SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

ARDUINO KIT USER MANUAL

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY
MECHANICAL ENGINEERING DEPARTMENT

Student Developers: Mr. Walelign Nikshi & Mr. Aaron Bost
Faculty Advisors: Dr. Mark Bedillion & Dr. Karim Muci-Küchler

VERSION 5.5 (OCTOBER 2017)

PREFACE

Welcome to the user manual for the Arduino kit. This manual introduces the hardware included in the kit and provides the reader with a very brief overview on how to interface and use devices such as LEDs, buttons, temperature sensors, light sensors, etc. with the Arduino Uno. It also shows how to use Arduino Uno compatible shields to do tasks such as driving motors and servos or receiving input from a PS3 controller. A Grove-Base Shield is used to avoid any soldering or bread-boarding. Therefore, the required sensors are already wired up and are "*plug-n-play*".

This manual does not assume that the reader has much familiarity with the Arduino hardware or with the Arduino IDE (Integrated Development Environment), which is the programming environment for Arduino. However, the reader is referred to the many books and web sites that are available to learn about the Arduino Uno and the Arduino IDE since this document is not intended to be a substitute for those references.

Finally, there is a companion PowerPoint presentation that instructors that adopt the kit can use in class and serves as a supplement for some of the topics discussed in this document.

ACKNOWLEDGMENTS

This document was prepared as part of a project supported by the Office of Naval Research under Award No. N00014-15-1-2419. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research or the U.S. Government.

CONTENTS

CHAPTER 1
HARDWARE COMPONENTS

HARDWARE COMPONENTS

OBJECTIVES

- Introduce the Components of the Arduino Kit

1.1 General Information About the Hardware Kit

There are different brands and vendors for Arduino. In this manual, we will consider the Arduino Uno Rev3 (see Figure 1-1). This is a microcontroller board based on the ATmega328P, an 8-bit microcontroller with 32KB of Flash memory and 2KB of SRAM. The board is compatible with many shields, which enables quick and easy project prototyping. It can interact with real-world sensors, control motors, display information, and perform near-instantaneous calculations. Code development occurs on a different system (a laptop or desktop computer for example) and is then uploaded to the Arduino through a USB cable.

Figure 1-1. Arduino UNO Rev3.

1.1.1 Components

The electronic components corresponding to the kit are listed below. These are the minimum components that are required. You can add more sensors and actuators as needed.

Microcontroller

- Arduino UNO Rev3

Shields

- Grove Base Shield (comes with the Grove Starter Kit)
- Adafruit Motor/Stepper/Servo Shield
- Adafruit 16-Channel 12-bit PWM/Servo Shield
- SparkFun USB Host Shield

Grove Starter Kit

- Grove Base Shield
- Grove LCD RGB Backlight
- Grove Smart Relay
- Grove Buzzer
- Grove Sound Sensor
- Grove Touch Sensor
- Grove Rotary Angle Sensor
- Grove Temperature Sensor
- Grove LED Module
- Grove Light Sensor
- Grove Button
- Blue LED
- Green LED
- Red LED
- Mini Servo
- Grove Cables (Qty: 10)
- 9V Battery to Barrel Jack Adapter
- Grove Starter Kit Manual

Other Components

- USB Mini-B Cable - 6 Foot
- USB A to B Cable - 6 Foot
- Sony Dual Shock 3, PS3 controller
- Bluetooth 4.0 USB Module
- Arduino Stackable Header Kit
- Arduino Uno Plastic Enclosure
- 3x4 Right Angle Male Header
- Battery pack (12V) with on/off switch: 8AA batteries, bare ends
- Battery pack (12V) with on/off switch: 8AA batteries, barrel jack plug (5.5mm / 2.1mm)
- Battery pack (6V) with on/off switch: 4AA batteries, bare ends

The following figures show the different components of the Arduino kit.

Microcontroller

Figure 1-2. Arduino UNO Rev3.

Shields

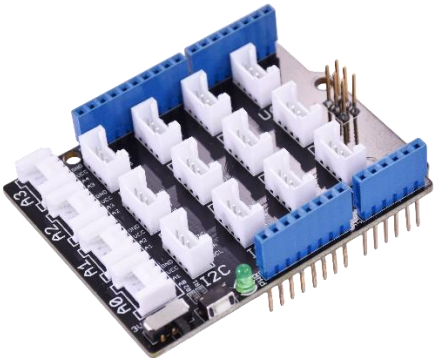


Figure 1-3. Grove Base Shield.

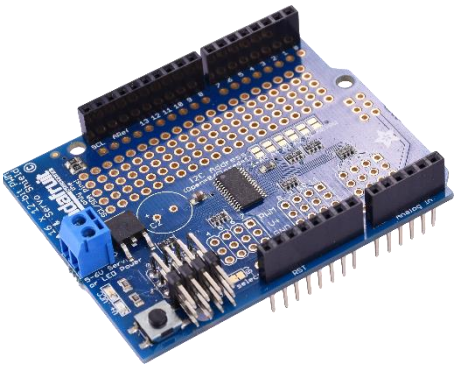


Figure 1-4. Adafruit 16-Channel 12-bit PWM/Servo Shield.

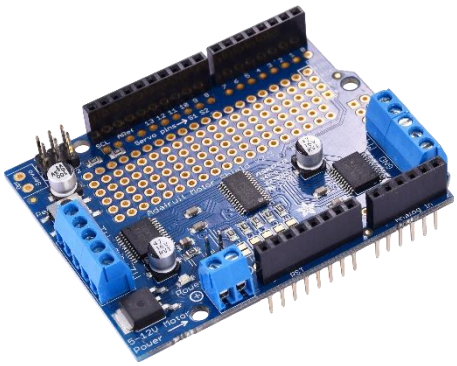


Figure 1-5. Adafruit Motor/Stepper/Servo Shield.

Figure 1-6. SparkFun USB Host Shield.

Grove Starter Kit

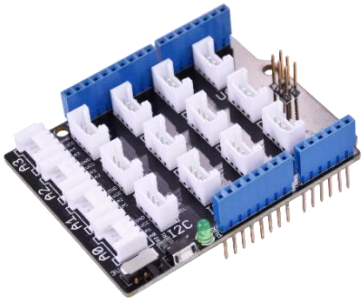


Figure 1-7. Grove Base Shield.

Figure 1-8. LCD RGB Backlight.

Figure 1-9 Smart Relay.

Figure 1-10. Buzzer.

Figure 1-11. Sound Sensor.

Figure 1-12. Touch Sensor.

Figure 1-13. Rotary Angle Sensor.

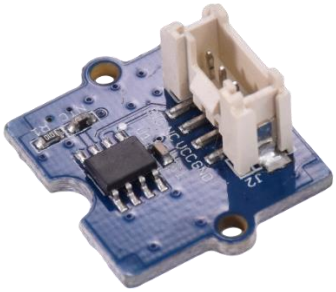


Figure 1-14. Temperature Sensor.

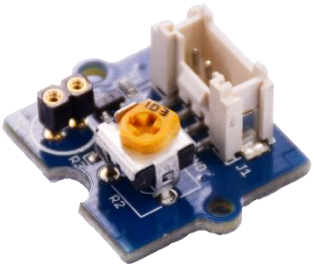


Figure 1-15. LED Module.

Figure 1-16. Light Sensor.

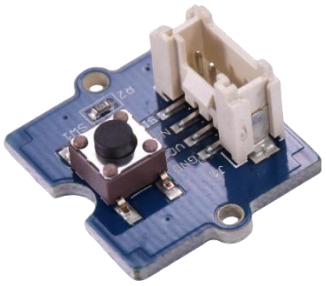


Figure 1-17. Button.

Figure 1-18. Cables.

Figure 1-19. LEDs.

Figure 1-20. 9V Battery Adapter.

Figure 1-21. Mini Servo Motor.

Other Components

Figure 1-22. Bluetooth Module.

Figure 1-23. Sony Dual Shock 3, PS3 controller.

Figure 1-24. USB Cable A to B - 6 Foot.

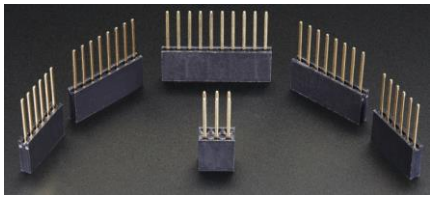


Figure 1-25. Shield stacking headers for Arduino (R3 Compatible).

Figure 1-26. 3 x 4 Right Angle Header.

Figure 1-27. 8 x AA battery holder with 5.5mm/2.1mm Plug and On/Off Switch.

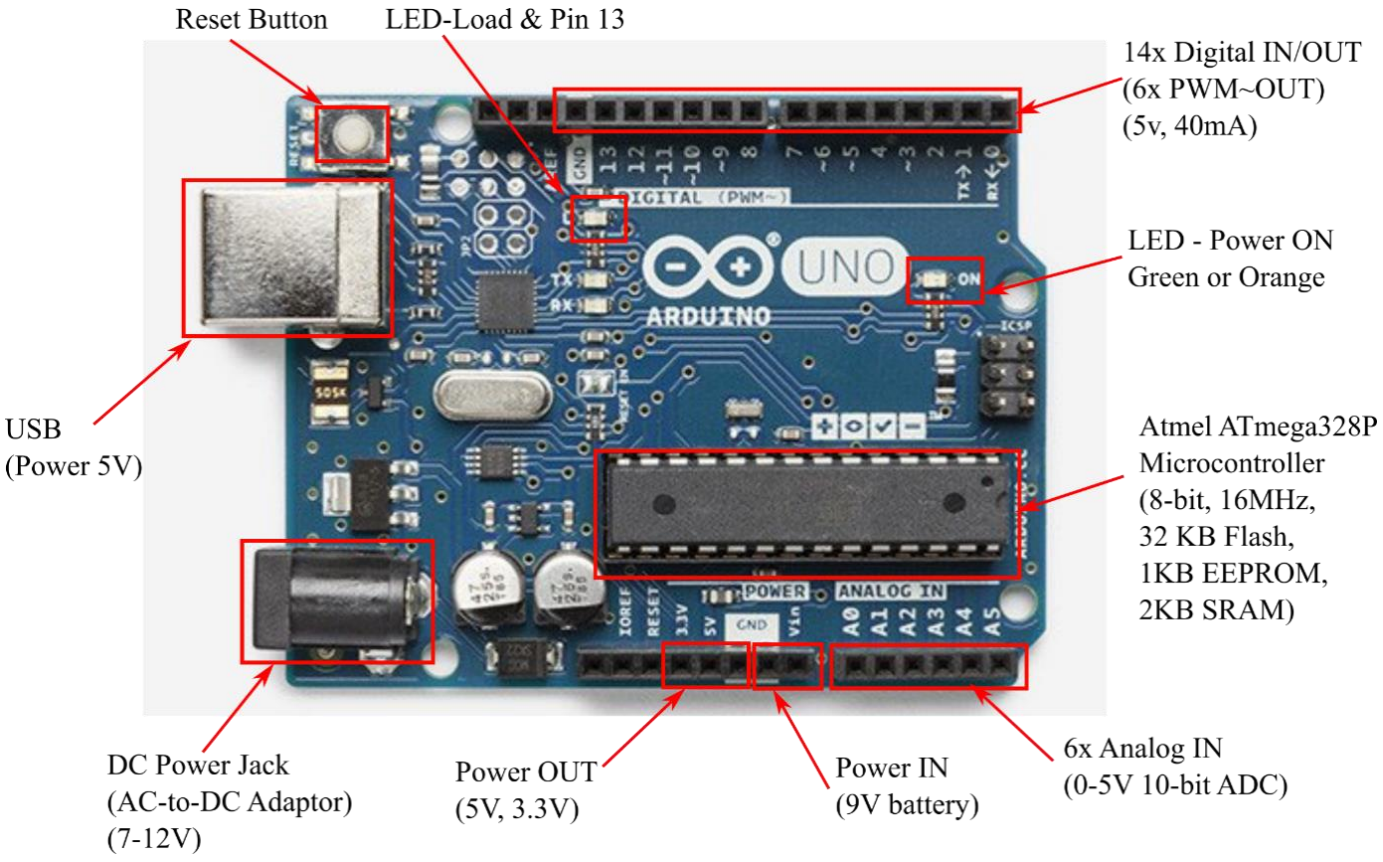
Figure 1-28. Holder 8 AA Cell W/Cover and switch.

Figure 1-29. Battery Holder 4xAA with Cover and Switch.

Figure 1-30. Arduino Uno Enclosure – Plastic.

Appendix B provides a list of the components mentioned in this chapter indicating a possible supplier for each one as well as its estimated unit cost. Some assembly/soldering is needed to get the Adafruit Motor/Stepper/Servo Shield, the Adafruit 16-Channel 12-bit PWM/Servo Shield, and the SparkFun USB Host Shield ready for use with the Arduino UNO. Appendix C shows the assembly/soldering steps required for each of those shields.

CHAPTER 2
GETTING STARTED



GETTING STARTED

OBJECTIVES

- Provide a general overview of the Arduino Uno
- Learn how to power the Arduino Uno
- Understand how to upload code to the Arduino

2.1 Overview of the Arduino UNO Rev3

The Arduino UNO Rev3 is a microcontroller based on the Atmega328P, an 8-bit microcontroller with 32KB of Flash memory and 2KB of SRAM. The Arduino UNO Rev3, which is referred as the **Arduino** from now on, is an open-source hardware platform used to learn about electronics and the basics of sensors and actuators, to do hands-on projects, and to quickly build prototypes. It can be programmed over a USB A to B cable using the Arduino IDE software installed on a PC. Plug in the board to the PC using the USB cable, select "Arduino UNO" in the "**Tools → Board**" menu of the Arduino IDE software, and you are ready to upload code to the Arduino. As shown in Figure 2-1, the Arduino has 14 Digital I/O pins with 6 PWM pins, 6 Analog Inputs, UART, SPI, and external interrupts.

The **digital pins** are the digital inputs and outputs of the Arduino. These are where you connect items such as buttons, LEDs, sensors, and others to interface the Arduino with other pieces of hardware. Pins marked with a tilde (~) (which are also called PWM pins) can be used to approximate an analog output, which you can use to dim LEDs or run servo motors or DC motor drivers.

There are six **analog inputs** on the analog header. These pins have analog-to-digital converters (ADC), which can be used to read in an analog voltage between 0 and 5V. These are useful if you need to read the output of analog sensors. All six analog pins can also serve as digital inputs and outputs.

The **power header** consists of voltage supply pins. These pins are used as power sources for other pieces of hardware (like LEDs, potentiometers, and other circuits). The '3.3V' and '5V' pins are regulated 3.3V and 5V voltage sources. The 'GND' pins are the common ground. 'VIN' is the input voltage; it will equal the voltage of your input supply if you have a wall adapter connected. If nothing is connected to the barrel jack, and you are powering the board via USB, 'VIN' should be around 5V.

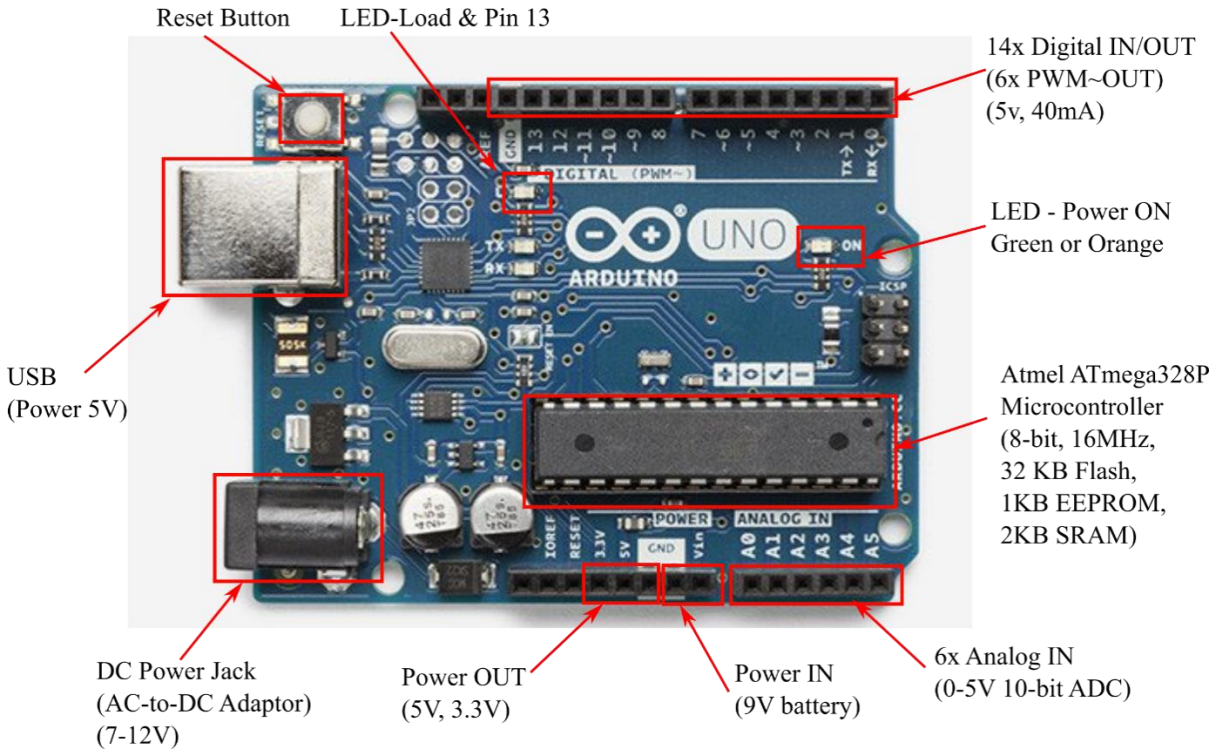


Figure 2-1. Arduino UNO Rev3 overview.

