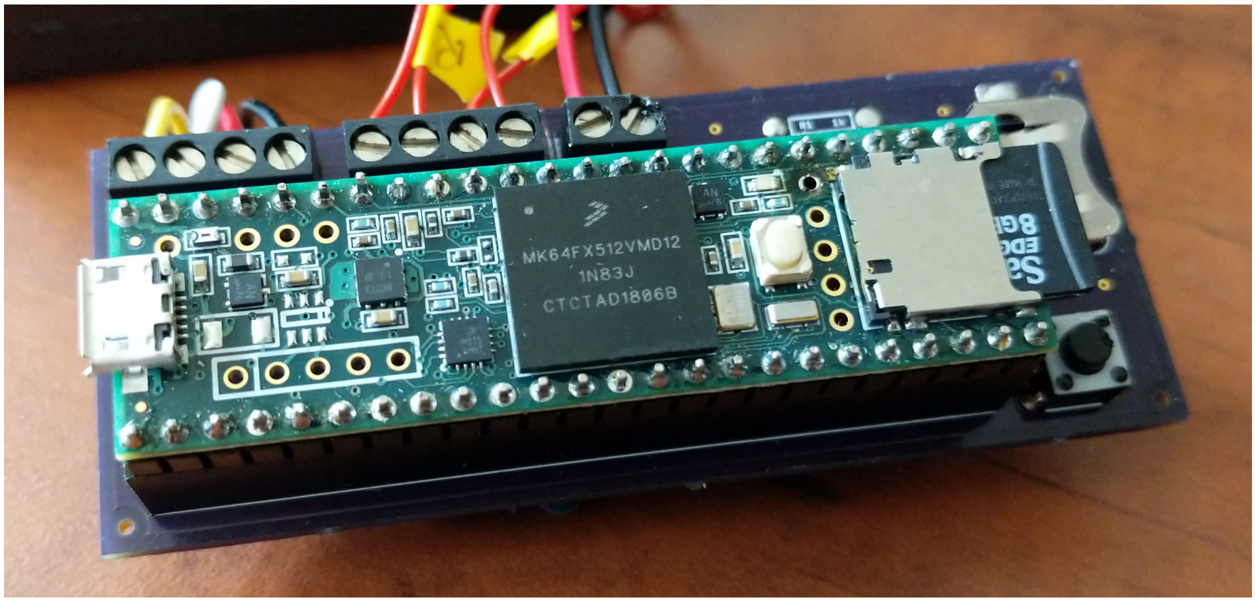
CHEM 191 Workbook: Building a Teensy Air Quality Monitor

Al Fischer

Western Carolina University

Contents



Preface

What is Arduino?

Over the course of the semester, we will build air quality monitors using the Teensy microcontroller platform. This platform is an offshoot of the popular Arduino microcontrollers, and they are very similar. This workbook will introduce you to the basics of Arduino and Teensy, and walk you through assembling and programming your air quality sensor.

Arduino is an open-source platform for developing interactive electronic devices. Arduinos of various flavors are used for all variety of things, from controlling motors and lights to reading data from sensors. Exercise 1 will provide a more thorough overview of Arduino.

OK, but what is Teensy?

Teensy is an offshoot of Arduino. It is capapble of most everything an Arduino can do and often much more. You might think of it as the supercar version of an Arduino. Aside from being more capable, it's also smaller – a fact that gives rise to it's name (Teensy). In this class, the names Teensy and Arduino will be used interchangeably. Exercise 1 will provide a more thorough overview of Teensy.

Using this Book

Some common typographical (formatting) conventions will be used throughout this book.

Code

The exercises will make extensive use of computer code, which will always be represented with `monospace` font with a grey background. It may be `inline` or

```
offset.
```

When you see code, you should take time to thoroughly digest it. Sometimes the code may work as-is; other times you may have to add to the code to make it work. And other times the code may be `pseudocode`, which is example code that looks like it would work but will not actually compile. The code will often be color-coded to highlight various features of the programming language.

Notes and Warnings

This book will also make use of notes and warnings. Examples are below.

> *A note is written in a quote box with italic font and represents background information or a bit of knowledge to think about.*

> A NOTE ALL CAPITAL LETTERS REPRESENTS A SAFETY WARNING. YOU **MUST** PAY ATTENTION TO THESE WARNING OR YOU MAY BREAK EQUIPMENT AND/OR HARM YOURSELF OR YOUR CLASSMATES.

Is this work dangerous?

Although electricity can be very dangerous and even lethal, the work we'll do in this class shouldn't pose any particular danger if done carefully. There are two important factors to consider when deciding how dangerous an electrical device is: (1) the voltage and (2) the current. Generally speaking, it takes both to be dangerous. Most of the voltage in these exercises will be either 3.3V or 5V, with currents on the order of 100 mA. These are relatively low voltages and low currents, are safe to work with, and do not pose a shock hazard. That said, connecting things incorrectly can still break components, cause them to get very hot (they might burn you), or even go up in smoke – be careful and double check yourself/your lab partner when connecting things!

Additional Resources

There are myriad books, blog posts, and websites about Arduino, but few specifically deal with the Teensy version. This workbook and the Teensy website are your best resources regarding Teensy-specific items. The code used to program the Teensy (and Arduinos) is a version of `C`; knowing that will be useful when trying to find help online. The Arduino language reference provides a list of available functionalities and syntax.

If you want more general information on programming Arduinos, the Arduino IDE, or electronics, you may check some of the following:

- Monk, Simon. ***Programming Arduino: Getting Started with Sketches***.
- Scherz, Paul and Simon Monk. ***Practical Electronics for Inventors***.
- The Arduino forum.
- Boxall, John. ***Arduino Workshop***.

Chapter 1

Getting Started with Arduino

Objective: Install Arduino and Teensyduino
To turn in: Nothing! Make sure your software is installed and working so you're ready for the next class period.

Introduction

Arduino is an open-source platform for developing interactive electronic devices. More specifically, Arduinos are *microcontrollers*, which are essentially small computers that can be programmed to perform a specific task over and over again. Microcontrollers are ubiquitous in the modern world and are embedded in scientific equipment, cars, cell phones, and nearly every other electronic device. Arduino provides a simple, standardized interface for microcontrollers, which has led it to become a very popular tool for hobby projects, citizen science tools, and even art installations. Arduino began in 2005 as a student project at the Interaction Design Institute Ivrea in Ivrea, Italy.

Using an Arduino requires a *program*, or a set of commands that are uploaded to the Arduino to have it perform the desired task. In the Arduino community, the term *sketch* is used synonymously with program. Examples of possible programs include having the Arduino **measure** things like temperature, light, or humidity, or **control** things like lights or motors. You could even have it measure the temperature and turn on a heater when it gets too cold. Although an Arduino can run as a stand-alone device once setup, it must first be programmed by a computer.

To program the Arduino, we'll use the a piece of software called an *integrated development environment* (IDE) that's available for free on the Arduino website. The IDE is shown in the figure below. At the top of the IDE is the menu bar, just like any other program. The middle part of the IDE is the "text area", and looks like a text editor or simple word processing program. The text area is where you write the code. Finally, the black bottom region of the editor is the message area; this is the area where important messages detailing errors and successes will be displayed. Arduino code is written in the programming language *C* – probably the most commonly used language in the world – but the IDE handles some of the programming behind the scenes to make it easier for the user.

Part of the beauty of Arduino is its open-source nature. Many spin-offs and flavors of Arduino exist due to it's open-source nature, each with its own unique benefits. In this class, we'll use a variant of Arduino called the ***Teensy*** (specifically Teensy 3.5). The Teensy is programmed in the same way as an Arduino, but has a smaller form-factor and more capabilities than a standard Arduino. The picture below shows a Teensy 3.5 and the functions of each connection on the Teensy. Seeing all the functions laid out like that can be overwhelming, but we'll walk through the ones we need to use step-by-step in future exercises.

