Fundamentals of Applied Microcontrollers Manual

Seth McNeill

Edition 0.01
2022 February 02

Contents

Chapter 1

Introduction

This book is the accompanying text to a class introducing microcontrollers to upper division, non-electrical engineering undergraduate students who have taken some C programming.

Chapter 2

Number Systems

This chapter introduces the concepts of different numbering systems that are relevant to microcontrollers.

2.1 Decimal Numbers

The standard number system used by humans is based on 10. This logically flows the usual numbers of fingers or toes humans have. When counting in the base 10 (decimal) we start at 0, count up to 9, then run out of numbers to use. When runs out of numbers, we put a number to the left of the column we were counting in, and then start over at zero again which gives us the number 10. A more clear way to count would be to count from 00 to 09, then increment the left digit and start the right digit back at 0. This can be continued until 99 is reached. But now we have a model to follow. If a column gets to 9, we increment a column to the left and start the current column over at 0. This leads to the number 100. This concept can be continued forever.

When given a particular number, 3254, in decimal the value is calculated as

$$3254 = 3 \cdot 1000 + 2 \cdot 100 + 5 \cdot 10 + 4 \tag{2.1}$$

It could also be represented as

$$3254 = 3 \cdot 10^3 + 2 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0 \tag{2.2}$$

This is the basis for a positional number system.

2.2 Binary Numbers

Computers run a base 2 system, also known as binary. This means that we only have two options at each position, 1 or 0, instead of the 10 available in the decimal system. However, counting is done in the same way, when you run out of symbols, increment the position to the left and then start over at zero in the current position.

Binary numbers are also positional so a number like 1011 is

$$1011 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11_{10} \tag{2.3}$$

where 11_{10} indicates that the number 11 is base 10.

The maximum value a binary number can have is determined by the number of bits it has. The formula is:

$$V_{max} = 2^n - 1 \tag{2.4}$$

where n is the number of bits. For example, an 8 bit number has a maximum value of $2^8 - 1 = 255$.

2.3 Hexadecimal Numbers

The problem for humans is that we have a very difficult time reading binary numbers. Especially once the numbers get long such as 11101101010010110101_2. The first thing we can do to improve intelligibility is to put a space in every 4 digits so that we aren't completely bowled over by all the digits. This gives us $1110\ 1101\ 0100\ 1011\ 0101_2$. This is better but still leaves some challenges. Someone pointed out that since 4 bits can have 16 different values, what if each 4 bits (also called a nibble) was represented by a single base 16 number. Base 16 is called hexadecimal. It has the problem that it runs out of digits once we get to 10, so it was decided to simply start on the alphabet so 10_{10} is A_{16}, 11_{10} is B_{16}, and so forth. Now the cumbersome binary number we've been playing with can be represented as $ED4B5_{16}$.

A comparison of the different numbering system representations of 0 through 15_{10} are shown in Table 2.1.

Decimal	Binary	Hexadecimal
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table 2.1: Decimal, binary, and hexadecimal numbers align as shown in this table.

2.4 Binary Background

The reasons for this stem from early computers using switches (relays) as their basic computing elements and the fact that telling the difference between a "high" voltage and a "low" voltage is easier to do than to differentiate 10 different voltages. It also allows for a gap between high and low that creates some immunity to noise. (TODO: insert figure or reference figure from logic chapter)

2.5 Converting Between Bases

2.5.1 Binary to Decimal

Converting to decimal is straightforward since we can simply sum each digit multiplied by the power of 2 it represents.

$$V_{10} = \sum_{i=0}^{p} d_i \cdot 2^i \tag{2.5}$$

In Equation 2.5 the p digits, d_i, of the binary number are summed from right to left while being multiplied by the power of 2 they represent. As an example, convert 1011 0111 to decimal.

$$\begin{aligned} V_{10} &= 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 + 1 \cdot 2^5 + 0 \cdot 2^6 + 1 \cdot 2^7 \\ &= 1 + 2 + 4 + 0 + 16 + 32 + 0 + 128 \\ &= 183 \end{aligned} \tag{2.6}$$

I usually find it easiest to just remember the powers of two for each place and add them up.

$$\overset{128}{1}\quad\overset{64}{0}\quad\overset{32}{1}\quad\overset{16}{1}\quad\overset{8}{0}\quad\overset{4}{1}\quad\overset{2}{1}\quad\overset{1}{1} = 1 + 2 + 4 + 16 + 32 + 128 \tag{2.7}$$

2.5.2 Decimal to Binary

Converting decimal numbers to binary involves dividing by 2 until the remainder is 0 or 1. The process goes as follows:

1. Divide the number by 2. If the remainder is 1 then the least significant bit is 1 otherwise if the remainder is 0 (it was an even number) then the least significant bit is 0.

2. Take the quotient from the previous step and divide by 2. The remainder is the next more significant bit in the binary representation.

3. Repeat Step 2 until the quotient is 0.

As an example, let's convert 11_{10} to binary.

$$\begin{aligned} 11 \div 2 &= 5R1 && \rightarrow 1 \text{ (LSB)} \\ 5 \div 2 &= 2R1 && \rightarrow 1 \\ 2 \div 2 &= 1R0 && \rightarrow 0 \\ 1 \div 2 &= 0R1 && \rightarrow 1 \text{ (MSB)} \end{aligned} \tag{2.8}$$

That means that 11_{10} is 1011_2. As another example let us convert 20_{10} to binary.

$$\begin{aligned}
20 \div 2 &= 10R0 && \rightarrow 0 \text{ (LSB)} \\
10 \div 2 &= 5R0 && \rightarrow 0 \\
5 \div 2 &= 2R1 && \rightarrow 1 \\
2 \div 2 &= 1R0 && \rightarrow 0 \\
1 \div 2 &= 0R1 && \rightarrow 1 \text{ (MSB)}
\end{aligned} \tag{2.9}$$

This shows that 20_{10} is 10100_2.

