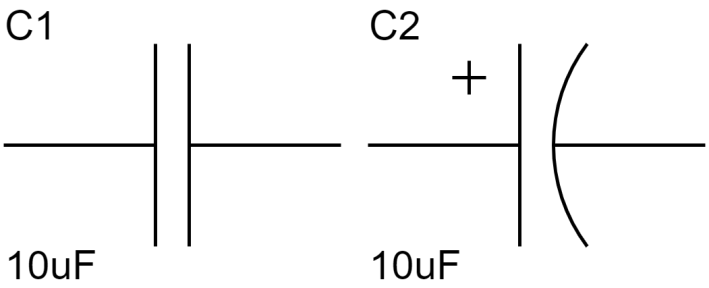



**Figure 3.13:** Schematic symbols for Capacitors. On the left is the symbol for a non-polarized bi-directional capacitor. On the right is the symbol for an electrolytic or polarized capacitor.

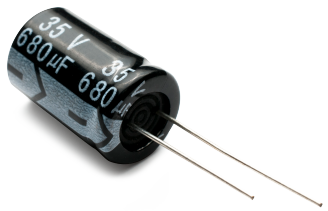



**Figure 3.14:** Electrolytic capacitor. Notice the longer lead indicates the positive terminal and the negative terminal is clearly marked with a stripe on the casing. Retrieved from SRG LLC

**Electrolytic Capacitors** are polarized or uni-directional capacitors that use a liquid electrolyte compared to the ceramic ones used in non-polarized capacitors. These capacitors must be placed in a specific orientation otherwise they may become damaged and could explode! On electrical schematics, the negative terminal of these capacitors is denoted by a curved line, like shown above. Even though, you must closely pay attention to how the capacitor is orientated, the electrolytic capacitor provides a much high energy density compared to the ceramic capacitors, making them more suited for high power application.

### Inductors

Inductors are tiny electromagnetic that wind conducting wire around an iron or air core. When alternating electrical current is passed through the inductor, some of the energy is buffered in the magnetic field of the electromagnet. You can think of them as electrical flywheels that store

moving charge and can discharge it to maintain a more constant "flow". This makes inductors useful for resisting changes in current or acting as the analogue to capacitors in AC circuits.


**Figure 3.15:** Schematic symbol for inductors.

**Diodes**

Diodes are used to restrict current flow in a certain direction. This is accomplished using some electrochemical wizardry and comes at the cost of a little bit of voltage drop through the diode. But, the main property of the diode makes it invaluable for protecting circuits against reverse polarity, dissipating AC current, or even rectifying AC voltages into DC.

**Figure 3.16:** Through hole diode as commonly found in Arduino kits. Retrieved from AnalyzeAMeter

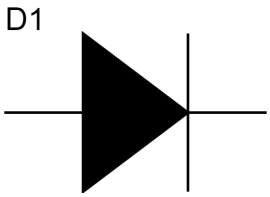



**Figure 3.17:** Schematic symbol for diodes.

**Light Emitting Diodes (LEDs)** are diodes that siphon a little bit of electrical energy to emit photons at specific wavelengths. These components are ubiquitous in any application that requires a simple human interface such as light indicators, RGB gaming products, or flashlights. LEDs are smaller, easily scaled to almost any production scale, and more energy efficient compared to most other light sources in use today.

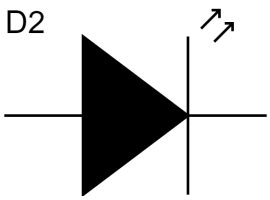



**Figure 3.18:** Schematic symbol for light emitting diodes.

**Zener Diode** are special diodes that restricts the current flow direction up to a certain breakdown voltage. When that voltage is achieved, current is allowed to flow against the diode, without damaging it, making it useful in applications like circuit protection.

**Schottky Diode** are other special diodes that have a very low voltage drop across it, making it ideal for applications where flow restriction is necessary, but power loss needs to be considered. These are commonly

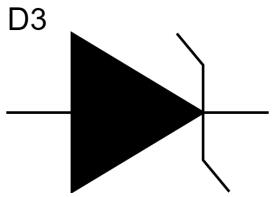



**Figure 3.19:** Schematic symbol for Zener diodes.

used to control the flow of charge within a battery pack of multiple cells. As these cells discharge, they can have slightly different voltages, meaning some energy loss will occur as one battery tried to equalize voltage with another in the pack. Schottky diodes ensure the charges flow only from the batteries to the draw outside the pack, without wasting much energy.

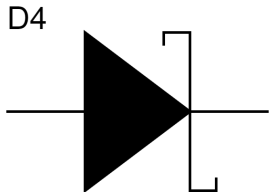



**Figure 3.20:** Schematic symbol for Schottky diodes.

**Crystal Oscillators**

Oscillators resonate at a certain frequency depending on the type of crystal used and the coupling capacitors matched to it. These driving most modern computational circuits as they generate a very clean DC square wave. Some components use the rising or falling edge of this wave to trigger register shifts, calculations, anything else needed to perform its duty. Therefore, for any circuit that requires a high frequency, high reliability clock signal, most often, crystal oscillators will be present

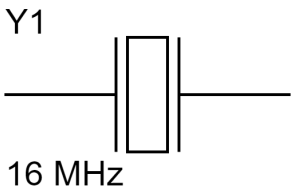
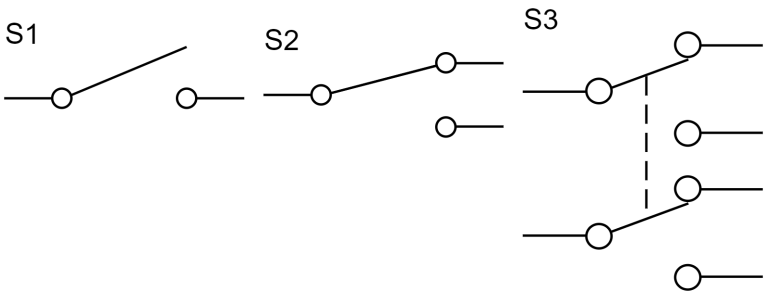



**Figure 3.21:** Schematic symbol for crystal oscillators.

**Switches**

Switches are mechanical contacts that physically direct the flow of electricity. They can have multiple independent circuits called "poles" that can allow current to flow from a common connector to an output connector called a "throw". They are named according to their architecture, e.g. Single Pole Double Throw (SPDT) switches have one circuit with two outputs that are connected to the common pin, depending on the switch position. Double Pole Triple Throw (DP3T) switches have two circuits that connect three output pins to a common input pin, depending on the switch position. Each circuit is mechanically linked so the same output connector in both circuits will be connected to the common. One downside to the mechanical switch

is that they can be welded together if a maximum current is exceeded or, they can wear out over thousands of cycles.




**Buttons** are typically Single Pole Single Throw (SPST) switches that are spring loaded to remain normally open until pressed. When the button is pressed, a conducting plate makes contact with the common and output pins, completing the circuit through the button. These are commonly used as human input into circuits for calculations, state machines, typing lecture notes, etc.

**Figure 3.23:** A common push button found in most Arduino kits. Retrieved from eBay

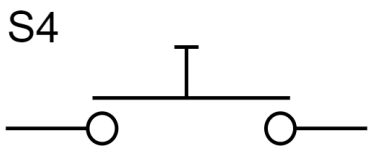



**Figure 3.24:** Schematic symbols for a normally open push button.

**Relays** are electromechanical switches that rely on an electromagnetic to move the electrical contact, rather than physical force. These are used for computers to manipulate large currents or direct electrical flow automatically, without human intervention. These relays follow the same nomenclature as normal switches and can be found in any number of configurations of poles and throws.

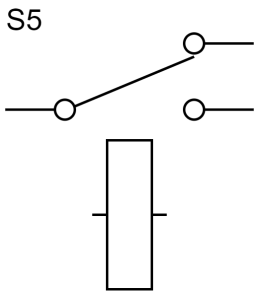



**Figure 3.25:** Schematic symbols for an electromechanical relay.

**Transistors**

Transistors are electrical solid-state switches that use electrochemical properties to control the flow of electricity through them. These are typically able to withstand millions, if not billions, of switching cycles before failing and have driven the modern age we find ourselves in today. Everything in


**Figure 3.26:** A common transistor found in most Arduino kits. Retrieved from Walmart

modern computing from the simple logic gate to the highest end computer and graphics processors use transistors to perform calculations or "think" about problems. Transistors have a gate that controls current flow from the source to the drain, depending on the voltage applied.

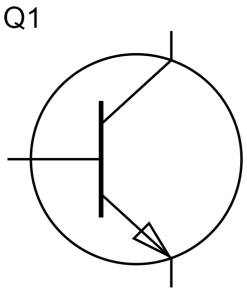



**Figure 3.27:** Schematic symbol for a transistor.

**MOSFETs** or Metal-Oxide-Semiconductor Field-Effect-Transistors are some of the most common types of transistors. They are able to be configured (doped) with different silicon structures that give them different electrochemical properties. N-Channel MOSFETs require a positive voltage on the Gate to function, whereas P-Channel MOSFETs require a negative gate voltage. Because of this, N-Channel MOSFETs are typically used for low-side switching (connect circuit to ground) and P-Channels are used for high switching (connect circuit to the voltage source). There are two main types: depletion, which requires a Gate-Source voltage (Vgs) to switch the gate OFF, equivalent to a normally-closed switch; and enhancement where a Vgs is required to switch the gate ON, acting like a normally-open switch. These transistors are very common in high power applications and signal amplification.

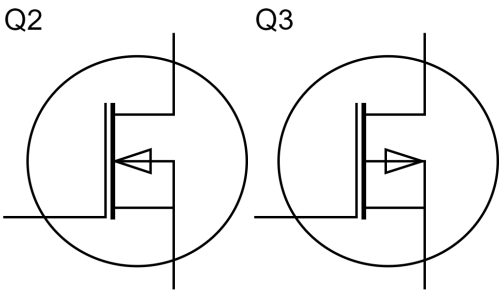



**Figure 3.28:** Schematic symbols for MOSFETs. On the left is a P-Channel MOSFET, on the right is an N-Channel MOSFET.

insert example from Thetis/EVE circuits - cherry pick some nice ones that use mostly discreet components

### 3.3.3 Integrated Circuits

Integrated Circuits (ICs) comprise the backbone of modern electrical design. They combine any combination of discreet components, logic gates, processors, storage, etc. into a single package that can be easily placed into a circuit. For schematic symbols, the representation varies wildly, but generally hold to a few core design features:

**Outlines** - represents the chip's presence on the schematic. It encompasses all of the information about the IC and usually is large enough to fit

descriptive text and allows the pins to be arranged in a cohesive manner.

**Pins** - schematic representations of the chip's physical contacts with the circuit. Each is uniquely coupled to the physical pin or pad on the real-world design, but does not have to be sequentially ordered in the schematic. Each pin should have a label for its purpose on the IC next to it, within the outline, as well as a label to identify its physical number on the IC.

**Name** - this is the unique identifier of the IC on the schematic

**Values** - this is typically the IC's part number for ordering information and implementation

It is important to note that some IC symbols will opt to include other handy information in the outline so it is more easily accessible without having to refer back to the datasheet constantly.

insert image of an IC symbol with leaders pointed to different parts

Inlcude image of IC circuits e.g. voltage regulator

# Software Basics and Architecture 4



While the world of physical hardware can be harnassed to develop systems that accomplish tasks, oftentimes these implementations become prohibitively complex as the scale of implementation grows larger. For example, a modern computer processor contains billions of transistors and it would be impossible to set and reset each one of these transistors by hand. Thankfully, as you probably already know, modern programming consists of several layers of abstraction above the transistor level, which allow us to develop meaningful software solutions while leveraging the expansive amount of hardware that we have access to.

This chapter will focus on introducing you to the fundamentals of hardware-based software development. You may find that several if not all of these topics are also covered in introductory programming courses. However, since we do not presume that everyone who has taken this class has had background programming experience, we will cover all the topics necessary for this course from the ground up.

## 4.1 Arduino Basics

Lorem Ipsum Dolores

## 4.2 Primitive Programming Datatypes

The baseline for every programming language is an understanding of datatypes. Datatypes are how we can classify the characteristics of the blocks of information that our program will process. These blocks of information can be collected data from the environment, information that the program stores internally, function inputs/outputs, etc. Additional information on arduino datatypes (and in general, most arduino syntax) can be found on tutorialspoint or the Arduino Language Reference

For the following sections, example code blocks will be given. As such, please note that the '\\' symbol is used to denote a 'comment' and therefore is not code that will execute and that the capitalization of keywords is deliberate (since Arduino C is case sensitive)

### 4.2.1 void

The most basic of datatypes, the **void** keyword is used to represent 'nothing'. For arduino programming, it is primarily used to denote that a function does not give us back any output. Do note, however, that the **void** datatype cannot be applied to variables.

We will go more depth into functions in a later section, but for the purposes of this section you can think of functions as boxes that produce output from a given input. In the case of **void** we have no output.

**Proper usage of Void** The following code block demonstrates the proper usage of the **void** datatype.

```
void function_name (\\ input variables) {
    \\ function code
    \\ note how there is no return statement (we are outputting nothing)
}
```

**Improper usage of void** The following code block demonstrates the improper usage of the **void** datatype.

```
void variable_name = \\ some value;
```

### 4.2.2 bool

Boolean datatypes are used to store logic values, in this case the logic values of **True** or **False**. The **bool** keyword is used in both function and variable declarations to classify the function output or variable type as boolean logic.

**Boolean usage** The following code block demonstrates how the **bool** keyword is used in a function declaration and a variable declaration

```
bool function_name (\\ input variables) {
    bool variable_name = True;
    return variable_name;
}
```

### 4.2.3 Char

Character datatypes are special when programming in Arduino C. While they do represent what you would conventionally think as a character (for example, the letter 'a'), they are defined based on their ascii value (ASCII Reference), which allow you to manipulate many more characters than what can be typed by a keyboard.

The **char** keyword is used in variable and function declarations to classify the result as a character, and there are two ways to syntaxically represent a character.

**Method 1: Using Single Quotes** The first form of syntax makes more sense visually, and involves wrapping a typed character within two single quotes

```
char variable_name = 'a';
```

Do note, however, that this method restricts you to the characters that can be typed using a conventional keyboard. In addition, when declaring a character in this form, you cannot wrap multiple characters inside the quotes.

```
char variable_name = 'abc'; \\ this is improper declaration
```

**Method 2: Using ASCII Values** As mentioned previously, characters are encoded via ASCII values when used in a programming context. As such, we can define a character using its corresponding ASCII value.

```
char variable_name = 97; \\ ASCII value 97 corresponds to character 'a'
```

Since one character is equivalent to one byte of data, the maximum decimal value that can be used in a character declaration is 255.

**Aside: Some quirks with characters**

For all intents and purposes, characters are 8-bit binary numbers (0-255 in decimal). They become 'characters' based on how they are interpreted when output to a display. Because of this dual interpretation, you can manipulate characters as if they were numbers. For example

$$'a' + 1 = 'b'$$

However, if you were to use '1' (the character representation of 1) as the input you would get

$$'a' + '1' = 97 + 49 = 156$$

Which, for those that are curious, 156 represents the ASCII character œ. In addition, there are special characters that can be typed using the single quotes method that do not strictly follow the 'only one character' rule. In particular, the $'\backslash n'$ (newline) and $'\backslash t'$ (tab) characters are often used.

### 4.2.4 unsigned char and byte

The unsigned character and byte datatypes are both datatypes that are used to represent a byte of data (8 bits, 0-255 in decimal). While they share the same range of values as the char datatype, they should not be used interchangeably with the char datatype. The **unsigned char** and **byte** keywords are used to define function output and variable values.

Generally speaking, it is conventional to use the **byte** keyword over the **unsigned char** keyword for both brevity and code readability

**unsigned char and byte usage** the following block of code demonstrates how to use the **unsigned char** and **byte** keywords

```
byte function_name (\\ input variables) {
    byte variable_name_1 = 0;
    unsigned char variable_name_2 = 128;
    return variable_name_1;
}
```

### 4.2.5 int and short

Both the **int** datatype and the **short** datatype are used to store 16-bit (2 byte) signed integers. This makes their effective range -32,768 to 32,767. By convention, the **int** datatype is the more commonly used one in variable and output declaration.

One of the things that you will have to look out for when using numeric datatypes is **overflow**; ie. when the number passed into a data field is larger than the range of that datatype. Overflow will **not** stop the code from compiling, but the code's behavior will arbitrarily differ from what you expect.

**int and short usage** the following block of code demonstrates how to use the **int** and **short** keywords

```
1        int function_name (\\ input variables) {
2            int variable_name_1 = -20000;
3            short variable_name_2 = 12763;
4            return variable_name_1;
5        }
```

### 4.2.6 unsigned int and word

Similar to the case with **char** and **unsigned char**, there are also unsigned variants of the **int** and **short** datatypes respectively known as **unsigned int** and **word**. These datatypes are both 16-bit(2 byte) integers; however, they span only the **positive** range of 16-bit integers (0 to 65,535).

The **unsigned int** datatype has **known** overflow behavior unlike **int**. When the value of an unsigned integer goes below zero, it wraps around to the max of the range (so -1 = 65535, -2 = 65534) and when it exceeds 65535, it wraps back to zero (so 65536 = 0, 65537 = 1)

**unsigned int and word usage** the following block of code demonstrates how to use the **unsigned int** and **word** keywords

```
1        unsigned int function_name (\\ input variables) {
2            unsigned int variable_name_1 = 1203;
3            word variable_name_2 = 55000;
4            return variable_name_1;
5        }
```

### 4.2.7 long and unsigned long

We had previously covered that a **short** is a 16-bit integer. Therefore, appropriately, a **long** is a 32-bit signed integer value encompassing the range -2,147,483,648 to 2,147,483,647. As per convention, the **unsigned long** store a 32-bit unsigned integer value with range 0 to 4,294,967,295.