This exercise will walk you through installation of the Arduino IDE, the software for the Teensy (Teensyduino), and an initial check of the board.

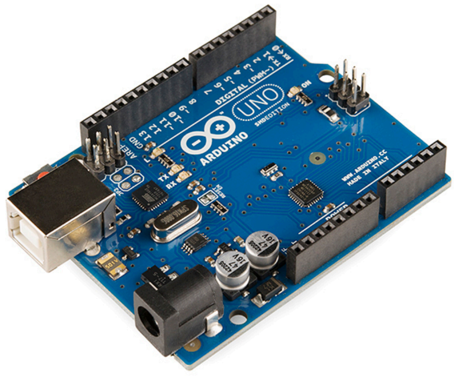
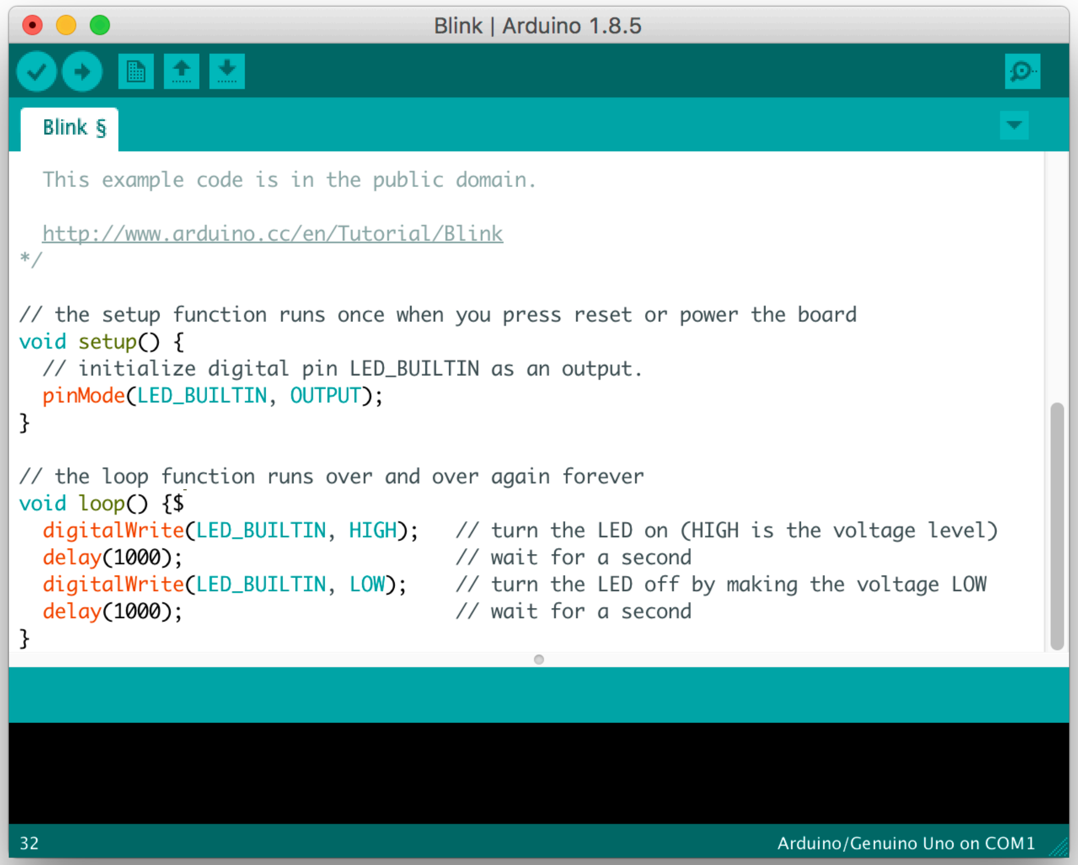