The Arduino can be powered via either the **USB** or **barrel jack** connectors. If you choose to power it via USB, the other end of the USB cable can be connected to either a computer as shown in Figure 2-2 or a 5V regulated USB wall charger. USB is usually the easiest way to power the board, especially when you are programming it, because the USB interface with the PC is required for uploading the code. However, its limited power can become a problem for some applications like driving a motor.

The power jack in the Arduino accepts a center-positive barrel jack connector with an outer diameter of 5.5mm and inner diameter of 2.1mm. 9V and 12V adapters are good choices if you are looking to power the Arduino through the barrel jack connector. Any wall adapter connected to this jack should supply a **DC voltage** between **7V and 12V**. You can also use a battery pack as shown in Figure 2-3.

Figure 2-2. Powering the Arduino using a USB cable.

Figure 2-3. Powering the Arduino using a barrel jack, 12V 8AA battery pack.

You need to use the barrel jack when you need more power than the USB port can supply. A USB port is usually only allowed to supply 500mA; if you need more than that, the barrel jack is the only choice. We can connect both a barrel jack and a USB connector at the same time as shown in Figure 2-4; the Arduino has power control circuitry to automatically select the best power source.

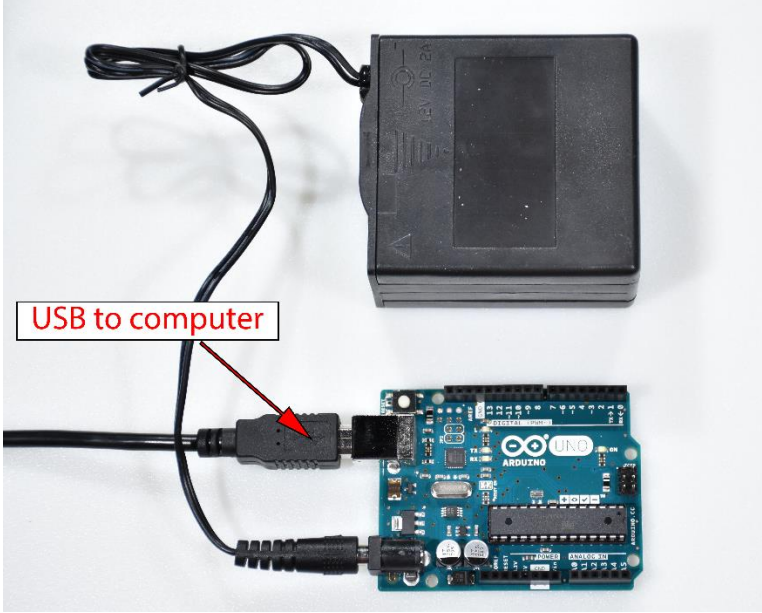


Figure 2-4. Powering the Arduino using both a USB cable and a battery pack.

2.2 Installing the Arduino IDE Software

Before using the Arduino, you need to install the Arduino IDE software on your PC (laptop or desktop). You also need to install some drivers for proper functionality of the board. The Arduino IDE can be installed in computers running any of the following operating systems: Windows, Mac, and Linux.

2.2.1 Windows 10, 8, 7, Vista, and XP

Before you plug in the Arduino to your PC, you need to install the Arduino IDE.

- Download the latest version of Arduino **installer** from: https://www.arduino.cc/en/Main/Software.
- Run the executable file to begin the installation.
- Follow the instructions provided and install all the required drivers.

When plugging the Arduino to your PC for the first time, your computer will apply the compatible drivers. You should see a notification like the one shown in Figure 2-5.

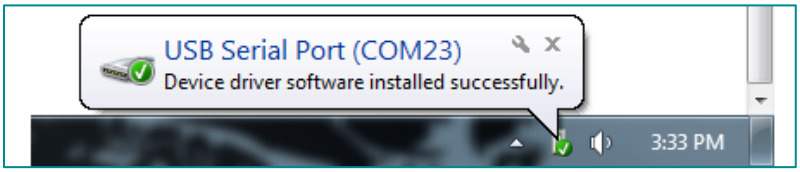


Figure 2-5. USB Serial Port Notice.

2.2.2 Mac

Before you plug in the Arduino to your PC, you need to install the Arduino IDE.

- Download the software from https://www.arduino.cc/en/Main/Software which is only offered in a .zip file format.
- Double-click the **.zip** file to unzip it and you will see an "**Arduino.app**" file.
- Copy **the Arduino application** into your applications folder to complete installation. For details see https://www.arduino.cc/en/Main/Software.

2.2.3 Linux

Before you plug in the Arduino to your PC, you need to install the Arduino IDE. There are many different distribution "flavors" of Linux. Installing Arduino is slightly different for many of these distributions and some already have the required software installed. Follow the procedure given in https://www.arduino.cc/en/Main/Software.

2.3 Uploading Code to the Arduino

2.3.1 Arduino IDE Software Overview

This section will familiarize you with the Arduino IDE software. When you run the Arduino software, the window will look like Figure 2-6.

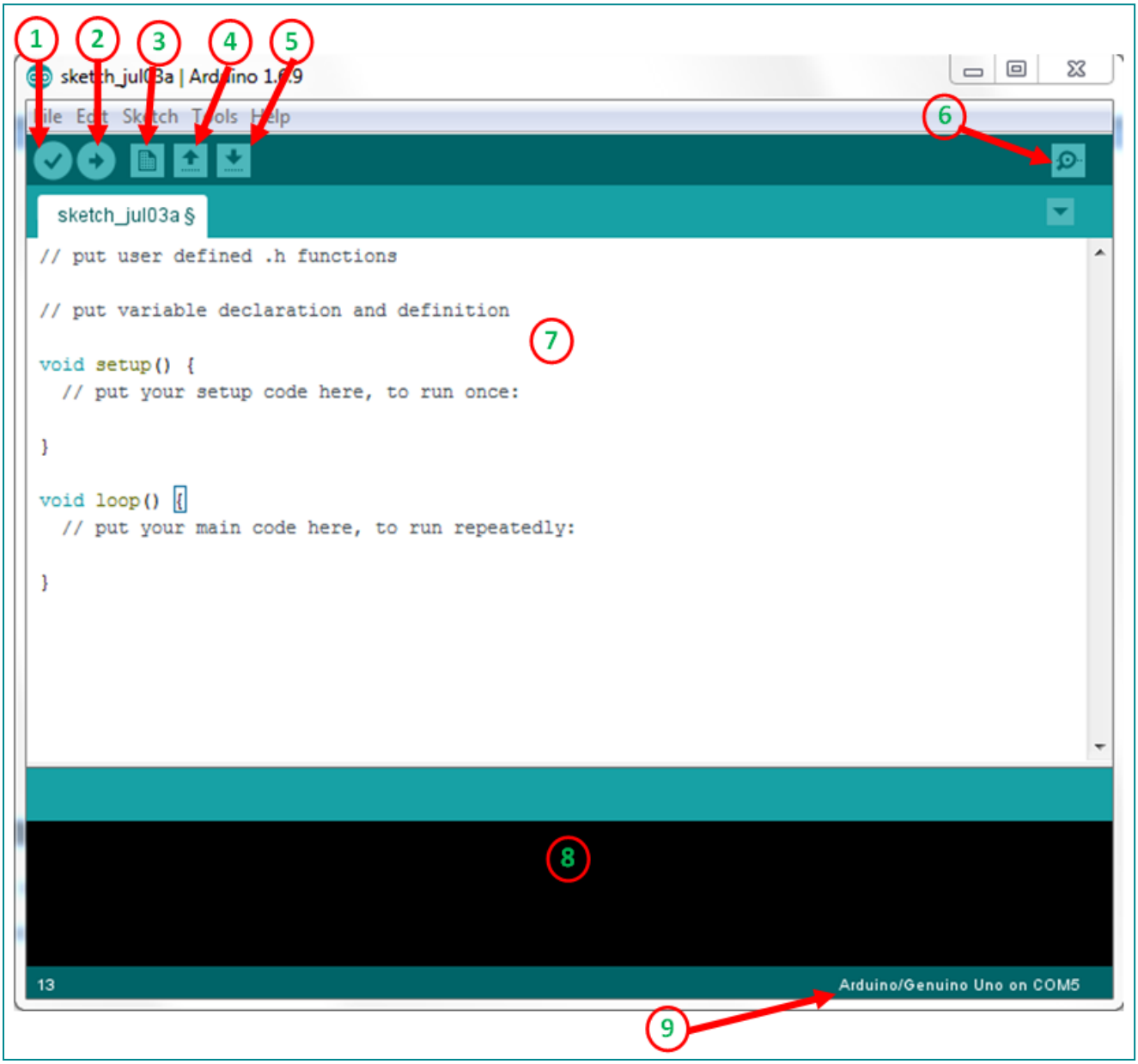


Figure 2-6. Arduino software overview.

The elements highlighted in Figure 2-6 correspond to the following:

1. **Verify** – Compiles and checks your code for errors.
2. **Upload** - Compiles your code and uploads it to the configured board.
3. **New** - Creates a new sketch.
4. **Open** – Opens an existing sketch.
5. **Save** - Saves your sketch.
6. **Serial Monitor** - Opens the Serial Monitor.
7. **Main Code Area** - Area where you write your code.
8. **Console** - Displays the information about the status of the code.
9. **Board and Serial Port Selections** - Displays the Arduino board and COM port in use.

2.3.2 Selecting a Board

Since many Arduino boards can be used, you need to tell the Arduino IDE which one you are using. Go to **Tools** on the menu bar, then place the mouse cursor over the **Board** menu option and select **Arduino/Genuino UNO** (see Figure 2-7).

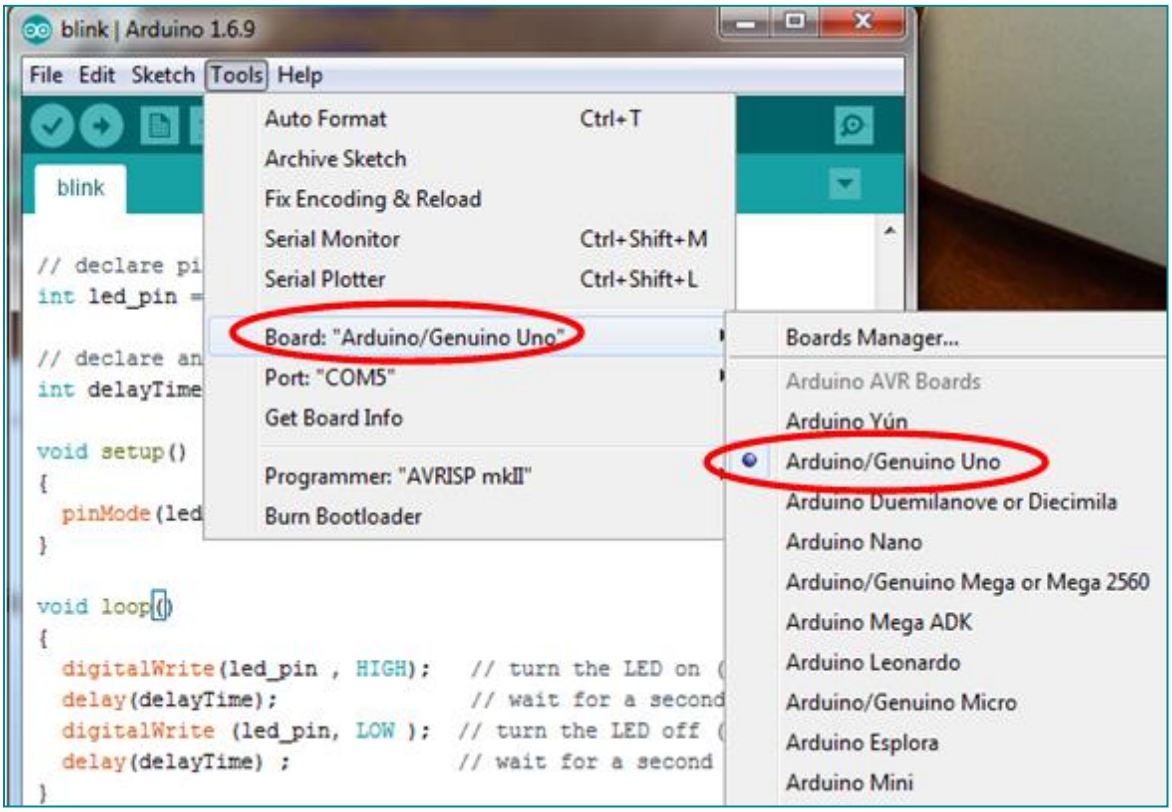


Figure 2-7. Selecting the Arduino board that will be used.

2.3.3 Selecting a Serial Port

Since your computer may have multiple COM ports, you need to tell the Arduino IDE which of your computer's serial ports the Arduino is connected to. Go to **Tools**, then place the mouse cursor over **Serial Port** and select your Arduino's COM port (see Figure 2-8). If you are not sure which of the serial ports is your Arduino, unplug it for a moment and check the menu to see which one disappears from the list.

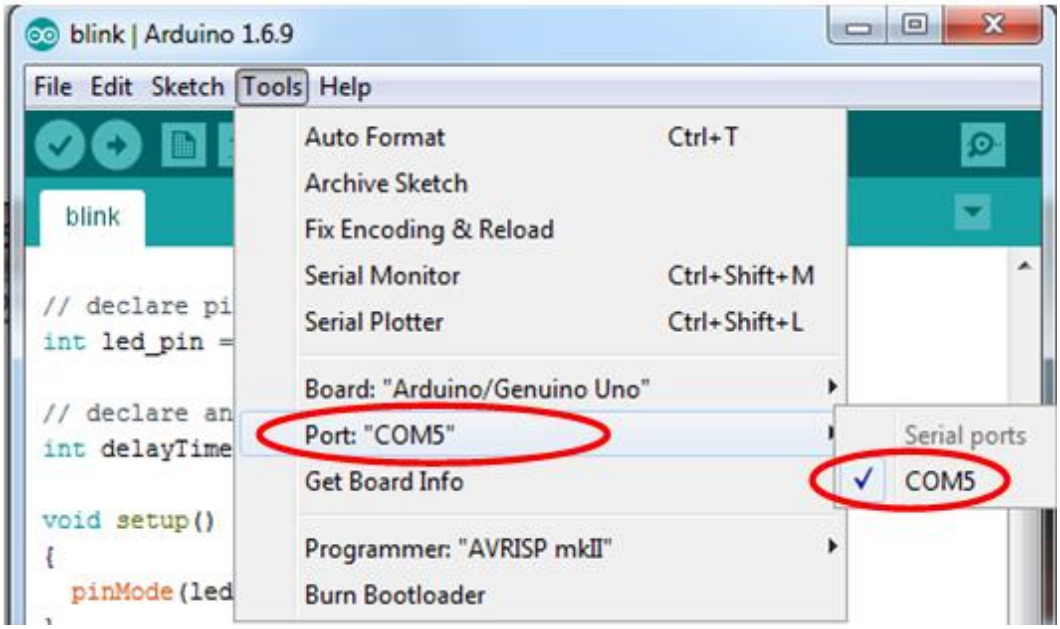


Figure 2-8. Selecting the COM port where the Arduino is connected.

2.3.4 Uploading the Code to the Arduino

When you are ready to upload the code, click the **Upload** button (#2 in Figure 2-6) and allow the IDE some time to compile and upload your code. When the code has uploaded correctly, you should see something like the message in Figure 2-9 in your console window.

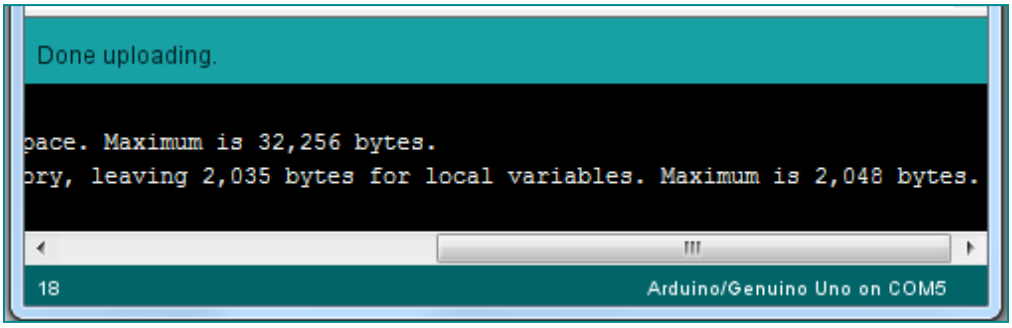


Figure 2-9. Message indicating that the code was successfully uploaded to the Arduino.

If you see an error message in the console window, it means something went wrong. There are some things you can check such as making sure that you selected the correct board and COM port.

2.4 Using the Serial Monitor

This section will familiarize you with the serial monitor using an example. Follow the steps in Example 2.1 to see if your serial port is working properly.

EXAMPLE 2.1

Serial Monitor Test

1. Connect the Arduino to your PC as shown in Figure 2-10.

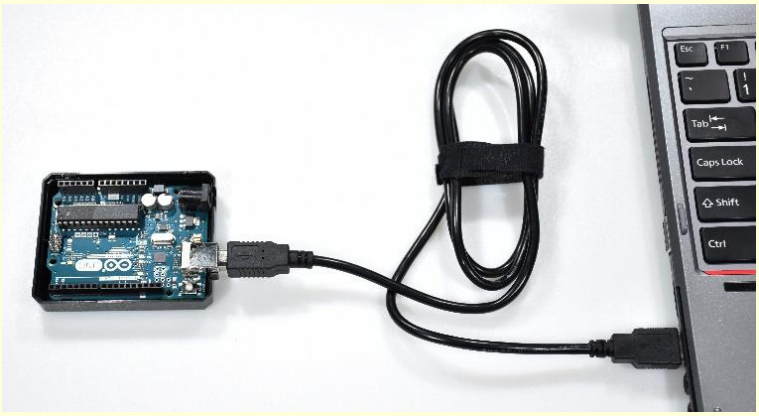


Figure 2-10. Test example for the Serial Monitor.