**long and unsigned long usage** the following block of code demonstrates how to use the **long** and **unsigned long** keywords

```
1        long function_name (\\ input variables) {
2            long variable_name_1 = -2000000;
3            unsigned long variable_name_2 = 1000000;
4            return variable_name_1;
5        }
```

### 4.2.8 float and double

There are also datatypes that are used to store decimal values vs. pure integer, these are the **float** and **double** datatypes. For the boards that you will be using, the **float** and the **double** datatypes are functionally the same. They both cover the range -3.4028235E+38 to 3.4028235E+38, and are both stored in memory as 32-bit values.

**float and double usage** the following block of code demonstrates how to use the **float** and **double** keywords

```
1        float function_name (\\ input variables) {
2            float variable_name_1 = -300.21;
3            double variable_name_2 = 600.123;
4            return variable_name_1;
5        }
```

**Floating point error** One of the issues that you may run into when dealing with floats and doubles is the floating point (roundoff) error. Due to how floating point numbers are represented (for more info on that, you can read up on the IEEE-754 Standard), and due to the fact that floating point numbers are stored in a finite number of bits (32), floating point numbers are **not absolutely accurate** and will differ from the true value of a datapoint by a small amount. For the majority of applications, this slight difference between actual values and real values is trivial and non-impactful to the application. However, for applications that store small magnitude numbers as data, or rely on high-precision calculations, floating point error is something to watch out for.

### 4.2.9 Arrays

Arrays are one of the most fundamental structures to understand when working with any programming language. In Arduino C, they are treated as a block of memory that is broken up into chunks, where each chunk stores a single value of a datatype. Arrays in Arduino follow an explicit set of rules that govern how they are defined and used, which we will cover below.

**Rule 1: Arrays can only hold 1 data type** Unlike some other languages, an array in Arduino (and in C in general) must be defined to store a single datatype. You cannot have an array that has both int and float datatypes for example. For the reasoning behind this rule, you can take a look at the following aside.

**Rule 2: Arrays must have a predefined number of values** When you initialize an array, you must tell the program how many values are going to be stored in the array. After this declaration, you cannot modify the size of that array. The reasoning behind this rule is also covered in the following aside.

**Aside: The weird world of memory allocation**

C (and its variants) are programming languages that deal directly with memory. While this makes them faster than languages that do not directly modify memory, it also restricts how flexible variables and memory structures can be when using them. When reading or writing to memory, the amount of memory you use and how that memory will be divided up must be explicitly defined. If the variable asks for 32 bits of memory, it will get 32 bits of memory, no more, no less. In a similar sense, arrays also must explicitly define how much memory they will use. An array that wants to store 10 ints (16-bit) will be allocated $10 * 16 = 160$ bits of memory divided into 10x 16-bit chunks. If you tried to store a long (32-bit) into one of those chunks, it will not fit and the program will error. Now, while it is possible to allocate a raw chunk of memory to be manually partitioned and used, how to do that is beyond the scope of this class. Those interested can read up on **memory allocation in C**

# Data Processing 5

Data collected from sensors can sometimes be useful on their own, but often times, it is necessary to process it to gain a better understanding of what is occurring. To that end, this chapter will be about how to process raw data into something more usable and some techniques that you can employ for your projects in the future.



## 5.1 Digital Filtering

One of the most fundamental data processing techniques is that of the digital filter. The idea behind these filters is to take in data samples and apply mathematical formulations on them to dampen undesirable signals within.

### 5.1.1 Background

This section assumes that you do not have a lot of background knowledge on basic statistics and terminology. It is important to understand here that mean and expected value are not the same thing. In many college lab courses, measurements are taken multiple times and averaged together to find the mean value (often denoted as $\mu$). This is assumed to be the absolute correct value with zero uncertainty. However, this is never the case in the real world as all measurements have uncertainty in some fashion. When the average measurement value is taken *with* uncertainties factored in, this becomes the expected value ($E$) and used to guess what the real (correct) value of the measured variable is.

So, how can quantify the uncertainties within the expected value? First, we need to know what the measurement variance is. This value is a description of how far the measured data is from the mean and is defined in Equation 5.1. [1]

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2 \tag{5.1}$$

where $\sigma^2$ is the variance

$N$ is the total number of samples

$x_n$ is the n-th sample

$\mu$ is the running average

1: For large datasets, variance is normalized by $\frac{1}{N-1}$

The standard deviation of the dataset $\sigma$ decribes the probabalistic distance a measurement can have from the mean and is defined by:

$$\sigma = \sqrt{\sigma^2}$$

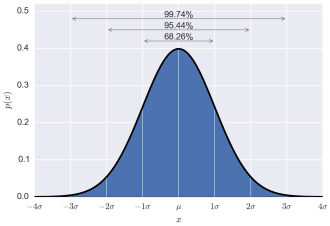



**Figure 5.1:** A graph of a Gaussian distribution with the 1st standard deviations shown. Retrieved from The Akkad and Koda Report

**Aside: Normal Distribution**

Normal distribution of data is the archetypical bell curve. It is also referred to Gaussian distribution and is defined by:

$$f(x, \mu, \sigma^2) = \frac{1}{2\pi\sigma^2} \exp \frac{-(x-\mu)^2}{2\sigma^2}$$

This function creates a Gaussian curve which is the Probability Density Function (PDF) for a normal distribution. Normally, measurements errors are Gaussian-distributed and the Kalman filter assumes this is always the case. Different error distributions will drastically increase the uncertainties within the filter and may negate it entirely.

Estimates are the algorithm's guessed value at the unknown system state. Here, accuracy quantifies how close the estimate is to the true value whereas precision quantifies the variability of the estimates with respect to uncertainty. High precision estimates have a low variance (uncertainty) in measurements. Low accuracy estimates display a bias that are systemic errors in measurement. Variance is determined by random measurement error and can be reduced by averaging or smoothing the measurements.

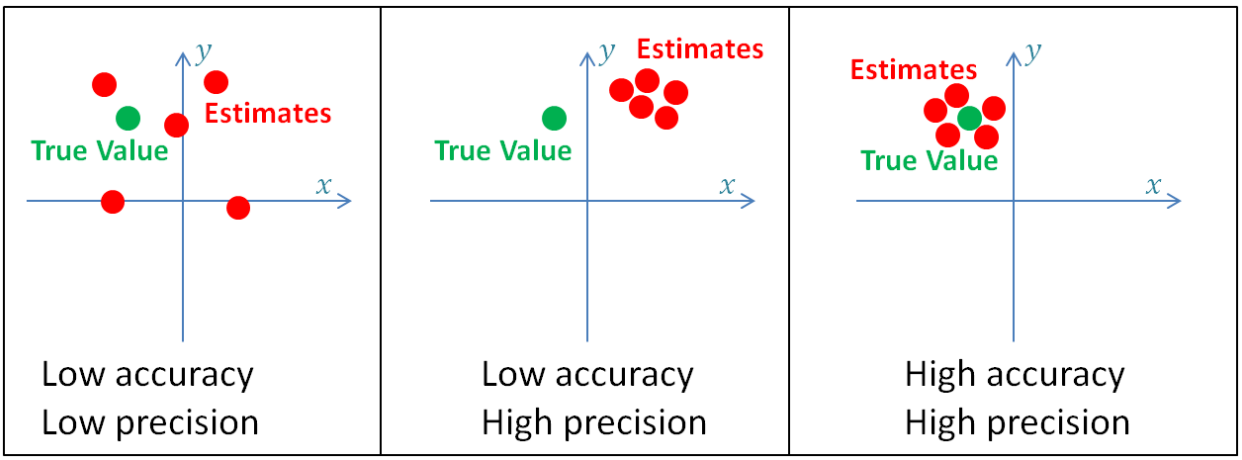



**Figure 5.2:** Different plots demonstrating the difference between accuracy and precision. Retrieved from KalmanFilter.net

### 5.1.2 The $\alpha$ Filter

At any sample, $N$, a value $\hat{x}_{N,N}$ can be estimated using:

$$\hat{x}_{n,n} = \frac{1}{N}\sum_{n=1}^{N}(z_n)$$

where $\hat{x}_{n,n}$ is the estimate of the true value at sample $n$

$N$ is the total number of samples taken

$n$ is the current sample number

$z$ is the current measurement

Since the estimate accuracy will improve with the number of measurements, we need to remember all the historical measurements. However, storing, accessing, and operating on discrete measurements exponentially increases in difficulty and computational power as more measurements are taken. Therefore, we can store the previous measurements in a state variable and add a small adjustment that is proportional to the newest measurement. This preserves the accuracy of storing many discrete data points and the computing and storage resources required to operate on data added to the set.

This algorithm is called the *State Update Equation* and is one of the five Kalman equations. It is defined by:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + \alpha_n(z_n - \hat{x}_{n,n-1}) \tag{5.2}$$

where $\hat{x}_{n,n}$ is the estimate of the true value at sample $n$

$\hat{x}_{n,n-1}$ is the estimate of the previous true value

$n$ is the current sample number

$\alpha_n$ is a correction gain

$z$ is the current measurement

The correction gain, $\alpha_n$ is unique to every example and can change with every sample. It is also called the Kalman gain, denoted by $K_n$

The term $(z_n - \hat{x}_{n,n-1})$ is the measurement residual or "innovation"; this contains the new information.

This algorithm uses zero-indexing, so the initial or starting guess will be $n = 0$. This value can be determined by reasoning or physical constraints of the system. The filter will always require an initial guess to kickstart the algorithm. A flow chart describing the algorithm's process is below.

**Example 5.1.1** Let's say that we are programming a scale to estimate the weight of an ROV placed atop it. For the sake of example, we know that the ROV weighs exactly 10 kilograms and we want to quantify the uncertainty of our scale.

**Iteration 0**

0 : We can guess that the initial weight of the ROV is 10-kg: $\hat{x}_{0,0} = 10$ kg

3 : Since the weight should not change, our prediction for the next measurement is: $\hat{x}_{1,0} = \hat{x}_{0,0} = 10$ kg

**Iteration 1**

1 : We make a measurement on the scale: $z_1 = 10.3$ kg

2a : We calculate the correction gain: $\alpha_1 = \frac{1}{N} = \frac{1}{1} = 1$

2b : We can then estimate the current state: $\hat{x}_{1,1} = \hat{x}_{1,0} + \alpha_1(z_1 - \hat{x}_{1,0}) = 10 + 1(10.3 - 10) = 10.3$ kg

3 : Since the weight should not change: $\hat{x}_{2,1} = \hat{x}_{1,1} = 10.3$ kg

**Iteration 2**

1 : We make a measurement on the scale: $z_2 = 9.9$ kg

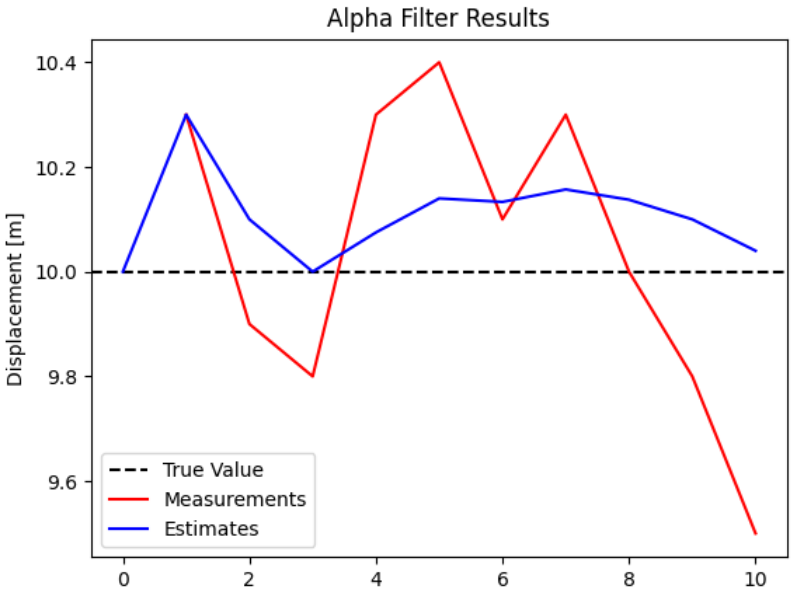
2a : We calculate the correction gain: $\alpha_2 = \frac{1}{N} = \frac{1}{2} = 0.5$

2b : We can then estimate the current state: $\hat{x}_{2,2} = \hat{x}_{2,1} + \alpha_2(z_2 - \hat{x}_{2,1}) = 10.3 + 0.5(9.9 - 10.3) = 10.1$ kg

3 : Since the weight should not change: $\hat{x}_{3,2} = \hat{x}_{2,2} = 10.1$ kg

The calculations and results for this example are summarized in the table and figure below:

| | Current Estimates | | | Predictions |
|---|---|---|---|---|
| n | $z_n$ | $\alpha_n$ | $\hat{x}_{n,n}$ | $\hat{x}_{n+1,n}$ |
| 0 | — | — | 10.0 | 10.0 |
| 1 | 10.3 | 1 | 10.3 | 10.3 |
| 2 | 9.9 | $\frac{1}{2}$ | 10.1 | 10.1 |
| 3 | 9.8 | $\frac{1}{3}$ | 10.0 | 10.0 |
| 4 | 10.3 | $\frac{1}{4}$ | 10.1 | 10.1 |
| 5 | 10.4 | $\frac{1}{5}$ | 10.1 | 10.1 |
| 6 | 10.1 | $\frac{1}{6}$ | 10.1 | 10.1 |
| 7 | 10.3 | $\frac{1}{7}$ | 10.2 | 10.2 |
| 8 | 10.0 | $\frac{1}{8}$ | 10.1 | 10.1 |
| 9 | 9.8 | $\frac{1}{9}$ | 10.1 | 10.1 |
| 10 | 9.5 | $\frac{1}{10}$ | 10.0 | — |




The $\alpha$-filter considerably smoothed out the noisy measurement data and will eventually converge on an estimate close to the true value (within 10%).

### 5.1.3 The $\alpha$ and $\beta$ Filter

As demonstrated in Example 5.1.1, the $\alpha$ filter makes a major assumption that the state does not change between samples. In reality, this is rarely the case as we typically want to measure a body's movement, or track a variable over time, or measure something that has dependency on another variable.

For example, if we wished to track a boat moving in a one-dimensional world at a constant velocity (no acceleration), we need to modify our model a bit. For nomenclature, $x_n$ will represent the displacement of the vessel from a known starting point. Since the vessel is moving with a constant velocity, we can model its time rate of change of position as:

$$\dot{x} = v = \frac{dx}{dt}$$

If we sample the vessel's displacement at intervals of $\Delta t$ time, then we can define a dynamic model of the vessels movement as:

$$x_{n+1} = x_n + \Delta t \dot{x}_n \tag{5.3a}$$
$$\dot{x}_{n+1} = \dot{x}_n \tag{5.3b}$$

2: or the *State Transistion Equation* or the *State Prediction Equation*

This is called the *State Extrapolation Equation* [2] and is another one of the five Kalman equations. If we assume the sample interval is $\Delta t = 5$ seconds at sample $n-1$ and the displacement and velocity of the vessel is estimated as 300-m and 10-m/s, respectively, then using Equations 5.3a and 5.3b, we can predict where the vessel will be and its velocity at sample $n$.

$$\hat{x}_{n,n-1} = \hat{x}_{n-1,n-1} + \Delta t \hat{\dot{x}}_{n-1,n-1} = 300 + 5(10) = 350 \text{ m}$$
$$\hat{\dot{x}}_{n,n-1} = \hat{x}_{n-1,n-1} = 10 \text{ m/s}$$

However, at sample $n$, the vessel records a displacement of 320-m, not the expected 350-m; there are two possible reasons:

1. the vessel displacement measurements are not precise or accurate
2. the vessel velocity changed unexpectedly [3]

3: The new velocity would have to be:

$$\frac{320 - 300}{5} = 4 \text{ m/s}$$

We know from the previous section that we can account for the measurement error using an $\alpha$ filter, so lets re-write Equation 5.3b to account for error in the velocity:

$$\hat{\dot{x}}_{n,n} = \hat{\dot{x}}_{n,n-1} + \beta\left(\frac{z_n - \hat{x}_{n,n-1}}{\Delta t}\right) \tag{5.4}$$

**Example 5.1.2** Consider a boat moving in one dimension. We are going to assume the following:

- $\alpha = 0.2$ (Positional gain)
- $\beta = 0.1$ (Velocity gain)
- $\Delta t = 5$ seconds

**Iteration 0**

0 : Initial conditions: $\hat{x}_{0,0} = 300$ m, $\hat{\dot{x}}_{0,0} = 4$ m/s

3 : Predict the next position: $\hat{x}_{1,0} = \hat{x}_{0,0} + \Delta t \hat{\dot{x}}_{0,0} = 300 + 5(4) =$ **320 m**

: Predict the next velocity: $\hat{\dot{x}}_{1,0} = \hat{\dot{x}}_{0,0} =$ **4 m/s**

**Iteration 1**

1 : The vessel makes a positional measurement: $z_1 =$ **310 m**

2 : We estimate the current position: $\hat{x}_{1,1} = \hat{x}_{1,0} + \alpha(z_1 - \hat{x}_{1,0}) = 320 + 0.2(310 - 320) =$ **318 m**

: We estimate the current velocity: $\hat{\dot{x}}_{1,1} = \hat{\dot{x}}_{1,0} + \beta(\frac{z_1 - \hat{x}_{1,0}}{\Delta t}) = 4 + 0.1(\frac{310 - 320}{5}) =$ **3.8 m/s**

3 : Predict the next position: $\hat{x}_{2,1} = \hat{x}_{1,1} + \Delta t \hat{\dot{x}}_{1,1} = 318 + 5(3.8) =$ **337 m**

: Predict the next velocity: $\hat{\dot{x}}_{2,1} = \hat{\dot{x}}_{1,1} =$ **3.8 m/s**

**Iteration 2**

1 : The vessel makes a positional measurement: $z_2 =$ **317 m**

2 : We estimate the current position: $\hat{x}_{2,2} = \hat{x}_{2,1} + \alpha(z_2 - \hat{x}_{2,1}) = 337 + 0.2(317 - 337) =$ **333 m**