Figure 1.1: *An Arduino Uno. This class will use a variant of the Arduino, called the Teensy, as described below. (image credit: Spark Fun, CC BY 2.0)*



```
  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {$
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Figure 1.2: *Arduino IDE showing an example program (image credit: Cedar101 [CC0], wikimedia.org)*

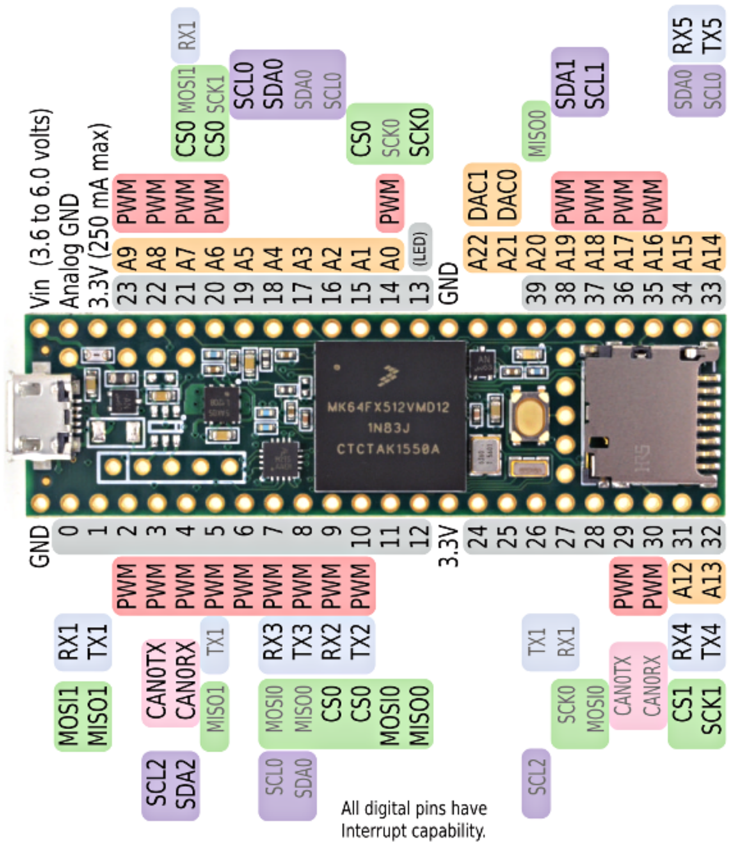


Figure 1.3: *A Teensy Board, with the pin (connection) numbers and functionalities shown (reprinted from PJRC.com).*

1.1 Install the Arduino IDE

You MUST follow the order of the instructions here or Teensyduino will not work!

1. Use a web browser to navigate to https://www.arduino.cc/en/Main/OldSoftwareReleases#previous.
2. Download the installer for version **1.8.6**. You must choose the installer appropriate for your operating system.
3. Choose **Just Download** to download the software, or make a donation if you're feeling generous!
4. Open the file that downloads and run the installer.
5. Open the Arduino program to make sure it runs and to prepare for the next steps.

1.2 Install Teensyduino

1. Use a web browser to navigate to https://www.pjrc.com/teensy/td_download.html.
2. Download the correct installer for your operatring system. Note that steps 1 and 2 on the Teensy page should have been completed when you installed the Arduino IDE, above.
3. Run the installer. When prompted:
4. Select the `Arduino/` folder for the install location. If **Next** is greyed out you probably downloaded the wrong version of the Arduino IDE.
5. Select **All** when asked which additional libraries to install.

1.3 Test the Teensy!

1.3.1 Check the board

1. Plug your Teensy into the computer via the USB connection.

   BE ***EXTREMELY*** CAREFUL WITH THE USB CONNECTION ON THE TEENSY. THEY BREAK ***VERY*** EASILY!!

2. You should see the orange light blinking on the Teensy. If you do not, something is wrong with your Teensy and you should notify your instructor.

1.3.2 Check the software

1. Open the Arduino IDE.
2. Click `Tools > Board > Teensy 3.5`
3. Click `File > Examples > 01.Basics > Blink`.
4. Find the part in the program that says `delay(1000)` and change it to `delay(500)`.
5. Click **Verify** in the software (checkmark button in upper left corner).
6. Press the white button on the top of the Teensy.
7. Press upload in the software (right arrow button, next to checkmark).
8. You should see the orange light blink more quickly. If you encountered any errors, something could be wrong with your software installation or you may have made a typo in the code. Notify your instructor of any problems.

Chapter 2

Blink an LED

Objective: Use Arduino to turn an LED on and off.
To turn in: Your working Arduino code and the Lab 2 Worksheet.

Introduction

This exercise will explore two concepts: (1) connecting devices to the Arduino and (2) the structure of the Arduino program used to *control* those devices. As a first example, we'll connect an LED (light-emitting diode) to the Teensy and write a program to turn it on and off.

How are things connected to a Teensy?

The Teensy can control all sorts of devices, including lights, motors, and various sensors. The device to be controlled must be connected to one of the Teensy's *pins*. You can think of a pin as a unique port or connection on the Teensy. Pins can send and receive information, so communication between the pin and the sensor goes both ways. One pin might be used to control a heater, while another might be used to read data from a temperature sensor. The pins that are available on the Teensy are shown in the figure below. Some pins provide power (3.3 V), some pins provide a connection to the circuit ground (GND), and some function to send and receive data. A lot of pins have several functions, each represented by a different color rectangle on the diagram. Note, though, that each pin can only have one function at any given time. For now, we'll just focus on the pin numbers. Those labeled 0-33 (shown in grey) are called *digital pins*; they function as both inputs (to receive data, or sense) and outputs (send data, or control), but we'll use just the output functionality in this exercise.

There are several ways to connect something to the pins of the Teensy. One way is to directly solder wires to them, in which a soft metal alloy is melted onto two the wire and the Teensy pins as they are connected to make a permanent, metal connection. This is best saved for a final, fully vetted design that won't ever need to be changed. We'll use another method for this exercise called a solderless breadboard.

The solderless breadboard

A breadboard is shown in the figure below. The holes in the breadboard are spaced in such a way that a Teensy can be plugged directly into the board. Columns, labelled A, B, C, ..., are connected across the board electrically, such that A1 and B1 are connected; rows are insulated from each other, such that A1 and A2 are not connected. The connections labeled with red and blue lines on each edge are connected together and are sometimes called rails; they provide an easy way to distribute power (e.g. the red rail) and ground (e.g. the blue rail) connections around the board. If that seems confusing, try watching this video this video.

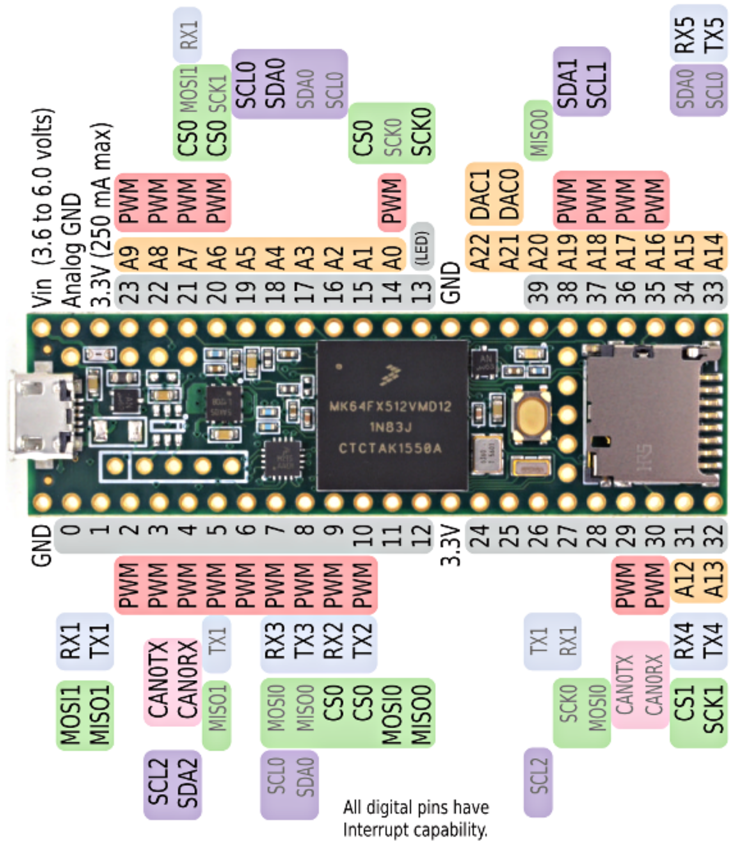


Figure 2.1: *A Teensy Board, with the pin (connection) numbers and functionalities shown (reprinted from PJRC.com).*

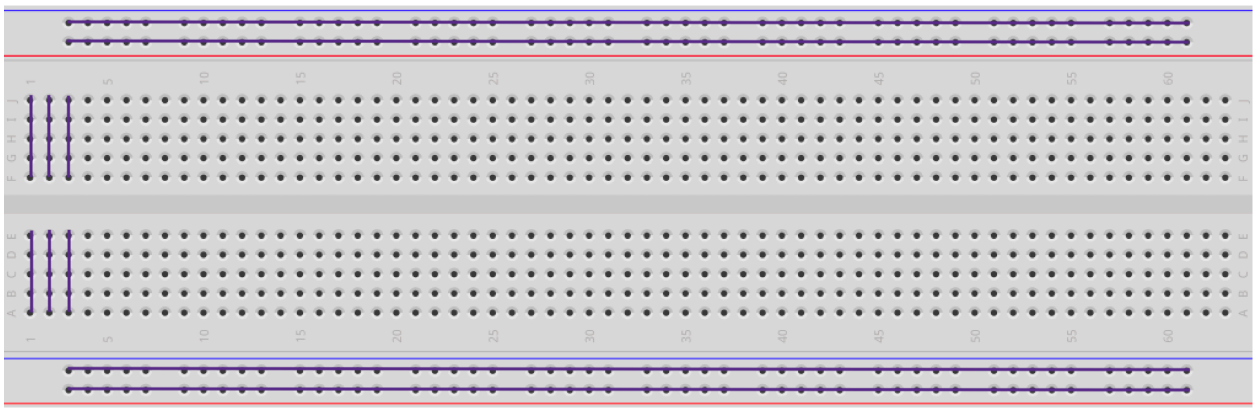


Figure 2.2: *A solderless breadboard with purple lines indicating connected holes; the pattern continues down the board. Holes with the same number are connected electrically, except across the gap in the middle, while holes with different numbers are electrically insulated.*

What is a program?

A *program*, or *sketch* in Arduino jargon, is a set of commands stored on the Teensy that tell it what to do. Although a Teensy can do most anything a computer would do, it can only run a single program over and over. For example, this exercise involves programming the Teensy to turn an LED on and off so that it blinks. Other options would be to collect data from a temperature sensor, or turn a motor on a off to move a robot (or to do all of those things at once).

An Arduino program always consists of two parts: (1) a `setup()` function and (2) a `loop()` function (more details about what a function is will come in future exercises). The `setup()` function runs once every time the Teensy is powered on. After that, the Teensy runs the `loop()` function over and over until it's turned off. Usually, some initial lines of code are included above the `setup()` function provide definitions used throughout the program. A "bare minimum" program with the `setup()` and `loop()` functions is shown below. Note that this program does nothing other than run.