2.5.3 Binary and Hexadecimal

For conversions between binary and hexadecimal I tend to use the table lookup method. After using it enough times you begin to memorize the conversions. In my head I'm usually converting from binary to decimal on each nibble and then converting decimal into hexadecimal. So if I see 0101 I remember it is 5 in decimal which is the same in hex. If I see 1010 I remember that it is $8 + 2 = 10$ which is one more than 9 so it is A in hex.

2.6 Colors

Colors for display on computers are represented as binary numbers. Colors are 24 bit which is broken down into 3 8 bit numbers representing red, green, and blue (RGB). Eight bits give values over the range of 0 to 255 so colors are represented by 3 numbers such as (255, 0, 255) which gives a color like this. It is good to note that (255, 255, 255) is white, (0, 0, 0) is black, and (X, X, X) where all three numbers are the same is gray.

Sometimes colors are represented as a 6 digit binary number in hexadecimal form prefixed by 0x such as 0xFF00FF for the color we looked at previously.

2.7 ASCII

Since computers operate using binary numbers, how do we get them to represent human languages? The method caught it is called American Standard Code for Information Interchange (ASCII). Basically, 7 bit numbers were mapped to letters, numbers, punctuation, symbols, and control characters. Some examples are A is 65, Z is 90, a is 97, z is 122, 0 is 48, 9 is 57. The most used control characters are carriage return (13 or \r) and line feed (10 or \n). Unix based operating systems (Linux, OS X) use line feed to indicate a new line. Windows uses both (\r\n).

Look up an ASCII table online when you need to know the values.

2.8 Adding and Subtracting Binary Numbers

2.8.1 Default method

2.8.1.1 Addition

Calculate the binary sum 0001 0011 1101 + 0000 1011 0111.

$$\begin{array}{ccccccccccccc} & & & & & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ + & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{array} \tag{2.10}$$

2.8.1.2 Subtraction

Calculate the binary difference 0001 0011 1101 − 0000 1011 0111.

$$\begin{array}{ccccccccccccc} & 0 & 0 & 0 & \cancelto{0}{1} & \cancelto{01}{0} & 0 & 1 & 1 & \cancelto{0}{1} & \cancelto{\cancelto{01}{0}}{1} & \cancelto{01}{0} & 1 \\ - & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ \hline & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \tag{2.11}$$

2.8.2 Twos Complement method

2.9 Gray Codes

When counting in binary, often times more than one bit changes as the number increments. This works fine in the ideal world, but in the real world, the logic controlling each bit might be slightly different. This will cause

Decimal	Two's Complement
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

Table 2.2: This table shows the decimal and two's complement numbers for 4-bits.

one bit to change at a slightly different time than another bit. That causes glitches in the counting that can be disruptive to the overall system.

Take as an example a robot that represents the cardinal directions as binary integers as illustrated in Figure 2.1.

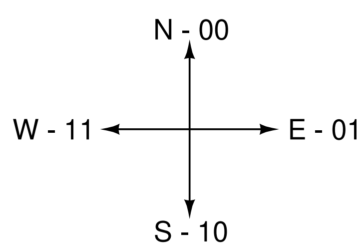

Figure 2.1: This figure shows using regular binary counting to represent the cardinal directions.

Using regular binary counting means that as the direction changes from East to South, two bits have to change. If the bits are driven by real switches or differing logic they may not change simultaneously leading to possible outputs of E - W - S or E - N - S. If the data is being used in a sequential manner it could lead to erroneous actions by the device (robot, car, etc.). Instead of using regular binary counting, we could use an encoding that only changes one bit at a time as the directions change as shown in Figure 2.2.

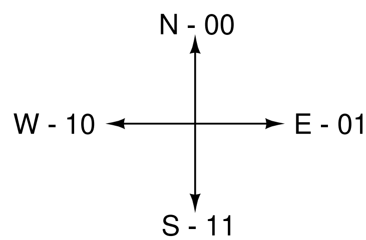

Figure 2.2: This figure shows using Gray codes to represent the cardinal directions.

Counting systems like this are called Gray code after Frank Gray or reflected binary code. A 4-bit example is shown in Table 2.3.

2.10 Binary Background

Originally, computers were just a bunch of switches, therefore, they only had two positions: on and off. This also has the benefit that there is a large amount of noise immunity. A repeater (buffer) can eliminate by recreating the original signal without noise. These are all great benefits of the binary system.

2.11 Assignment

There is a numbers quiz on Canvas as the assignment for this lab.

Gray Code
0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

Table 2.3: This table shows 4-bit Gray codes with horizontal lines showing the break for 2- and 3-bit Gray codes.

Chapter 3

Boolean Logic

3.1 Introduction

This chapter introduces some basic Boolean logic including gates and Boolean algebra.

When can you drive through an intersection? When the light is NOT red. This is the first and simplest logic operator–the NOT element. It simply changes any TRUE to FALSE or FALSE to TRUE (you can substitute 1 for TRUE and 0 for FALSE, or on/off).

How do you start a car? In most of the cars I have driven I have to press on the brake at the same time as I turn the key. To say it another way, the car starts when I press the break AND turn the key.

$$pressBreak \text{ AND } turnKey = startedCar \tag{3.1}$$

Again on a car, a particular blinker light will turn on if you turn on the turn signal or if you turn on the 4-way blinker.

$$turnSignal \text{ OR } 4wayBlinker = blinking \tag{3.2}$$

The AND and OR in the equation are Boolean operators.

3.2 Methods of Representing Logic

It is important to differentiate between different forms of representation because $\overline{\text{EN}}$ means **active low** enable. It does not mean NOT(E AND N). The understanding of which method is being used is usually derived from context. Using the dot $(\cdot)$ everywhere can become burdensome, so when context makes it obvious (usually examples involving A, B, and C or X and Y) we may drop the dots and just use adjacency to represent the AND function.