2. Open the Arduino IDE software and delete everything on the default sketch.
3. Copy the code presented below and paste it in the code area of the Arduino IDE.
4. Save the code as ***SerialPortTest.ino***.
5. Make sure that the appropriate Arduino board is selected.
6. Make sure that the correct COM port is selected.
7. Upload the code to the Arduino.
8. Open the Serial Monitor indicated by #6 in Figure 2-6, and make sure that baud rate matches the one highlighted in yellow in the code below.

```
/*
SerialPortTest.ino
*/
/* Serial Port Test code*/
void setup()
      {
        /* put your setup code here, to run once: */
        Serial.begin(115200);
        Serial.println("Serial port is open");
```

```
        }

void loop()
        {
        /* put your main code here, to run repeatedly: */
        Serial.println("Serial port Test");
        delay(500);
        }
```

If you see the requested messages in the Serial Monitor (see Figure 2-11), the serial communication was successful.

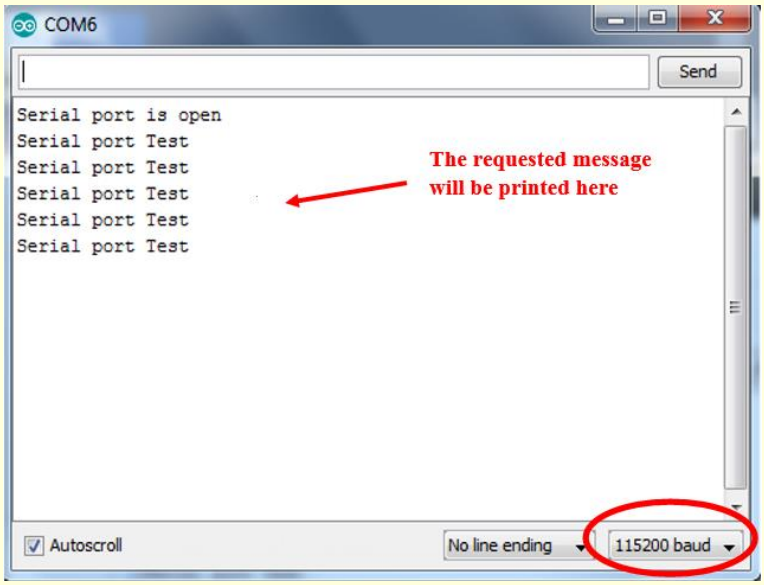


Figure 2-11 Serial Monitor display message.

2.5 Using the Arduino Built-In LED

The following example uses the Arduino built-in LED that is connected to digital pin 13. Although the code is very simple, it is useful in that it demonstrates that the Arduino is properly connected to the PC.

EXAMPLE 2.2

Blink the built-in Arduino LED

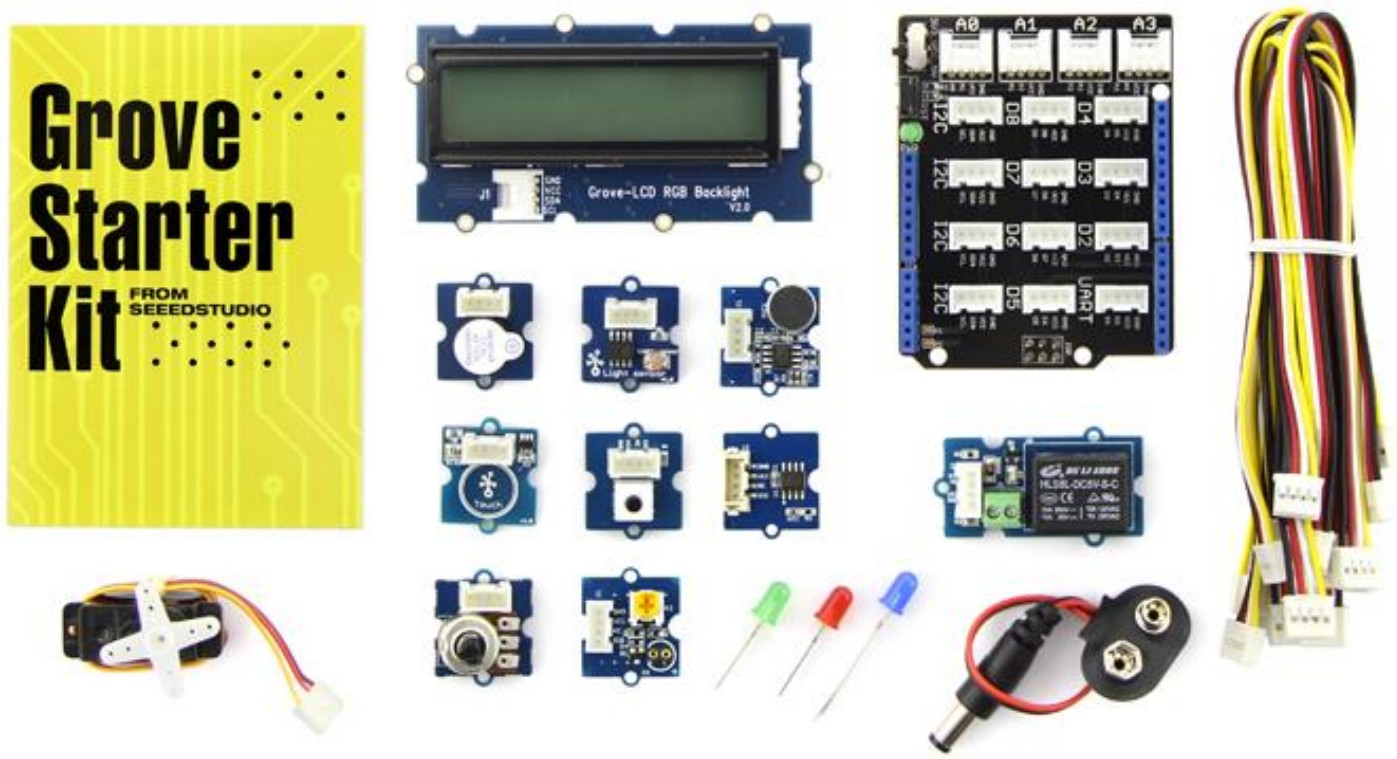
This example will blink the built-in LED that is connected to digital pin 13. The LED will blink every half a second and the number of blinks will be displayed in the Serial Monitor. For the time being, don't worry about the specific commands that are used in the code. You will become acquainted with them in the next chapter. Instead of copying and pasting the code in the Arduino IDE, try typing it.

```
int counter = 0;
int pin= 13;

void setup()
{
      // put your setup code here, to run once:
      Serial.begin(9600);
      pinMode(pin,OUTPUT);
      digitalWrite(pin,LOW);
}

void loop()
{
      // put your main code here, to run repeatedly:
      counter = counter + 1;
      digitalWrite(pin,HIGH);
      Serial.print("Blink #");
      Serial.println(counter);
      delay(500);
      digitalWrite(pin,LOW);
      delay(500);
}
```

CHAPTER 3
INTRODUCTION TO THE GROVE BASE SHIELD



Grove Starter Kit for Arduino

INTRODUCTION TO THE GROVE BASE SHIELD

OBJECTIVES

- Become acquainted with the Grove Starter Kit
- Learn about digital and analog signals
- Understand the concept of Pulse Width Modulation (PWM)

3.1 Introduction

A quick search on the Internet reveals that using a microcontroller, such as the Arduino UNO, is only limited by the level of knowledge of the person using it. Coding for microcontrollers can range from simple to very complex. A good way to start learning how to write code for microcontrollers is to select a "friendly" hardware platform, begin with very simple programs and progressively increase the level of difficulty.

3.2 Grove Base Shield Overview

The Arduino Uno board has headers that bring out all the analog and digital I/O of the microcontroller. There are many interface boards, known as shields, ready to use with the Arduino. One of those interface boards is the ***Grove Base Shield***. The Grove base shield is designed to simplify the process of learning how to use an Arduino microcontroller. Grove is a modular, standardized connector prototyping system. Grove takes a building block approach to assembling electronics. Compared to the jumper or solder based systems, it is easier to connect, experiment and build. The Grove Base Shield simply connects the header pins to standard polarized connectors for easy interfacing to sensors and actuators. There are 16 Grove ports on the base shield, which can be divided into three different functional areas: digital ports (8), analog ports (4), and I2C (4).

As shown in Figure 3-1, there are 8 digital ports (one is labeled UART and the other seven are labeled D2 to D8) that are connected to digital pins 0/1 through 8/9 on the Arduino Uno. They are used when reading a digital sensor that only outputs 0 or 1 (5V) or when writing to a digital device. The digital pins 0 and 1 are reserved for UART (an integrated circuit used for serial communication over a computer or peripheral device serial port). On the left-hand-side there are four Grove ports (labeled A0

to A3) for taking analog readings. Analog sensors can return readings ranging from 0 to 1023. Below the Grove digital ports there are four I2C Grove ports (labeled I2C). I2C is a low-speed bus protocol that transfers data via two wires: **SCL** and **SDA**. SCL is the clock line that synchronizes data transfer over the I2C bus, and the SDA is the data line. For details about this shield go to the following link: http://www.seeedstudio.com/wiki/Base_shield_v2.

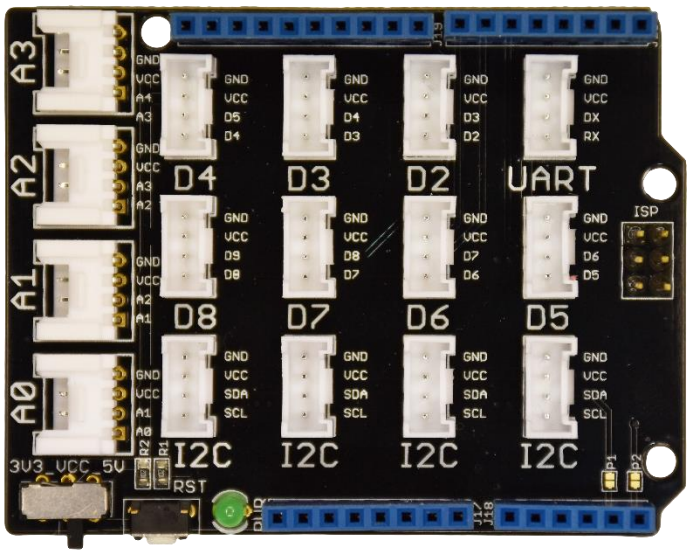


Figure 3-1. Grove Base Shield.

Each of the Grove connectors has four lines: normally two signal lines, power and ground. Sensors normally need at least one signal line and the power and ground lines. This system eliminates the need for external wiring. The lowest digital pin number listed in each Grove plug/socket is the one that is connected to the sensor. For example, if you want to connect the sensor to digital pin 2, you should connect it to D2 in the Grove Base Shield. Similarly, digital pin 3 is connected to D3 in the Grove Base Shield**.**

3.3 Adding the Grove Starter Kit Sketchbook and Library to the Arduino IDE

The Grove Starter Kit comes with its own documentation and with code examples for the sensors, actuators, and the LCD included in the kit. A small manual is included with the Grove Starter Kit and you can find additional documentation at http://wiki.seeed.cc/Grove_Starter_Kit_v3/.

In this section, we will show how to add the sketchbook and library that comes with the Grove Starter Kit to the Arduino IDE and how to access the example code available for each component. In contrast

to the normal way of adding Libraries to the Arduino IDE (i.e., download a Library → put it into the Arduino libraries directory → open an example and use it), we directly use a sketchbook provided by the supplier (Seeed Studio) to avoid the risk of compilation errors due to missing libraries. You can find out how to use the sketchbook that is available for the Grove Starter Kit at the following web site: http://wiki.seeedstudio.com/wiki/How_To_Use_Sketchbook.

Follow the steps below to add the sketchbook and library to your IDE and use the example code.

1. Download the sketchbook **Sketchbook_Starter_Kit_V2.0** from https://github.com/Seeed-Studio/Sketchbook_Starter_Kit_V2.0 and unzip the folder. The sketchbook is actually a folder and you can put it anywhere you like. Let's put the sketchbook folder in the "**Documents**" folder associated with your user name.
2. Open the Arduino IDE and click **File** on the menu bar.
3. Click **Preferences** and a pop-up screen will open (see Figure 3-2).

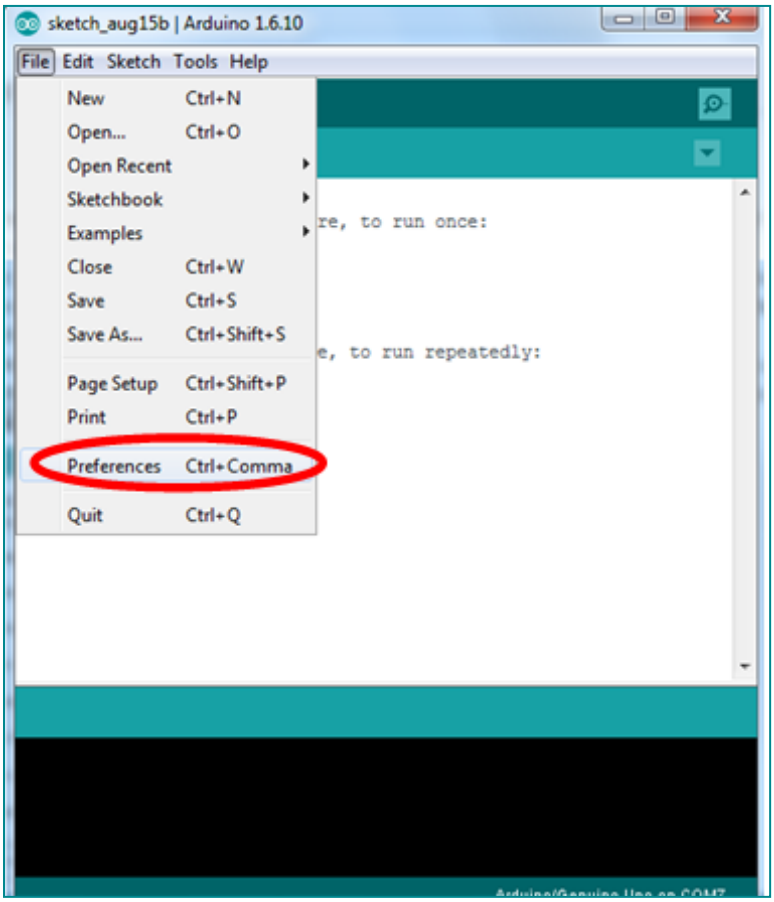


Figure 3-2. Selecting the sketchbook provided with the Grove Starter Kit as the default sketchbook.

4. Click **Browse**, locate the sketchbook folder you saved earlier, and click **OK** (see Figure 3-3).

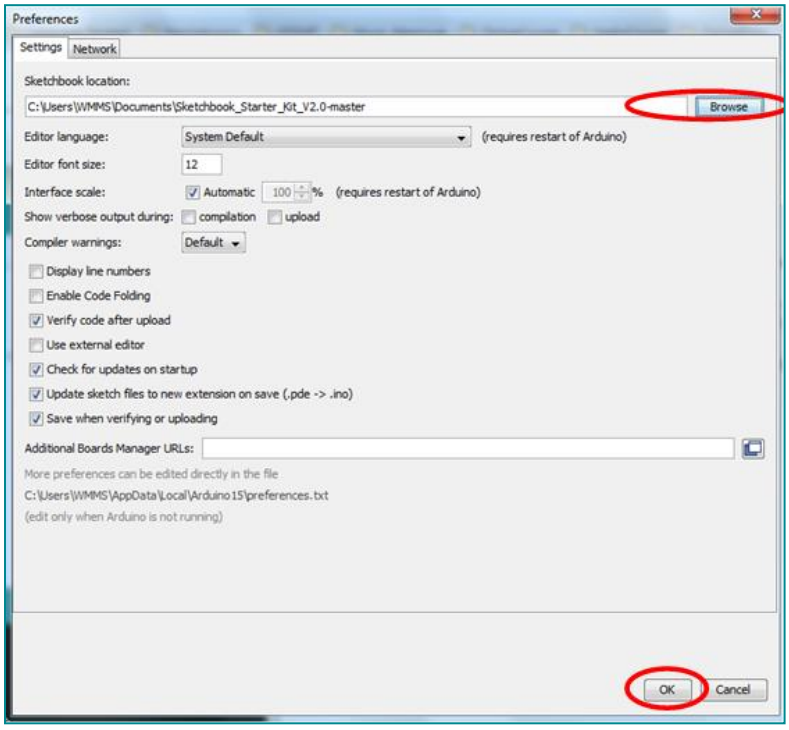


Figure 3-3. Locate the Sketchbook_Starter_Kit_V2.0-master folder.

5. Now the sketchbook is ready to use. To access the examples, click **File** on the menu bar and place the mouse cursor over the **Sketchbook** menu option to see all the available examples (see Figure 3-4**).**

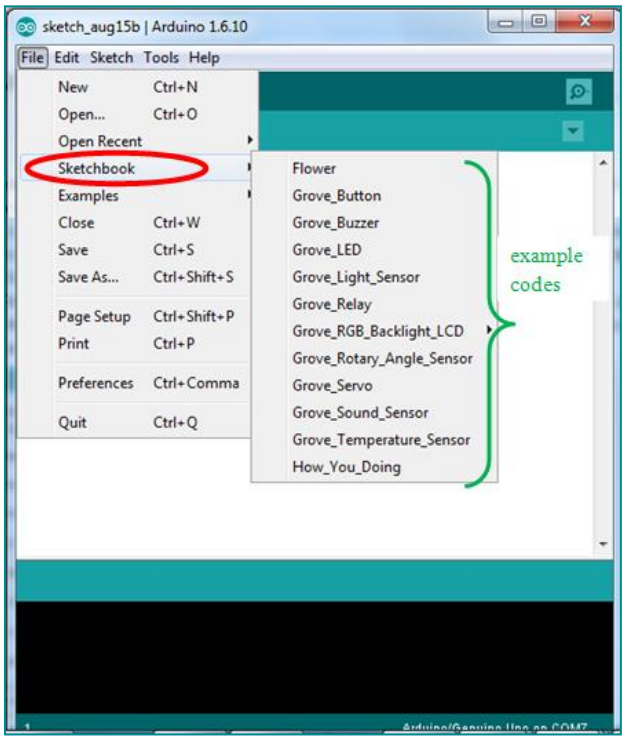


Figure 3-4. Access the example code for the components in the Grove Base Shield.