```
void setup() {
  // anything written here will execute exactly once on startup
}

void loop() {
  // anything written here will execute over and over indefinitely until the power is turned off
}
```

Arduino sketches (programs) are written with the Arduino IDE (see Chapter 1). To begin, you open the IDE and write your code in the text area of the program. When your're done writing your code, you compile it and upload it to the board to run it. If it compiles and uploads correctly, you will see a "Success" message in the message are of the IDE; if not, you'll see an error message and will need to troubleshoot your code.

You should use comments when writing your program to clairfy what each part does. Comments are not read by the compiler/board, and do not execute. In other words, they are only to help humans reading the code understand it more easily. To add a comment, just use the \\ symbol; anything after that symbol won't be read by the program. For example, you can make an entire line a comment, like the first line below.

```
// Pin 13 has the LED on Teensy 3.0
int led = 13;
```

You could also make a portion of a line a comment, like the second half of the line below.

```
int led = 13; // Pin 13 has the LED on Teensy 3.0
```

Finally, note that every line in the program *must* end with a semi-colon, `;`.

In the exercise, we'll connect an 3-color LED to a Teensy and write a sketch to control it.

2.1 Connecting the LED

1. Push your Teensy into the breadboard provided and connect the USB cable. Try not to bend the pins as you push it in. Refer to the diagram below if necessary.

   REMEMBER, BE ***EXTREMELY*** CAREFUL WITH THE USB CONNECTION ON THE TEENSY. THEY BREAK ***VERY*** EASILY!!

2. Push the LED into the breadboard such that none of the leads (wires) are connected to anything else (each one is in it's own row). Make sure you note which row the longer lead goes into.

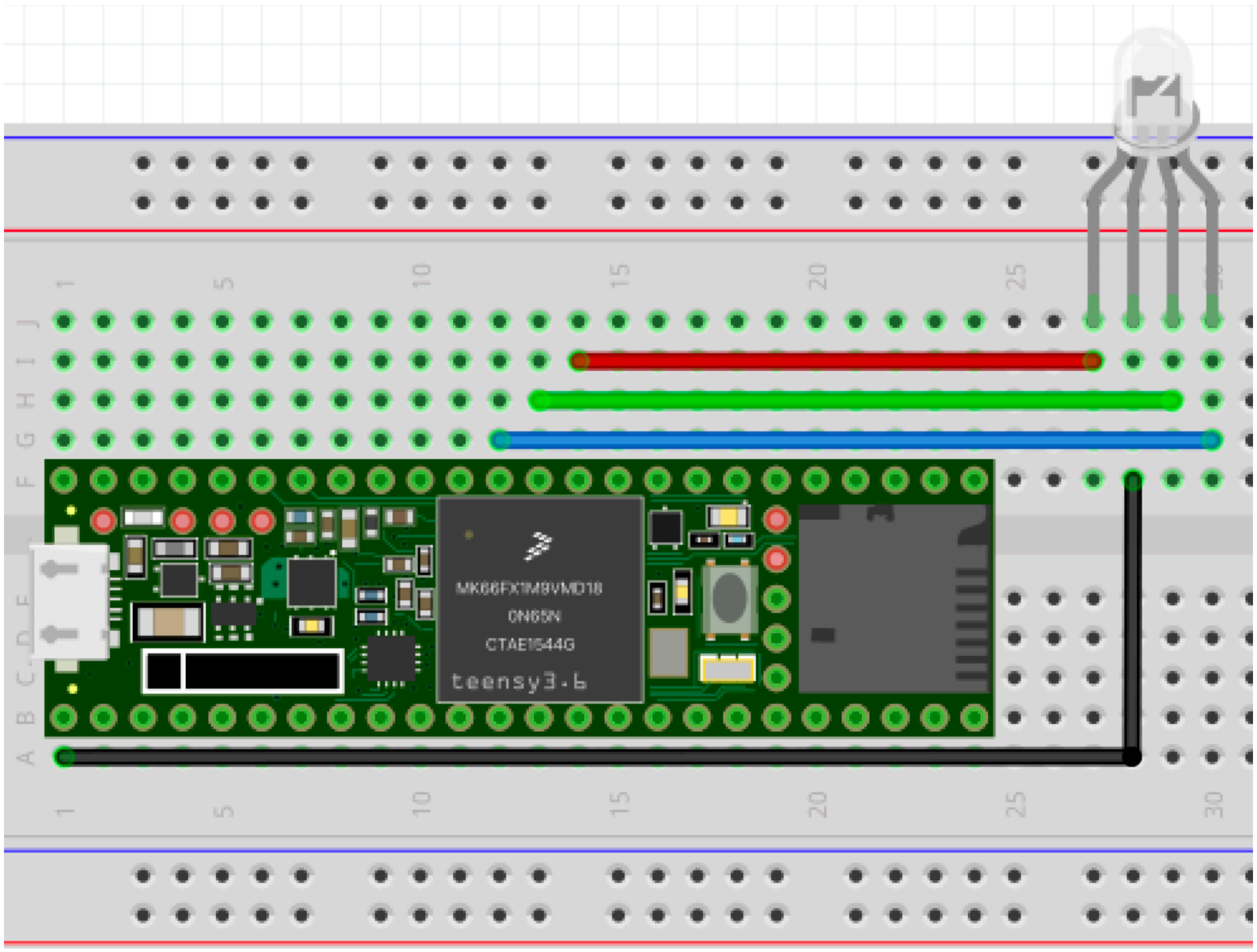


Figure 2.3: *Your final setup should look something like this picture.*

> *Always work with the Teensy unplugged from the computer. This will help ensure no "magic smoke" escapes.*

3. Use a jumper wire to connect the **longer lead** on the LED to the **GND** pin on the Teensy.
4. Use 3 more jumper wires to connect the remaining three leads to pins **13, 14, and 15**.

2.2 Program the Teensy

2.2.1 One Color

1. Open the Arduino IDE and load the Blink example (`File > Examples > 0.1 Basic > Blink`)
2. Go to `File > Save As` to save the example under a new name in the default location (`Documents/Arduino`). Call it **lastnameFirstname_blink.ino**.

   > *Remember, files that don't follow the naming convention **exactly** will incur point deductions.*

3. You should see the following at the top of the sketch:

```
    // Pin 13 has the LED on Teensy 3.0
    // give it a name:
    int led = 13;
```

Remember, anything following a `//` is a comment; it does not affect the Arduino program. Get in the habit of using comments to describe what each line of code does.

The line `int led = 13` defines the name of pin 13 as `led`. This name could be anything you wish. In this case, `led` is useful because the LED is connected to pin 13.

4. Next is the `setup()` function. In this case, the only setup necessary is to set the mode of the `led` pin as an output.

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

5. Finally, the `loop()` function, which runs over and over.

```
digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
delay(1000);               // wait for a second
digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
delay(1000);               // wait for a second
```

6. Assuming all those sections look good, press the **Upload** to button (right arrow in top left) to save, compile, and upload your sketch.

2.2.2 Three Colors

You should have seen a single color blinking on and off. Now, make some modifications so that all three colors blink on and off.

1. Define the two extra pins. In this case, it could be useful to refer to them by color.

```
  int blue = 13;   // blue pin connected to pin 13
  int green = 14;  // green pin connected to pin 14
  int red = 15;    // red pin connected to pin 15
```

*Have you saved your work yet? The IDE doesn't autosave, so make sure you save it **frequently!***

2. Set each of those to outputs in the `setup()` function. You will have to fill in the blanks for the red and green pins below.

```
// the setup routine runs once when you press reset:
void setup() {
  pinMode(blue, OUTPUT);     // initialize the blue pin as an output.
  pinMode(____, OUTPUT);     // initialize the green pin as an output.
  ______________________     // initialize the red pin as an output.
}
```

3. Finally, add the extra colors to the `loop()` function.