3.3 Boolean Algebra

3.3.1 Theorems of Boolean Algebra

Ways to show NOT:

$$NOT(X) = \overline{X} = X' \tag{3.3a}$$

$$\overline{(\overline{X})} = (X')' = X \tag{3.3b}$$

Rules of AND and OR:

$$\begin{array}{rll} \text{AND} & \text{OR} & \\ 0 \cdot 0 = 0 & 1 + 1 = 1 & (3.4a) \\ 1 \cdot 1 = 1 & 0 + 0 = 0 & (3.4b) \\ 0 \cdot 1 = 1 \cdot 0 = 0 & 1 + 0 = 0 + 1 = 1 & (3.4c) \end{array}$$

$$\begin{array}{rll} \text{AND} & \text{OR} & \\ X \cdot 1 = X & X + 0 = X & (3.5a) \\ X \cdot 0 = 0 & X + 1 = 1 & (3.5b) \\ X \cdot X = X & X + X = X & (3.5c) \\ X \cdot \overline{X} = 0 & X + \overline{X} = 1 & (3.5d) \end{array}$$

For two and three variables you have the following useful equations:

$$\begin{array}{rll} \text{AND} & \text{OR} & \\ X \cdot Y = Y \cdot X & X + Y = Y + X & (3.6a) \\ (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) & (X + Y) + Z = X + (Y + Z) & (3.6b) \\ (X + Y) \cdot (X + Z) = X + Y \cdot Z & X \cdot Y + X \cdot Z = X \cdot (Y + Z) & (3.6c) \\ X \cdot (X + Y) = X & X + X \cdot Y = X & (3.6d) \\ (X + Y) \cdot (X + \overline{Y}) = X & X \cdot Y + X \cdot \overline{Y} = X & (3.6e) \\ X \cdot (\overline{X} + Y) = X \cdot Y & X + \overline{X} \cdot Y = X + Y & (3.6f) \end{array}$$

$$X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z \tag{3.6g}$$

$$(X + Y) \cdot (\overline{X} + Z) \cdot (Y + Z) = (X + Y) \cdot (\overline{X} + Z) \tag{3.6h}$$

A very important pair of equations are DeMorgan's theorems which allow us to switch between sums of products and products of sums.

$$\overline{(X_1 \cdot X_2 \cdot \ \cdots \ \cdot X_n)} = \overline{X_1} + \overline{X_2} + \cdots + \overline{X_n} \tag{3.7a}$$

$$\overline{(X_1 + X_2 + \cdots + X_n)} = \overline{X_1} \cdot \overline{X_2} \cdot \ \cdots \ \cdot \overline{X_n} \tag{3.7b}$$

The following are very important to remember:

$$\overline{A} \cdot \overline{B} \neq \overline{AB} \tag{3.8a}$$

$$\overline{A} + \overline{B} \neq \overline{A + B} \tag{3.8b}$$

3.4 Logic Gates and Truth Tables

It is important to be able to transform between equation, diagrams/circuits, and truth tables.

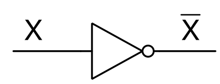

Figure 3.1: The NOT gate, also called an inverter, outputs NOT(A).

A	X
0	1
1	0

Table 3.1: This is the truth table for a NOT gate.

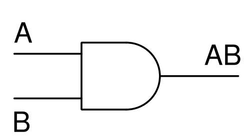

Figure 3.2: The AND gate outputs A AND B.

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Table 3.2: This is the truth table for an AND gate.

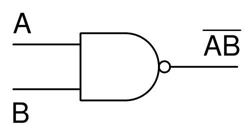

Figure 3.3: The NAND gate outputs NOT(A AND B).

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.3: This is the truth table for an NAND gate.

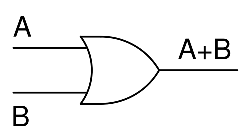

Figure 3.4: The OR gate outputs A OR B.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Table 3.4: This is the truth table for an OR gate.

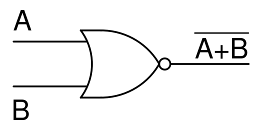

Figure 3.5: The NOR gate outputs NOT(A OR B).

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Table 3.5: This is the truth table for an NOR gate.

Chapter 4

Arduino Startup

4.1 Introduction

This chapter gives the students an introduction to the hardware we are using and gets them started with the Arduino IDE.

4.2 Datasheets

4.2.1 Arduino Nano Connect RP2040

The data sheet for the Arduino Nano Connect RP2040 is located at https://docs.arduino.cc/resources/datasheets/ABX00053-datasheet.pdf.

The main website for it is at https://docs.arduino.cc/hardware/nano-rp2040-connect

The pinout is at https://content.arduino.cc/assets/Pinout_NanoRP2040_latest.png.

4.2.2 RP2040 Microcontroller

The RP2040 microcontroller datasheet (all 654 pages) is at https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf.

If you are ever interested in putting the microcontroller onto a circuit board yourself, there is a reference design at https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf.

4.2.3 IMU - ST LSM6DSOXTR

The IMU datasheet is at
https://www.st.com/resource/en/datasheet/lsm6dsox.pdf

4.2.4 Mic - ST MP34DT06JTR

The microphone datasheet is at
https://www.st.com/resource/en/datasheet/mp34dt06j.pdf

The overview page shows that the microphone is still actively being produced.

4.2.5 WiFi and Bluetooth - U-blox® Nina W102

The main page for the wireless unit is at
https://www.u-blox.com/en/product/nina-w10-series-open-cpu.

The datasheet is at
https://www.u-blox.com/sites/default/files/NINA-W10_DataSheet_UBX-17065507.pdf.

4.2.6 Cryptographic IC - Microchip® ATECC608A

Note that Microchip doesn't suggest using this chip in new designs so expect that some future versions of the Nano RP2040 Connect to use the successor (ATTECC608B).

The datasheet for the A is here:
https://ww1.microchip.com/downloads/en/DeviceDoc/ATECC608A-CryptoAuthentication-Device-Summary-Data-Sheet-DS40001977B.pdf.

Note that it says it is a summary datasheet. You have to sign an NDA with Microchip to see the full datasheet.

4.3 Installing the IDE

These directions assume that you are using the lab computers and One Drive.

1. Go to software download page: https://www.arduino.cc/en/software

2. Download the Windows ZIP file (not the first link or the app)

3. Open the zip file and copy the folder inside (arduino-1.8.19 as of this writing) into your One Drive folder. This may take a while. If you are on your own computer, you can use any of the programs.