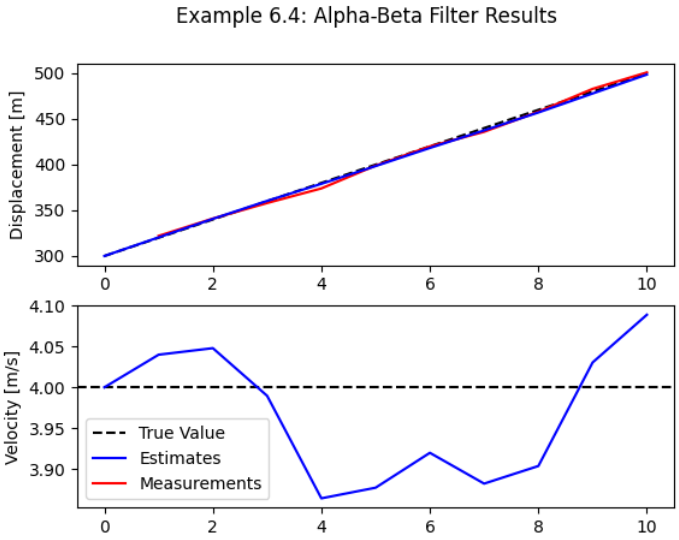
: We estimate the current velocity: $\hat{\dot{x}}_{2,2} = \hat{\dot{x}}_{2,1} + \beta(\frac{z_2 - \hat{x}_{2,1}}{\Delta t}) = 3.8 + 0.1(\frac{337 - 317}{5}) =$ **3.4 m/s**

3 : Predict the next position: $\hat{x}_{3,2} = \hat{x}_{2,2} + \Delta t \hat{\dot{x}}_{2,2} = 333 + 5(3.4) =$ **350 m**

: Predict the next velocity: $\hat{\dot{x}}_{3,2} = \hat{\dot{x}}_{2,2} =$ **3.4 m/s**

The data for this example is summarized in the table and figure below:

| | Current Estimates | | | Predictions | |
|---|---|---|---|---|---|
| n | $z_n$ | $\hat{x}_{n,n}$ | $\hat{\dot{x}}_{n,n}$ | $\hat{x}_{n+1,n}$ | $\hat{\dot{x}}_{n+1,n}$ |
| 0 | — | 300.0 | 4.00 | 320.0 | 4.00 |
| 1 | 322.0 | 320.4 | 4.04 | 340.6 | 4.04 |
| 2 | 341.0 | 340.7 | 4.05 | 360.9 | 4.05 |
| 3 | 358.0 | 360.3 | 3.99 | 380.3 | 3.99 |
| 4 | 374.0 | 379.0 | 3.86 | 398.3 | 3.86 |
| 5 | 399.0 | 398.5 | 3.88 | 417.9 | 3.88 |
| 6 | 420.0 | 418.3 | 3.92 | 437.9 | 3.92 |
| 7 | 436.0 | 437.5 | 3.88 | 456.9 | 3.88 |
| 8 | 458.0 | 457.1 | 3.90 | 476.7 | 3.90 |
| 9 | 483.0 | 477.9 | 4.03 | 476.7 | 4.03 |
| 10 | 501.0 | 498.7 | 4.09 | 498.1 | 4.03 |




From the figure, we can see that the filter provides a good "smoothing" effort for the noisy measurement data. High values for $\alpha$ and $\beta$ will result in less "smoothing" in certain cases. The current state estimate may be very close to the measurement values with a higher error in predicted values

The value of $\beta$ depends on the precision of the vessel's navigation system. If the positional precision is $\sigma = 0.1$ m, then it is very likely the vessel speed changed. Therefore, $\beta$ would be very high to adjust the estimated velocity. Conversely, if the positional precision is $\sigma = 10$ m, then it is likely the vessel's position measurements are wrong and $\beta$ can be smaller.

$\alpha$ and $\beta$ depend on the measurement precision. Very precise measurements would prefer a high $\alpha$ and $\beta$ gains that will cause the filter to converge quickly and can adapt to velocity changes. Lower precision measurements would prefer lower $\alpha$ and $\beta$ gains that allow any uncertainties to be smoothed out; this will cause the filter to react slowly to velocity changes or external perturbations, though.

**The Importance of Accurate Modelling** Lets expand upon Example 5.1.2. Here, we will consider the same vessel under the same assumptions and initial conditions, but we will have a flawed dynamic model. The vessel will begin transiting at a constant velocity of 5-m/s for 15 seconds, then accelerate at a constant 2-m/s/s for 15 seconds. The kinematic plots are displayed below in Figure **??** and the numerical results can be found in Table **??**.

| | Current Estimates | | | Predictions | |
|---|---|---|---|---|---|
| n | $z_n$ | $\hat{x}_{n,n}$ | $\hat{\dot{x}}_{n,n}$ | $\hat{x}_{n+1,n}$ | $\hat{\dot{x}}_{n+1,n}$ |
| 0 | — | 30.0 | 5.00 | 55.0 | 5.00 |
| 1 | 60.0 | 56.0 | 5.10 | 81.5 | 5.10 |
| 2 | 77.0 | 80.6 | 5.01 | 105.7 | 5.01 |
| 3 | 107.0 | 105.9 | 5.03 | 131.1 | 5.03 |
| 4 | 154.0 | 135.7 | 5.50 | 163.2 | 5.50 |
| 5 | 252.0 | 181.0 | 7.30 | 217.5 | 7.30 |
| 6 | 407.0 | 255.4 | 11.1 | — | — |

**Table 5.1:** Numerical results for the $\alpha - \beta$ filter not accounting for acceleration

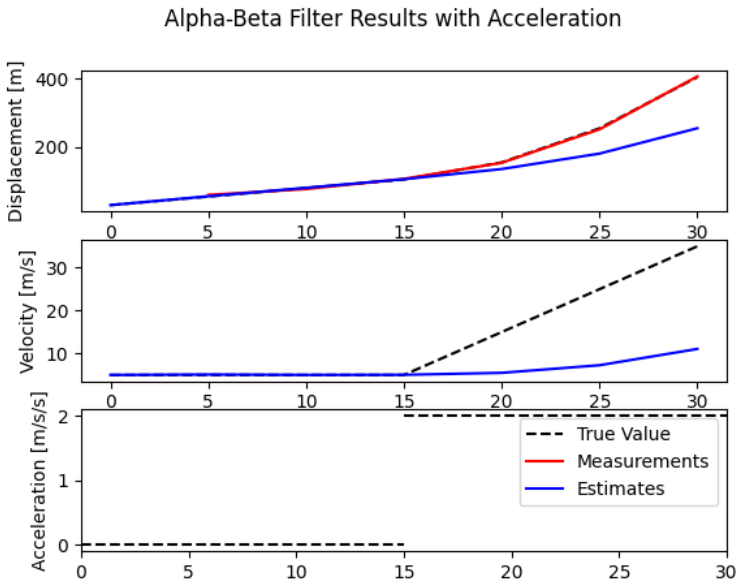



**Figure 5.3:** Graphical results for the $\alpha - \beta$ filter not accounting for acceleration

The difference between the real plot (black) and the estimated plot (blue) is called the lag error. The lag error causes the estimates to drastically diverge from the real values as they change with time. Increasing the $\beta$ gain may allow the estimates to converge faster to the real values, but there is a better way.

### 5.1.4 The $\alpha$ $\beta$ and $\gamma$ Filter

This next filter considers the change of the rate of the change variable. In the case of moving targets, this means acceleration. This changes the State Extrapolation Equations to:

$$\hat{x}_{n+1,n} = \hat{x}_{n,n} + \Delta t \hat{\dot{x}}_{n,n} + \frac{1}{2}\hat{\ddot{x}}_{n,n}\Delta t^2 \tag{5.5a}$$

$$\hat{\dot{x}}_{n+1,n} = \hat{\dot{x}}_{n,n} + \hat{\ddot{x}}_{n,n}\Delta t \tag{5.5b}$$

$$\hat{\ddot{x}}_{n+1,n} = \hat{\ddot{x}}_{n,n} \tag{5.5c}$$

Similarly, the State Update Equations become:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + \alpha(z_n - \hat{x}_{n,n-1}) \tag{5.6a}$$

$$\hat{\dot{x}}_{n,n} = \hat{\dot{x}}_{n,n-1} + \beta(\frac{z_n - \hat{x}_{n,n-1}}{\Delta t}) \tag{5.6b}$$

$$\hat{\ddot{x}}_{n,n} = \hat{\ddot{x}}_{n,n-1} + \gamma(\frac{z_n - \hat{x}_{n,n-1}}{\frac{1}{2}\Delta t^2}) \tag{5.6c}$$

This type of filter eliminates the lag error by accounting for a constant acceleration. If we want to account for "jerk", or a change in acceleration, we need to implement a more advanced filter like the Kalman filter.

### 5.1.5 Kalman Filter

#### Introduction

The Kalman Filter is a recursive algorithm introduced in the 1960's as a method to track, estimate, and predict the state of a system and corresponding uncertainties. This filter integrates a dynamic (linear) model of the system, control inputs, measurements, and biases/uncertainties into a single algorithm. This effectively fuses together system inputs and responses and extrapolates what the system is currently doing and expected to do. One key advantage of this algorithm is that it only requires the guess of the previous state to estimate the current state. This massively decreases the memory and processing costs as the history of inputs, measurements, and uncertainties does not need to be remembered or analyzed. However, it does have some limitations when the sensor data is noisy or the control inputs cannot be linearly mapped to the system state. Random errors in the sensor data may cause the filter to behave unpredictably and non-linearity prevents proper fusion entirely.[4]

4: Though this can be managed with an Extended Kalman Filter

This section will discuss the Kalman filter in a higher-order method. Many resources around with Internet can discuss the in-depth mathematics governing this filter and you are free to browse them at your leisure. For this guide, my primary concern is to get you acquainted with the Kalman filter and its broader practical applications.

**Kalman Filter in One Dimension**

The uni-dimensional Kalman filter is a special, idealized case for the Kalman filter. It is more convenient as a teaching tool as it does not include the complex matrix and vector operations the general multi-dimensional Kalman filter requires. However, this only makes this type useful for tracking and estimating a single variable.

Multiple uni-dimensional Kalman filters can be run in parallel or chained together to mimic a multi-dimensional filter, however, this will unnecessarily increase the computational resources required for the calculations and will reduce the quality of the filtering overall.

**The Kalman Gain** In a Kalman filter, the $\alpha - \beta - \gamma$ parameters are calculated at every filter iteration. These parameters are combined together into the Kalman Gain, denoted as $K_n$, which is the third Kalman equation. The Kalman gain is bounded by: $0 <= Kn <= 1$.

$$\begin{aligned} K_n &= \frac{\text{Estimate Uncertainty}}{\text{Estimate Uncertainty + Measurement Uncertainty}} \\ &= \frac{p_{n,n-1}}{p_{n,n-1} + r_n} \end{aligned} \tag{5.7}$$

If we refresh the State Update Equation (Eq. 5.2) with the Kalman gain, we get:

$$\begin{aligned} \hat{x}_{n,n} &= \hat{x}_{n,n-1} + K_n(z_n - \hat{x}_{n,n-1}) \\ &= (1 - K_n)\hat{x}_{n,n-1} + K_n z_n \end{aligned} \tag{5.8}$$

In the above equation, $K_n$ is the weight (importance) given to the measurement. Conversely, $(1 - K_n)$ is the weight given to the estimate.

**Important note:** When measurement uncertainty is very large, and the estimate uncertainty is small, $K_n << 1$, hence big weight to the estimate and small weight to the measurement. When the opposite is true, $K_n - > 1$, meaning a large weight to the measurement and a small weight to the estimate. This is how the Kalman filter can regulate and smooth out noisy data by knowing the uncertainties.

**The Estimate Uncertainty Update Equation** the fourth Kalman equation is the *Estimate Uncertainty Update*. [5] The estimate uncertainty should approach (converge) to 0 with each filter iteration as the filter improves its guessing accuracy. However, if the measurement uncertainty is large ($K_n << 1$), the estimate uncertainty will converge more slowly. The opposite is true if the measurement uncertainty is small. Basically, the more precise your measurements are, the faster the Kalman filter will converge on the best estimate.

5: Also referred to as the *Covariance Update Equation*.

$$p_{n,n} = (1 - K_n)p_{n,n-1} \tag{5.9}$$

where $p_{n,n}$ is the estimate uncertainty at the current state

$K_n$ is the Kalman gain at the current state

$p_{n,n-1}$ is the estimate uncertainty of the previous state

**The Estimate Uncertainty Extrapolation Equation** The fifth and final Kalman equation is how the filter predicts future uncertainties and is called the *Estimate Uncertainty Extrapolation Equation*. [6] Like with the State Extrapolation Equations, this is done with dynamic models and will be unique to every example. If we refer to the previous models from Equations 5.3a and 5.3b, we can define the Estimate Uncertainty Extrapolation Equations as:

6: Also referred to as the *Covariance Extrapolation Equation*

$$p^{x}_{n+1,n} = p^{x}_{n,n} + \Delta t p^{\dot{x}}_{n,n} \tag{5.10a}$$

$$p^{\dot{x}}_{n+1,n} = p^{\dot{x}}_{n,n} \tag{5.10b}$$

REFERENCE FOR VARIANCe

For more information on variance and its derivation, see

**Putting it all Together** First, the filter is initialized with a first guess ($\hat{x}_{0,0}$) and an associated uncertainty ($p_{0,0}$). These values are passed into the dynamic model equations and Equation 5.10 to predict the state at the first measurement ($x_{1,0}$) and the associated uncertainty ($p_{1,0}$). This is shown as Step 0 in Figure 5.4. Then, we can take a measurement ($z_n$) and record its uncertainty ($r_n$), using the latter to determine the Kalman gain with Equation 5.7. We can then estimate the current state value using Equation 5.8 using the recorded measurement and the Kalman gain ($K_n$). The current state estimate and its uncertainty can then be outputted from the filter and used in applications; this process is shown as Steps 1, 2a, and 2b in the figure below. These values are then fed into the dynamic model to predict the next state and estimate the associated uncertainty (Step 3 in the below diagram). Optionally, these predicted values can be outputted from the model as well, as the application requires. This process repeats for every measurement with $n$ incrementing every time.

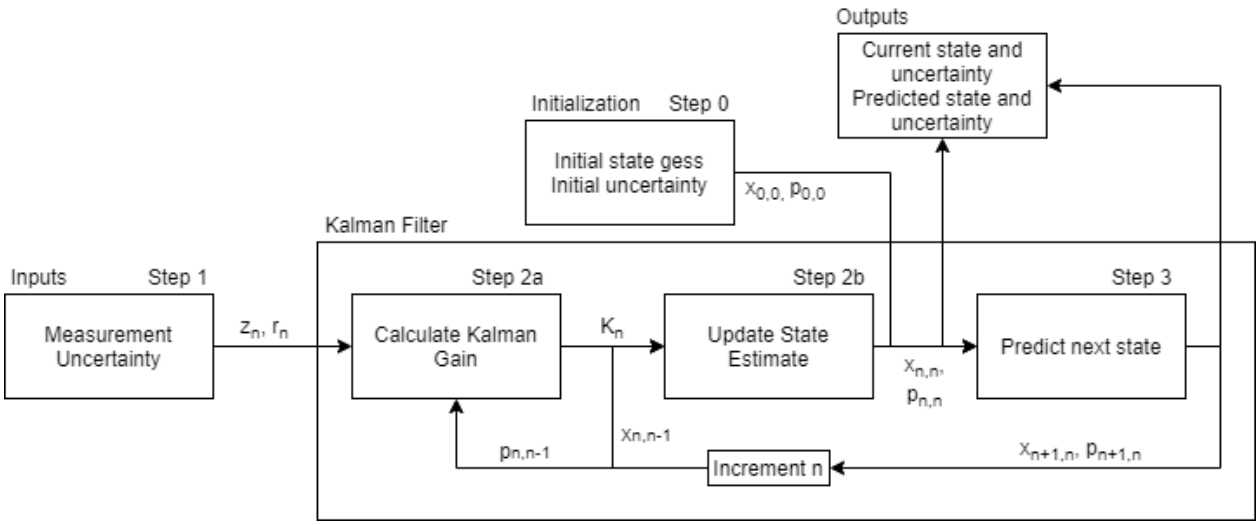



**Figure 5.4:** Process diagram for a Kalman filter.

**Example 5.1.3** In this example, we will be estimating the depth of an ROV using a 1D Kalman filter. We will be assuming the ROV depth remains constant and its location in space does not matter or change. For example's sake, we know for an absolute fact that the ROV is at a depth of 50-meters ($x = 50$ m). We will also assume that our depth sensor has a standard deviation of 5-meters ($\sigma = 5$ m), therefore we will have a measurement variance or uncertainty of 25-meters ($r = 25$ m$^2$)

**Iteration 0**

0 : Estimate depth: $\hat{x}_{0,0} =$ **60 m**

: Estimate uncertainty: $p_{0,0} =$ **225 m**$^2$

3 : Predicted depth: $\hat{x}_{1,0} = \hat{x}_{0,0} =$ **60 m**

: Prediction uncertainty: $p_{1,0} = p_{0,0} =$ **225 m**$^2$

**Iteration 1**

1 : Measure depth: $z_1 =$ **48.54 m**

: Measurement uncertainty: $r_1 =$ **25 m**$^2$

2 : Kalman gain$K_1 = \dfrac{p_{1,0}}{p_{1,0} + r_1} = \dfrac{255}{255 + 25} =$ **0.9**

: Estimated depth: $\hat{x}_{1,1} = \hat{x}_{1,0} + K_1(z_1 - \hat{x}_{1,0}) = 60 + 0.9(48.54 - 60) =$ **49.69 m**

: Estimate uncertainty: $p_{1,1} = (1 - K_1)p_{1,0} = (1 - 0.9)225 =$ **22.5 m**$^2$

3 : Predicted depth: $\hat{x}_{2,1} = \hat{x}_{1,1} =$ **49.69 m**

: Prediction uncertainty: $p_{1,0} = p_{1,1} =$ **22.5 m**$^2$

**Iteration 2**

1 : Measure depth: $z_2 =$ **47.11 m**

: Measurement uncertainty: $r_2 =$ **25 m**$^2$

2 : Kalman gain$K_2 = \dfrac{p_{2,1}}{p_{2,1} + r_2} = \dfrac{22.5}{22.5 + 25} =$ **0.47**