```
// blue
digitalWrite(blue, HIGH);   // turn the LED on (HIGH is the voltage level)
delay(1000);                // wait for a second
digitalWrite(blue, LOW);    // turn the LED off by making the voltage LOW
delay(1000);                // wait for a second

// green
digitalWrite(green, HIGH);   // turn the LED on (HIGH is the voltage level)
delay(1000);                 // wait for a second
digitalWrite(green, LOW);    // turn the LED off by making the voltage LOW
delay(1000);                 // wait for a second

// red
digitalWrite(red, HIGH);     // turn the LED on (HIGH is the voltage level)
delay(1000);                 // wait for a second
digitalWrite(red, LOW);      // turn the LED off by making the voltage LOW
delay(1000);                 // wait for a second
```

4. Make sure you show your code and blinking LED to your instructor before you move on.

2.2.3 Make Your Own Changes

1. Before you leave, modify your code to change the blink pattern. You can change the sequence of the colors, the time between blinks, the number of blinks, or the amount of time each color is on.

2.2.4 Turn in

1. A hardcopy of your completed Exercise 2 Worksheet.
2. An electronic copy of your final Blink code (submit to Dropbox).

*Remember, files that don't follow the naming convention **exactly** will incur point deductions.*

Chapter 3

Voltage and Communication

Objective: Measure a temperature with Teensy and display the result to the computer screen using serial communication.
To turn in:

Introduction

In the last exercise you used a Teensy to turn a light on and off, although we did not discuss what was happening electrically. This exercise will explore the electrical workings of the Teensy and walk you through how to write code to *sense* signals from a device. More specifically, you'll investigate (1) the concept of **voltage** while sensing temperature and (2) communicating .

Electricity

When first learning about electricity, it may help to think of it in analogous terms to water. **Current** is the flow of electricity through a circuit, and is analogous to the current in a river. Just like water molecules move through the river, charge carriers (electrons) in a move through the wires of the circuit. **Voltage** is a *potential energy difference* between two points in a circuit. This is analogous to the potential energy difference in water stored behind a dam and water at the bottom of the dam. Current will not flow without a voltage difference, just as water will not flow without a difference in height or pressure.

Recall that in the last exercise you turned an LED on and off using a pin on the Teensy. You did this by alternately setting the pin to `HIGH` and `LOW`, as in:

```
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
```

But what did this do internally in the Teensy? When the Teensy gets the command `digitalWrite(13, HIGH);` it sets the voltage of pin 13 to 3.3 volts (3.3 V is standard voltage for microcontrollers). Alternatively, when it gets the command `digitalWrite(13, LOW);`, it sets the voltage to 0 V. Thus, when the voltage is 3.3 V, there is a potential difference across the LED from 3.3 V on the controlling pin (anode) to 0V on the ground pin (cathode). This causes current to flow through the LED, which in turn causes the LED to light up. When the voltage is set back to 0, there is no longer a potential difference across the LED, current does not flow, and the LED does not light up.

You may have gotten the idea that voltage and current are inextricably linked, and that's true. In fact, **Ohm's Law** provides a mathmatical relation between the two:

$$\text{Voltage (volts)} = \text{Current (amps)} \times \text{Resistance (ohms, }\Omega)V = iR$$

This introduces a new concept, resistance. **Resistance** is simply the opposition to flow of electrical current. Resistance can be introduced to a circuit purposely with a device called a **resistor**. Usually these are a defined, fixed resistance introduced for a specific purpose. However, many sensors have a resistance that changes proporitionally to some stimulus. Thus, the voltage across the device will change in response to that stimulus, and the stimulus can be measured simply by measuring the voltage. As an example of how useful this is, consider a temperature sensor. A computer or microcontroller cannot sense the temperature directly – it only understands voltages. A device that converts changes in temperature to changes in voltage allows the computer to infer changes in temperature.

Although the voltage output from the Teensy pin was either HIGH or LOW, the voltage output from the temperature sensor can be any value. A signal that can be only HIGH or LOW (i.e. on or off, 1 or 0), is called a **digital signal**. Digital signals are often used to communicate between computers and to control many devices because they are relatively immune to outside noise. Many sensors, on the other hand, produce signals that vary *continuously*, and such signals are called **analog signals**.

The Teensy can read *digital* signals on any of its digital I/O (input/output) pins (grey labels on the pinout diagram). The Teensy can read *analog* signals on any of the analog input pin, which are shown with orange labels in the pinout diagram and numbered A0 through A22 (the A stands for analog).

Bits and Volts

Even though Teensy can "read" analog signals, it only understands digital signals. So every analog signal read by the Teensy gets sent through an *analog to digital converter* (ADC). This converts the analog signal from a sensor into a digital signal with discrete steps. The size of these steps is determined by the number of **bits** the ADC has. More bits equals more steps, and more steps equals a digital signal that more closely approximates the real analog signal.

Communicating via the serial port

In the previous exercise, you used digital signals to control an LED, but there was no communication between the Teensy and computer after the sketch was uploaded. When reading signals, it's often useful to display the reading on the computer screen to have real-time feedback from the program. To do this, the data collected on the Teensy board must be sent to the computer, the computer must read the data, and the computer must display it on the screen. The communication protocol used to do this is called **serial communication**. This is a very widely used protocol, and most computers used to come with a serial port built in. In our case, we'll conduct serial communication over the USB port.

Using serial communication on the Teensy is relatively simple. The first step is to open the serial port and tell it what data transfer speed to use (basically how quickly the Teensy and the computer talk to each other).

```
    Serial.begin(9600);
```

Once the connection is open, you can print most anything to the serial port you wish. For example, a simple hello world program would look something like:

```
    Serial.begin(9600);
    Serial.print("Hello, World!");
```

The above pseudocode would print "Hello, World!" to the computer's serial port. The result could be displayed on the computer screen by opening the serial monitor in the Arduino IDE. There are a few more details you'll learn as you go.

In this exercise, you'll measure ambient temperatures using a temperature sensor and display the result to the computer screen via the serial port.

3.1 Connect the Temperature Sensor

1. Push your Teensy into the breadboard provided and connect the USB cable. Try not to bend the pins as you push it in. Refer to the diagram below if necessary.

   REMEMBER, BE ***EXTREMELY*** CAREFUL WITH THE USB CONNECTION ON THE TEENSY. THEY BREAK ***VERY*** EASILY!!

2. Push the sensor into the breadboard such that neither of the leads (wires) are connected to anything else (each one is in it's own row). The orientation does not matter; both legs are the same.

   Always work with the Teensy unplugged from the computer. This will help ensure no "magic smoke" escapes.

3. Use a jumper wire to connect one side of the sensor to Vin/5V on the Teensy (side 1).
4. Connect the other side (side 2) of the sensor to a 10 kilo-ohm (k
   Ohm) resistor, and then connect the other side of the resistor to ground (GND on the Teensy). You may need to use a jumper wire to connect the resistor to GND.
5. Use a jumper wire to connect side 2 of the sensor to an analog input on the Teensy. In the example below, pin A0 is used.

3.2 Program the Teensy

3.2.1 Bare Minimum

1. Open the Arduino IDE and load a new file (if one doesn't open automatically).
2. Go to `File > Save As` to save the example under a new name in the default location (`Documents/Arduino`). Call it **groupName_voltage.ino**.

   *Remember, files that don't follow the naming convention **exactly** will incur point deductions.*

3. Create a `setup()` function and start serial communication within it:

```
    void setup() {
        Serial.begin(9600);  // start serial communication at a rate of 9600 bits per second (the stand
    }
```

4. Create a `loop()` function, read the sensor data, and print it to the serial port:

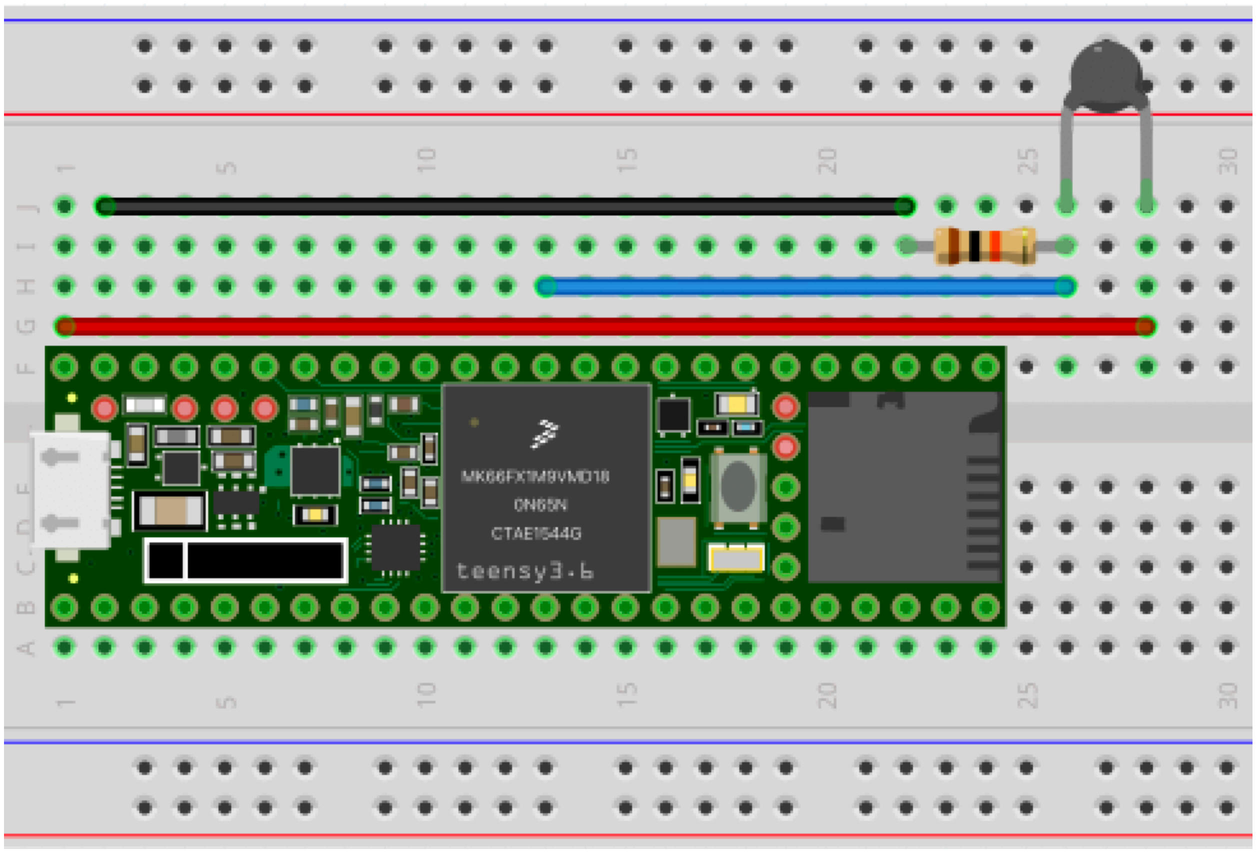


Figure 3.1: *Your final setup should look something like this picture. Red = 5V, black = GND, and blue = signal.*

```
    void loop() {
        value = analogRead(0); // read sensor
        Serial.println(value); // print result to serial port
        delay(1000);           // wait for 1 second
    }
```

5. Press the **Upload** button to send your sketch to the Teensy. Once you get it loaded, click the **Serial Monitor** button to see if the communication is working. You should see a number between 400 and 500 if you sensor is working correctly.

3.2.2 Converting to Temperature

The Teensy reports values in bits, which isn't very meaningful to humans. What we really care about is the temperature. The formula to get from bits to degrees C (for this sensor connected to a Teensy) is:

```
$$
    ^{\circ}C = 25 + (\text{value} - 512) / 11.3
$$
```

1. Add the above forumla to your code to convert to temperature. Also change the code to print the temperature to the serial port instead of the raw value.

```
    void loop() {
        value = analogRead(0); // read sensor
        temp  = ____________________________; // convert to temperature
        ______________________; // print the temperature to the serial port.
        delay(1000);             // wait for 1 second
    }
```

1. Upload your code and see if it works.

3.2.3 Cleaning up the Output

You can also print text to the serial port. When you use `Serial.print()` or `Serial.println()` to print text, it should always be in quote marks (this is technically called a *string*). For example, `Serial.println(Hello, world!)` would NOT work, but `Serial.println("Hello, world!")` would. To clean up you data output, try printing "*C" after the temperature to indicate the units.

1. Change `Serial.println(temp)` to `Serial.print(temp)`. These two functions are very similar, with one difference: `println` creates a new line when it's finished (like pressing enter on the keyboard) whereas `print` does not.
2. Add a line just below `Serial.print(temp)` to print the "*C" symbology:

```
    Serial.print(temp);
    Serial.println("*C");
```

3.2.4 Make Your Own Changes

1. Before you leave, add a line in your code to convert from degrees C to degrees F.
2. Once you've converted to F, print the results for both degrees C and degrees F. Your values should:
   - Be printed to the serial port *on the same line.*
   - Include the units "*C" and "*F" where needed.
   - Be separated from each other with a space (either " " or 's').

3.2.5 Turn in

1. A hardcopy of your completed Exercise 3 Worksheet.

2. An electronic copy of your final Voltage code (submit to Dropbox).

   Remember, files that don't follow the naming convention ***exactly*** *will incur point deductions.*

Chapter 4

Functions: Building Blocks for Programs

Objective: Create a `blink()` function in Arduino.

Introduction

In previous exercises discussed the two function, `setup()` and `loop()` that are necessary for every Arduino sketch. You've also (perhaps unknowingly) used pre-defined functions to achieve tasks. In programming, a **function** is a defined piece of reusable code that can be used to perform an action. You could think of functions as modules that can be stacked together to create a program. For example, the items `digitalWrite()` and `Serial.print()` are both functions. The first, as we've seen, controls the logic level of a digital I/O pin on the Teensy while the latter prints data to the serial port.

Note that functions all have a defined stucture, or *syntax*. This may vary from language to langages, but will be the same within any language. In Arduino (C), functions always take the form `function(argument_1, argument 2, ...);`, where `function()` is the function and `argument_1`, etc., are the entities the function acts on or uses in some way while executing the function. This is very similar to functions in mathmatics. Assume, for example:

$$f(x) = 3x$$

Here, we've defined a function, f, that acts on argument x. If we write $f(2)$, we are saying we wish to substitute 2 for x in the function, yielding $f(x) = 3 \times 2 = 6$.

We could just as easily do this in Arduino. Using the same example, we can define a function `f()`:

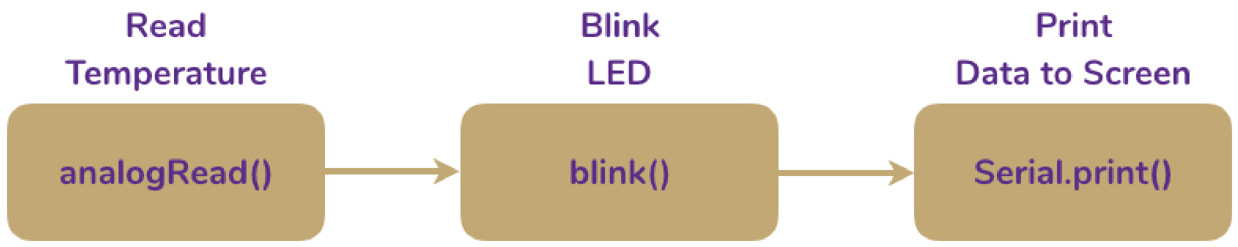
```
int f(int x) {
  return 3*x
}
```

This function is equivalent to the mathmatical expression above. First, we define a function `f()` that takes one argument, `x` (contained within the parentheses). Within the function (denoted by the curly braces), we ask Arduino to output, or **return**, $3 \times x$. After defining the function, we could write `f(2)` and Arduino would output `6`.

*What would Arduino output if you wrote `f(4)`?

4.0.1 Using Functions

Although most of the work done in this class will rely on pre-written functions, You'll explore the methods for writing your own function in this exercise to help you understand how they work. As mentioned above, you can think of function as modules or building blocks that can be combined to create desired outcome. For example, if you wished to create a program that flashed a light everytime the temperature was read and then displayed the result to the screen, you might wish to combine blocks for reading the temperature, flashing a light, and printing data to the computer screen:



Creating the **block diagrams** can be very helpful in visualizing what a program will do, and just the process of creating one can help you think through what the program needs to achieve line-by-line. The program above has one problem: it assumes there is a function called `blink()` that will flash the LED. This is not true, but you'll make one in this exercise. A list of built in functions can be found on the Arduino website; other functions specific to this class can be found in [Appendix C][Appendix C: Functions].

4.1 Writing a `blink()` Function

1. On your worksheet, create a diagram like the one above for a function that would blink and LED on and off. Your function should:

- Be called `blink()`.
- Turn the LED on for a set amount of time.
- Turn the LED off for a set amount of time.

2. Define your function in Arduino. Open a blank sketch by clicking `File > Examples > Bare Minimum`. Add your variable definitions to the top and set the appropriate pins as outputs in your `setup()` function using `pinMode`. Refer to your previous exercises if you can't remember how to do those things.