3.4 Installing the Grove Base Shield

Installing shields can be a little bit tricky because there are many delicate pins that must be connected at the same time to the Arduino headers. Incorrect installation of shields to the Arduino microcontroller can lead to bent or broken pins, damaged boards, and annoying frustrations with components not working properly.

Before we begin installing the Grove Base Shield:

- Double check that all the headers/pins are properly attached to the shield. It is possible that you could get a version of the Grove Base Shield that does not have the headers/pins attached. In that case, please refer to "APPENDIX C - Required Hardware Preparations" for instructions on how to attach the headers/pins.

- Also, now is a good time to check that all the pins are straight. If you see that any of the pins are not straight (see example in Figure 3-5 – the picture corresponds to a different shield), straighten them out before installing the shield. The best way to straighten a pin is to carefully use needle nose pliers to bend it back to a straight position.

Figure 3-5. Watch out for bent pins, like the one pictured above, before installing shields.

- Check to make sure that all the pins are present. It is easy to break a pin and not notice the missing pin when installing the shield. This, of course, will lead to components not working properly.

To install the shield, first align all the pins into the headers, as shown in Figure 3-6.

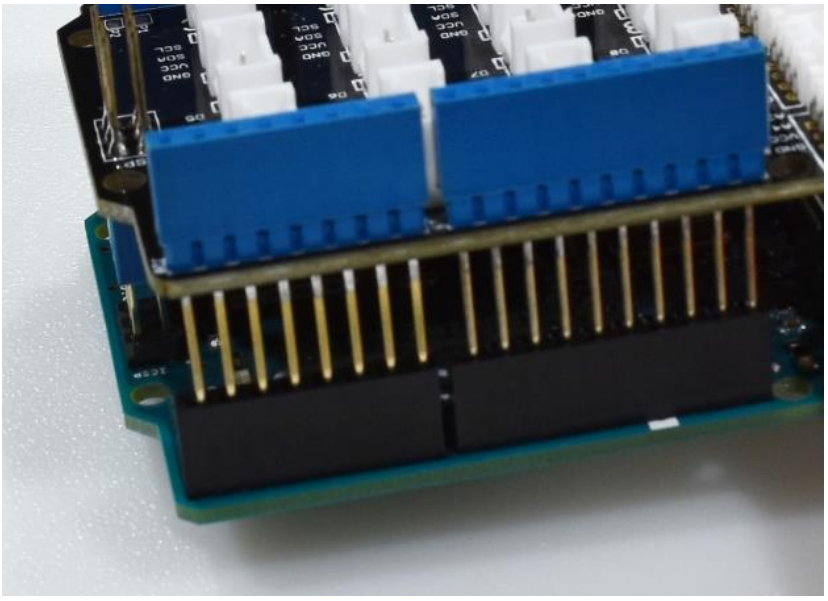


Figure 3-6. Aligning the pins before pushing them into the headers.

Next, gently push on both sides until the shield is inserted into the headers, as shown in Figure 3-7. It is crucial that the pins in the shield slide into the headers evenly because if the shield is not pushed down evenly the pins could bend.

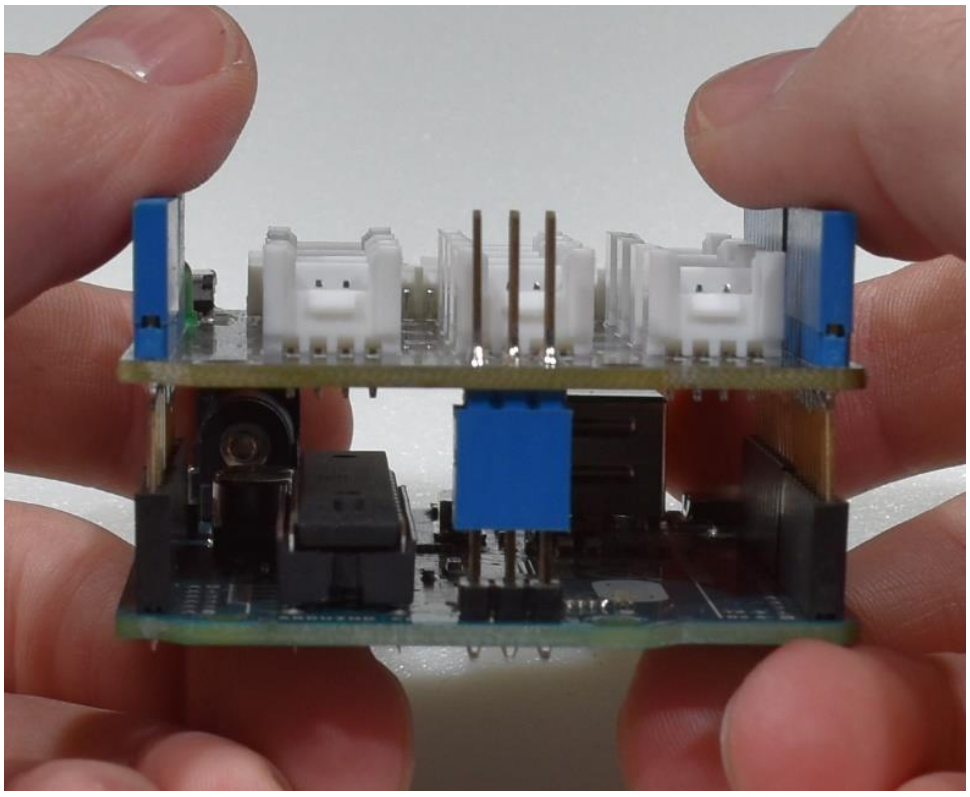


Figure 3-7. Pushing the pins in the shield into the headers.

3.5 Digital Input and Output

A digital signal is a signal that represents a sequence of discrete values. A logical signal is a digital signal with only two possible values. The Arduino represents the two discrete possible values as 0 or 1 ("LOW" or "HIGH"), which correspond to 0 and 5 V, respectively. The Arduino has 14 digital pins which can be configured as either inputs or outputs. They are used when reading digital inputs such as push buttons that only provide values of 0 or 1 ("LOW" or "HIGH"), and outputting a digital value such as turning on or off an actuator or LED. Two examples are used to illustrate the two concepts: turning on and off (blinking) an LED, and using a push button as a digital input to turn on or off an LED.

EXAMPLE 3.1

Digital Output: Blinking LED

In this example, we will demonstrate how to use the Grove LED module with the Grove Base Shield and the Arduino UNO. The program presented below will have an LED flash on and off every second. To connect the LED to the Arduino and flash it on and off, follow the steps below.

1. Make sure that the voltage selector in the Grove Base Shield is set to 5V. (Note: the shield has a switch next to the "A0" socket that can be used to select either 3.3V or 5V).
2. Prepare the LED module for use. The longer "leg" of the LED needs to be connected to the "+" side. Also, the module has a variable resistor (it has a yellow "cap") that can be adjusted with a Phillips screwdriver. The variable resistor controls the amount of current that passes through the LED and, consequently, its brightness when it is turned on.
3. Connect one end of a Grove cable (the cable is polarized. i.e. it will go into the socket in only one way) to the Grove LED Module, as shown by #1 in Figure 3-8.
4. Connect the other end of the Grove cable to the socket labeled D2 (which is the second digital pin) on the Grove Base Shield, as shown by #2 in Figure 3-8.
5. Connect the Arduino board to the computer via a USB cable.
6. Open the Arduino software, delete everything on the default sketch, copy the code provided on the next page (the code is located below Figure 3-8), and save the sketch as ***blink.ino***.

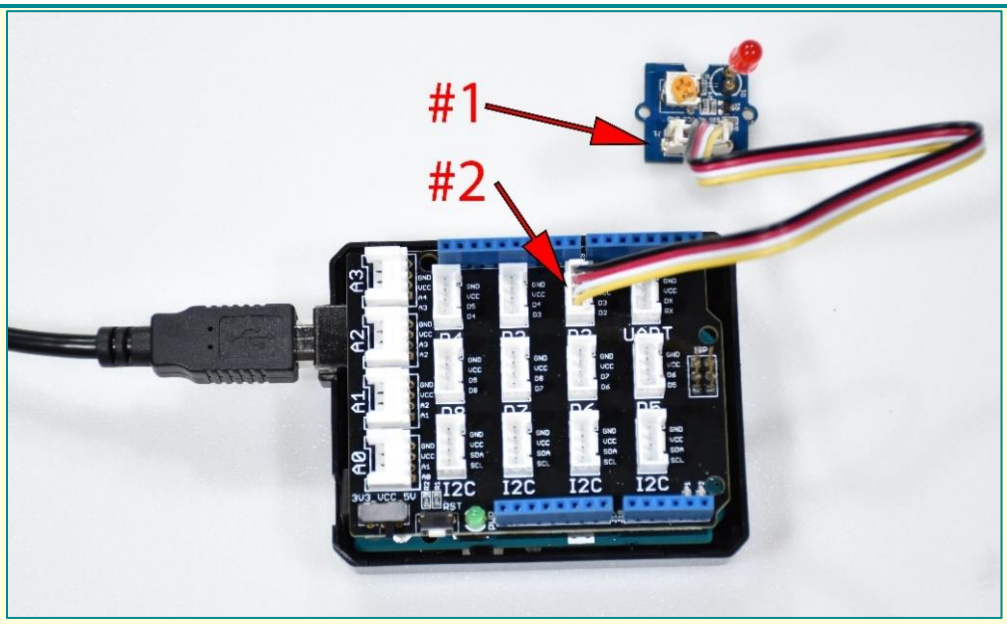


Figure 3-8. Connecting the LED module to the Arduino.

```
/* blink.ino */

/* declare pin numbers */
int led_pin= 2;
/* declare and initialize variables*/
int delayTime= 1000;  /* 1s = 1000milliseconds*/

void setup()
      {
        pinMode(led_pin,OUTPUT);/* Set the led_pin as output*/
      }

void loop()
      {
        digitalWrite(led_pin,HIGH);/* turn the LED on (HIGH) */
        delay(delayTime);          /* wait for a second*/
        digitalWrite(led_pin,LOW );/* turn the LED off (LOW) */
        delay(delayTime);          /* wait for a second*/
      }
```

7. Make sure that the Arduino UNO board is selected**.**
8. Make sure that the correct COM port is selected.
9. Upload the code to the Arduino. The console message should not show any error messages when the upload process is successful (see Figure 3-9).

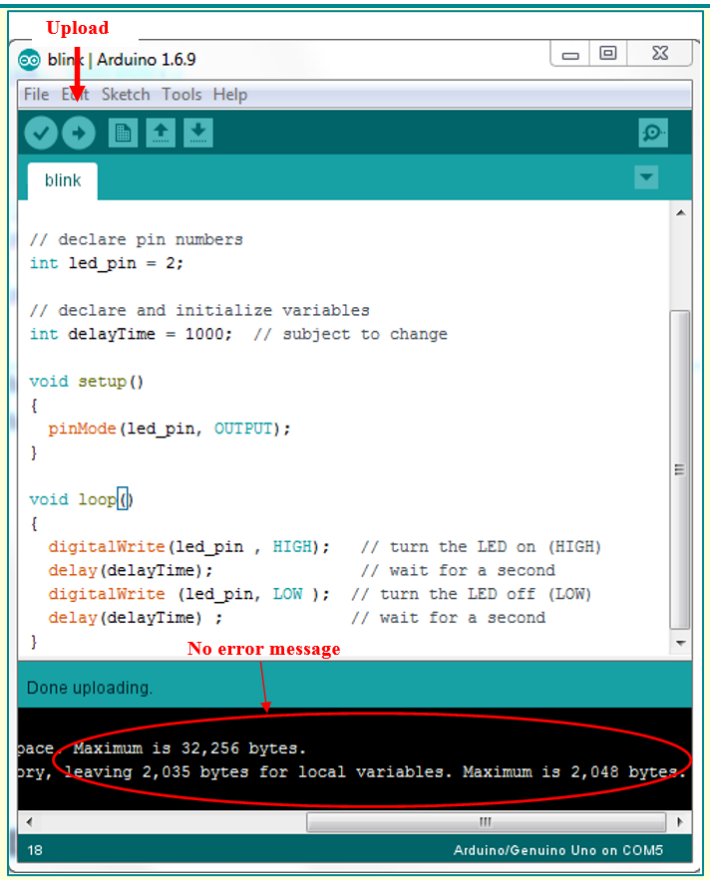


Figure 3-9. Uploading the code to the Arduino.

EXAMPLE 3.1 DISCUSSION

You should see the LED blinking on and off every 1 second. You can change the delay time to whatever time you want the LED to be on and off. The argument of the delay() function is time in milliseconds (1 second = 1000 milliseconds). You have now successfully completed the task of making an LED blink! While this may not seem like much, it indicates that your hardware and software environments are working properly. You are now ready to take on bigger challenges.

EXAMPLE 3.2

Digital Input: Push Button

A push button is a component that connects two points in a circuit when you press it. The push button used in this case is normally open which means that when you press the button, it will close the circuit. Follow the steps below to connect the Grove push button and the Grove LED module to the Grove Base Shield, and to turn on the LED when you press the button.

1. Connect one end of a Grove cable to the Grove push button module and the other end to the socket labeled D3 as shown in Figure 3-10.
2. Connect one end of a Grove cable to the LED module and the other end to the socket labeled D7 as shown in Figure 3-10.

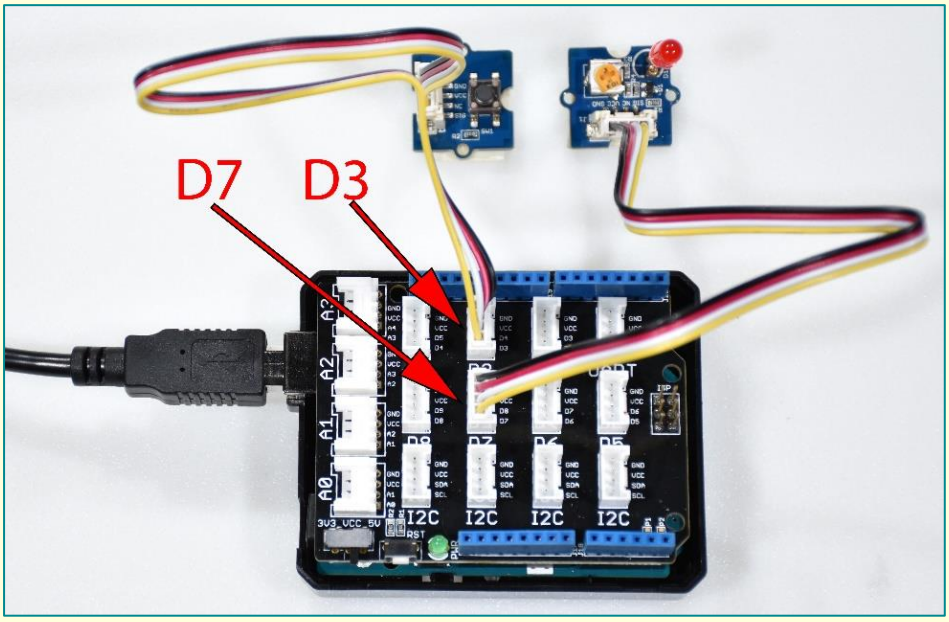


Figure 3-10. Connecting the push button module to the LED module.

3. Copy and paste the code below to a new sketch in the Arduino IDE and save the sketch as ***button.ino***.

```
/*
button.ino
Define the pins to which the button and LED are connected.
*/
const int pinButton = 3;
const int pinLed    = 7;

void setup()
{
    /* Configure the button's pin for input signals. */
    pinMode(pinButton,INPUT);
    /* Configure the LED's pin for output. */
    pinMode(pinLed,OUTPUT);
}

void loop()
{
    if(digitalRead(pinButton))
    {
        /* When the button is pressed, turn the LED on. */
        digitalWrite(pinLed,HIGH);
    }
    else
    {
```

```
        /* Otherwise, turn the LED off. */
        digitalWrite(pinLed,LOW);
    }
    delay(10);
}
```

4. Upload the code to the Arduino.
5. Press the button and observe the LED turn on. Release the button and observe the LED turn off.

EXAMPLE 3.2 DISCUSSION

The code is written in such a way that whenever you press the push button the circuit will close and the LED will be turned on. Otherwise the LED will be turned off. The button is read using a digital pin and can have the value of "HIGH" ("True" or 1 – switch closed) or "LOW" ("False" or 0 – switch open). Then we write the corresponding value to the digital pin to which the LED is connected. Read the code to note the relationship between the "digital read" of the push button and the "digital write" to the LED.

SUGGESTED PRACTICE

EXERCISE 3.1: Traffic light control

Use three LED modules to mimic a traffic light. Turn the green LED on for 5 seconds, followed by the blue LED for 2 seconds, and finally the red LED for 5 seconds.

EXERCISE 3.2: Use the digital input to control the states of the digital outputs

Use a push button to control the state of two LEDs. When the button is pressed turn the red LED ON and the green LED OFF. When the button is not pressed, the green LED should be ON and the red LED should be OFF.

3.6 PWM (“Analog”) Output

In the digital world, we can only work with ones and zeros (“high” and “low” values). However, in real life most things are “analog” which means that they can take a range of values. Components like motors can use an analog signal to control some parameter like rotational speed, but the Arduino doesn’t have pins providing an analog output. Instead, it uses Pulse Width Modulation (PWM) to create a digital approximation to an analog voltage. For a detailed discussion on PWM see Appendix D. Digital pins 3, 5, 6, 9, 10, and 11, can be configured for PWM output.