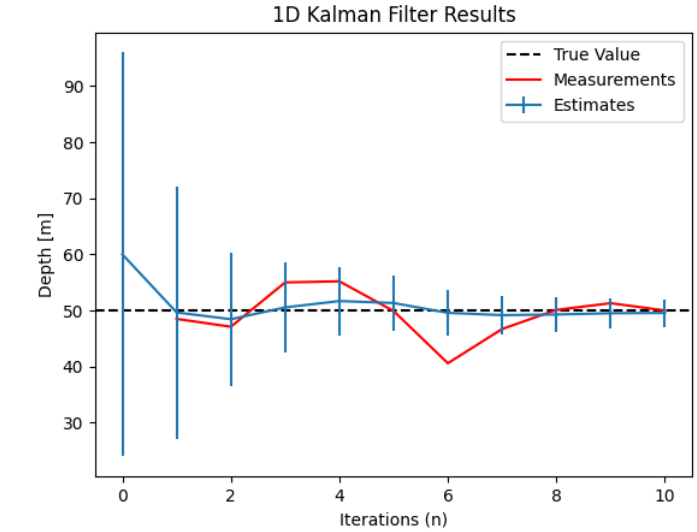
: Estimated depth: $\hat{x}_{2,2} = \hat{x}_{2,1} + K_2(z_2 - \hat{x}_{2,1}) = 49.69 + 0.47(47.11 - 49.69) =$ **48.47 m**

: Estimate uncertainty: $p_{2,2} = (1 - K_2)p_{2,1} = (1 - 0.47)22.5 =$ **11.84 m**$^2$

3 : Predicted depth: $\hat{x}_{3,2} = \hat{x}_{2,2} =$ **48.47 m**

: Prediction uncertainty: $p_{3,2} = p_{2,2} =$ **11.84 m**$^2$

| | Measurements | | Current Estimates | | | Predictions | |
|---|---|---|---|---|---|---|---|
| n | $z_n$ | $r_n$ | $K_n$ | $\hat{x}_{n,n}$ | $p_{n,n}$ | $\hat{x}_{n+1,n}$ | $p_{n+1,n}$ |
| 0 | —- | – | —- | 60.0 | 225 | 60.0 | 225 |
| 1 | 48.5 | 25 | 0.90 | 49.7 | 22.5 | 49.7 | 22.5 |
| 2 | 47.1 | 25 | 0.47 | 48.4 | 11.8 | 48.4 | 11.8 |
| 3 | 55.0 | 25 | 0.32 | 50.6 | 8.03 | 50.6 | 8.03 |
| 4 | 55.2 | 25 | 0.24 | 51.7 | 6.08 | 51.7 | 6.08 |
| 5 | 49.9 | 25 | 0.20 | 51.3 | 4.89 | 51.3 | 4.89 |
| 6 | 40.6 | 25 | 0.16 | 49.6 | 4.09 | 49.6 | 4.09 |
| 7 | 46.7 | 25 | 0.14 | 49.2 | 3.52 | 49.2 | 3.52 |
| 8 | 50.1 | 25 | 0.12 | 49.3 | 3.08 | 49.3 | 3.08 |
| 9 | 51.3 | 25 | 0.11 | 49.5 | 2.74 | 49.5 | 2.74 |
| 10 | 50.0 | 25 | 0.10 | 49.6 | 2.47 | 49.6 | 2.47 |




We can see from the figure that the Kalman filter eventually converges on the true value and considerably smoothes out the noisy measurements. The error bars on the estimate plot (blue) also show that the Kalman filter increases its confidence with every iteration as it converges to the true value. The oscillation shown in the estimates are a result of the gains being slightly off from their ideal value. Tuning the process noise variable (q) can cause the filter to converge quickly and confidently to the true value in fewer iterations.

**Process Noise** In the real world, there are uncertainties in the system's dynamic model. Uncertainty is caused by unanticipated changes in the system due to external factors. This can be drift caused by ocean current, wind blowing a rocket to the side, drag, friction, even time dilation in extreme cases. Generally, these uncertainties can be combined into the Process Noise gain denoted by "$q$" To account for process noise, it must be included in the Covariance Extrapolation Equation (Equation 5.10)

If the model is not known to be good or is very noisy, we can increase the process noise gain to reduce the lag error.

$$p_{n+1,n} = p_{n,n} + q \tag{5.11}$$

# Appendix

# A

# Syllabus

## Course Description

This course is intended to give students exposure to practical electronic design and grant them additional programming experience. All of this will be accomplished through the lens of developing an instrumentation package that can be deployed as a project and capture/record data. Students will use basic Arduino learning kits to cultivate their interests. Graduate students taking the course will be required to follow more industry-related programming practices for the Arduino programming, will be held to a higher product quality standard, and will be required to develop a more advanced instrumentation package than the undergraduates. Course lectures will be supplemented by weekly (or as weekly as possible) demonstrations using real measurement equipment used by the Ocean Engineering department.

### Meeting Times

#### Lectures

Monday, Wednesday, Friday; 10:00-10:50 (0h50min)
Fall 2022, August 22 - December 16
Link 325

#### Office Hours

Monday, Wednesday, Friday; 13:00-15:00
Or, by appointment (preferred)
Link 155 or Frueauff 100

### Objectives & Outcomes

The goal of this course is the provide a basic instruction on practical electrical engineering and computer programming through the lens of instrumentation design. After introducing and refreshing basic concepts like electrical components, digital electronics, and C++ basics, students will be taught how industry professionals make real-world measurements using various instruments. At the end of the course, students will be able to:

1. Understand how various measurement devices capture, record, and transmit information to researchers and engineers
2. Have a fundamental knowledge of programming Arduino-based microcontrollers
3. Have a fundamental knowledge of Printed Circuit Board design, manufacture, and assembly

4. Apply the fundamental concepts learned in lecture and through demonstrations on a practical Individual Course Project.

## Target Audience & Prerequisites

On the undergraduate side, this course is intended for students who do not have a basic familiarity with basic electronics and programming with Arduino. The ELEGOO Arduino Starter Kit recommended for this class has supplementary information that will assist students in understanding exactly what is occurring with various projects. Students are strongly encouraged to delve deeper into the topics covered in class and pursue a challenging Individual Course Project. Individuals with a strong drive to learn independently will benefit greatly during this course.

On the graduate side, this course is designed for students who have a basic knowledge of electronics and programming with Arduino or C++. Embedded design experience is also desired as it will be most beneficial during the Individual Course Project. While the ELEGOO Arduino Starter Kit will give the basics, graduate students will be expected to expand upon the projects covered in the kit and use more sophisticated programming techniques. The Individual Course Project will also need to be well considered and adequately demonstrate the student's knowledge and motivation to learn outside the classroom.

## Course Resources

The course material will be regularly updated on Canvas and GitHub. The GitHub repository will have all of the supplementary and study material required by students. The Canvas page may also contain these files, but GitHub will be the most up to date.

Students will be *required* to purchase their own Arduino learning kit for this course. They can be easily found on Amazon.

1. ELEGOO Mega R3 Complete Ultimate Starter Kit - $66
2. UCTRONICS Mega R3 Complete Ultimate Starter Kit - $69
3. LAFVIN Mega R3 Starter Kit - $46

Graduate students will also be required to order a Raspberry Pi - or equivalent Single Board Computer (SBC), for this course. It is strongly recommended to purchase a Raspberry Pi 400 Full Computer Kit (Adafruit) with an accompanying T-Cobbler GPIO Breakout (Adafruit).

Due to the recent chip shortage, Raspberry Pis may be hard to find or prohibitively expensive. If students are unable to acquire their own Raspberry Pis or similar SBCs, they may ask for a loaner unit from the instructor. **Please note that if the loaner unit is *not* returned by the end of the semester, the student will be given an "Incomplete" grade until the instructor has been given the device.**

**Recommended Readings** Students are not required to purchase any particular textbooks or reference guides, but the following are recommended:

1. The Art of Electronics
2. The Art of Electronics: Lab Course
3. The Arduino Cookbook
4. Programming Arduino
5. The Arduino Technical Reference Handbook

## Grading Policies

This course covers several student performance metrics: (i) Assignments, (ii) Participation, (iii) Midterm Exam, and (iv) an Individual Course Project. The weighting for this metrics is below:

| Metric | Weight |
|---|---|
| Assignments | 20% |
| Participation | 10% |
| Midterm Exam | 30% |
| Individual Course Project | 40% |

Students will be assigned the following letter grade and GPA quality points based on their weighted sum assignment scores according to:

| Score | Letter Grade | Quality Points |
|---|---|---|
| 90-100 | A | 4 |
| 80-89 | B | 3 |
| 70-79 | C | 2 |
| 60-69* | D | 1 |
| <60 | F | 0 |

## Course Schedule

| Week | Monday | Wednesday | Friday |
|---|---|---|---|
| 1 | Syllabus | Software Setup | Project and Kit Discussion |
| 2 | Project 1 Review | Digital Logic | Multiplexers, Registers, States |
| 3 | **NO CLASS** | Project 2 Review | Electrical Components |
| 4 | Analog Measurements | Distance Sensors and Measurement | YSI/HOBO Demo |
| 5 | Project 3 Review | Environmental Sensors | Launchsonde Demo |
| 6 | Inertial Measurement Units | Sensor Fusion/AHRS Design | Lowell/Thetis Demo |
| 7 | Work Day | Transducers | Sidescan SONAR Demo |
| 8 | **NO CLASS** | Project 4 Review | Midterm Exam |
| 9 | Midterm Review | Intro to Fusion 360 | ECAD-Introduction |
| 10 | ECAD-Schematics | ECAD-Libraries and Creation | Schematic Review |
| 11 | ECAD-PCB Basics | Measurement Filtering Techniques | Data Processing Demo |
| 12 | Gerber Gen. and Order | PCB Manufacture Techniques | **NO CLASS** |
| 13 | PCB assembly (hand) | PCB assembly (PnP) | Robotics Demo |
| 14 | Work Day | **NO CLASS** | **NO CLASS** |
| 15 | Data Storage/Transmission | Power Considerations | Work Day |
| 16 | ICP Presentations | ICP Presentations | ICP Presentations |

* Undergraduate students only. **Graduate students will fail below a 70.**

## Assignment Schedule

| Assignment | Due Date |
|---|---|
| Get Arduino Kit, Raspberry Pi[†] | August 26 |
| Project 1: RGB LED | August 29 |
| Project 2: Digital inputs and interrupts | September 7 |
| Project 3: 7-Segment Counter, ICP: Proposal[†] | September 19 |
| Project 4: Sensor module and display | October 12 |
| Midterm Exam | October 14 |
| ICP: Schematic | October 28 |
| ICP: PCB design | November 7 |
| ICP: Assembled PCB | December 2 |
| ICP: Final presentation | December 5 |
| ICP: Final report | December 12 |

## Other Important Dates

| Event | Date |
|---|---|
| Last day to drop without a "W" | August 31 |
| Last day to drop with a "W" | October 31 |

## Course Policies

### Online Course Management

This course will be published in the online learning tool, Canvas, and will be made readily available to all students. Canvas provides an online cross-platform solution for students and instructors to engage and will handle all of the assignment submissions and *preliminary* grades for students. Assignments will be issued and submitted through the Canvas platform and students will be expected to submit the required documents by the due date and time listed on the assignment submission box. If, for whatever reason, assignments are unable to be turned in through Canvas, they must be emailed to the instructor and timestamped by the date and time established.

Grades will also be posted on Canvas for students to track their progress in status in the course. However, there is no guarantee that the posted grade in Canvas will represent the final course grade submitted to the registrar, nor may it be up-to-date if grade corrections are necessary. In the event a student is not satisfied by their grade posted in Canvas, they are more than welcome to schedule a *face-to-face* meeting with the instructor to discuss. Emailed requests to change grades may be considered but it will be more effective to meet with the instructor in-person to ensure the change is made properly.

Canvas will also have a "Files" section where students can find relevant course resources and documents to aid in their studies. Students may request certain documents be uploaded to Canvas to share with their classmates and they may download all files freely if they wish to have their own local copies. The Canvas may be periodically updated to reflect changes or addendums to course content, but it may not necessarily reflect the most up-to-date

[†] Graduate students only

information. For the most updated course material, please consult the class GitHub Repository. *You may be surprised what you find in there...*

**Remote Learning** The COVID-19 pandemic has radically changed the learning environment we are in. The traditional ideology of in-person, all the time, has been fundamentally destroyed by two years of remote working. As the pandemic has lessened, Florida Tech has resumed in-person classes, full-time. However, the benefits of remote and asynchronous learning cannot be understated to those that can effictively learn that way, or need to for whatever reason.

Therefore, the class will be taught with a hybrid option. Lectures, projects, and assignments will be primarily done in-person in accordance with Florida Tech policies and instructor preferences but, students will have the option to perform assignments or attend lectures remotely. There is a Zoom link below that will be active for all of the class lectures; lecture presentations and classes will be recorded and uploaded to Panopto. Students who elect to perform remote learning will need to submit a video of their working projects on Canvas as a part of their submission. They are also encouraged to be in-person for the ICP presentations at the end of the course to minimize technical difficulties. However, accomodations will be made if the presentation is given remotely.

## Weekly Projects

For the first couple of weeks of the course, students will be expected to put their Arduino starter kits to use with various projects. These are intended to slowly ramp up in complexity and give students a glimpse of what practical electronics and programming looks like. Projects will be due by the start of class on the day specified in the Assignment Schedule section.

**Late Policy** Submissions for assignments will be accepted up to the date of the students ICP presentation. However, for every day (24 hours) after the deadline, starting the minute after the deadline passes, 10 points will be deducted from the assignment down to an absolute maximum of 50%.

## Individual Course Project

The Individual Course Project (ICP) is designed to encompass all of the elements taught throughout the course into a single package. Undergraduate students will be tasked with taking one of the Arduino projects available in their starter kits and converting it to an Arduino-compatible shield. Essentially, taking the circuit they already made on a breadboard, drawing the schematic in an ECAD software like Fusion 360, routing the PCB in the same software, and order and assembling the final PCB. Students will be tasked with writing up their ICP in a comprehensive report and giving a quick final presentation at the end of the semester. Graduate students will be asked to perform the same task, but with a task that has appropriate difficulty and merit pursuant to their experience. These students will be required to submit an ICP proposal before beginning their assignment that must be approved by the instructor.

**Financial Policy** Students will be required to purchase their PCBs and any additional components for their ICP not already available to them in their Arduino kits or by the Underwater Technology Lab. Currently, each student is expected to pay around $5 for the PCB order and most undergraduate students should be using components already present in their Arduino kits. The final amount each student will have to pay for their PCB will be determined at the time of ordering by the instructor.

Sponsorship by the UTL may be requested and may be granted on a case-by-case basis. Students with ICPs that prove useful to the lab's operation may have their fees waived depending on several external factors. This will

be determined by the time of ordering. Students wishing to be sponsored should contact the instructor ASAP for details.

**Failure Policy** *FAILURE IS AN OPTION.* It is well-understood that students may have a non-functional ICP at the end of the semester. **That is okay!** The university environment is about learning and especially learning while failing. Therefore, students who do not have a functional PCB have two options for recourse:

1. they may explain, in copious detail, what the failure on the PCB was and how it would be addressed in the next revision, were it to be manufactured.
2. they will have until the start of their ICP presentation to submit a fixed PCB that is working, but may have "bodges" or other modifications

If students elect for Option 1, they will receive penalty points on their ICP. These points will be subjectively detracted according to the instructor. As a general guideline, *the simpler the mistake, the harsher the penalty will be*. For instance, a missing connection in the schematic or trace on the PCB will have more points deducted than two components not talking due to EMI or other nuanced electrical engineering problem. *Attention to detail is important!*

**Late Policy** Late submissions on any part of the ICP will not be tolerated. Starting one minute after the assignment is due and for every 24 hours thereafter, 10% will be deducted from the **overall** ICP grade. Simply put, if four ICP-related assignments are turned in a minute late, then the student will be limited to a maximum grade of a C in the course.

Special consideration may be made for students who experience extraordinary hardship in submitting their ICP-related assignment before the deadline. Appeals for special consideration must be made **within one hour** of the specified deadline passing. Appeal requests made after this window will not be considered. Period. If an appeal is granted, no penalty points will be applied and the deal reached by the instructor and student will supersede the assignment requirements. *Please let the instructor know if you have any difficulties at all. We will try to work around them with you!*

Additionally, students who do not submit a PCB gerber package by the order deadline (November 11) may end up forfeiting their potential grades. This deadline is in place to ensure ample time for PCB assembly and testing and all orders will be placed at the same time. If a student does not submit the PCB gerber package by the deadline, the onus and financial responsibility will be on them to order the PCBs in a timely manner such that they can still assemble and test their ICP.

## For Students with Handicaps and/or Disabilities

Students with handicaps and/or disabilities will be given special considerations depending on their condition and Florida Tech policy. Please meet with the instructor privately to discuss any concerns or arrangements.

## Academic Dishonesty Policy

Students who are caught cheating or plagiarizing will be given an audience with the instructor to discuss the situation. Some cases are simple mistakes or coincidences with no ill-intent and can be rectified with a penalty to the assignment grade. Severe cases of rampant cheating or plagiarism *with* malice or gross negligence will be referred to the Dean of Students in accordance with academic policy present in the Florida Tech handbook. Students who are referred to the Dean of Students may forfeit their overall grades for the course and may face academic probation, suspension, or expulsion from Florida Tech - as the Dean determines.

Academic Dishonesty incidents will not be considered until the assignment is officially submitted. Therefore, students are strongly encouraged to meet with the instructor with questions about if a portion of their assignment could be considered academically dishonest.

## Title IX

Title IX of the Educational Amendments Act of 1972 is the federal law prohibiting discrimination based on sex under any education program and/or activity operated by an institution receiving and/or benefiting from federal financial assistance. Behaviors that can be considered "sexual discrimination" include sexual assault, sexual harassment, stalking, relationship abuse (dating violence and domestic violence), sexual misconduct, and gender discrimination. You are encouraged to report these behaviors. The instructor and all other faculty and staff are *legally required* to report an incidents they know about, directly or otherwise, to institute administration.