```
// define variables here

void setup() {
  // set pinMode() here
}

void loop() {
  // leave blank for now
}
```

3. Define your `blink()` function at the bottom of your sketch. Follow the format below (don't worry about the `void` part now – just know that you need to use it).

```
// define variables here

void setup() {
  // set pinMode() here
}

void loop() {
  // leave blank for now
}

void blink(int ledPin) {
  digitalWrite(ledPin, HIGH); // turn LED on
  ___________________________ // wait x milliseconds
  ___________________________ // turn LED off
  ___________________________ // wait x milliseconds
}
```

4.2 Using Functions

1. Add the function into your code and test it out.

```
// define variables here

void setup() {
  // set pinMode() here
}

void loop() {
  blink(________);
}

void blink(int ledPin) {
  digitalWrite(ledPin, HIGH); // turn LED on
  ___________________________ // wait x milliseconds
  ___________________________ // turn LED off
  ___________________________ // wait x milliseconds
}
```

4.3 Write Functions With Multiple Arguments

1. Create a second function called `blink2();`. It should do the same thing as blink but allow for a variable blinking time.

```
void blink(int ledPin, int interval) {
  // turn LED on
  // wait x milliseconds
  // turn LED off
  // wait x milliseconds
}
```

2. Add this to your code and use it with `blink()`.

4.4 Turn In

1. Turn in a hardcopy of your completed worksheet.
2. Turn in an electronic copy of your code containing the functions `blink();` and `blink2()` that shows use of both.

Chapter 5

Soldering and PCB Assembly

Objective: Assemble/solder air quality sensor printed circuit board.

5.1 Introduction

Up to this point, we've used solderless breadboards and jumper wires to connect things to the Teensy. Those provide quick, temporary connections that can be used for prototyping and learning. However, they can be prone to failure over time as the connections inadvertently come loose. In situations where a design won't change, metal solder joints provide a more reliable, permanent connection.

Although soldering seems difficult to many people, it's actually a simple process if you have the correct tools and know the right technique. Hand soldering has been done on an industrial scale by assembly technicians for decades, and modern electronics with very small components are usually soldered by machine. In short, the soldering process involves heating up two metal parts to be joined with a *soldering iron*, applying a compound called *flux* to the joint, and then melting a soft metal alloy (the *solder*) onto the joint. Once the solder cools (after a second or two), a permanent connection is made between the two components.

> Good soldering technique, contrary to popular belief, is not hard. In fact, it's incredibly easy!
>
> If you've got the right tools and now the very simple, basic techniques, good soldering is a piece of cake! Anyone can do it, straight up! -*Dave Jones, EEVBlog*

This exercise will walk you through the basics of soldering, and by the end you should have completed assembly of your air sensor printed circuit board (PCB). The following video tutorials may be helpful in knowing what to expect before you come to class. Please watch the beginner tutorial, and then watch the longer tutorials by the EEVBlog if you would like to learn more details.

- Beginner tutorial
- In depth tutorial @ EEVBlog:

    1. Tools
    2. Through-hole Soldering
    3. Surface-mount Soldering

- schematic diagrams

5.2 Soldering Basics

5.2.1 The Supplies

Good tools and supplies can really make or break a soldering project. The two most important things are the solder and the soldering iron, but there are a host of other supplies that can help out.

- **Solder** is traditionally made from a lead alloy, although many modern solders are lead free due to the toxic nature of lead. For most hobby applications, either option will work. Solder comes in many different sizes, but in general narrower solder (<0.5 mm diameter) is better for electronics work. Many solders come with rosin (basically pin sap) embedded in the core, which acts as *flux* and eliminates the need for pre-application of flux (described below).
- **Flux** is a special compound that removes oxidation from the surface of the metals to be joined. Many metal (including copper, which is what we are soldering) form oxide layers on the surface of the metal. Although these are often difficult to see, they are always there. They will prohibit the solder from *wetting* (flowing onto and adhereing to) the items to be joined. It is usually very difficult or impossible to solder a joing without flux, and joints made with it will eventually crack and fail. Many solders come with flux (rosin) embedded in the middle of the solder to act as a flux; with these solders no additional flux is necessary. Any residue left from flux can be cleaned up with isopropyl (rubbing) alcohol or special flux cleaner.
- A **Soldering iron** is a pen-shaped device with a fine metal tip that is heated, usually with an electric heating element. It's always best to invest in a temperature-controlled soldering iron and skip the cheaper ones – the proper temperature is important for good soldering. A good soldering iron should also have a removable/interchangeable tip and a safety stand to prevent fires.
- A **sponge** is just a regular kitchen sponge that is dampened with water and used to clean the tip of the soldering iron during use. Keeping the tip clean helps the solder melt easily. As an alternative, a **brass sponge** (think of a metal kitchen scrubbie) can be used.
- A **third hand** is just a metal stand with clips on it to hold your workpiece while you solder to it. Tape, sticky tack, and other items can be used as well.
- **Flush side cutters, wire cutters, and/or wire strippers** are useful for trimming leads after they've been soldered and stripping wires prior to soldering.
- **Isopropyl alcohol** or **flux remover** and a small brush are useful in removing stick residue left by the flux.
- **Solder wick** and a **solder sucker** are tools used to remove solder when necessary. (Even though it said solder is "permanent" above, it can be removed, albeit not always in an easy manner!)
- A **Fume extractor** collects the smoke released during soldering and prevents it from being breathed in by the solderer.

5.2.2 The Technique

The main thing to remember when soldering is to use heat and wettability to determine where the solder will end up. If you want solder on one part of a circuit board and not another, make sure you apply the heat and flux to that part to be soldered and no where else; the solder will folow dutifully.

Setup

1. Check for the recommended iron temperature for the solder you're using. If you're unsure, 350°C is a good starting point. If your part to be soldered is very large or you're soldering to a ground plane with a large thermal mass you made need to increase the temperature.
2. Turn on the soldering iron to the correct temperature.
3. Wet the sponge on the soldering iron stand.
4. Layout all the tools necessary.
5. Turn on the fume extractor.

Soldering

1. **Create a mechanical connection if possible.** If joining two wires, twist them together. If connecting a component to a circuit board, bend the leads down to hold in the place. The mechanical connection is not strictly necessary, but it will add stregnth to the solder joint.
2. **Stabilize the parts to be soldered.** It takes two hands to solder – one to hold the soldering iron and one to apply the solder. Make sure your workpiece is secure before you being soldering.
3. [**Apply flux (if not using rosin-core solder).** Apply a small amount only to the parts to be soldered. Remember, the solder will folow the flux.]
4. **Clean the tip of the iron.** The soldering iron tip will develop an oxide layer as it sits at high temperature. Give a one or two wipes across the wet sponge each time you pick it up.
5. **Tin the tip of the iron.** Just barely touch the tip of the iron
6. **Apply heat to the joint.** Try to maxmize contact between the tip of the iron and the joint to be soldered. Apply heat to the solder joint, not the solder. This ensures the joint is hot enough to flow the solder and helps create a strong joint.
7. **Apply solder to the joint.** Touch the end of the solder to the joint to be soldered. If the joint is hot enough the solder should flow immediately throughout the joint.
8. **Let it cool.** Remove the solder, then the iron, and let the joint cool for a second or two before moving it. Wait longer before touching the joint as it will be hot enough to burn you.
9. **Inspect the joint.** A properly soldered joint should have a shiny, domed appearance. An improperly soldered joint will appear dull and blobby. Remove solder if necessary and repeat until a good joint is achieved.

5.2.3 Safety

- Obviously, the soldering iron and anything it touches are hot. Don't touch the tip of the iron or any of the parts that have recently been soldered.
- Always be mindful of where you set the iron down. It's hot enough to start fires and/or melt work surfaces and power cords.
- Don't breath in the fumes. They mostly result from the flux. Use a fume extractor or a fume hood to keep the fumes out of your lungs!

5.3 Assembling the Board

You will add components to a printed circuit board (PCB) to build the air sensor. The PCB has internal connections that ensure all components will be connected correctly. The purple areas of the board are called *solder mask*. Solder strongly prefers to stick to metal and will (almost) not stick to the solder mask. Use this to your advantage!

1. Solder each component on one at a time. Remember to consider which ones you need to solder first and which ones last, as some components may make it hard to get the iron into areas of the PCB.
2. Each component is labelled on the board. Refer to the model at the front of the room if you're unsure where something goes.
3. Make sure your have the component in the right place and the right orientation before making the solder connection! You may not be able to remove it if you do it incorrectly!!

Appendix A: Function Library

blink()

Blinks the Teensy's onboard LED.