EXAMPLE 3.3

PWM Output: Change the brightness of an LED

The brightness of an LED can be modified by changing the PWM duty cycle. In this example, we will send a PWM signal to digital pin 6.

1. Connect the Grove LED module to socket D6 with a Grove cable (see Figure 3-11).

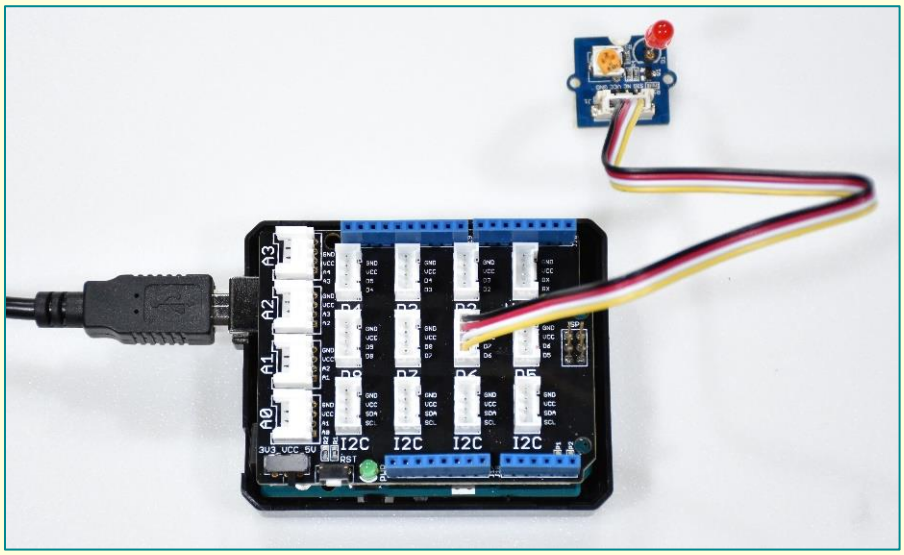


Figure 3-11. LED module connected to the PWM D6 socket.

2. Copy and paste the code below to a new sketch in the Arduino IDE and save the sketch as ***pwm.ino***.
3. Upload the code to the Arduino.

```
/* pwm.ino
Generate a PWM signal to pin 6
*/

/* Initialize data types and variable values*/
```

```
int pwmPin= 6;
int MAX_PWM_LEVEL = 255; /* value can change between 0-255*/
int value = 0;

void setup()
{
/* Configure the serial communication line at 115200 bauds (bits
per second.) */
      Serial.begin(115200);
      Serial.print("PWM value set for pin,");
      Serial.println(pwmPin);
}
void loop()
{
      value += 50;
      if (value > MAX_PWM_LEVEL)
      {
            value = 0;
      }
      analogWrite(pwmPin, value);
      Serial.println(value);     /* print PWM*/
      delay(500); /* delay half a second*/
}
```

4. Open the Serial Monitor and observe that the PWM value is changing by 50 every half a second. After the value is equal to 250, it is reset to 0.

   Note: A PWM value of 0 corresponds to 0V and a value of 255 corresponds to 5V.

EXAMPLE 3.3 DISCUSSION

In this case the PWM value to the LED changes by 50 units every half a second. The LED will be dim at first and then it will gradually get brighter. At the PWM value of 250 the LED will be the brightest and then it will reset to zero making the LED dim again. To test your skills, you can modify the code to make the dimming be gradual as well!

SUGGESTED PRACTICE

EXERCISE 3.3: Change the intensity of two LED’s

Change the intensity of two LED’s using PWM. Use the red and green LEDs. Change the intensity of the red LED from dim to bright and the intensity of the green LED from bright to dim at the same time. Observe the effect of using smaller and larger step values for changing the PWM value.

3.7 Analog Input

An analog signal is any continuous signal for which the time varying feature (variable) of the signal is a representation of some other time varying physical quantity, i.e. analogous to another time varying signal. Analog sensors can return readings ranging from 0 to 1023. Compared to digital sensors, analog readings are more detailed and precise.

EXAMPLE 3.4

Analog Input: Temperature Sensor

Temperature sensing is a common application in embedded systems. The temperature sensor included in the Grove Starter Kit is a thermistor that can be easily used to measure ambient temperature. This program will read the analog value provided by the temperature sensor included in the Grove Starter Kit, convert it to a temperature value in degrees Celsius using appropriate equations, and print the temperature value to the Serial Monitor. Follow the steps below to learn how to use the temperature sensor.

1. Connect the temperature sensor to the analog pin A0 as shown in Figure 3-12.

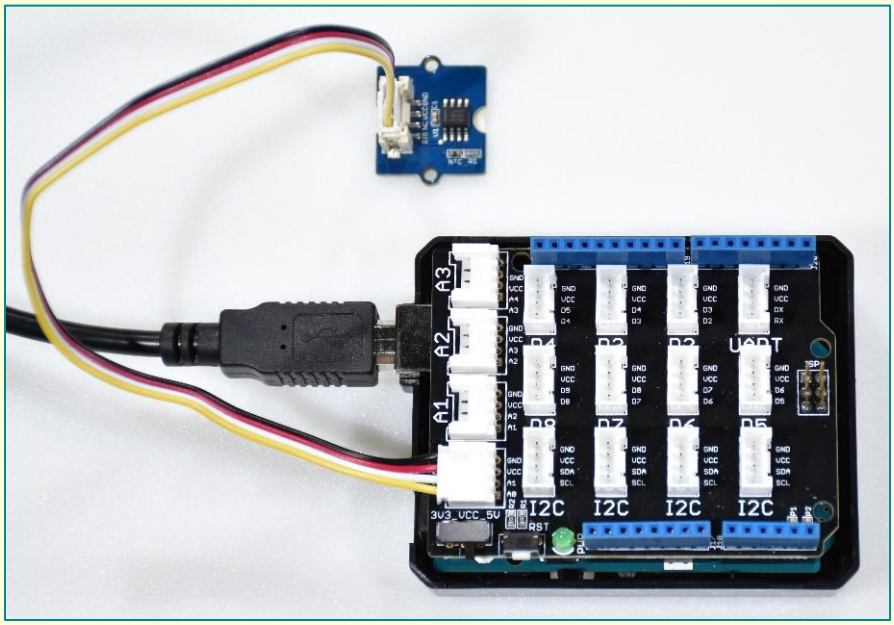


Figure 3-12. Grove temperature sensor connected to the analog pin A0.

2. We need to define the "B value" of the thermistor in the code so that the Arduino can report the correct temperature. The specifications listed in the product web site give a B

value range of 4,250K to 4,299K. Any value within that range can be used, but a good rule of thumb is to use the central value, which in this case is about 4,275K.

Notes:

- The datasheet/specifications for the temperature sensor can be found at: http://wiki.seeed.cc/Grove-Temperature_Sensor_V1.2/
- To learn more about what B values are and how they are calculated visit the web site: http://www.electronics-tutorials.ws/io/thermistors.html

3. Copy and paste the code below in the Arduino IDE and save it as ***temp_sensor.ino***.

```
/* temp_sensor.ino */

/* Define the pin to which the temperature sensor is connected.
*/
const int pinTemp =A0;

/* Define the B-value of the thermistor. */
const int B= 4275;

void setup()
{
   Serial.begin(115200); /* Open serial port*/
}

void loop()
{
      /* Get the analog value from the temperature sensor. */
      int val=analogRead(pinTemp);

      /* Determine the current resistance of the thermistor based
      on the sensor value. */
      float resistance = (float)(1023-val)*10000/val; /*10000ohm
      = 10Kohm is the zero power resistance*/

      /* Calculate the temperature based on the resistance value.
      */
      float temperature = 1/(log(resistance/10000)/B+1/298.15)-
      273.15;

      /* Print the temperature to the serial console. */
      Serial.print("temperature = ");
      Serial.println(temperature);

      /* Wait one second between measurements. */
      delay(1000);
}
```

4. Upload the code to the Arduino.
5. Open the Serial Monitor and observe the ambient air temperature value in degrees Celsius.

EXAMPLE 3.4 DISCUSSION

In the Serial Monitor you should be able to see the room temperature. When you blow on or grab the temperature sensor, you should see a change in the temperature value.

SUGGESTED PRACTICE

EXERCISE 3.4: Change the state of three LEDs using the reading from the temperature sensor

In this exercise, we will read the temperature sensor and use the temperature value to light up different LEDs. Let's assume that the ambient temperature is between 64 and 73 degrees Fahrenheit. If the reading is within this range, turn ON the green LED. If it is less than 64 degrees Fahrenheit turn ON the blue LED. If it is above 73 degrees Fahrenheit turn ON the red LED. Now grab the temperature sensor with your fingers to test the high temperature case. Next, place a cold item against the sensor to test the cold temperature case.

3.8 I2C Communication

I2C stands for an Inter-Integrated Circuit Protocol. It is a protocol intended to allow multiple "slave" digital integrated circuits ("chips") to communicate with one or more "master" chips. It is a low-speed bus protocol that transfers data via two wires: **SCL** and **SDA**. SCL is the clock line that synchronizes data transfer over the I2C bus, and SDA is the data line. The I2C protocol is usually preferred over the asynchronous serial port protocol. One of the reasons is that it is possible to connect multiple devices to a single serial port, while asynchronous serial ports are inherently suited to communications between two, and only two, devices. For more detailed information on I2C communication, please visit the following web site: https://learn.sparkfun.com/tutorials/i2c.

EXAMPLE 3.5

I2C Communication: Grove LCD RGB Backlight

In this example, you will use the LCD module included in the Grove Starter Kit. The Grove LCD has a backlight that uses the RGB (Red, Green, Blue) color model and can have 16,777,216 (i.e., 256 x 256 x 256) different color values. Thus, you can basically set the color of the backlight to whatever you like. With I2C, the number of pins required for data exchange and backlight control shrinks from about 10 to 2.

1. Connect the Grove LCD RGB Backlight to one of the I2C sockets on the Grove Base Shield, as shown in Figure 3-13.
   Note: It doesn't matter which I2C socket you decide to use.
2. Copy and paste the code below in the Arduino IDE and save it as ***Grove_RGB_LCD.ino***.
   Notes:
   - The code uses two different libraries: ***Wire*** and ***rgb_lcd***. The ***Wire*** library comes standard with the base Arduino IDE installation. The ***rgb_lcd*** library is a **contributed library** and must be present in the ***libraries*** folder of the sketchbook that you are using.
   - The difference between the **<,>** and the **",**" in the **#include** statements in the code is that the first (e.g., **#include <Wire.h>**) indicates that the library is in a standard location whereas the second (e.g., **#include "rgb_lcd.h"**) indicates that the library is local.

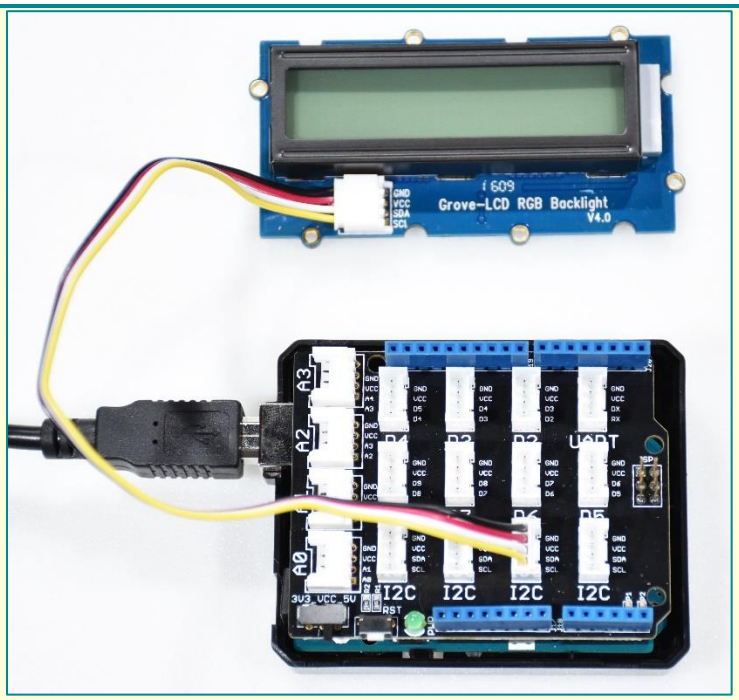


Figure 3-13. Grove LCD connected to one of the I2C sockets.

3. Upload the code to the Arduino.

```
/*
Grove-RGB_LCD.ino
This code will set the color of the LCD backlight and keep counting
the seconds since the last reset of the Arduino
*/

#include <Wire.h>
#include "rgb_lcd.h"

rgb_lcd lcd;

/* This is the color code. You can change it to whatever you want */
const int colorR= 255;
const int colorG= 0;
const int colorB= 0;

void setup()
{
      /* set up the number of columns and rows in the LCD */
      lcd.begin(16, 2);
      lcd.setRGB(colorR,colorG,colorB);
      /* Print a message to the LCD. */
      lcd.print("Hello Friends!");
      delay(1000);
}
void loop()
{
      /* set the cursor to column 0, line 1 (note: line 1 is the
      second row, since counting begins with 0): */
      lcd.setCursor(0, 1);
      /* print the number of seconds since reset: */
      lcd.print(millis()/1000);
      delay(100);
}
```

EXAMPLE 3.5 DISCUSSION

The color of the LCD backlight can be changed by changing the red (R), green (G), and blue (B) values between 0 and 255. Since we are using the I2C communication protocol, it does not matter which I2C socket of the Grove Base Shield you use.

SUGGESTED PRACTICE

EXERCISE 3.5: Make a color of the LCD fade or brighten

Change the value of red, green, or blue from 255 to 0 or from 0 to 255 to make the backlight color fade or brighten.

EXERCISE 3.6: Change the color of the LCD using a push button

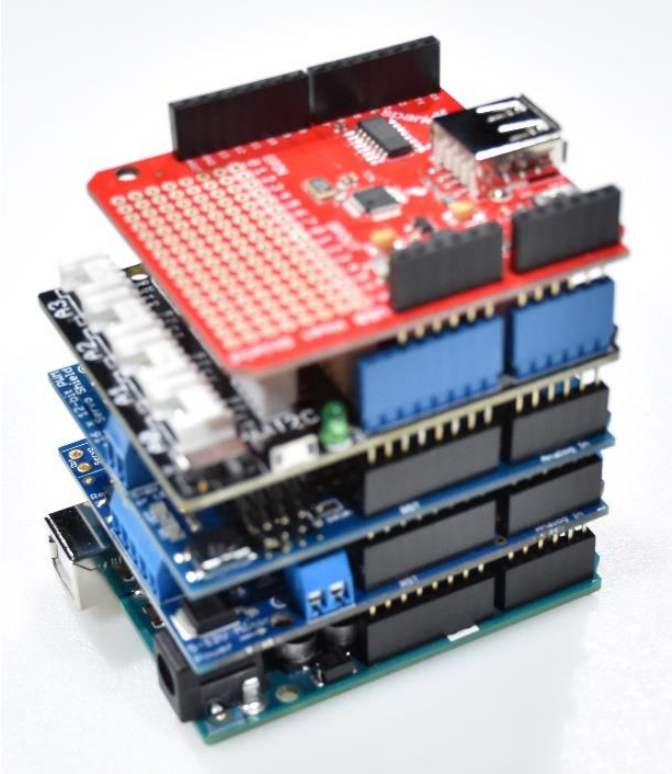
Use a push button to change the backlight color of the LCD in a random fashion. To do this, generate a random number between 0 and 255 for each of the three colors: red, green, and blue. Then, set the color of the backlight using those random numbers. The backlight color should remain the same until the push button is pressed again.

NOTES:

- To practice and understand how the other sensors from the Grove Starter Kit work, you can go through the short manual that comes with the kit and test the demo codes provided by the manufacturer (Seeed Studio). All the demo codes can be accessed as shown in Figure 3-4.
- The next chapter explains how to use the additional hardware items that are not a part of the Grove Starter Kit.

CHAPTER 4

INTRODUCTION TO THE OTHER SHIELDS IN THE KIT



INTRODUCTION TO THE OTHER SHIELDS IN THE KIT

OBJECTIVES

- Introduce the other shields in the kit
- Learn how to use the other shields in the kit
- Show the suggested way to stack the shields

In this chapter, we will be briefly go over the other shields that are part of the kit. Code examples are provided to show the basic functions of each one. The information presented is only intended to help you get started using the shields. Please refer to the documentation provided by the manufacturer of each shield for additional details.

When shields are stacked, it is not possible to see the information printed on the boards. Appendix A has pictures showing the layout of the Arduino Uno as well as the other shields in the kit. We suggest that you have a hardcopy of Appendix A at hand when you are working with the shields in case you need to see the layout of the boards.

4.1 Adafruit Motor/Stepper/Servo Shield

The motor shield can drive up to 4 DC motors bi-directionally (i.e., the motors can be driven forward and backward). The shield handles all the motor and speed controls over **I2C**. Thus, it only requires two pins (**SDA** and **SCL**) to drive multiple motors. Also, since the shield uses I2C you can connect other I2C devices or shields to the same two pins.

4.1.1 Adding the Library Required for Using the Adafruit Motor/Stepper/Servo Shield

Before using this shield, you need to download and install the "**Adafruit Motor Shield V2 Library**". Follow the steps below to do so.

1. Download the library from:
   https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software.

2. Rename the ".zip" file as "**Adafruit_MotorShield.zip**".
3. Open the Arduino IDE, go to the **Sketch** tab and move the mouse cursor to the **Include Library** menu option (see Figure 4-1).