**Reporting** Florida Tech can better support students in trouble if we know about what is happening. Reporting also helps us to identify patterns that might arise - for example, if more than one complainant reports having been assaulted or harassed by the same individual. We can only help if you let us.

## Regarding Unusual or Extraneous Circumstances

*"If anything can go wrong, it will" - Murphy's Law*

It is well-understood that incidents will occur in this fickle thing we call life. Students and instructors alike are human and we cannot predict what will happen in the next five minutes let alone the next few days. In the event of something unexpected or unusual, please contact the instructor ASAP. Each case will be considered for its severity, impact on student well-being, and impact to the student's education. Some unforeseen circumstances may warrant an extension to an assignment deadline, a reschedule of a test, or other remedies depending on its severity.

Conversely, the instructor may have incidents where class may need to be cancelled or an assignment delayed. The instructor reserves the right to maneuver the class as they see fit, but it would not be possible without the participation of the students. If the instructor is late to class, please allow for up to 15 minutes before departing. If the instructor needs to cancel a class, there will be an announcement made through the Canvas page, ideally, well ahead of the class start time. Additionally, the instructor may elect to hold class in a remote session through Zoom, should that be a necessary option.

Students, please try to be flexible and understanding, and the instructors will do the same.

## Course Evaluations

Since this course is a relatively new offering, student feedback is incredibly important to the instructor and administration. Therefore, students will be granted an extra 2% to their overall grade in the course, if *and only if*, the course evaluations have a 100% response rate. This includes both the undergraduate and graduate sections.

As always, honest feedback is appreciated and these evaluations are anonymous. They are intended for the students to identify problems with the course material, teaching style, or teacher conduct during the course such that the course and instructor may improve for the next semester. Students are highly encouraged to submit feedback for other courses as well.

# B

# Project 1: RGB LED Cycler

## Overview

In this project, you will use the RGB LED within your Arduino kit to learn the basics of digital inputs, PWM output, switch-case statements, functions, and various loops. The goal of this project will be to activate different modes on the RGB LED using a button to cycle through them and a switch-case statement to execute the appropriate functions. Since this is a beginning project, there is some psuedocode below to give you an idea of how the code should be structured. It will be up to you to determine what specific functions you want your LED to perform.

## Requirements

For this project, you must successfully hook up a button, an RGB LED, and any supporting passive components your Arduino and breadboard. You must also have at least three different functions that your LED performs - one of which must include loop. A great example would be the rainbow loop or the breathe loop.

**Reminder:** The LED *must* be connected to the Arduino pins with resistors in series in order to protect the diodes. Please consult your Arduino kit manual for specific resistor values

## Submission

You will be required to submit the following on Canvas:

1. a well-organized and documented schematic of the project setup
2. the source code file

Please package all of these items into a compressed (zipped) folder and upload them to Canvas.

## Grading

You will be graded along the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 60 |
| Schematic neatness | 20 |
| Code neatness | 20 |
| Mystery extra credit | 10 |

## Extra Credit

**Hint:** Research the problem with button inputs

If you are willing to dig in a little bit more, this project has a couple of opportunities to earn extra credit points at the discretion of the instructor. Implementing something unique with the RGB LED or the button input will warrant the extra credit points. If you want to try and get the extra credit points, please let the instructor know in the submission and detail why you believe you earn the points.

## Psuedocode

```
Program: RGB LED Cycler

Define an input button pin as some Arduino pin
Define the red LED pin as some Arduino pin
Define the green LED pin as some Arduino pin
Define the blue LED pin as some Arduino pin
Define the number of LED functions as the number of modes you want

Initialize the current LED mode as 0

Function: Setup
    Initialize Serial communication for debugging
    Initialize input pins
    Initialize output pins

Function: Loop
    Check for button pressed
    If button is pressed then
        If the current LED mode is the last LED mode
            Reset the current LED mode to the first one
        Else
            Increment the LED mode by 1
    Switch the current LED mode
        Case the current LED mode is set to 0
            Execute first LED function
        Case the current LED mode is set to 1
            Execute second LED function
        Case the current LED mode is set to 2
            Execute third LED function
        ...

Function: First LED Function
    [YOUR CODE HERE]
...
```

# C

# Project 2: Digital Inputs and Interrupts

## Overview

For more information on digital inputs, interrupts, and filtering, please consult Chapter **??** and Example **??**

Make these parts and put it into the book

This project is designed to introduce you to concepts of filtering digital inputs and handling interrupts inside a microcontroller's program. It will also introduce a little bit about the Serial monitor. You will wire up three buttons to your Arduino and use them to turn on and off an LED. One button will not have any filtering or regulation at all; another will have a software-defined debounce filter; and the third will have a low-pass RC circuit filter. You will use an interrupt attached to each input pin to increment a counter and display the current press count to the Serial monitor. This will demonstrate how inputs to a program need to be considered as false readings can create undesired program behaviors. You should notice that the unregulated button has difficulty incrementing the counter by 1 every time you press it and may not always switch the light from off to on or vice versa. The software-regulated button input should be a little more accurate and reliable, but you may still notice some problems. The hardware-regulated button input should throw few, if any false positives - always incrementing by 1 with every press and switching the LED state consistently.

## Special Consideration

For information on which pins to use and how to set up the interrupts, see `https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/`

The project as intended, requires three interrupt-capable digital input pins. The Arduino Uno and other ATmega328 devices can only handle two interrupt pins. The Arduino Mega and some other devices can handle more than two interrupt pins. If you have an Arduino Uno, you may request a Mega from the instructor or one of your class mates for this project. Please note that if you elect to loan a unit from the instructor, this project *will not* be graded until that loaner unit is returned.

Alternatively, you may demonstrate the project in two parts. In the first part, you will implement the unregulated and software-defined filtered button input. In the second part, you will implement the unregulated and hardware filtered button input. Please discuss this with your instructor for additional information.

## Requirements

**Reminder:** The LED *must* be connected to the Arduino pins with resistors in series in order to protect the diodes. Please consult your Arduino kit manual for specific resistor values

For this project you will be required to demonstrate:

- appropriately wiring three buttons and three LEDs to your Arduino and breadboard with all supporting components
- reading in the three digital inputs and attaching appropriate interrupt service routines to the pins
- incrementing counters for each button press and switching the corresponding LED on and off
- appropriately printing the counters to the Serial monitor

## Submission

You will be required to submit the following on Canvas:

- a video of the project working with narration of what is occurring
- a well-organized and documented schematic of the project setup
- the source code file
- a printout of the Serial monitor showing a counter increment when you press the corresponding button

Please package all of these items into a compressed (zipped) folder and upload them to Canvas.

## Grading

You will be graded along the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 60 |
| Printout of Serial monitor | 20 |
| Schematic neatness | 10 |
| Code neatness | 10 |
| Extra credit | 5 |

## Extra Credit

An extra credit opportunity exists with this project! To earn the credits, you must program the buttons to increment their counters and change LED states *after* holding the button for a certain period of time. How you implement this is up to you. If you want to try and get the extra credit points, please let the instructor know in the submission and detail why you believe you earn the points.

# D

# Project 3: 7-Segment Display Counter

## Overview

This project will introduce you to the shift register, the seven segment display, and the active piezoelectric buzzer. You will incorporate many of the lessons previously learned including digital inputs and filtering, interrupts, digital outputs, mode selection, and counting.

## Undergraduate Students

You will only be focussed on working with the four digit seven segment display counter. You wire up the display with the 74HC595 shift register IC and increment the display based on the current millisecond reading of the Arduino since time on *or* a count reset button press. When the reset count button is pressed, the counter will reset back to 0 and a buzzer will sound for auditory feedback.

In decimal, the display can only display up to four non-negative digits meaning you can only display a maximum of 9999. Make sure your code is capable of re-setting the counter when this value is reached. *For extra credit:* figure out how to make it count higher than 9999.

For heaps of extra credit, you may elect to do the graduate student section of this project.

Since this project is more complicated than previous ones, the schematic for the breadboard and the psuedocode are provided in this document. It is *strongly* suggested you start this project early and do not wait to work on it.

### Requirements

*For extra credit:* Figure out how to change the buzzer's volume without changing the circuit every time

For this project you will be required to have the following:

- A button wired to an interrupt-capable input
  - These buttons will be attached to digital interrupts to trigger the counter reset and sound a buzzer
- The current counter value will be displayed on a 4-digit, 7-segment display

## Pseudocode

```
Program: 4-Digit 7-Segment Display Counter

Define the 74HC595 data pin as some Arduino pin
Define the 74HC595 latch pin as some Arduino pin
Define the 74HC595 clock pin as some Arduino pin
Define the digit control pins as an array of some Arduino pins
Initialize the display digital values as an array of values with Digit One being ind

Initialize an array of hex values corresponding to letters and numbers

Define the button reset pin on some interrupt-capable Arduino pin

Define the buzzer pin as some Arduino pin
Define the buzzer active time as some time in milliseconds
Initialize the buzzer active flag as false
Initialize the buzzer start time as 0

Initialize the start time as 0 or the current millisecond value

Function: Setup
    Set up 7-segment display pins as outputs
    Set up button input pin as an input pullup
    Attach the counter reset ISR function to the button input to trigger on the fall
    Set up the buzzer output pin
    Set the buzzer output pin to LOW to ensure buzzer is off

Function: Loop
    Update the display
    If one second has passed since the last execution then
        Reset execution timer
        update the counter

    If the buzzer active flag is true then
        If the buzzer active time has not elapsed then
            Turn the buzzer on
        Else
            Set the buzzer active flag to false
            Turn off the buzzer

Function: Turn off Display
    For every pin in the digit display pins array
        Set the pin to low to turn off the digit

Function: Display
    Arguments: Index of value to be displayed, found in the table of values
    Set the latch pin to low to enable shift register writing
    Shift out the value of table at the specified index to the shift register
    Set the latch pin to high to disable shift register writing

Function: Update Display
    For every digit in the display digits array
        Turn off the display
```

```
        Display the value of the current digit
        Turn on the specific digit in the display
        Delay a little bit to give everything time to process (~1 ms)
    Turn off the display to reduce power consumption

Function: Update Counter
    If every digit in the digit display array is equal to 9 then
        Reset every value in the digit display array to 0

    Initialize the an increment value flag as true
    For every digit in the digit display array
        Store the value of the specific digit in a temporary variable
        If the increment value flag is true then
            Increment the temporary variable by one
            Set the increment value flag to false
            If the temporary variable is greater than 9 then
                Reset the specific digit to 0
                Set the increment value flag to true
            Else
                Set the specific digit to the new value of the temporary variable

Function: Counter Reset ISR
    For every digit in the display digits array
        Set the specific digit value to 0
    Set the buzzer active flag to true
    Set the buzzer start time to the current millisecond value

```

## Schematic

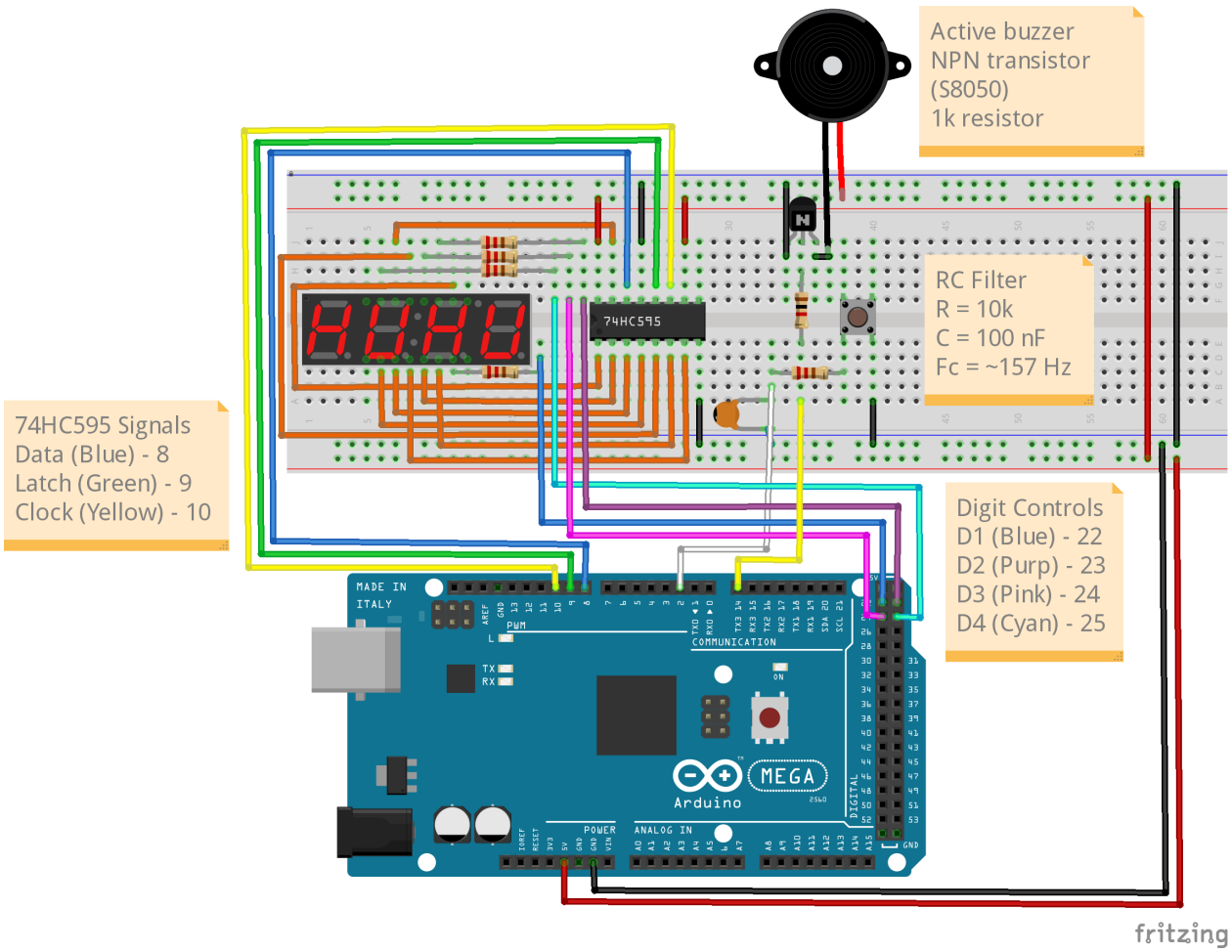



**Figure D.1:** Breadboard schematic for Project 3 (Undergraduate Version)

## Graduate Students

You will wire up five (5) buttons to your Arduino as digital inputs. Three buttons will determine the counter increment (e.g. 1, 10, 100), and two buttons will determine the increment direction (increasing or decreasing) and be attached to interrupts to trigger the counter change. When you hold one of the size buttons and then press either the decrement or increment button, an internal counter will increase or decrease by the size specified and update the number in the 4-digit 7-segment display. When the decrement button is held for a certain amount of time, the counter will be reset back to 0. For each press of the decrement or increment button, you will make an active buzzer sound as audible feedback to your inputs. *For extra credit:* Figure out how to play a different tone for increment or decrement.

*Reminder:* Some filtering, either software or hardware-based will be required to accurately change the counter and prevent false readings.

Since this project is a little more complicated than previous ones, the schematic has been provided. However, the coding portion is up to you.

The display is only capable of showing up to four non-negative digits. So the limits of your counter will be between 0 and 9999. *For extra credit:* Figure out how to make it count higher.

### Requirements

For this project you will be required to have the following:

- A button wired to an interrupt-capable input
  - Three (3) of those buttons will determine how much the counter changes
  - Two (2) of them will determine the counter's change in direction (either increment or decrement)
    - These buttons will be attached to digital interrupts to trigger the change in the counter
    - Each press of these buttons will sound a buzzer for auditory feedback
- The current counter value will be displayed on a 4-digit, 7-segment display

## Schematic

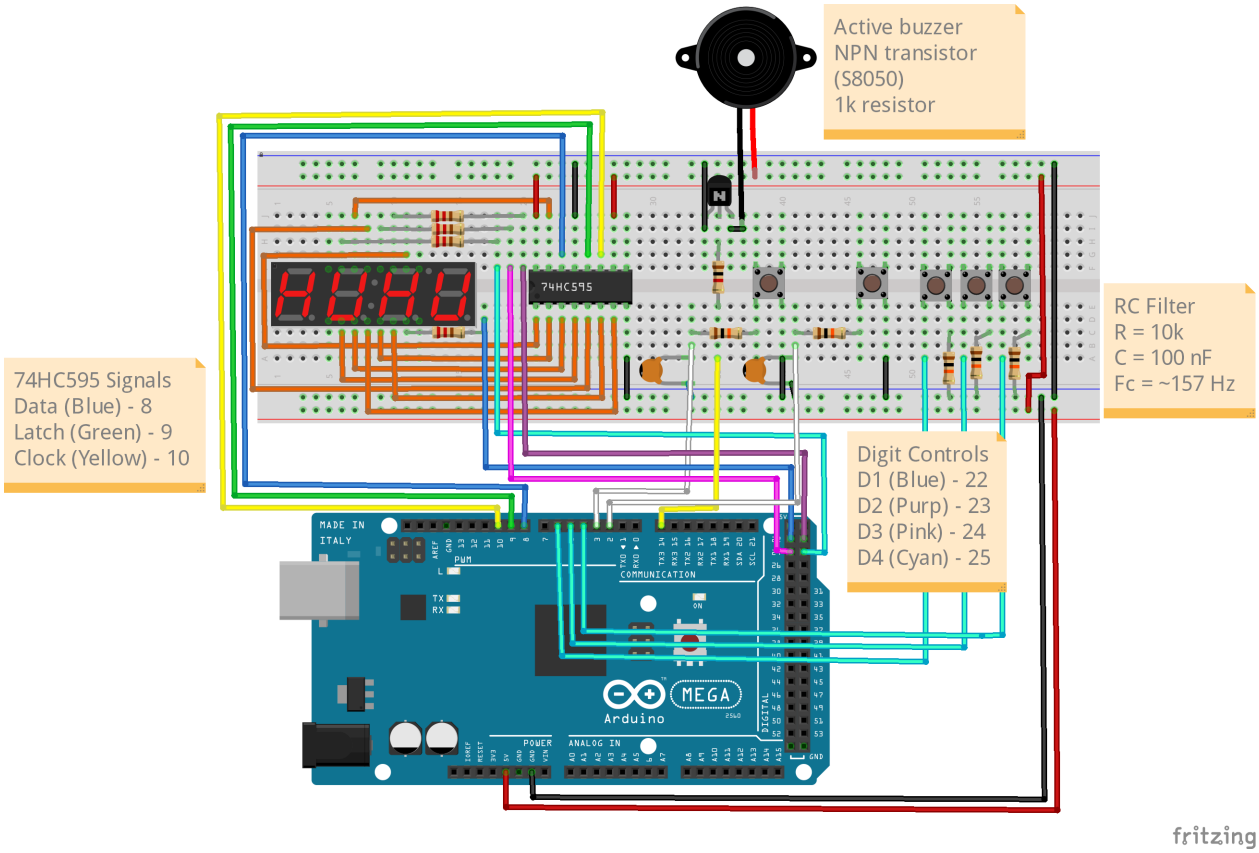



**Figure D.2:** Breadboard schematic for Project 3 (Graduate Version)

## Submission

You will be required to submit the following on Canvas:

- a video of the project working with narration of what is occurring
- the source code file
- a still image of your breadboard and Arduino setup

## Grading

You will be graded along the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 60 |
| Breadboard neatness | 20 |
| Code neatness | 20 |
| Extra credit | 10 |
| Extra extra credit * | 25 |

If you are willing to dig in a little bit more, this project has a couple of opportunities to earn extra credit points! If you want to try and get the extra credit points, please let the instructor know in the submission and detail why you believe you earn the points.