4. Once that transfer finishes, go into the folder and run arduino.exe. Windows will try to save you, but if you click More Info you can click Run Anyway.

5. Windows Defender Firewall will also complain. Uncheck the box that is checked and/or click Cancel.

6. It should load up with a window that looks like Figure 4.1.

7. In order to get it to connect correctly to your board, you need to install the Arduino Nano Connect RP2040 board.

   (a) Navigate to Tools→Board: "Arduino Uno" (or similar)→Boards Manager

   (b) It should load as shown in Figure 4.2.

   (c) In the search bar, type "arduino nano connect" (without the quotes)

   (d) The first item should be Arduino Mbed OS Nano Boards and should list the Arduino Nano RP2040 Connect.

   (e) Move your cursor over it and it should show an Install button. Click it to install the board library.

   (f) Wait for it to finish.

   (g) While you are waiting, plug your Nano Connect into your computer and let it install it.

   (h) As it finished, I received a User Account Control warning asking if I wanted to let dpinst-amd64.exe make changes to my device. I said yes.

   (i) Next it asked me if I wanted to install Arduino Universal Serial Bus devices. Again, click to Install.

   (j) It popped up again and I clicked Install again. Now it should say that the Arduino Mbed OS Nano Boards has been installed.

   (k) Close the Boards Manager.

8. Now go to Files→Examples→01.Basics→Blink.

9. This will open another window with the Blink program.

10. Go to Tools→Board→Arduino Mbed OS Nano Boards and select the Arduino Nano RP2040 Connect

11. Go to Tools→Port and select the COM that isn't COM1 (mine showed up as COM5)

12. Click the right arrow under the word edit in the menu to Uplaod the sketch to the Arduino board.

13. It should say "Compiling sketch..." in the lower left and show a progress bar on the lower right.

14. Then it should switch to Uploading... and finally Done Uploading.

15. An orange light near the USB port on your board should be blinking.

16. Congratulations! You have programmed your board!

17. Now look in the program for the two delay statements. Try changing the values inside the parentheses and re-uploading it. Does the blinking change?

18. In order to save files and have it portable, you need to change the directory where the Arduino IDE stores it's sketchbooks

    (a) Go to File→Preferences

    (b) Change the Sketchbook location to your OneDrive and a folder named arduino (lowercase is good)

    (c) My OneDrive was in C:\Users\mcneils2\OneDrive - Embry-Riddle Aeronautical University\arduino

19. Now try saving the blink sketch with your changed values.

20. Demonstrate your working blink and it's storage location to your instructor/TA

21. Here are some other Examples to test:

(a) Basics → fade: change the variable led to be LED_BUILTIN, watch the red/orange LED pulse

(b) Analog → AnalogInOutSerial: change analogInPin to A1 and analogOutPin to LED_BUILTIN. Turn the potentiometer (R6) with a screwdriver and watch the LED change accordingly. If you press the magnifying glass button in the top right the actual values will scroll by. You may need to set the baud rate to 9600.

(c) Digital → DigitalInputPullup: Change the first pinMode call to use 9 instead of 2. The same for the digitalRead command (2→9). Press the left button and see the LED blink. The right button is pin 10. Note that this program isn't written as well as the others since you have to change a number in two places. Could you rewrite it better?

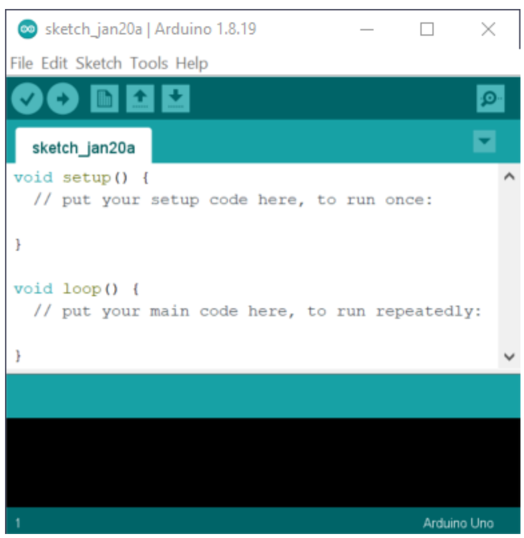

Figure 4.1: This is what the Arduino IDE should load up to.

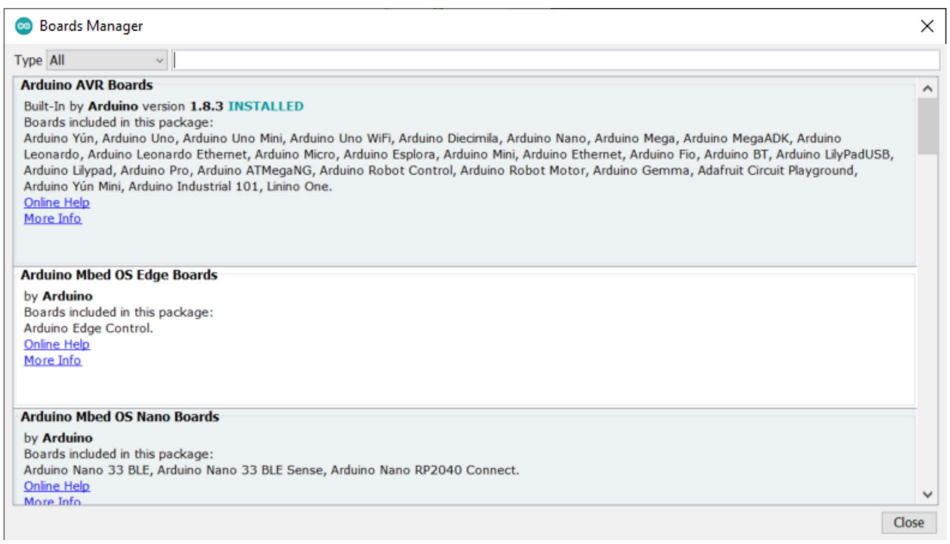

Figure 4.2: This what the Boards Manager loads up to.