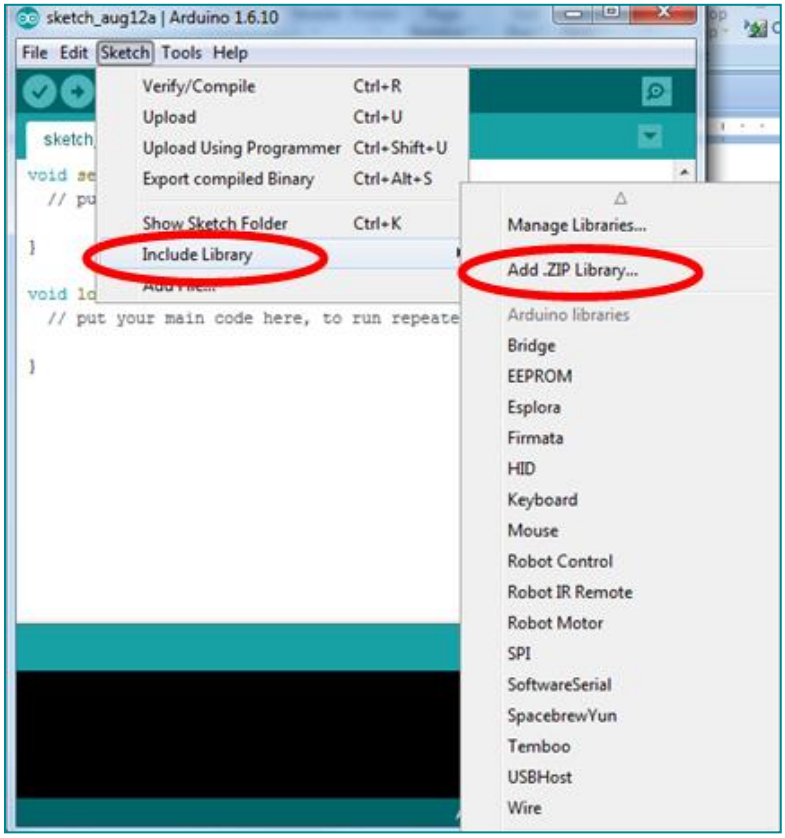


Figure 4-1. Adding the library required by the Adafruit motor shield.

4. Click on the **Add .ZIP Library…** menu option and locate and select the folder where you saved the library for the Adafruit motor shield.
5. Close and reopen the Arduino IDE and the library should be ready to use.

   Notes:

   - Every time you add a new library you need to close and reopen the Arduino IDE to be able to start using it.
   - If the library that you added is not showing in the Arduino IDE menu items related to libraries, it means that the library was not installed correctly or that you forgot to restart the Arduino IDE.
   - When you change the default sketchbook that the Arduino IDE is using, it is a good idea to restart the Arduino IDE.

Additional Resources

To learn more about the motor shield and its associated library visit the following web sites:

- https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-dc-motors.

- https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/library-reference.

4.1.2 Powering the Adafruit Motor/Stepper/Servo Shield

Before providing details on how to use the motor shield, we present some information regarding the selection of the DC motors that you will be controlling with the shield. The specification/data sheet for a DC motor will provide important information to help you decide if that motor is a good choice for your project. Some motors may need a lot of power to operate and the motor shield has limitations in the voltage range and amount of current that it can provide. You need to read the documentation provided by the manufacturer of the motor shield to become acquainted with the capabilities and limitations of the hardware.

The first aspect to consider is the voltage indicated in the motor specification/data sheet. The motor controllers on the shield are designed to run between **5V** and **12V**. Commonly you will be using small DC motors with the shield that need between **6V** and **12V**. In this regard, it is important to point out that most **1.5V** to **3V** motors will not work with the shield.

The motor shield needs an external power supply to drive the motors. Here we will use a **12V** battery pack for that purpose. The battery pack that we selected uses eight **1.5V** AA batteries connected in series, has an on/off switch, a cover, and two lead wires with bare ends. We strongly suggest that you use external power supplies that have an on/off switch and that you make sure that the switch is in the off position when you are not driving motors with the shield or when you store the battery pack.

As shown in Figure 4-2, connecting the battery pack to the motor shield is relatively simple. However, since you are working with DC, you need to be very careful with the polarity. The battery pack has a positive (“+”) lead wire and a negative (“-”) lead wire. If they are not clearly marked in the battery pack that you have, you can use a multimeter that is set to measure DC voltage to figure out which wire is the “+” and which wire is the “-”. The information printed on the board of the motor shield clearly indicates where the lead wires of the battery pack need to be connected. In particular, the board clearly indicates the connection point for the “+” lead wire and for the “-” lead wire of the battery pack.

If you are not familiar with DC, there are many references on the subject available online. The company SparkFun has different tutorials that you may find helpful. The following is a link to one of their tutorials: https://learn.sparkfun.com/tutorials/voltage-current-resistance-and-ohms-law.

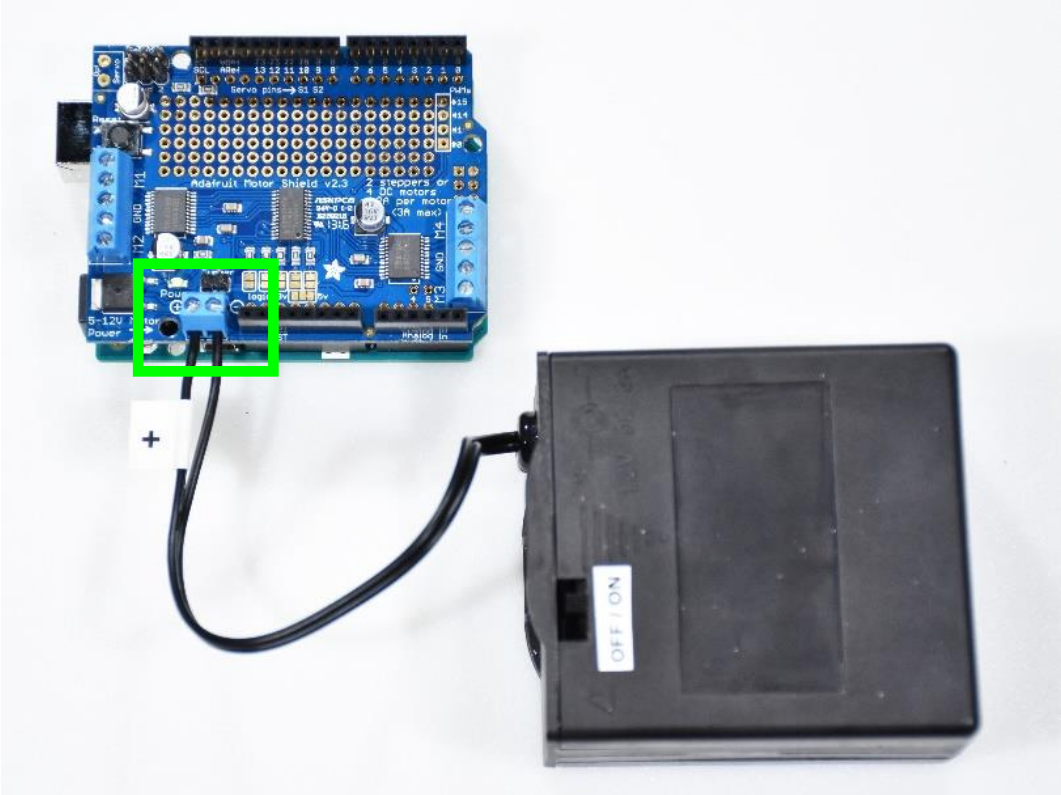


Figure 4-2. Powering the Adafruit Motor/Stepper/Servo shield.

4.1.3 Using the Adafruit Motor/Stepper/Servo Shield

We will learn how to drive motors using the Adafruit Motor/Stepper/Servo Shield by considering an example.

EXAMPLE 4.1

Driving DC motors using the Adafruit Motor/Stepper/Servo Shield

In this example, we will drive two DC motors forward and backward.

Before we begin:

- Remove any shields that you may have on the Arduino.
- Carefully place the Adafruit Motor/Stepper/Servo Shield on the Arduino.
- Have two 12V battery packs ready: one with a barrel jack connector and one with lead wires that have bare ends. The battery pack with the barrel jack connector will be used to power the Arduino when the Arduino is not connected to a PC via a USB cable. The battery pack with lead wires that have bare ends will be used to power the motor shield. We recommend

that you use battery packs that have an on/off switch and that you keep the switch in the off position when you are not using the Arduino and the motor shield.

- Select the DC motors that you will be controlling with the motor shield. One possible option is to use VEX EDR 393 motors from VEX Robotics.
- The shield can control up to four DC motors. The places where motors can be connected to the shield (i.e., the connection pins for motors) are clearly labeled on the board as "M1", "M2", "M3", and "M4".

Follow the steps below to make the required hardware connections and upload the example code to the Arduino.

1. Connect one DC motor to "M1" and one DC motor to "M2", as shown in Figure 4-3. From now on, we will refer to those two motors as ***Motor 1*** and ***Motor2***, respectively.

   Notes:

   - In order to drive motors bi-directionally, be sure to connect the motor leads between the two terminals for each motor. DO NOT connect one of the leads to ground.
   - Sometimes when you are working on a project the direction in which a motor is rotating is the opposite of what you expected (i.e., the "forward" and "backward" directions are reversed). To solve this problem, you only need to reverse the way in which the lead wires of the motor are connected to the shield.
   - If you want, you can connect two more DC motors to the shield: One to "M3" and one to "M4".

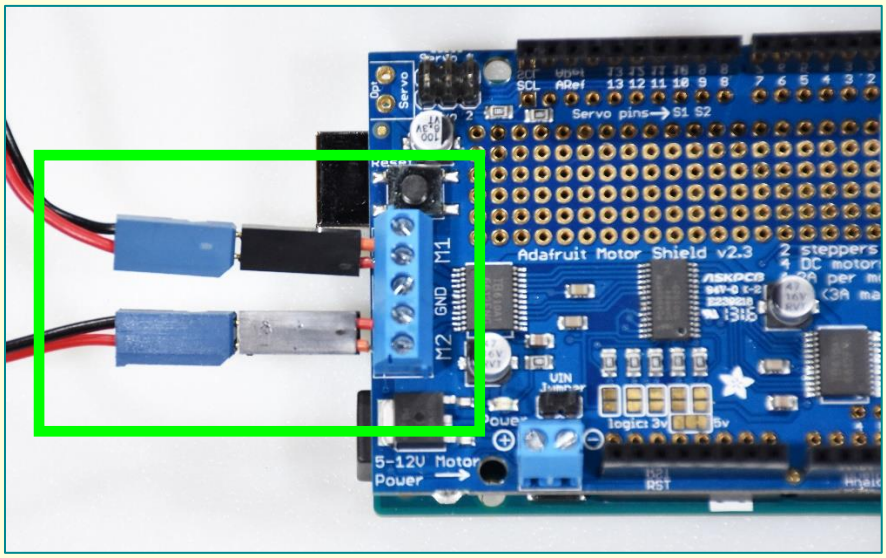


Figure 4-3. Connecting the DC motors to the Adafruit Motor/Stepper/Servo shield. Note: The short jumpers used to make the connections to "M1 and "M2" allow easy connections to the motors if another shield is stacked on top of the motor shield.

2. Connect the 12V external power supply to the motor shield (see Figure 4-2). The external power supply is mainly used to power all the motors connected to the shield.
3. Do a final check to make sure that the motors and battery packs are connected as shown in Figure 4-4.

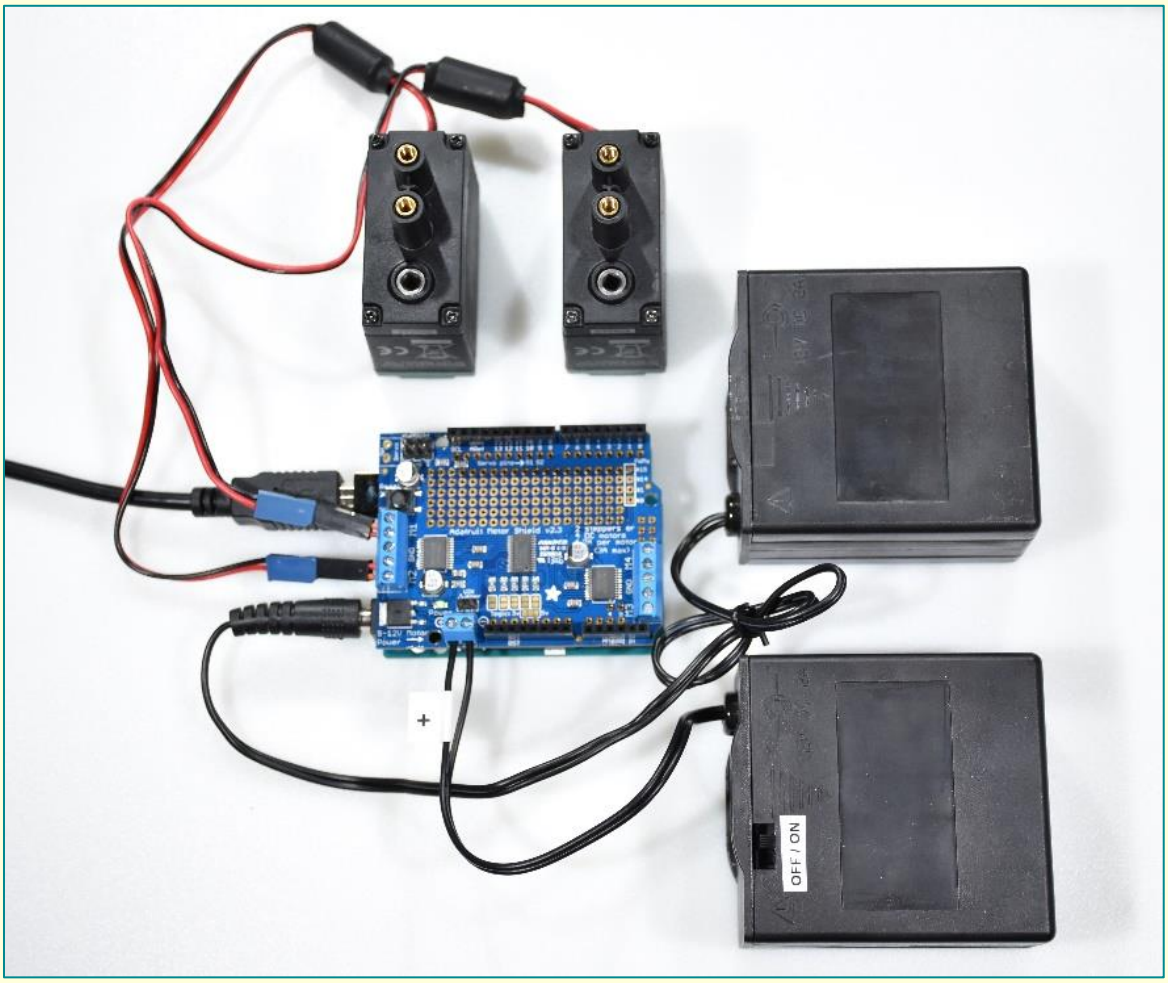


Figure 4-4. Overview of how to connect the motors and battery packs to use the Adafruit Motor/Stepper/Servo shield.

4. Copy and paste the code below in the Arduino IDE and save it as ***dcmotor.ino***.
5. Upload the code to the Arduino.

Important Notes:

- The first three ***#include*** lines in the example code should always be added at the beginning of programs that use the Adafruit Motor/Stepper/Servo shield. They include all the libraries that are required to use the shield.
- The shield uses PWM to drive the motors.

- The rotational speed of a motor is set by using the function ***setSpeed(speed)*** where the value of the variable ***speed*** can range from 0 (stopped) to 255 (full speed). You can set the speed of the motor to whatever value you want in that range. Note that you are not actually setting the speed, but rather the PWM duty cycle that is sent to the motor. The actual motor speed will depend on the load that the motor experiences.
- To direction of rotation of a motor is set using the function ***run(direction)*** where the variable ***direction*** can take the following values: ***FORWARD***, ***BACKWARD***, or ***RELEASE***.

```
/*
  dcmotor.ino
  This code drives two dc motors forward and backward.
  The general layout of functions that can be used to drive a vehicle
  left, right, and to stop the vehicle are also included so that
  you can use them in the future if you need them.
*/

#include<Wire.h>
#include<Adafruit_MotorShield.h>
#include"utility/Adafruit_MS_PWMServoDriver.h"

/* Create the motor shield object with the default I2C address */
Adafruit_MotorShield AFMS =Adafruit_MotorShield();

/* or, if you have more than one motor shield stacked on top of each
 other, you can create the motor shield object with a different I2C
 address */
//Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

/* Select to which 'port' on the shield (M1, M2, M3 or M4) each
 motor is connected */
Adafruit_DCMotor*myMotor_1 =AFMS.getMotor(1);
Adafruit_DCMotor*myMotor_2 =AFMS.getMotor(2);

void setup() {
  Serial.begin(9600); /* set up the serial port at 9600 bps*/
  Serial.println("Adafruit Motorshield v2 - DC Motor test!");
  AFMS.begin();  /* create with the default frequency of 1.6KHz */
  //AFMS.begin(1000);  /* OR with a different frequency, say 1KHz*/
```

```
}

void loop() {
  int Level;
  int Speed[3] = {75, 150, 255};
  /* Run forward and backward at each speed level */
  for (Level = 0; Level < 3; Level++) {
    Serial.print("Speed Level ");
    Serial.println(Level);
    forward(Speed[Level]); /* The forward function is defined below
*/
    delay(1000);
    Stop();
    delay(1000);
    backward(Speed[Level]); /* The backward function is defined
below */
    delay(1000);
    Stop(); /* The Stop function is defined below */
    delay(2000);
  }
}
void forward(int spd) /* Drive all the motors forward */
{
  myMotor_1->setSpeed(spd);
  myMotor_1->run(FORWARD);
  myMotor_2->setSpeed(spd);
  myMotor_2->run(FORWARD);
}
void backward(int spd) /* Drive all the motors backward */
{
  myMotor_1->setSpeed(spd);
  myMotor_1->run(BACKWARD);
  myMotor_2->setSpeed(spd);
  myMotor_2->run(BACKWARD);
}
/* Function to drive M1 and M2 backward and M3 and M4 forward */
void TurnLeft(int spd)
{
  // motor 1
  myMotor_1->setSpeed(spd);
```

```
  myMotor_1->run(BACKWARD);
  // motor 2
  myMotor_2->setSpeed(spd);
  myMotor_2->run(BACKWARD);
}
/* Function to drive M1 and M2 forward and M3 and M4 backward */
void TurnRight(int spd)
{
  // motor 1
  myMotor_1->setSpeed(spd);
  myMotor_1->run(FORWARD);
  // motor 2
  myMotor_2->setSpeed(spd);
  myMotor_2->run(FORWARD);
}

/* Function to stop all Motors */
void Stop()
{
  // motor 1
  myMotor_1->setSpeed(0);
  myMotor_1->run(RELEASE);
  // motor 2
  myMotor_2->setSpeed(0);
  myMotor_2->run(RELEASE);
}
```

EXAMPLE 4.1 DISCUSSION

The code drives the two DC motors by endlessly repeating a prescribed sequence. First, the motors move “forward” with a PWM value of 75 for 1 second, stop for 1 second, move “backward” with a PWM value of 75 for 1 second, and stop for 2 seconds. Then the PWM is set to 150 and the forward and backward cycle is repeated. Finally, the PWM is set to 255 and the forward and backward cycle is repeated once more.