## Miscellanea

Since working with bits and registers and 7-segment displays can be tricky for beginners, the binary sequences for various numbers and letters are provided in the table below. Note that these values will change depending on how the shift register is wired to the 7-segment display. The values provided are known and tested to have worked with the schematics shown above.

* Undergraduate students only

| Character | Hex Code |
|---|---|
| 0 | 0x3F |
| 1 | 0x06 |
| 2 | 0x5B |
| 3 | 0x4F |
| 4 | 0x66 |
| 5 | 0x6D |
| 6 | 0x7D |
| 7 | 0x07 |
| 8 | 0x7F |
| 9 | 0x6F |
| A | 0x77 |
| b | 0x7c |
| C | 0x39 |
| d | 0x5E |
| E | 0x79 |
| F | 0x71 |
| | 0x00 |

# E

# Project 4: MPU6050 Accelerometer and LCD Display

## Overview

This project will introduce to you the concepts of reading sensor values, data storage inside a structure, and displaying recorded values on an LCD display. You will read the accelerometer and gyroscope readings from the MPU6050 sensor (also referred to as the GY-521 module) and store them in an internal structured data packet for reporting to the LCD module. Your LCD1602 module will have several different pages: one page will show the accelerometer values, another will show the gyroscope values, a third will show roll and pitch data, and others will show any additional information you desire. A button will be used to cycle between the different pages and a buzzer will be used to provide auditory feedback on every press.

See the Extra Credit section for more details.

## Graduate Students

You will have some additional work for this project. Since an accelerometer provides accelerations, it will be your task to extrapolate (integrate) velocity and position as well as the other tasks outline above. You will be required to store this information in your storage packet and display the velocity and position values on separate pages on the LCD module. Since the MPU6050 module lacks a magnetometer, it will also be your task to integrate the heading angle and save it to the telemetry packet. You will also program a button to act as velocity, position, and heading reset after you hold the button down for a certain amount of time.

## Requirements

For completion of this project, you must demonstrate the following:

- Successful wiring of the MPU6050 IMU, LCD1602 module, piezoelectric buzzer, and button input (with appropriate debounce filtering)
- Reading and calculating a variety of values from the MPU6050 sensor and storing them in a telemetry packet.
    - Acceleration X,
    - Acceleration Y,
    - Acceleration Z,
    - Gyroscope X,

  - Gyroscope Y,
  - Gyroscope Z,
  - Roll,
  - Pitch
- Display the values on distinct pages on the LCD display with appropriate labels and units
  - Page 1: Acceleration values
  - Page 2: Gyroscope values
  - Page 3: Orientation values
- Use a button tied to an interrupt service routine on the Arduino to cycle between the different LCD pages. Use a buzzer to provide auditory feedback on each button press.

## Submission

You will be required to submit the following on Canvas:

- a well-organized and documented schematic of the project setup
- the source code file

## Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 50 |
| Well organized and neat schematic | 20 |
| Well organized and neat source code | 30 |
| Extra Credit Challenge 1 * | 25 |
| Extra Credit Challenge 2 | 25 |
| Extra Credit Challenge 3 | 25 |
| Extra Credit Challenge 4 | 25 |

## Extra Credit

There exists many opportunities for extra credit on this assignment. Ultimately, this is your chance to really explore and learn how to interact with sensors on a low level and gain an in-depth understanding of the relationship between microcontrollers and sensors. To that end, it is highly encouraged to pursue the following challenges and earn as many points as possible. A successful demonstration of *all* of the challenges below will earn you **an exemption from the midterm exam**. If you fail to finish all the

* Graduate students: this will count towards your normal score

challenges, the appropriate amount of points will still be given to you and used as standard extra credit for this assignment.

In your submission, please include a note of which challenges you have completed. Your source code will also need to broken into specific files for each challenge, or have each challenge highlighted clearly in different sections. *If it is not clear where your challenge-specific code is, it may not be counted.*

## Challenge 1: Sensor Fusion - Attitude and Heading Reference System

**Note:** This challenge is *mandatory* for graduate students.

In this challenge, you are tasked with combining the accelerometer and gyroscope readings from the MPU6050 into a unified AHRS. There are a multitude of different ways to accomplish this task, but at the end, you should be able to store the values into a telemetry packet and display them on separate pages on your LCD module. You will be required to capture and store:

- Acceleration (X, Y, Z)
- Gyroscope (X, Y, Z)
- Orientation (Roll, Pitch, Yaw [heading])
- Velocity (X, Y, Z)
- Displacement (X, Y, Z)

A button will be used to reset the velocity, position, and heading when held for a certain amount of time.

**FOR CREDIT:** You must scroll through the different pages of the LCD module and show that you are capturing the appropriate data. You must also demonstrate resetting the velocity, position, and heading values with the reset button.

## Challenge 2: AHRS Filtering

For information, consult the lecture notes Chapter 5, Section 5.1.5 and This excellent tutorial (focus on the 1-dimensional application for now).

You should have noticed from the AHRS challenge that the velocity, displacement, and orientation values are extremely inaccurate and drift significantly over time. This is due to the inherent inaccuracies and drift built into the accelerometer and gyroscope readings. In that challenge, you were working directly with the raw values without filtering them out. These values accumulate over time, causing the massive divergence.

Therefore, to combat these inaccuracies, we will implement some filters to increase the estimation accuracy. First, you will implement a Kalman filter to "smooth" out your raw acceleration and gyroscope data to give you better readings to use. Here, it will be beneficial to a 6-axis Kalman filter which can be created using the Arduino Kalman libary by *rfetick*. It is strongly recommended to study Kalman filtering and the basic theory before attempting this. It will make a lot more sense after some studying!

You will then feed the filtered acceleration and gyroscope data into a Mahony Filter that can generate the appropriate orientation data. You will not need to understand the theory of this filter, you may just use the example code provided by the Arduino Mahony library.

If you established and tuned the Kalman filter correctly, you should have some more accurate readings of velocity, position, and orientation. Again, update the telemetry packet with these values and display them on the LCD module on separate pages.

**FOR CREDIT:** you must submit a capture of a plot created from your data that shows one accelerometer axis (X, Y, or Z). One line must be the method you used previously, and one line must be the filtered estimate. Quickly discuss the results in your submission.

## Challenge 3: Removing Gravity

For this challenge, we will convert the acceleration signals into linear acceleration, put that information into the telemetry packet, and display it on a separate page of the LCD module. For this challenge, it is strongly recommended to use a quaternion rotation as this method is immune to gimbal lock - a phenomenon where rotations require additional euler transformations at certain orientations.

To assist you, the code to convert from Euler angles to Quaternion is below: [1]

1: You may also want to look into the Adafruit IMU maths helper libraries

```
struct Quaternion
{
    double w, x, y, z;
};

// yaw (Z), pitch (Y), roll (X)
Quaternion ToQuaternion(double yaw, double pitch, double roll)
{
    // Abbreviations for the various angular functions
    double cy = cos(yaw * 0.5);
    double sy = sin(yaw * 0.5);
    double cp = cos(pitch * 0.5);
    double sp = sin(pitch * 0.5);
    double cr = cos(roll * 0.5);
    double sr = sin(roll * 0.5);

    Quaternion q;
    q.w = cr * cp * cy + sr * sp * sy;
    q.x = sr * cp * cy - cr * sp * sy;
    q.y = cr * sp * cy + sr * cp * sy;
    q.z = cr * cp * sy - sr * sp * cy;

    return q;
}

```

From there, you will define a Quaternion vector for gravity defined in the North, East, Down (NED) reference frame. [2] Then, you will rotate the gravity quaternion from the NED reference frame to the body reference frame using:

2: *Hint:* it's just a normal gravitational acceleration vector with another term for the 'w' component of the Quaternion.

$$Q_{g,b} = Q_b^{-1} * Q_{g,NED} * Q_b$$

Finally, subtract the X,Y,Z components of the rotated Quaternion vector for gravity from the recorded body accelerations and you will have linear acceleration. *Easy right!?*

As always, display these values on a new page of the LCD module!

## Challenge 4: MPU6050 Motion Detect Interrupt

The interrupt pin on the MPU6050 can be configured to serve as a motion detection trigger. This challenge's task will be to put the Arduino into a sleep mode after the initialization and use the motion detection interrupt to wake the Arduino up and read sensor data for a defined period of time before going back to sleep.

**FOR CREDIT:** You must demonstrate the Arduino not capturing any data (asleep), then you jolting the Arduino awake by moving the accelerometer, and the values changing on the LCD module. After a set time, these readings should stop updating as the Arduino goes back to sleep.

# F

# Individual Course Project: Undergraduate

## Overview

As part of this course, you are tasked with designing, ordering, and assembling your own Printed Circuit Board (PCB) to perform a measurement task. This will be your first foray into the world of instrumentation design and analysis so it is important you consider your project choice carefully. You only have a couple weeks to accomplish the goal, so consider your workload, capabilities, and experience when scoping out your ICP idea.

It is generally recommended that you consider taking one of the four previous projects and making that into your ICP. For the most part, you already have the schematics and code finished, it is just up to the PCB design and assembly and what you want to do for that. You can also make your PCB into an Arduino Uno-compatible shield, for a really slick looking product!

For those that are interested in more of a challenge, you may examine other projects you could do within your Arduino starter kit, or refer to ICP topics addendum. There are several projects that will challenge you, but will be sponsored by the UTL outright and will probably be used within the lab for other projects or as a demonstration or teaching tool.

For completion of this project you must submit the following assignments on time, on the date specified in the syllabus and on Canvas, and as specified.

## Assignment 1: Schematic

Your first assignment will be to design a schematic for your ICP using industry standard practices, symbols, and nomenclature. The lecture notes contain some examples and you are encouraged to look at companies like Adafruit for inspiration on layout, formatting, etc. The instructor will also be available during office hours to offer advice and look over your schematic before you submit it. It is highly encouraged you take advantage of the instructor's availability to get the best schematic possible.

Also keep in mind what components you have on hand. Do not specify a component

### Requirements

Your schematic must have the following:

- A clear frame sized for *either* US Letter/ANSI A (8.5in x 11in) or US Tabloid/ANSI B (11in x 17in)
- Your name, project name, revision, and date created, as well other pertinent information within the title box of the frame.
- A well organized and logical layout for the parts used
- Clear and consistent nomenclature for the part values, net/bus names, and section names
- Effort towards presentability and readability (i.e. it should be easy to read!)

### Submission

You will submit this assignment on Canvas before 23:59 on the date that it is due in a PDF format as well as the source schematic in an EAGLE-compatible format. Please consult the late policy found in the syllabus for additional details.

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy/Completeness | 40 |
| Neatness and organization | 40 |
| Feasibility | 10 |
| Component usage | 10 |

## Assignment 2: Printed Circuit Board Design

Your second assignment will be to take your schematic and turn it into a PCB for manufacturing. This will involve component placement, layout, and wire routing, all of which can be intimidating so please make sure you leave yourself ample time to do the assignment right! As always, consult the instructor during their office hours for assistance. They are there to help you!

Here, it is important to try and keep things as neat as possible and make your routes direct and to the point to minimize potential issues. This will be especially important when you are laying out components before routing them; don't place two components that need to be connected on opposite ends of the boards, unless necessary!

Before submission, run a Design Rules Check to ensure all of the connections have been made, there are not any overlaps or restricted-zone violations, or anything else that may jeopardize the manufacture process. If you have any questions or concerns, please consult the instructor!

### Requirements

Your PCB design must have the following:

- A clear board that is of appropriate size (< 6in x 6in)
- Your name, revision, date of design (month/year), and project name (optional) in a silkscreen layer somewhere easily visible and unobstructed on the board.
- A well organized and logical layout for the parts used
- A well organized and thorough Bill of Materials for the parts used.
    - Shall be in an Excel workbook format with the following headers: Item, Manufacturer, Part Number, Supplier, Supplier Part Number, Reference, Value, Package, Price, Qty Used, Component Price, Location, Datasheet, Notes
- Gerber files for manufacture
- Effort towards aesthetics and user interactions

### Submission

You will submit this assignment on Canvas before class on the date that it is due in a zipped archive containing the following:

- PDF containing the overall PCB design file, the top layers, and the bottom layers, all scaled to a readable amount
- Source file in an EAGLE-compatible format
- A complete Bill of Materials in Excel workbook format with the headers specified in the requirements
- Gerber files in a zipped archive as generated by your ECAD software

Please note that if this submission is not made on time, you risk your PCB not being ordered with enough time for delivery and assembly for the final report/presentation!

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy/Completeness | 30 |
| Neatness and organization | 30 |
| Bill of Materials | 30 |
| Gerber files | 10 |
| Extra Credit | 25 |

### Extra Credit

There will be an extra credit opportunity with this assignment! If you want to, design a project logo or design that can be silk screened onto your PCB for some custom flair! Alternatively, design your PCb to be in a creative or unique shape. These points will be awarded subjectively by the instructor, but you must specify in your submission that you wish to have the extra credit points.

## Assignment 3: Assembled Printed Circuit Board

Your fourth assignment will be to assemble your PCB and test it! You may assemble it by any means you desire. For extra credit, you may program and use the pick and place machine located within the Underwater Technology Lab. Be advised though, that machine is still in a prototype phase and may be difficult to program and use. Please give yourself adequate time to learn it and use it. Effort to use will still result in extra credit though, depending on how much time you dedicate to it.

Even though it may be your first time soldering, please ensure that all the solder joints are neat looking and the board is clean. The PCB should be as presentable as possible!

If there are any issues, please consult the failure policy listed in the syllabus!

As always, please work with the instructor if you have any questions or concerns about assembly. If you feel unsafe working with the soldering machines, talk to the instructor and something can be arranged.

### Requirements

Your PCB assembly must have the following:

- All components cleanly attached to the PCB
- Working PCB

### Submission

You will bring your completed PCB to class on the day the assignment is due and perform a quick ( 3 minute) show-and-tell for the instructor and your peers. Please have a suitable demonstration and talking points prepared!

Please note that if this submission is not made on time, it will not be good for your grade. Even a non-functional ICP is better than nothing!

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 80 |
| Neatness and organization | 10 |
| Demonstration | 10 |

## Assignment 4: Assembled Printed Circuit Board

Your fourth assignment will be to assemble your PCB and test it! You may assemble it by any means you desire. For extra credit, you may program and use the pick and place machine located within the Underwater Technology Lab. Be advised though, that machine is still in a prototype phase and may be difficult to program and use. Please give yourself adequate time to learn it and use it. Effort to use will still result in extra credit though, depending on how much time you dedicate to it.

*Note:* You should try to keep a comprehensive assembly and testing log for future reference!

Even though it may be your first time soldering, please ensure that all the solder joints are neat looking and the board is clean. The PCB should be as presentable as possible!

If there are any issues, please consult the failure policy listed in the syllabus!

As always, please work with the instructor if you have any questions or concerns about assembly. If you feel unsafe working with the soldering machines, talk to the instructor and something can be arranged.

### Requirements

Your PCB assembly must have the following:

- All components cleanly attached to the PCB
- Working PCB

### Submission

You will bring your completed PCB to class on the day the assignment is due and perform a quick ( 3 minute) show-and-tell for the instructor and your peers. Please have a suitable demonstration and talking points prepared!

Please note that if this submission is not made on time, it will not be good for your grade. Even a non-functional ICP is better than nothing!

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 80 |
| Neatness and organization | 10 |
| Demonstration | 10 |

## Assignment 5: Final Presentation

Your fifth assignment will be to give a comprehensive presentation on your ICP. This presentation will be neatly formatted and cover the following topics, at a minimum:

- Purpose
- Design Considerations and Philosophy
- Assembly Notes (include build process)
- Testing Notes
- Conclusions

Remember to include citations and plenty of pictures, as needed!

### Submission

This will be submitted on Canvas in a PDF and/or PowerPoint-compatible form by the time of your ICP presentation. Please consult the late policy in the syllabus for more details.

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Purpose | 5 |
| Design Process | 30 |
| Testing Process | 30 |
| Conclusion | 15 |
| Presentation Style and Delivery | 10 |
| Formatting | 10 |

## Assignment 6: Final Report

Your sixth and final assignment will be to write a comprehensive report on your ICP. This report will be neatly formatted and cover the following topics, at a minimum:

- Executive Summary
- Introduction and Purpose
- Design Considerations and Philosophy
- Assembly Notes (include build process)
- Testing Notes
- Conclusions

Remember to include citations and plenty of pictures, as needed!