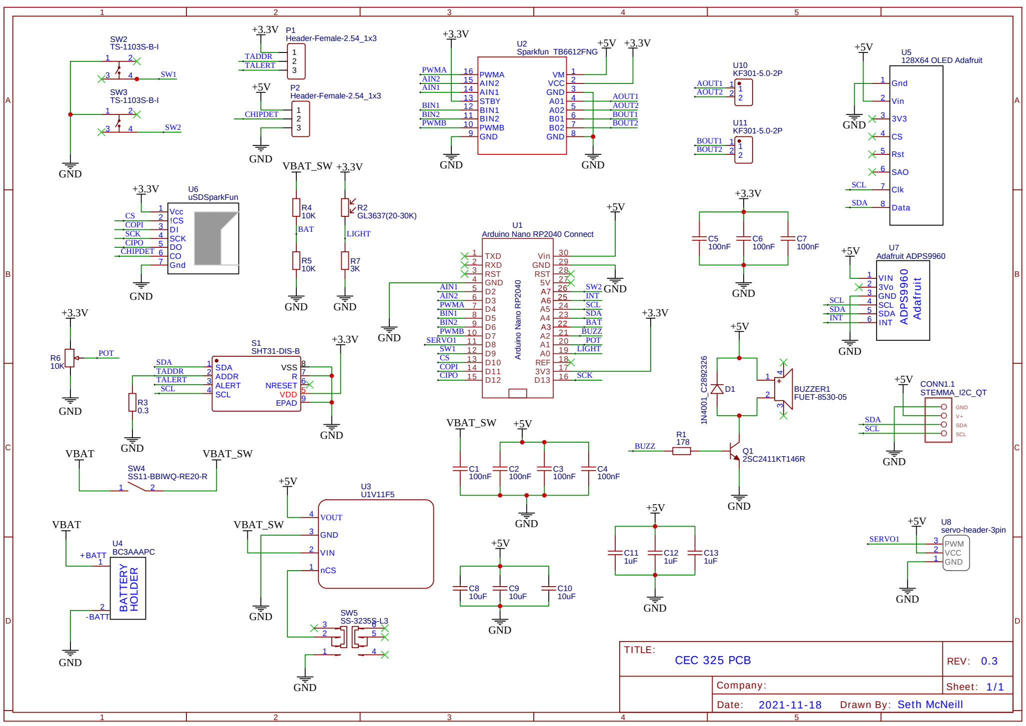

Figure 4.3: This is the schematic of version 0.3 of the board. This is the board used in Spring 2022.

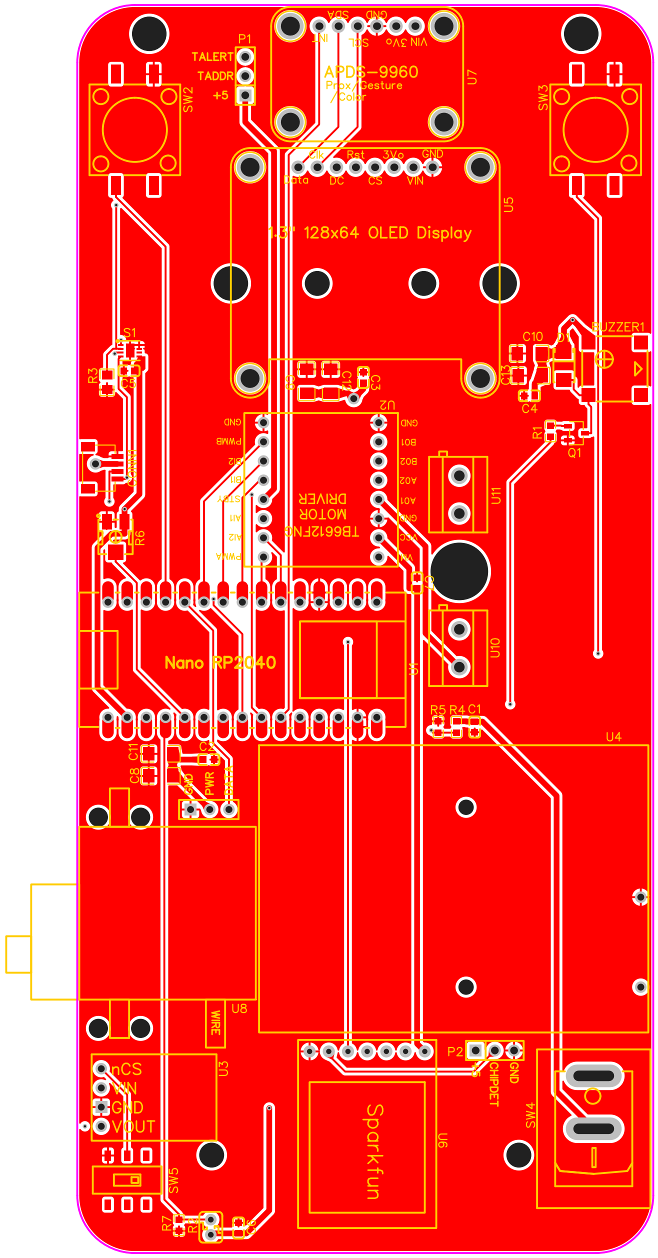

Figure 4.4: This shows the top of version 0.3 of the board for this class to help locate components.

Chapter 5

Buttons and Serial Communications

5.1 Introduction

This chapter introduces students to using buttons and serial communications.