So that you can have an idea of how PWM is used to control the speed of a motor, a PWM value of 75 corresponds to a duty cycle of 29.5%, which means the power is “on” for 29.5% of the period of the signal and “off” for 70.5% of the period of the signal. You can change this value

between **0** and **255,** which mimics voltages between **0V** and **the maximum voltage** that the shield can supply to the motor.

To learn more about PWM see the Appendix D.

SUGGESTED PRACTICE

EXERCISE 4.1: Driving four motors

In Example 4.1 we used the Adafruit Motor/Stepper/Servo shield to drive two DC motors. In this exercise, you will drive four DC motors as in a **differential drive robot**. Consider a scenario in which the motors are being used as part of a robot that has four wheels: The motors on "M1" and "M2" are driving the two wheels located on one side of the robot and the motors on "M3" and "M4" are driving the two wheels located on the other side of the robot.

Notes:

- Drive all the motors in the "forward" direction to move the robot forward along a "straight" line.
- Drive all the motors in the "backward" direction to move the robot backward along a "straight" line.
- Drive motors "M1" and "M2" in the "forward" direction and motors "M3" and "M4" in the "backward" direction to make the robot turn "right".
- Drive motors "M1" and "M2" in the "backward" direction and motors "M3" and "M4" in the "forward" direction to make the robot turn "left".
- Write your code in such a way that the robot makes a square loop, driving in a straight line on each side of the square for 2 seconds. Do you think that the robot will come back to its starting point? If your answer is "No", what do you think the problem is? How do you think you can fix it?

4.2 Adafruit 16-Channel 12-Bit PWM/Servo Shield

A servo motor is an assembly of four devices: a normal DC motor, a gear reduction unit, a position-sensing device (usually a potentiometer), and a control circuit. The function of a servo is to receive a control signal that represents the desired angular position of its shaft, and apply power to its DC motor until the shaft turns to that angular position. A servo has a 3-wire connection: power, ground, and control. The power source must be constantly applied and the control signal uses PWM (see Appendix D) to define the desired servo angle. The PWM used in controlling servo motors is different than that used in the ***analogWrite()*** command and hence we will use different commands to control servo motors. The pulse width of the control signal determines the angular position of the servo shaft. There is a particular pulse width value that corresponds to the center angular position of the servo shaft. A longer pulse makes the servo turn to a "clockwise-from-center" angular position, and a shorter pulse makes the servo turn to a "counter-clockwise-from-center" angular position.

The Adafruit 16-Channel 12-Bit PWM/Servo Shield can drive up 16 servos. The only Arduino pins required to run the shield are the **Ground**, **5V**, and the **SDA** and **SCL I2C** control pins. Since I2C is a "shared bus", you can still connect other I2C devices to the SDA and SCL pins of the Arduino as long as they do not have conflicting I2C addresses. The default address for the Adafruit servo shield is 0x40.

Although the Adafruit 16-Channel 12-Bit PWM/Servo Shield can be used to drive up to 16 servos, only 4 servos can be easily connected to the shield due to issues arising from stacking the different shields in the kit on top of each other. A 3x4 right angle male header was soldered to the shield to facilitate the connection of the 4 servos. If additional servos are needed for your project, you can find ways to use the other pin connections in the shield. However, it is very likely that some soldering will be required.

4.2.1 Adding the Library Required for Using the Adafruit PWM/Servo Shield

Before using the Adafruit PWM/Servo Shield you need to download and install the "**Adafruit PWM/Servo Library**". Follow the steps below to do so.

1. Download the library from: https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield/using-the-adafruit-library.
2. Unzip the folder that you downloaded and rename it "**Adafruit_PWMServoDriver**".

3. Check that the "**Adafruit_PWMServoDriver**" folder contains the following two files: ***Adafruit_PWMServoDriver.cpp*** and ***Adafruit_PWMServoDriver.h***.
4. Place the "**Adafruit_PWMServoDriver**" folder in the "**libraries"** folder of the Arduino sketchbook that you are using to save your sketches.
5. Close and reopen the Arduino IDE software and the library required by the Adafruit PWM/Servo Shield should be ready to use.

Additional Resources

To learn more about the servo shield and its associated library visit the following web sites:

- https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield/using-the-adafruit-library.
- https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield/library-reference.

4.2.2 Powering the Adafruit PWM/Servo Shield

We need to provide power to both the **Vin** pin (**3V** to **5V** for the logic level) and the "**V+**" terminal block (**5V** to **6V** for the servos) of the Adafruit PWM/Servo Shield. The **Vin** pin receives power from the Arduino board when we stack the shield on top of the Arduino. The "**V+**" terminal block receives power from an external power supply. Here we will use a **6V** battery pack to provide external power to the shield. The battery pack that we selected uses four **1.5V** AA batteries connected in series, has an on/off switch, a cover, and two color-coded (red for positive and black for negative) lead wires with bare ends. The battery pack is connected to the shield as shown in Figure 4-5 and Figure 4-6.

Important: The shield has a red LED and a green LED to indicate that the two power supplies that it needs to operate are properly connected. **Both the red and the green LEDs on the shield must be lit for the shield to work!**

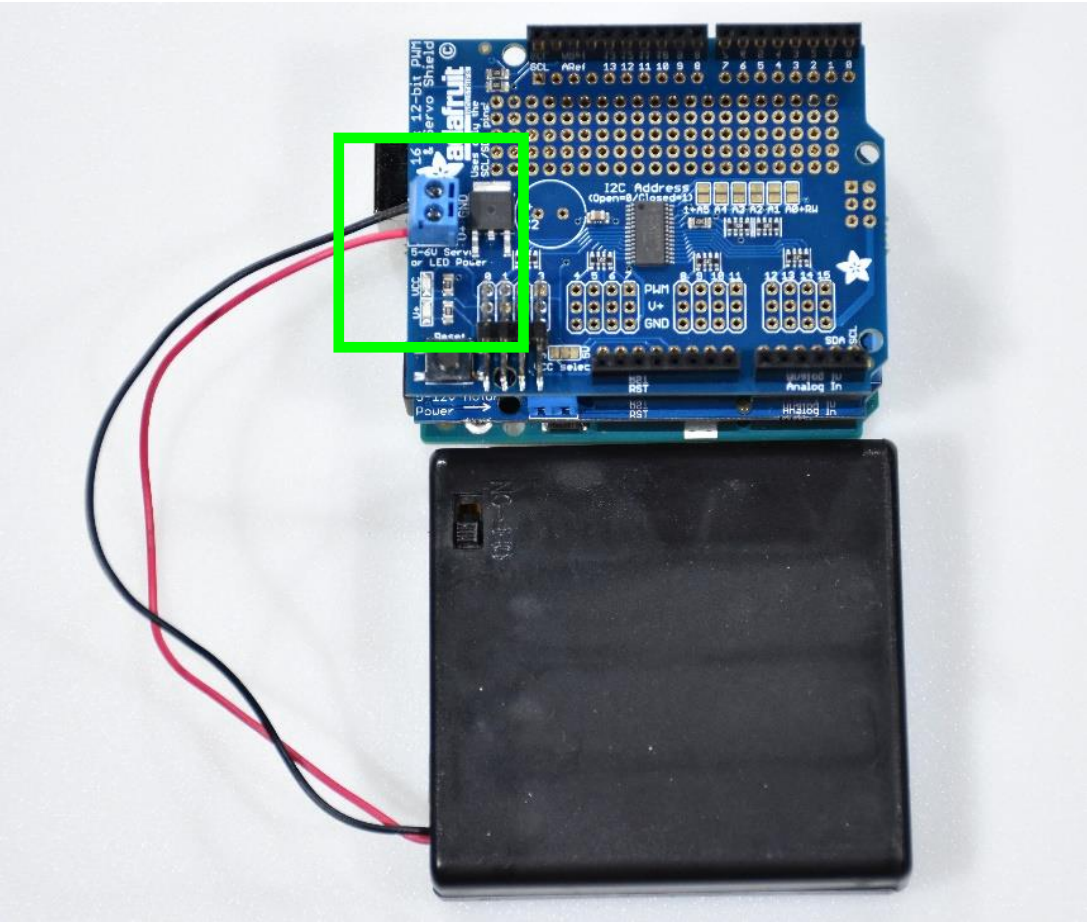


Figure 4-5. Powering the Adafruit PWM/Servo Shield with a 6V, 4 AA battery pack.

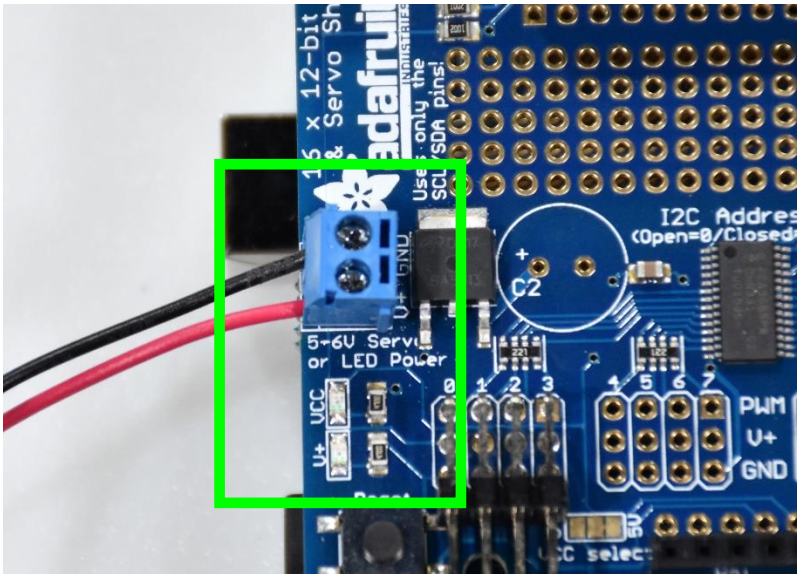


Figure 4-6. Close up of the connection of the positive (red) and negative (black) cables from the 6V battery pack to the "V+" terminal block of the Adafruit PWM/Servo Shield. Getting this connection backwards could result in the servos not working correctly and/or in damage to the servos.

4.2.3 Using the Adafruit PWM/Servo Shield

We will learn how to use the Adafruit PWM/Servo Shield by considering a simple example.

EXAMPLE 4.2

Driving servos using the Adafruit 16-Channel 12-Bit PWM/Servo Shield

In this example we will be driving two servos using the Adafruit PWM/Servo Shield.

Before we begin:

- We only need the Arduino UNO and the Adafruit 16-Channel 12-Bit PWM/Servo Shield for this example.
- If you did Example 4.1 and have the Adafruit Motor/Stepper/Servo Shield stacked on top of the Arduino, you don't need to remove that shield. Simply stack the Adafruit PWM/Servo Shield on top of the motor shield.

  Notes:
  - Since the code in this example doesn't make use of the Adafruit Motor/Stepper/Servo Shield, you can disconnect the external power supply (i.e., the 12V battery pack) and the DC motors from that shield.
  - Remember that the Adafruit Motor/Stepper/Servo Shield and the Adafruit PWM/Servo Shield use different battery packs. The motor shield uses a 12V battery pack whereas the servo shield uses a 6V battery pack.
- We will be using VEX EDR 3-Wire Servos in this example. However, you can use other servos if you like. In that case, some lines in the code will need to be adjusted based on the characteristics of the servos that you selected.
- An important characteristic of a servo motor is the angle that it can rotate. The VEX EDR 3-Wire Servo only has 100 degrees of rotation.
- We will use the labels ***Servo 1*** and ***Servo 2*** to refer to the two servos that we will be using in this example.

Follow the steps below to make the required hardware connections and upload the example code to the Arduino.

1. Connect the 6V battery pack to the Adafruit PWM/Servo Shield as shown in Figure 4-6. Remember that both the red and the green LEDs on the shield must be lit for the shield to work.
2. Connect ***Servo 1*** and ***Servo 2*** to the first 2 sets of pins of the 3x4 right angle male header as shown in Figure 4-7.

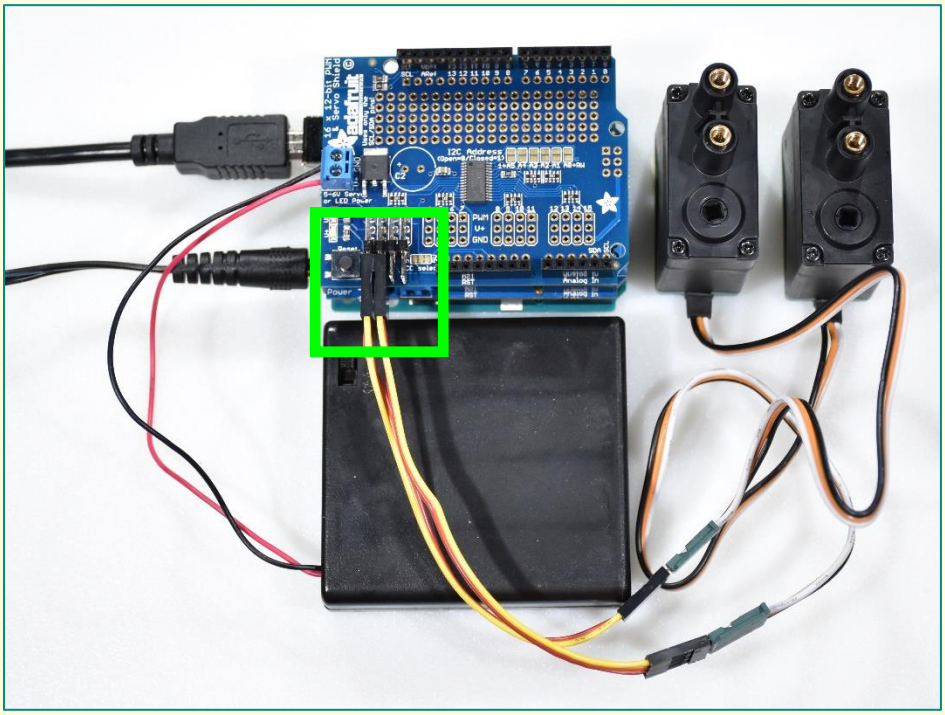


Figure 4-7. Overview of how to connect the servos to the Adafruit PWM/Servo Shield.

3. Copy and paste the code below in the Arduino IDE and save it as ***servo.ino***.
4. Upload the code to the Arduino.

Important Notes:

- The first two ***#include*** lines in the example code should always be added at the beginning of programs that use the Adafruit PWM/Servo Shield. They include all the libraries that are required to use the shield.
- The function ***setPWMFreq(freq)*** can be used to adjust the PWM frequency (***freq***), which determines how many full "pulses" per second are generated. In other words, the frequency (***freq***) determines how "long" each pulse is in duration from start to finish, considering both the high and low segments of the pulse. The value of ***freq*** is given in **Hz** and needs to be in the range between **40** and **1000**. This frequency should match the

frequency used by your servo; see the datasheet for your device to choose the correct value.

- The function ***setPWM(Channel, On, Off)*** sets the start (***On***) and end (***Off***) of the high segment of the PWM pulse on a specific channel (***Channel***). You specify the "tick" value between **0** and **4095** when the signal will turn on (***On***), and when it will turn off (***Off***). ***Channel*** indicates which of the 16 PWM outputs that the shield has (which are labeled **0** to **15** on the board) should be updated with the new values. If you set the value of ***On*** to 0, then the value that you assign to ***Off*** will determine the pulse width.
- The rotation angle of the servos that you selected can be used together with the standard Arduino ***map(value, fromLow, fromHigh, toLow, toHigh)*** function to convert a desired angular position to the corresponding pulse width of the control signal. For that purpose, you can use a statement such as

  pulseWidth = ***map(degrees, 0, rotationAngle, SERVOMIN, SERVOMAX)***

  to convert the desired angular position ***degrees*** into the corresponding pulse width value ***pulseWidth***. The parameters ***rotationAngle***, ***SERVOMIN***, and ***SERVOMAX*** need to be replaced by specific values applicable to the servo that is being controlled. As with the frequency, these values may be obtained from your servo's datasheet.
- The rotation angle (***rotationAngle***) and the pulse width values corresponding to the minimum (***SERVOMIN*** → 0 degrees) and maximum (***SERVOMAX*** → ***rotationAngle*** degrees) angular positions of a servo vary between different brands and models. For precise position control, you can calibrate the minimum and maximum pulse widths in your code to match the minimum and maximum angular positions of the servo. To find the minimum (***SERVOMIN***), use the example code and edit the ***SERVOMIN*** value until the low-point of the servo reaches the minimum range of travel. Approach this gradually and stop before the physical limit of travel is reached. Do the same procedure to find out the maximum (***SERVOMAX***). While you can use the values provided by the manufacturer, this calibration approach allows you to correct for any part-to-part variation between servo motors.
- For VEX EDR 3-Wire Servos you need to use a value of 100 for the parameter ***rotationAngle***.
- The ***#define*** directive allows the definition of macros within your code. These macro definitions allow constant values to be declared for use throughout your code. Macro definitions are not variables and cannot be changed by your program code like variables.