```
void blink() {
    digitalWriteFast(LED_BUILTIN, HIGH);
    delay(15);
    digitalWriteFast(LED_BUILTIN, LOW);
    delay(15);
}
```

checkConnection()

Checks serial connection to PM sensor and connection to SD card and returns 5 and 2 LED blinks, respectively, if successful.

```
void checkConnection() {
  if (my_status == 1){
    for (int i = 1; i < 6; i++){
      blink();
    }

    delay(3000);
  }
  else{
    digitalWrite(LED_BUILTIN, HIGH);
    delay(5000);
    digitalWrite(LED_BUILTIN, LOW);
  }

   if (!SD.begin(chipSelect)) {
    return;
  }
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
        // don't do anything more:
        while (1) {
            blink();
            blink();
        }
    }
    for (int i = 1; i < 3; i++){
      blink();
    }
}
```

createFileName(int y, int m, int d)

Creates file name based on the current date.

```
String createFileName(int y, int m, int d) {
  return String(fillDigits(y)) + String(fillDigits(m)) + String(fillDigits(d)) + deviceID + ".txt";
}
```

fillDigits(int digits)

Adds a leading zero to a number if < 10.

```
String fillDigits(int digits) {
  if (digits < 10) {
    String digitsout = "0" + String(digits);
    return digitsout;
  }
  else return digits;
}
```

getTeensy3Time()

This function comes from the RTC example. It syncs the RTC time on the Teensy.

```
time_t getTeensy3Time() {
  return Teensy3Clock.get();
}
```

performPMReading()

Reads PM_2.5 data from sensor and returns it as a float.

```
float performPMReading() {
    float pavg = 0.0;
    float psum = 0.0;

    for (int i = 1; i < 21; i++) {
      my_status = my_hpm.read(&p25, &p10);
      if (i > 15) {          // throw away first points while it warms up
          psum = psum + p25;
          pavg = (psum)/(float(i)-15.0);
          }
          delay(1000);
    }
    return pavg;
}
```

printData()

Returns data string in standard format based on sensor ID string and measurements values (floats).

```
String printData(String ID, float t, float h, float p, float g, float pm25) {
  return String(ID) + '\t' + printDateTime(printDate(year(), month(), day()), printTime(hour(), minute(
}
```

printDate(int y, int m, int d)

Prints date string in standard YYYY-MM-DD format given Y, M, D integers.

```
String printDate(int y, int m, int d) {
  return String(y) + "-" + fillDigits(m) + "-" + fillDigits(d);
}
```

printDateTime(String d, String t)

Concatenates date (`printDate()`) and time (`printTime()`) strings as ‘YYYY-MM-DD HH:MM:SS’.

```
String printDateTime(String d, String t) {
  return d + ' ' + t;
}
```

printTime(int h, int m, int s)

Returns time string as HH:MM:SS based on integer H, M, S inputs.

```
String printTime(int h, int m, int s) {
  return fillDigits(h) + ":" + fillDigits(m) + ":" + fillDigits(s);
}
```

setBMESamplingParameters()

Sets the sampling and filtering parameters for the BME sensor.

```
void setBMESamplingParameters() {
  if (!bme.begin(0x76)) {
    return;
  }
  // Set up oversampling and filter initialization
  bme.setTemperatureOversampling(BME680_OS_8X);
  bme.setHumidityOversampling(BME680_OS_2X);
  bme.setPressureOversampling(BME680_OS_4X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
}
```

turnSensorsOff()

Turns sensors off for hibernate.

```
void turnSensorsOff() {
  my_status = my_hpm.stop_measurement();
  Serial1.end();
  delay(500);
  digitalWrite(33, LOW);
}
```

turnSensorsOn()

Turns on the sensors and sets the RTC time. Can be used to wake from hibernate or start from off.

```
void turnSensorsOn() {
  digitalWrite(33, HIGH); // turn on relay
  delay(5000); // wait to stabilize
  setSyncProvider(getTeensy3Time);

  Serial1.begin(9600);
  while (!Serial1) {
    ; // wait for serial port to connect. Needed for native USB
  }
  Serial1.flush();
  my_status = my_hpm.stop_autosend();
  delay(500);
  my_status = my_hpm.start_measurement();
  delay(500);
}
```

writeFile(char filename[16])

Writes the measurement file to the SD card.

```
void writeFile(char filename[16], String dataString) {
    File dataFile = SD.open(filename, FILE_WRITE);
    if (dataFile) {
        dataFile.println(dataString);
        delay(500);
        dataFile.close();
     }
     else return;
}
```

Appendix B: Strategies for Troubleshooting

Appendix C: Glossary and Abbreviations

Current (i)

The flow of electical energy, measured in **amperes (A)** or **amps** for short. By convention, current flows from the positive side of a circuit to the negative side. [Note that the actual charge carriers, electrons, flow from the negative side the positive side!]

Electricity

A form of energy resulting from the accumulation or movement of charged particles, typically electrons. Practically speaking, electricity is a form of energy that we use to generate heat, light, and movement.

LED

Light emitting diode; a semi-conductor designed to emit light when a current is applied.

Power (P)

The rate of work, measured in **Watts (W)**. Practically speaking, power is the rate at which a circuit converts electrical energy to another form of energy (such as light).

$$P = Vi$$

Resistance (R)

The resistance to current flow, measured in **ohms** (Ω).

$$R = \frac{V}{i}$$

Voltage (V)

The potential energy difference between two points in a circuit measured in **volts (V)**. A larger voltage indicates a stronger "push" to drive electric current.