5.2 Buttons

A typical button circuit is shown in Figure 5.1. The input is high until the button is pressed. Unfortunately, being mechanical, buttons do not always create nice, clean switched inputs as is shown in Figure 5.2. Maxim Integrated has a nice paper on the topic advertizing their devices to solve the problem. However, if we don't have the option of adding their hardware, some work can be done in software to debounce an input. As Maxim mentions, software debouncing is not free. It does incur some overhead so you probably do not want to do it for many inputs. Am example of software debouncing is in Listing 5.1.

```
/*
Debounce

Each time the input pin goes from LOW to HIGH (e.g.
 because of a push-button
press), the output pin is toggled from LOW to HIGH
 or HIGH to LOW. There's a
```

```
minimum delay between toggles to debounce the
 circuit (i.e. to ignore noise).

The circuit:
- LED attached from pin 13 to ground through 220 ohm
  resistor
- pushbutton attached from pin 2 to +5V
- 10 kilohm resistor attached from pin 2 to ground

- Note: On most Arduino boards, there is already an
 LED on the board connected
  to pin 13, so you don't need any extra components
 for this example.

created 21 Nov 2006
by David A. Mellis
modified 30 Aug 2011
by Limor Fried
modified 28 Dec 2012
by Mike Walters
modified 30 Aug 2016
by Arturo Guadalupi

This example code is in the public domain.

https://www.arduino.cc/en/Tutorial/BuiltInExamples/
 Debounce
*/

// constants won't change. They're used here to set
 pin numbers:
const int buttonPin = 9;     // the number of the
 pushbutton pin
const int ledPin = 13;       // the number of the LED
  pin

// Variables will change:
```

```
int ledState = LOW;         // the current state of
 the output pin
int buttonState;             // the current reading
 from the input pin
int lastButtonState = LOW;   // the previous reading
  from the input pin

// the following variables are unsigned longs
 because the time, measured in
// milliseconds, will quickly become a bigger number
  than can be stored in an int.
unsigned long lastDebounceTime = 0;  // the last
 time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce
  time; increase if the output flickers

void setup() {
Serial.begin(115200);
while(!Serial) delay(10);
Serial.println("Starting...");

pinMode(buttonPin, INPUT);
pinMode(ledPin, OUTPUT);

// set initial LED state
digitalWrite(ledPin, ledState);
}

void loop() {
// read the state of the switch into a local
 variable:
int reading = digitalRead(buttonPin);

// check to see if you just pressed the button
// (i.e. the input went from LOW to HIGH), and you'
 ve waited long enough
// since the last press to ignore any noise:
```

```
// If the switch changed, due to noise or pressing:
if (reading != lastButtonState) {
  // reset the debouncing timer
  lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
  // whatever the reading is at, it's been there for
  longer than the debounce
  // delay, so take it as the actual current state:

  // if the button state has changed:
  if (reading != buttonState) {
  buttonState = reading;

  // only toggle the LED if the new button state is
 HIGH
  if (buttonState == HIGH) {
    ledState = !ledState;
  }
  }
}

// set the LED:
digitalWrite(ledPin, ledState);

// save the reading. Next time through the loop, it'
 ll be the lastButtonState:
lastButtonState = reading;

}
```

Listing 5.1: This is the Arduino example of software debouncing.

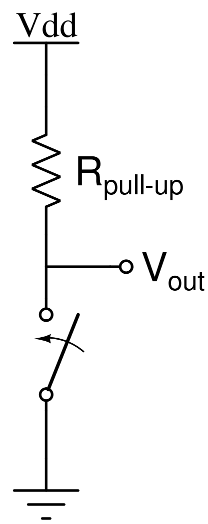

Figure 5.1: This is a typical button input circuit. It is active low in that the output signal will be high until the button is pressed.

5.3 Serial Communications

Serial communications sends data one bit at a time from one device to another as shown in Figure 5.3. This is in contrast to parallel communications where multiple bits (8 in the example) are transferred simultaneously between devices as shown in Figure 5.4. As can be seen parallel communications is much faster than serial since you can send so many bits simultaneously. However, parallel requires many more pins on each device to communicate. This is challenging in the embedded systems world since most microcontrollers don't have many pins. Also, connectors with many pins tend to fail more often than connectors with fewer pins. Because of these reasons, the embedded world communicates primarily with serial protocols.

A table of serial protocols is listed in Table 5.1.

5.3.1 Universal Asynchronous Receiver-Transmitter

The Universal Asynchronous Receiver-Transmitter (UART) protocol has been around quite a while. Older computers used to ship with serial ports that used this protocol, usually implemented as the RS-232 protocol. UARTs are used to communicate between two devices. It does not allow for more than

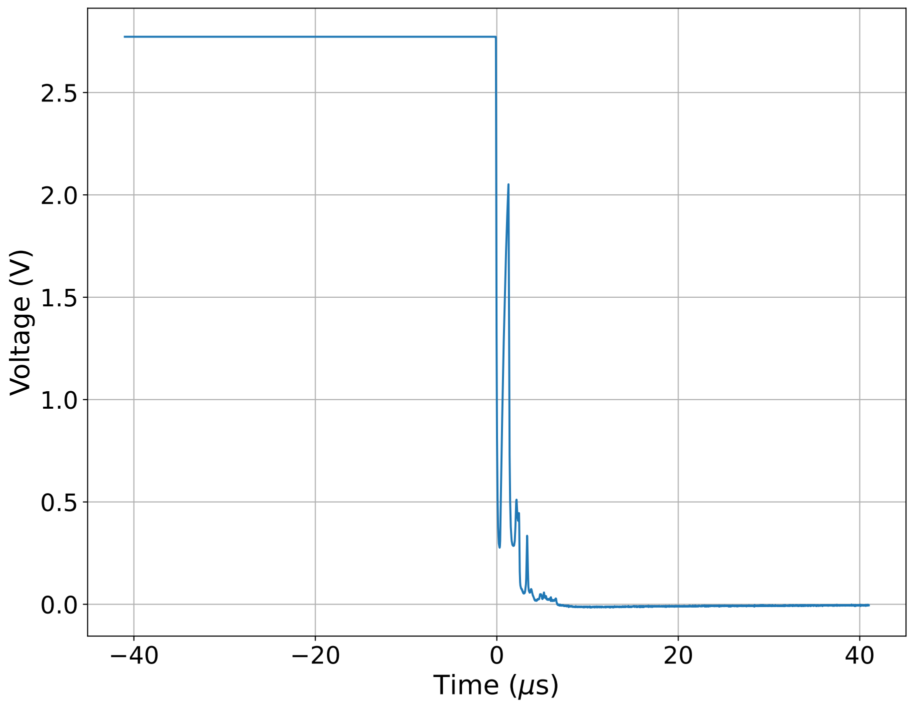

Figure 5.2: This is an example of what the output signal from a button with the circuit in Figure 5.1 could look like.