### Submission

This will be submitted on Canvas in a PDF form by 23:59 on the date specified in the syllabus. Please consult the late policy in the syllabus for more details.

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Executive Summary | 15 |
| Purpose | 5 |
| Design Process | 30 |
| Testing Process | 30 |
| Conclusion | 15 |
| Formatting | 5 |

# G

# Individual Course Project: Graduate

## Overview

As part of this course, you are tasked with designing, ordering, and assembling your own Printed Circuit Board (PCB) to perform a measurement task. This will be your first foray into the world of instrumentation design and analysis so it is important you consider your project choice carefully. You only have a couple weeks to accomplish the goal, so consider your workload, capabilities, and experience when scoping out your ICP idea.

It is generally recommended that you consider taking one of the four previous projects and making that into your ICP. For the most part, you already have the schematics and code finished, it is just up to the PCB design and assembly and what you want to do for that. You can also make your PCB into an Arduino Uno-compatible shield, for a really slick looking product!

For those that are interested in more of a challenge, you may examine other projects you could do within your Arduino starter kit, or refer to ICP topics addendum. There are several projects that will challenge you, but will be sponsored by the UTL outright and will probably be used within the lab for other projects or as a demonstration or teaching tool.

For completion of this project you must submit the following assignments on time, on the date specified in the syllabus and on Canvas, and as specified.

## Assignment 1: ICP Proposal

### Requirements

### Submission

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy/Completeness | 40 |
| Neatness and organization | 40 |
| Feasibility | 10 |
| Component usage | 10 |

## Assignment 2: Schematic

Your second assignment will be to design a schematic for your ICP using industry standard practices, symbols, and nomenclature. The lecture notes contain some examples and you are encouraged to look at companies like Adafruit for inspiration on layout, formatting, etc. The instructor will also be available during office hours to offer advice and look over your schematic before you submit it. It is highly encouraged you take advantage of the instructor's availability to get the best schematic possible.

Also keep in mind what components you have on hand. Do not specify a component

### Requirements

Your schematic must have the following:

- A clear frame sized for *either* US Letter/ANSI A (8.5in x 11in) or US Tabloid/ANSI B (11in x 17in)
- Your name, project name, revision, and date created, as well other pertinent information within the title box of the frame.
- A well organized and logical layout for the parts used
- Clear and consistent nomenclature for the part values, net/bus names, and section names
- Effort towards presentability and readability (i.e. it should be easy to read!)

### Submission

You will submit this assignment on Canvas before 23:59 on the date that it is due in a PDF format as well as the source schematic in an EAGLE-compatible format. Please consult the late policy found in the syllabus for additional details.

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy/Completeness | 40 |
| Neatness and organization | 40 |
| Feasibility | 10 |
| Component usage | 10 |

## Assignment 3: Printed Circuit Board Design

Your third assignment will be to take your schematic and turn it into a PCB for manufacturing. This will involve component placement, layout, and wire routing, all of which can be intimidating so please make sure you leave yourself ample time to do the assignment right! As always, consult the instructor during their office hours for assistance. They are there to help you!

Here, it is important to try and keep things as neat as possible and make your routes direct and to the point to minimize potential issues. This will be especially important when you are laying out components before routing them; don't place two components that need to be connected on opposite ends of the boards, unless necessary!

Before submission, run a Design Rules Check to ensure all of the connections have been made, there are not any overlaps or restricted-zone violations, or anything else that may jeopardize the manufacture process. If you have any questions or concerns, please consult the instructor!

## Requirements

Your PCB design must have the following:

- A clear board that is of appropriate size (< 6in x 6in)
- Your name, revision, date of design (month/year), and project name (optional) in a silkscreen layer somewhere easily visible and unobstructed on the board.
- A well organized and logical layout for the parts used
- A well organized and thorough Bill of Materials for the parts used.
    - Shall be in an Excel workbook format with the following headers: Item, Manufacturer, Part Number, Supplier, Supplier Part Number, Reference, Value, Package, Price, Qty Used, Component Price, Location, Datasheet, Notes
- Gerber files for manufacture
- Effort towards aesthetics and user interactions

## Submission

You will submit this assignment on Canvas before class on the date that it is due in a zipped archive containing the following:

- PDF containing the overall PCB design file, the top layers, and the bottom layers, all scaled to a readable amount
- Source file in an EAGLE-compatible format
- A complete Bill of Materials in Excel workbook format with the headers specified in the requirements
- Gerber files in a zipped archive as generated by your ECAD software

Please note that if this submission is not made on time, you risk your PCB not being ordered with enough time for delivery and assembly for the final report/presentation!

## Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy/Completeness | 30 |
| Neatness and organization | 30 |
| Bill of Materials | 30 |
| Gerber files | 10 |
| Extra Credit | 25 |

### Extra Credit

There will be an extra credit opportunity with this assignment! If you want to, design a project logo or design that can be silk screened onto your PCB for some custom flair! Alternatively, design your PCb to be in a creative or unique shape. These points will be awarded subjectively by the instructor, but you must specify in your submission that you wish to have the extra credit points.

## Assignment 4: Assembled Printed Circuit Board

Your fourth assignment will be to assemble your PCB and test it! You may assemble it by any means you desire. For extra credit, you may program and use the pick and place machine located within the Underwater Technology Lab. Be advised though, that machine is still in a prototype phase and may be difficult to program and use. Please give yourself adequate time to learn it and use it. Effort to use will still result in extra credit though, depending on how much time you dedicate to it.

Even though it may be your first time soldering, please ensure that all the solder joints are neat looking and the board is clean. The PCB should be as presentable as possible!

If there are any issues, please consult the failure policy listed in the syllabus!

As always, please work with the instructor if you have any questions or concerns about assembly. If you feel unsafe working with the soldering machines, talk to the instructor and something can be arranged.

### Requirements

Your PCB assembly must have the following:

- All components cleanly attached to the PCB
- Working PCB

### Submission

You will bring your completed PCB to class on the day the assignment is due and perform a quick ( 3 minute) show-and-tell for the instructor and your peers. Please have a suitable demonstration and talking points prepared!

Please note that if this submission is not made on time, it will not be good for your grade. Even a non-functional ICP is better than nothing!

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 80 |
| Neatness and organization | 10 |
| Demonstration | 10 |

# Assignment 5: Assembled Printed Circuit Board

Your fifth assignment will be to assemble your PCB and test it! You may assemble it by any means you desire. For extra credit, you may program and use the pick and place machine located within the Underwater Technology Lab. Be advised though, that machine is still in a prototype phase and may be difficult to program and use. Please give yourself adequate time to learn it and use it. Effort to use will still result in extra credit though, depending on how much time you dedicate to it.

*Note:* You should try to keep a comprehensive assembly and testing log for future reference!

Even though it may be your first time soldering, please ensure that all the solder joints are neat looking and the board is clean. The PCB should be as presentable as possible!

If there are any issues, please consult the failure policy listed in the syllabus!

As always, please work with the instructor if you have any questions or concerns about assembly. If you feel unsafe working with the soldering machines, talk to the instructor and something can be arranged.

## Requirements

Your PCB assembly must have the following:

- All components cleanly attached to the PCB
- Working PCB

## Submission

You will bring your completed PCB to class on the day the assignment is due and perform a quick ( 3 minute) show-and-tell for the instructor and your peers. Please have a suitable demonstration and talking points prepared!

Please note that if this submission is not made on time, it will not be good for your grade. Even a non-functional ICP is better than nothing!

## Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Efficacy | 80 |
| Neatness and organization | 10 |
| Demonstration | 10 |

## Assignment 6: Final Presentation

Your sixth assignment will be to give a comprehensive presentation on your ICP. This presentation will be neatly formatted and cover the following topics, at a minimum:

- Purpose
- Design Considerations and Philosophy
- Assembly Notes (include build process)
- Testing Notes
- Conclusions

Remember to include citations and plenty of pictures, as needed!

### Submission

This will be submitted on Canvas in a PDF and/or PowerPoint-compatible form by the time of your ICP presentation. Please consult the late policy in the syllabus for more details.

### Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Purpose | 5 |
| Design Process | 30 |
| Testing Process | 30 |
| Conclusion | 15 |
| Presentation Style and Delivery | 10 |
| Formatting | 10 |

## Assignment 7: Final Report

Your seventh and final assignment will be to write a comprehensive report on your ICP. This report will be neatly formatted and cover the following topics, at a minimum:

- Executive Summary
- Introduction and Purpose
- Design Considerations and Philosophy
- Assembly Notes (include build process)
- Testing Notes
- Conclusions

Remember to include citations and plenty of pictures, as needed!

## Submission

This will be submitted on Canvas in a PDF form by 23:59 on the date specified in the syllabus. Please consult the late policy in the syllabus for more details.

## Grading

You will be graded on the following criteria:

| Criterion | Points |
|---|---|
| Executive Summary | 15 |
| Purpose | 5 |
| Design Process | 30 |
| Testing Process | 30 |
| Conclusion | 15 |
| Formatting | 5 |

# H

# In-Circuit Serial Programming Guide

Microcontrollers are special embedded processors that are capable of executing specific code with a high efficiency. They are used throughout the world in various devices ranging from handheld gaming devices, medical units, military hardware, and digital signage. Recently, the Arduino foundation made playing with microcontrollers easier as they created a platform where users could write code, plug in a microcontroller over USB, flash the chip, and see the results - nearly in real-time. They helped usher in the Maker Renaissance we found ourselves in today, by obfuscating the more complex tasks in microcontroller programming using high-level software and other microcontrollers.

The purpose of this guide is to help de-obfuscate low-level microcontroller programming by showing you what is really happening when you give the "Upload" command in the Arduino IDE.

## What is In-Circuit Serial Programming?

In-Circuit Serial Programming (ICSP) is the fundamental way to program a microcontroller. Microcontrollers typically store their programs on SPI flash memory which can be read or written over as the programmer demands. By temporarily disabling the microcontroller and overwriting the contents of the SPI flash memory, we can reprogram the microcontroller with different code.

On the Arduino boards, when you upload code over USB, a secondary microcontroller on the board interprets the USB communications and translates them to the UART serial protocol. The bootloader on the main microcontroller then uses the data coming from the UART bus to reprogram the SPI flash memory, thus reprogramming the Arduino. This is a form of ICSP, but adds complexity to both the circuit and fundamental microcontroller firmware.

A simpler version of ICSP is accessing the SPI flash memory directly. On the Arduino Uno board, the ICSP header is easily visible and can be used by any device that has an SPI bus to reprogram the microcontroller. If, for instance, the USB-Serial bridge on the board has its program memory corrupted ICSP can be used to re-flash the correct firmware onto the chip. Alternatively, if the chip is completely non-functional, the primary microcontroller can still be used with new code being deployed over the ICSP pins.

Note that different microcontrollers may have different implementations of ICSP. Another popular form is the Single Wire interface. Please refer to your microcontroller's datasheet for specific information.

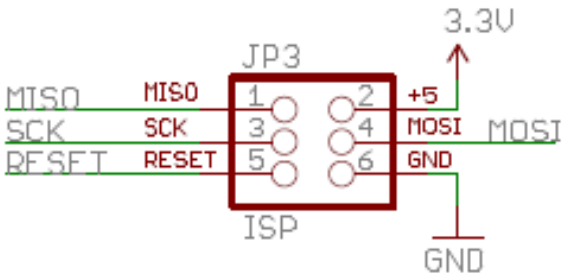



**Figure H.1:** Pinout of the Arduino Uno and Mega ICSP header

## Arduino as In-circuit Serial Programmer

For this example, we will be using an Arduino as an In-circuit Serial Programmer (ISP) to program another Arduino over ICSP. The Arduino Foundation provides a sketch in the examples folder of the Arduino IDE to configure an Arduino board as an ISP. First, plug in the Arduino Uno to the ICSP header as shown in Figure H.2 and according to Table H.1. On most boards, there is no reverse-polarity protection, so make sure the 5V and GND wires are plugged in correctly.

**Table H.1:** ArduinoISP hookup table for target and programmer

| Programmer | Target |
|---|---|
| DIO 12 | MISO (Pin 1) |
| DIO 13 | SCK (Pin 2) |
| DIO 10 | RESET (Pin 3) |
| 5V | 5V (Pin 4) |
| DIO 11 | MOSI (Pin 5) |
| GND | GND (Pin 6) |

After wiring up the boards, plug the Arduino Uno into the programming computer. This will apply power to the system and should allow you to program the Uno as you normally would. Then, navigate the ArduinoISP sketch located in the “examples” folder of the Arduino IDE (Figure H.3) and upload the sketch to the Arduino Uno.

Once the ArduinoISP sketch is uploaded, navigate to a sketch to upload to the target board configuring the compiler to the target board and processor (Figure H.4). Use the same COM port as the Arduino Uno and use the “Upload Using Programmer” option shown in Figure H.5. If you get any errors such as “Unable to communicate with device” or “Invalid device signature”, check your wiring and try again.

While this example used the basic Blink example code, this process can be done for any appropriate Arduino sketch onto most Arduino-compatible boards. On some embedded devices, a common USB interface may not be accessible and therefore ICSP is the only programming option.

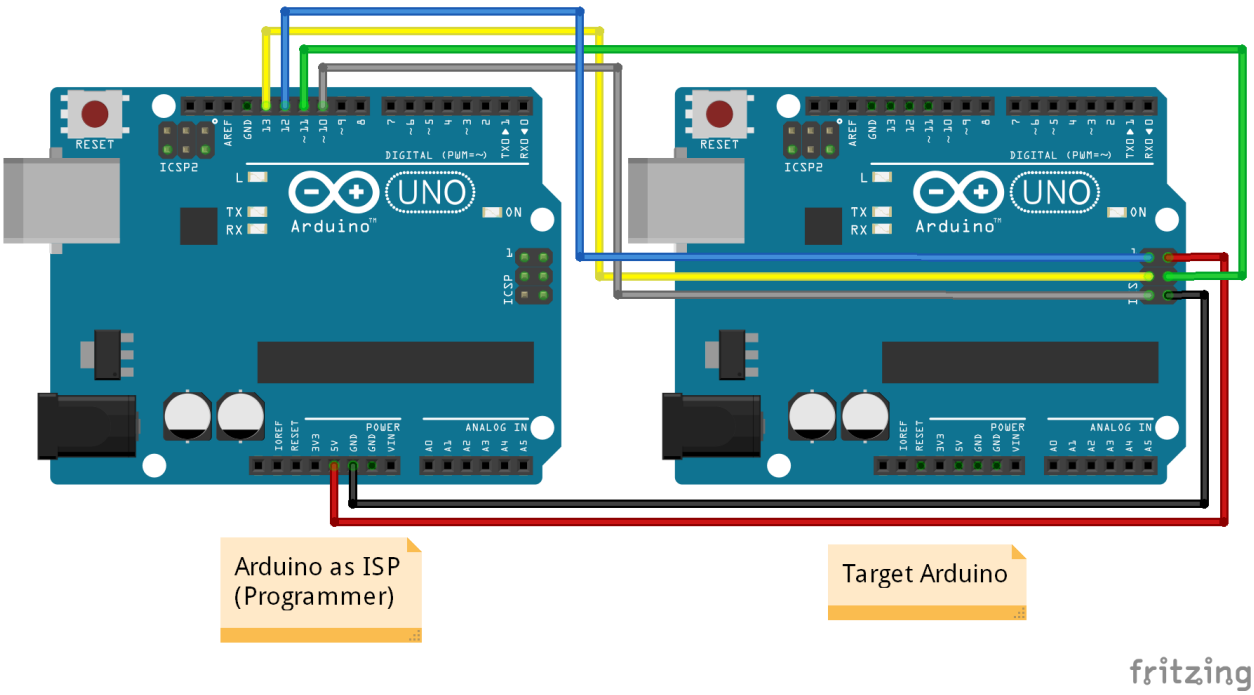



**Figure H.2:** Diagram showing how to hook up an ArduinoISP programmer and Arduino ICSP target.Retrieved from https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader/hardware-hookup