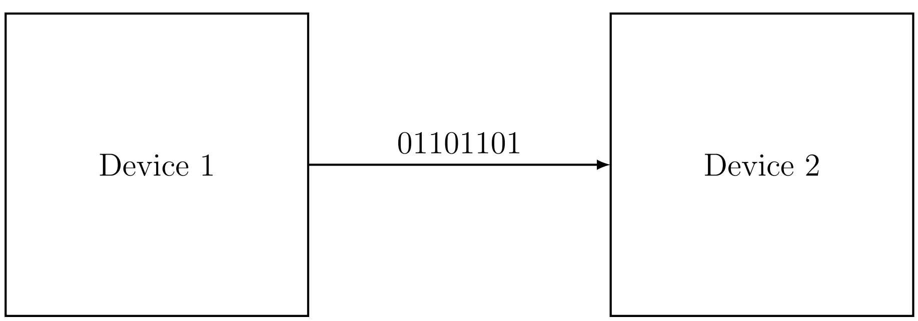

Figure 5.3: Serial transfers one bit at a time.

two devices. It is full duplex and communicates over 2 wires (plus a ground for reference). One wire transmits data from Device 1 to Device 2 and is

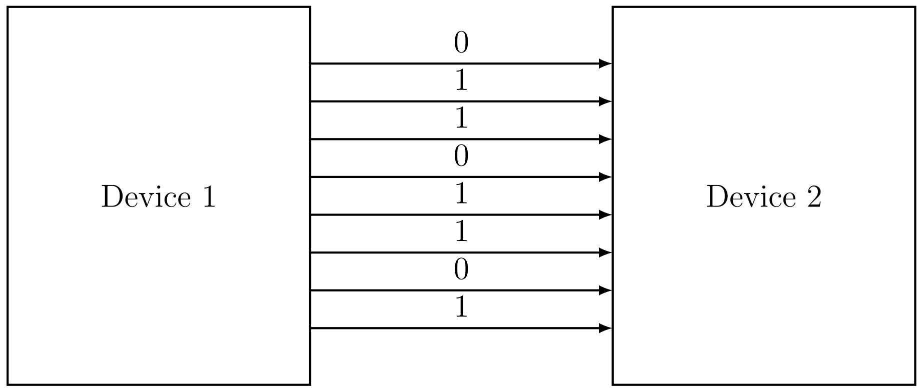

Figure 5.4: Parallel transfers multiple bits at a time.

Protocol	Description
UART	Used to communicate between computers and Arduino boards
SPI	Used for higher speed communications between devices on circuit boards
I^2C	Only uses two wires and allows for multiple controllers and peripherals
CAN	*Often used in the automotive industry*
RS-485	*Differential signaling for robustness (noisy and long wires)*
1-Wire	*Only requires 1 wire and ground (no power) to communicate*
USB	*Ubiquitous on computers now*

Table 5.1: This is a list of some of the more common serial protocols. The grayed out protocols will not be discussed further.

connected to Device 1's TX pin and Device 2's RX pin. The second wire transmits data from Device 2 to Device 1 and is connected to Device 1's RX pin and Device 2's TX pin. This is illustrated in Figure 5.5. This protocol does require both devices to use the same specified transmission speed. Some common speeds are 9600, 14,400, 57,600, and 115,200. It can go faster. Nowadays, it is generally best to go as fast as possible so that the communications takes less of the processor's time. The most common speed used now seems to be 115,200. The asynchronous part of the name comes from the fact that there is no clock line for UARTs. It typically has a maximum speed of 1.5 megabits per second (Mbps).

UART is the protocol used to communicate between the computer and your Arduino board. You can see this in the code in the setup function where `Serial.begin(115200)` or `Serial.begin(9600)` shows up often.

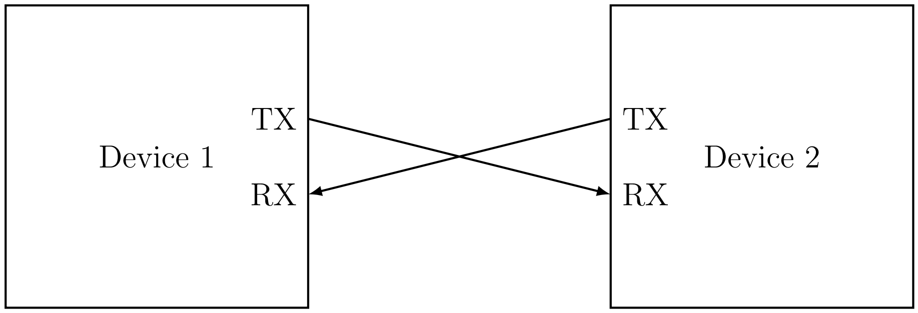

Figure 5.5: A UART has full duplex between two entities.

5.3.2 Serial Peripheral Interface

The Serial Peripheral Interface (SPI) is a full-duplex serial interface shown in Figure 5.6. It is setup in a controller-peripheral (previously known as master-slave) architecture with only one controller on the bus. It does allow for multiple peripherals by giving each peripheral its own chip select (CS) line. Since it is synchronous, it does require a clock line. This means that it takes a minimum of 4 wires as listed in Table 5.2. The SCLK signal is the clock signal used by both the controller and peripheral. There is one CS line per peripheral. The CS line lets the controller specify which peripheral it is communicating with. The COPI line is the data going from the controller to the peripheral. The CIPO line is the data going from the peripheral to the controller.

SPI has a maximum data rate of 60 Mbps, which is the fastest of the 3 protocols commonly used in the embedded world. Because of this, it is used in more data intense situations like SD Cards (where we will be using it) and displays.

Signal	Description
SCLK	Clock signal to keep everything synchronized
CS	Chip select–tells a peripheral that the controller is communicating with it
COPI	Controller Out, Peripheral In (used to be MOSI)
CIPO	Controller In, Peripheral Out (used to be MISO)

Table 5.2: The SPI protocol uses these signals to connect.

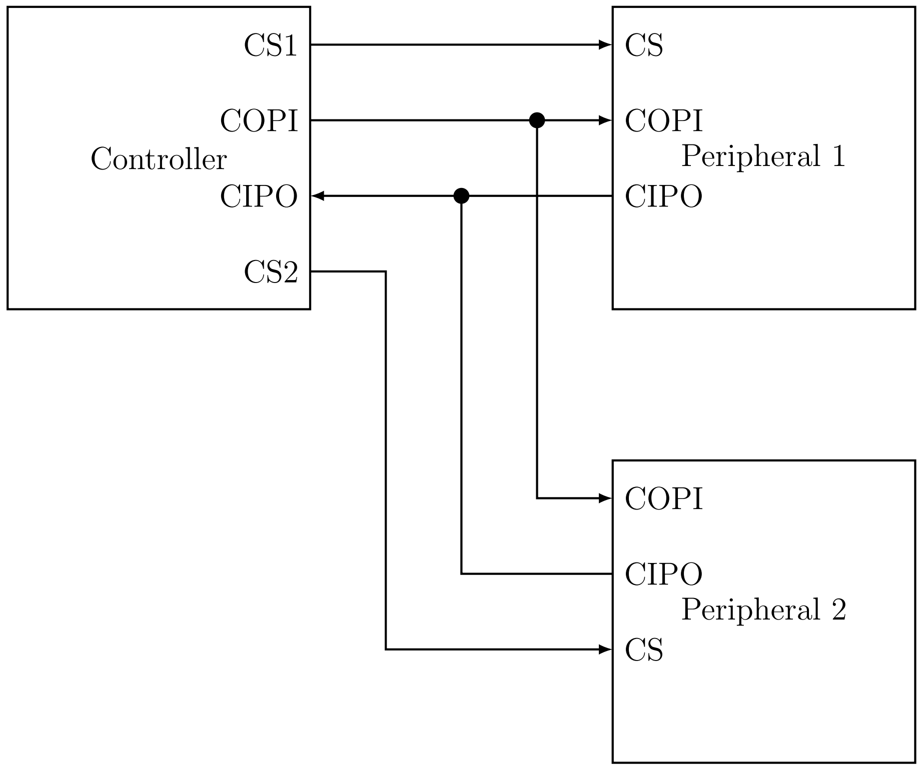

Figure 5.6: SPI allows for one (sometime more) controller and multiple peripherals.

5.3.3 Inter-Integrated Circuit

The Inter-Integrated Circuit (I^2C, IIC, or I2C) protocol is multi-controller and multi-peripheral. Basically, any device connected to the bus can drive the communication. It is single ended so data only flows one way at a time. It only requires 3 wires, SCL - clock, SDA - Data, and a ground reference. SCL and SDA do require pull-up resistors so that devices only have to pull the lines to ground to communicate as shown in Figure 5.7. Instead of using chip select lines like SPI, I2C requires each peripheral to have a unique address. I2C addresses for modules that are on the board or may be used with the board can be seen in Table 5.3. I2C has a max speed of 3.4 Mbps but is only 100 kbps in standard mode.

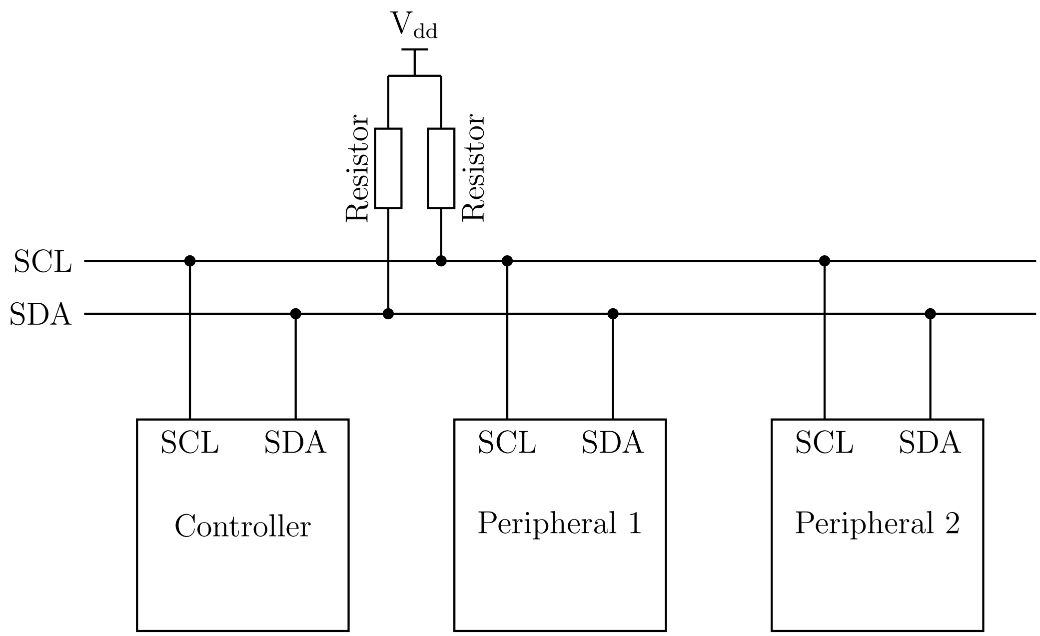

Figure 5.7: I^2C allows for multiple controllers and peripherals on the same bus.

Address (HEX)	Module
0x44 or 0x45	SHT31-DIS Temperature/Humidity
0x3D	1.3” 128x64 OLED Display
0x39	APDS-9960 Light, Color, Proximity, Gesture
0x77	BME688 Temperature, Humidity, Gas
0x2D, 0x53, and 0x57	ST25DV16 Dynamic NFC/RFID Tag IC
0x30 or other	NeoKey 1x4 QT breakout board
0x10	STEMMA MiniGPS

Table 5.3: I2C addresses for relevant modules.