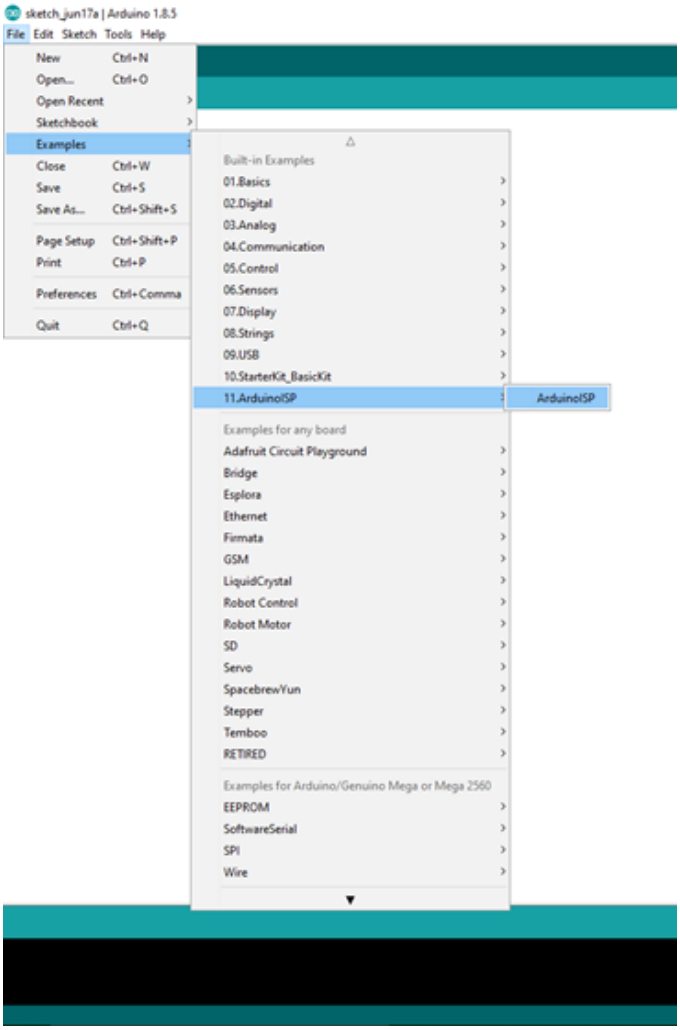



**Figure H.3:** Location of ArduinoISP sketch in the Arduino IDE

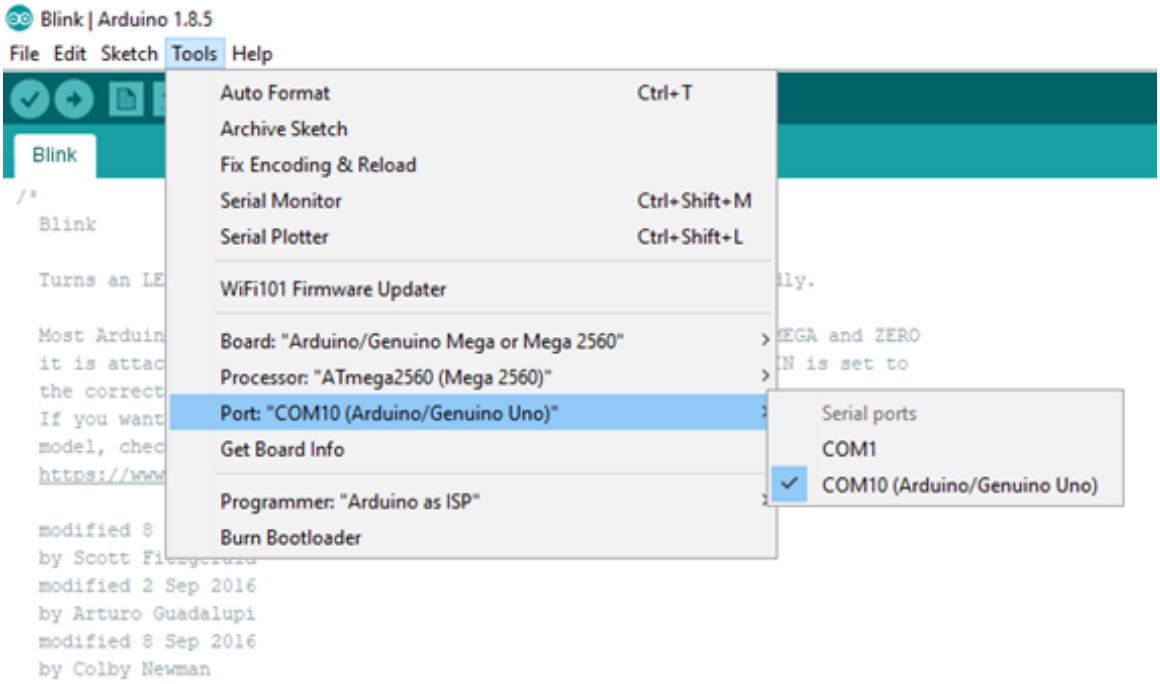



**Figure H.4:** Configuring the Arduino IDE to compile for the correct board

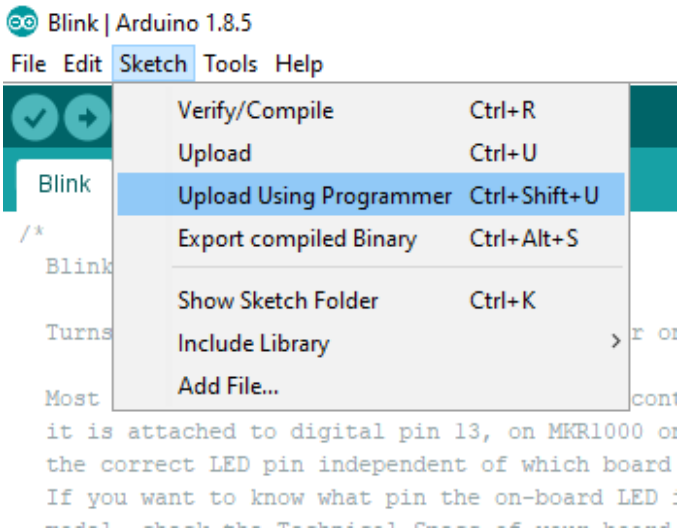



**Figure H.5:** Upload Using Programmer sketch option

## Using a Raspberry Pi as ISP

For the graduate portion of this class, we will be using a Raspberry Pi to upload code to the Arduino boards over ICSP. The Raspberry Pi uses 3.3V logic levels which are not completely compatible with the 5V logic levels of the Arduino board. Therefore, it is strongly recommended to use a breadboard to breakout the Raspberry Pi's pins to a logic level converter and connect it from there to the Arduino's ICSP header. Please refer to Figure H.6 for hookup information. The pinout for this configuration can be found in the figure, which we will use later with AVRDUDE

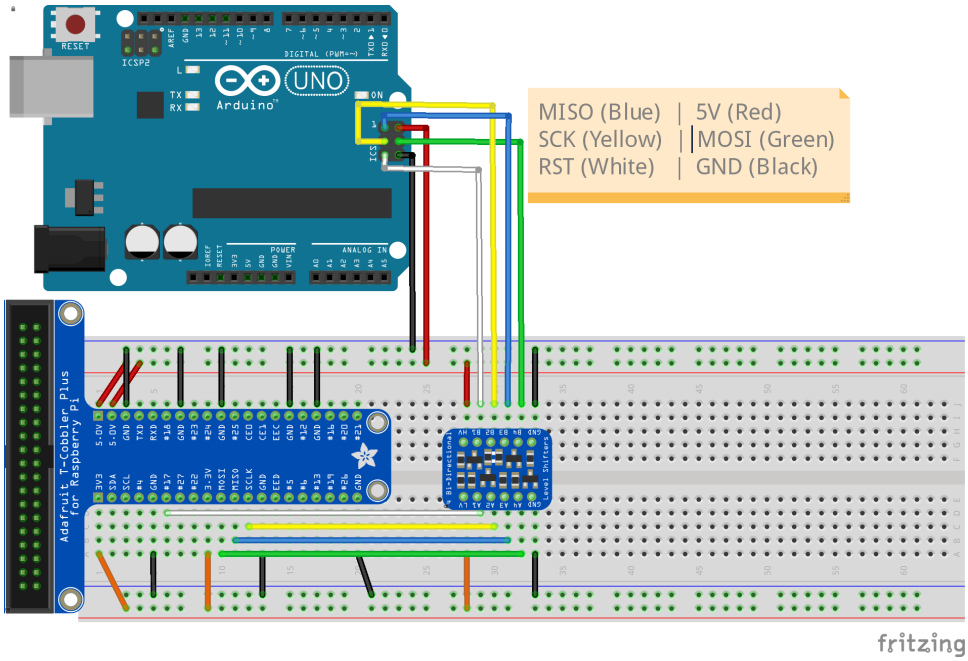



**Figure H.6:** A wiring diagram of how the Raspberry Pi should be connected to the Arduino ICSP headers through a breadboard and a logic-level converter. Created using Fritzing https://fritzing.org

**Table H.2:** Raspberry Pi as ISP hookup table for target and programmer

| Programmer | Logic Level Converter | Target |
|---|---|---|
| 3V3 | LV | NC* |
| NC* | HV | 5V |
| GPIO 17 | A1/B1 | RESET |
| GPIO 11 (SCLK) | A2/B2 | SCK |
| GPIO 9 (MISO) | A3/B3 | MISO |
| GPIO 10 (MOSI) | A4/B4 | MOSI |
| GND | GND | GND |

### Setting up AVRDUDE

AVRDUDE is the software package that will upload machine binary to the microcontroller's program storage. The following steps will assume you have already set up the Raspberry Pi with RaspberryPiOS and updated

* Not connected

the latest packages. Through this example, we will be operating inside a command terminal through a remote secure login shell, but this process can be repeated inside the command terminal of a headed set up.

First, in the open terminal execute the following command:

```
sudo apt-get install avrdude
```

This will install AVRDUDE on the Raspberry Pi so we can flash .hex files onto the microcontroller. To verify the installation, execute:

```
avrdude -v
```

If the output is something like:

```
avrdude: Version 6.3-20171130
    Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
    Copyright (c) 2007-2014 Joerg Wunsch

    System wide configuration file is "/etc/avrdude.conf"
    User configuration file is "/home/pi/.avrduderc"
    User configuration file does not exist or is not a regular file, skipping

avrdude: no programmer has been specified on the command line or the config file
    Specify a programmer using the -c option and try again
```

Then the installation is good and AVRDUDE has created a user configuration file that we can edit. To begin this process, execute the following commands in the terminal:

```
cp /etc/avrdude.conf ~/avrdude_gpio.conf

nano ~/avrdude_gpio.conf
```

The first command copies the default AVRDUDE configuration file to a new file in the home directory of user. The second command will open this file in the Nano file editor, pulling up a screen like Figure H.7.

Once in the file, press CTRL_, then CTRLV to navigate to the bottom of the file. There, paste the following block of code:

```
programmer
    id = "gpio_icsp";
    desc = "Use the Linux sysfs interface to bitbang GPIO lines for programming the Arduino;
    type = "linuxgpio";
    reset = 17;
    sck = 11;
    mosi = 10;
    miso = 9;
;
```

This creates an AVRDUDE programmer that uses the GPIO pins specified in Table H.2 to program the Arduino over ICSP. Press CTRL+X then Y then ENTER to save and exit the file; the AVRDUDE programming tool is now configured.

## Preparing a Sketch for ICSP Uploading (Arduino)

Once you have a sketch ready for the microcontroller to run, configure the compiler for your board, same as Figure H.4. Select the “Export compiled Binary” option in the Arduino IDE (Figure H.8). This will create two

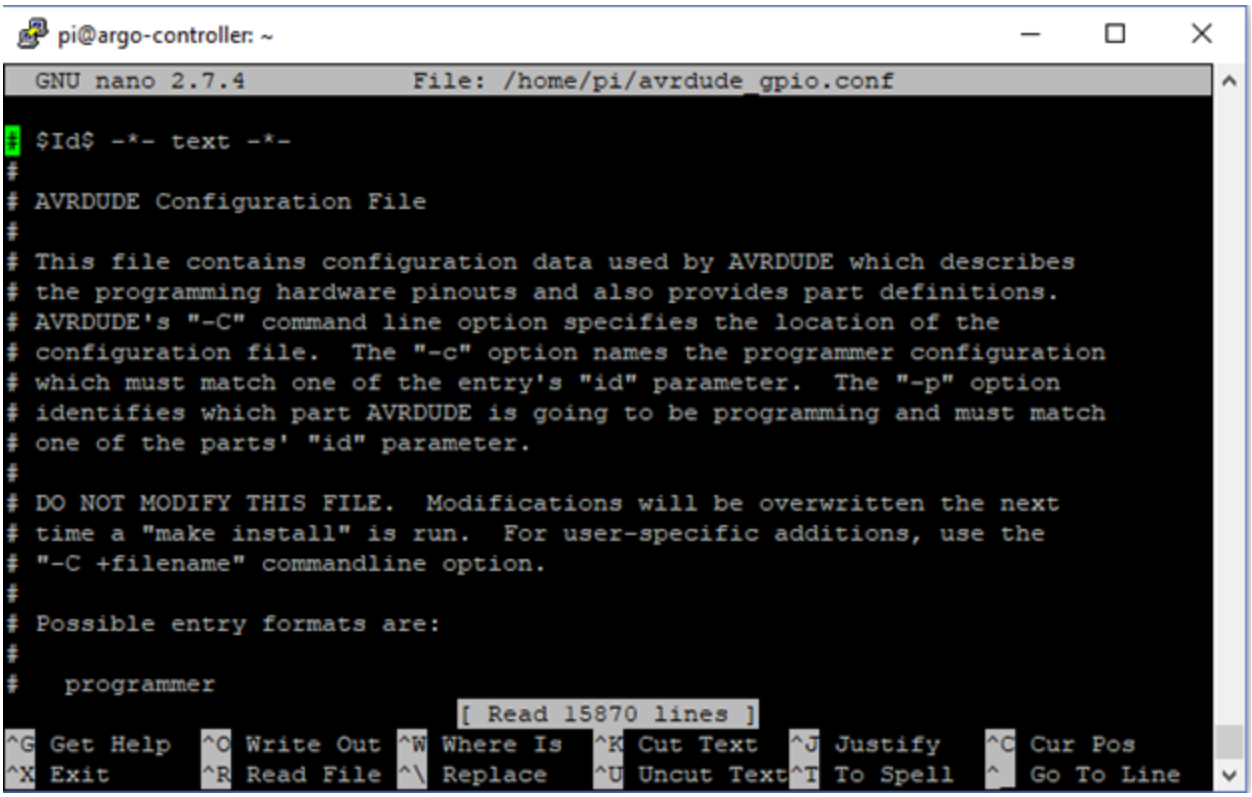



**Figure H.7:** The top of the avrdude_gpio.conf file in Nano

files in the sketch's directory, both ending with ".hex", but one will have ".with_bootloader" in the middle. The file we want to upload is the one without the bootloader, highlighted in Figure H.9. †

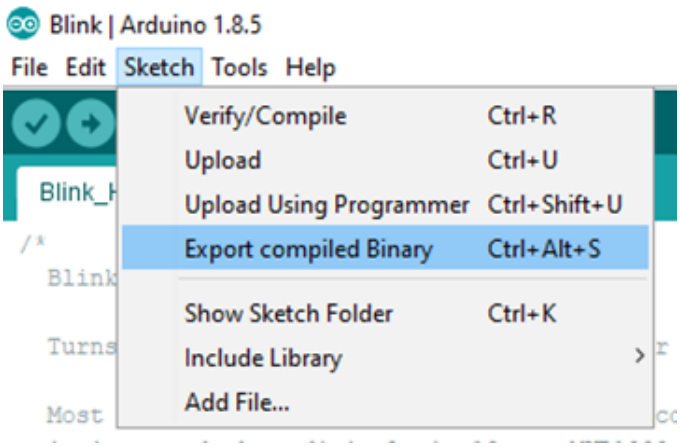



**Figure H.8:** Arduino IDE "Export compiled Binary" option

† This will disable the USB programming bootloader. This will need to be re-flashed with the bootloader if you want to upload code to the microcontroller via USB again.

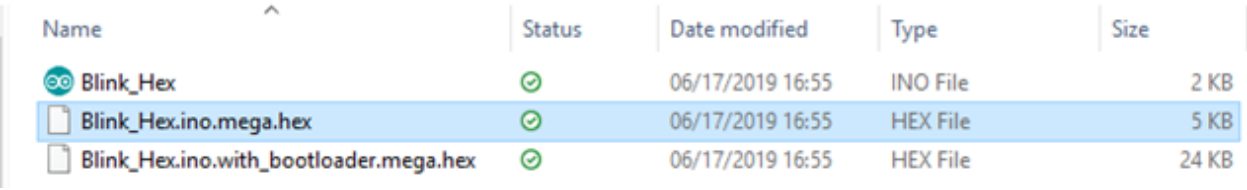



**Figure H.9:** The sketch and its compiled binaries in their folder

## Preparing a Sketch for ICSP Uploading (VS Code)

This part assumes you have already initialized the Arduino development environment within VS Code. In VS Code, open the "arduino.json" file in the /.vscode folder of your workspace. Add the following line to the bottom of the file:

```
"output": ".arduinobuild"
```

This will create a new folder in the root directory of the workspace called ".arduinobuild" that will hold all of pre-compiled binaries for your Arduino sketch, logs, and other things that are not important right now. Inside this folder will be two files: "[sketch_name].ino.bin" and "[sketch_name].bootloader.bin". As before, the one we want to upload is the one without the bootloader.*

## Programming the Microcontroller

Transfer the binary file to the Raspberry Pi using your preferred method of choice. This is most easily done over a USB stick or a File Transfer Protocol like Secure Copy. Applications like WinSCP make this process easy for beginners. Place the file in a working destination directory - ideally, a project folder you have already set up beforehand. Then, open a terminal on the Raspberry Pi and execute the following command:

```
sudo avrdude -p [Microcontroller] -C ~/avrdude_gpio.conf -c gpio_icsp -v
```

Note: the **[Microcontroller]** in this command needs to be replaced with the name of the microcontroller in use (e.g. "atmega328p" for the Arduino Uno or "atmega2560" for the Arduino Mega).

This will verify that the Raspberry Pi can talk to the microcontroller and verifies that the chip is operating nominally.

If there are any issues, make sure you are executing this command with "sudo" and that the configuration file matches with the GPIO pins used in the schematic; check the wiring connections to the Arduino; and check that the file path for "avrdude_gpio.conf" is correct .‡

Once you have established communications with the Arduino, execute:

```
sudo avrdude -p [Microcontroller] -C ~/avrdude_gpio.conf -c gpio_icsp -v -U flash:w:[filename]:i
```

Where again **[Microcontroller]** needs to be replaced with the name of the microcontroller and **[filename]** needs to be replaced with the path and name of the binary sketch file you copied to the Raspberry Pi. The end of a successful write should look something like this:

‡ The "~" in the file path only denotes the relative directory you are currently in. If the file is NOT in the same directory that you are executing the command in, you must put the path in lieu of the "~".

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e9801 (probably m2560)
avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D8
avrdude: safemode: efuse reads as FD
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "Blink_Hex.ino.mega.hex"
avrdude: writing flash (1462 bytes):

Writing | ################################################## | 100% 0.41s

avrdude: 1462 bytes of flash written
avrdude: verifying flash memory against Blink_Hex.ino.mega.hex:
avrdude: load data flash data from input file Blink_Hex.ino.mega.hex:
avrdude: input file Blink_Hex.ino.mega.hex contains 1462 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.82s

avrdude: verifying ...
avrdude: 1462 bytes of flash verified

avrdude: safemode: lfuse reads as FF
avrdude: safemode: hfuse reads as D8
avrdude: safemode: efuse reads as FD
avrdude: safemode: Fuses OK (E:FD, H:D8, L:FF)

avrdude done.  Thank you.
```

If there is an error (most commonly with a file name) it will likely occur after the first reading block. The -v argument of the command gives error statements in the output so you can error trace and find the problem.