You generally use this option when creating constants that represent numbers, strings or expressions. The syntax is ***#define CNAME value***.

```
#include<Wire.h>
#include<Adafruit_PWMServoDriver.h>

/* When called in the way shown below it uses the default
address 0x40 */
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

/* You can also call it with another valid address as shown
below */
//Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);

/* The pulse width min and max may vary depending on your servo.
You want these values to be as small and large as possible without
hitting the hard stop on both ends of the range. You will have
to tweak the values to match the servo that you are using. */

/* Calibrate the min and max pulse width values according to the
instructions given in:
https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-
shield/using-the-adafruit-library */

#define SERVOMIN 200 // this is the 'minimum' pulse length count
#define SERVOMAX 400 // this is the 'maximum' pulse length count

/* Use the following command to convert degrees between 0 and
MaxAngle to pulselength in the range from SERVOMIN to SERVOMAX */

/*
pulselength = map(degrees, Minangle, MaxAngle, SERVOMIN, SERVOMAX);
*/

int MinAngle = 0, MaxAngle = 100, increment = 20, angle, pulselength;

// our servo # counter
uint8_t servonum_1 = 0;
uint8_t servonum_2 = 1;
```

```
/* Note: uint8_t corresponds to an unsigned integer of length 8 bits
(1 byte = 8 bits) */

void setup() {
  Serial.begin(115200);
  Serial.println("16 channel servo shield test!");
  pwm.begin();
/* Analog servos run at approximately 60 Hz updates */
  pwm.setPWMFreq(60);
/* Don't worry about the yield function included below */
 yield();
}

void loop() {
  /* Drive each servo one at a time using a statement like */
  // pwm.setPWM(servonum_1, 0, pulselength)

  pwm.setPWM(servonum_1, 0, SERVOMAX);
  pwm.setPWM(servonum_2, 0, SERVOMAX);
  delay(2000);
  pwm.setPWM(servonum_1, 0, SERVOMIN);
  pwm.setPWM(servonum_2, 0, SERVOMIN);
  delay(2000);

  angle = 0;
  while (angle < MaxAngle) {
    pulselength=map(angle, MinAngle, MaxAngle, SERVOMIN, SERVOMAX);
    pwm.setPWM(servonum_1, 0, pulselength);
    pwm.setPWM(servonum_2, 0, pulselength);
    angle = angle + increment;
    delay(1000);
  }
  while (angle >= MinAngle) {
    pulselength=map(angle, MinAngle, MaxAngle, SERVOMIN, SERVOMAX);
    pwm.setPWM(servonum_1, 0, pulselength);
    pwm.setPWM(servonum_2, 0, pulselength);
    angle = angle - increment;
    delay(1000);
  }
```

```
  yield();
  delay(2000);
}
```

EXAMPLE 4.2 DISCUSSION

When the Arduino runs the code presented above the following events take place. First, the servo shafts are rotated to their maximum angular position and remain there for 2 seconds. Then, the servo shafts are rotated to their minimum angular position and remain there for 2 seconds. Next, the angular position of the servo shafts is gradually increased from the minimum angle to the maximum angle based on the specified increment and with a wait time of 1 second between changes in angular position. Afterwards, the angular position of the servo shafts is gradually decreased from the maximum angle to the minimum angle based on the specified increment and with a wait time of 1 second between changes in angular position. Finally, after a wait time of two seconds, the execution returns to the beginning of the ***loop()*** function and the sequence is repeated.

SUGGESTED PRACTICE

EXERCISE 4.2: Find the minimum and maximum pulse width for your servo

In the example given above, the SERVOMIN and SERVOMAX values will be different for different servos. The approximate value for the servo used in this manual is given. Try changing these values to see the maximum range for your specific device.

4.3 SparkFun USB Host Shield, Bluetooth Dongle, and Sony PS3 Controller

4.3.1 SparkFun USB Host Shield

The SparkFun USB Host Shield uses the MAX3421E USB Peripheral/Host Controller with Serial Peripheral Interface (SPI). A four-wire serial interface is used to communicate with the host controller chip and the shield connects the Arduino's SPI pins (D10 to D13) to the MAX3421E. In addition, the shield uses the Arduino digital pins D7, D8, and D9 for reset, GPX, and INT, respectively. Thus, when stacked on the Arduino, the Arduino digital pins **D7 to D13** are taken by the SparkFun USB Host Shield and will not be available for any other use.

The SparkFun USB Host Shield takes its power from the "**Vin**" pin of the Arduino. Power from that pin is regulated to both **5V** and **3.3V** on the shield.

4.3.2 Adding the Library Required for Using the SparkFun USB Host Shield

Before using the SparkFun USB Host Shield you need to download and install the "**USB-Host_Shield_2.0 Library**". Follow the steps below to do so.

1. Download the library from: https://github.com/felis/USB_Host_Shield_2.0.
2. Rename the .zip file as "**USB_Host_Shield.zip**".
3. Open the Arduino IDE, go to the **Sketch** tab and move the mouse cursor to the **Include Library** menu option.
4. Click on the **Add .ZIP Library…** menu option and locate and select the folder where you saved the library for the SparkFun USB Host Shield.
5. Close and reopen the Arduino IDE and the library should be ready to use.
6. Now you can access the examples that come with the library to learn how to use the SparkFun USB Host Shield. For example, the code for the **USB shield – Bluetooth PS3 controller** interconnection can be accessed as shown in Figure 4-8**.**

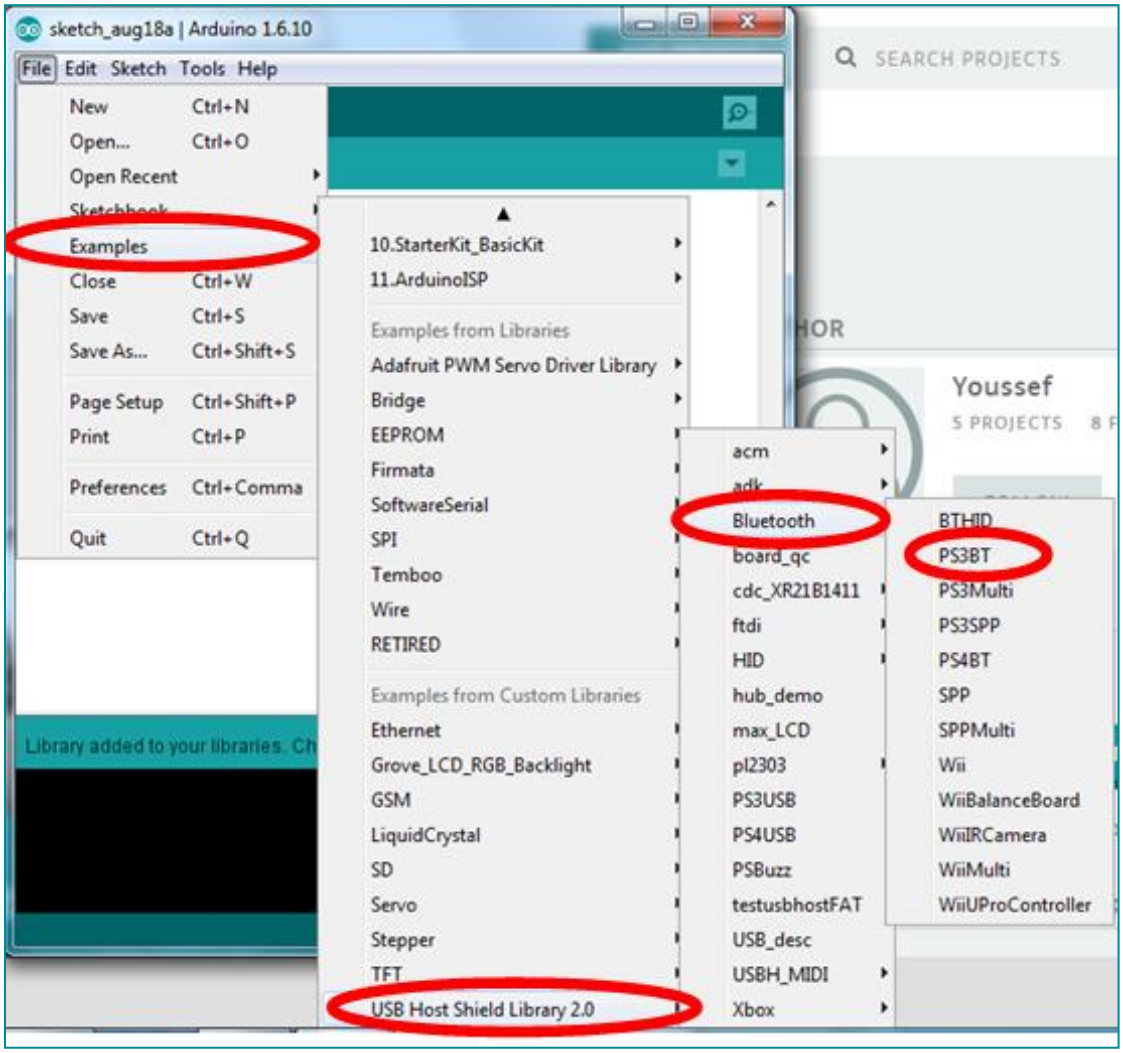


Figure 4-8. Accessing the examples that come with the library for the SparkFun USB Host Shield.

Additional Resources

To learn more about the SparkFun USB Host Shield visit the following web site:

- https://www.sparkfun.com/products/9947.

4.3.3 Bluetooth Dongle

A Bluetooth dongle with unique **MAC** (Media Access Control) address is used to make a wireless connection between the Sony PS3 controller and the SparkFun USB Host Shield. The unique MAC address allows pairing of a Sony PS3 controller with a specific Bluetooth dongle. This avoids problems when using two or more kits that are close to each other at the same time.

Follow the steps provided below to get the **MAC** address of a Bluetooth dongle.

1. Plug in the Bluetooth dongle into a USB port in your computer and wait until your computer recognizes the Bluetooth device and installs the required drivers.
2. Go to the Windows "**Device Manager**" to see the list of "**Bluetooth Radios**" that are connected to your computer (see Figure 4-9).

   Note: You can find the Windows "**Device Manager**" in the Windows "**Control Panel**".

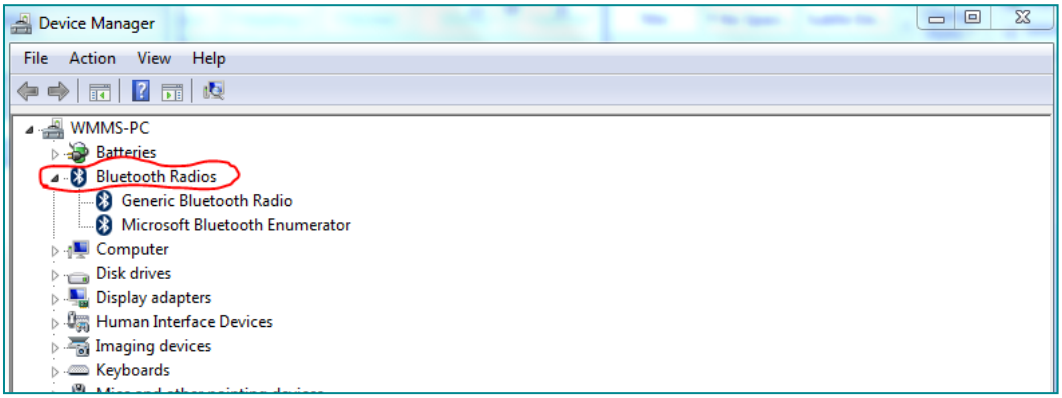


Figure 4-9. Locating the "Bluetooth Radios" connected to your computer in the Windows "Device Manager".

3. Right click the "**Generic Bluetooth Radio**" and click "**Properties**". You should see a "window" like the one shown in Figure 4-10.

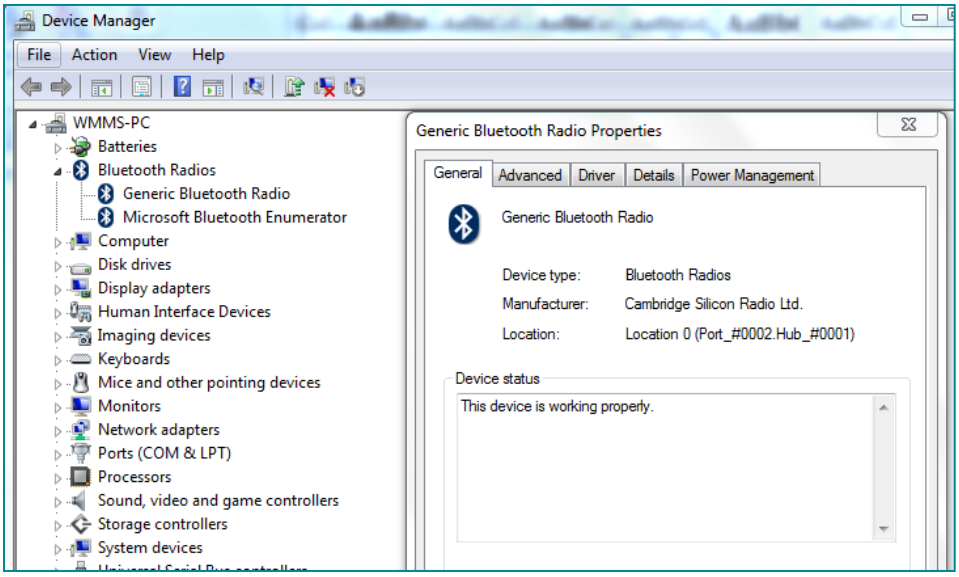


Figure 4-10. Locating the properties of the "Generic Bluetooth Radio".

4. Click the "**Advanced**" tab on the "**Generic Bluetooth Radio Properties**". You should see a "window" like the one shown in Figure 4-11.

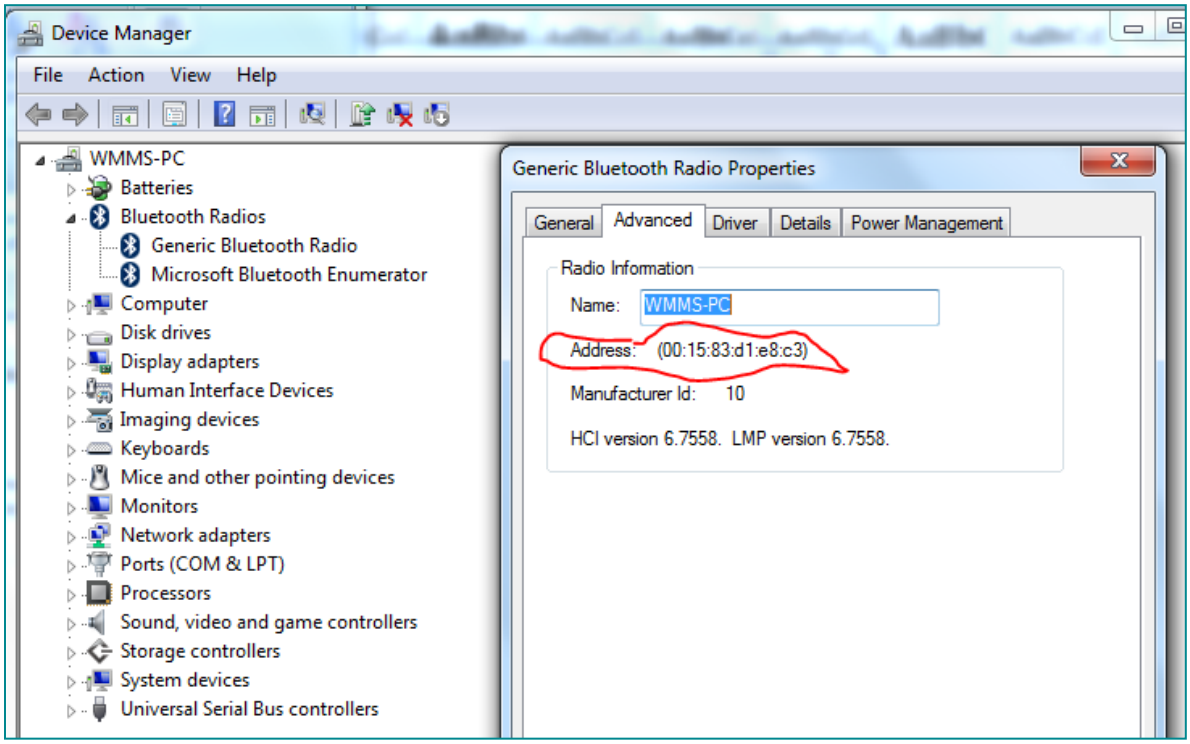


Figure 4-11. Locating the MAC address of the Bluetooth dongle.

5. The numbers next to the word "Address" in Figure 4-11 are the MAC address of the Bluetooth dongle. For example, the MAC address of the Bluetooth dongle corresponding to Figure 4-11 is 0x00:0x15:0x83:0xd1:0xe8:0xc3 (00:15:83:d1:e8:c3), where the prefix "0x" simply indicates that the number is given in hexadecimal. The MAC address will be used to connect the Sony PS3 controller to the Bluetooth dongle.

4.3.4 Sony PS3 Controller

As can be seen in Figure 4-12, the Sony PS3 controller can provide analog inputs via two two-axis joysticks and two "trigger" buttons (which are like single-axis joysticks). In addition, the Sony PS3 controller can provide digital inputs via ten different buttons (see Figure 4-12). This provides more than enough inputs for most projects without using additional digital and analog pins from the Arduino. Furthermore, the use of the Sony PS3 controller helps avoid any wiring to get those functionalities.

It is important to point out that when we tried to use a PS3 controller that was not manufactured by Sony, we were unable to connect it to the SparkFun USB Host Shield using the Bluetooth dongle. Thus, we strongly suggest that you use a Sony PS3 controller to avoid experiencing connection problems.

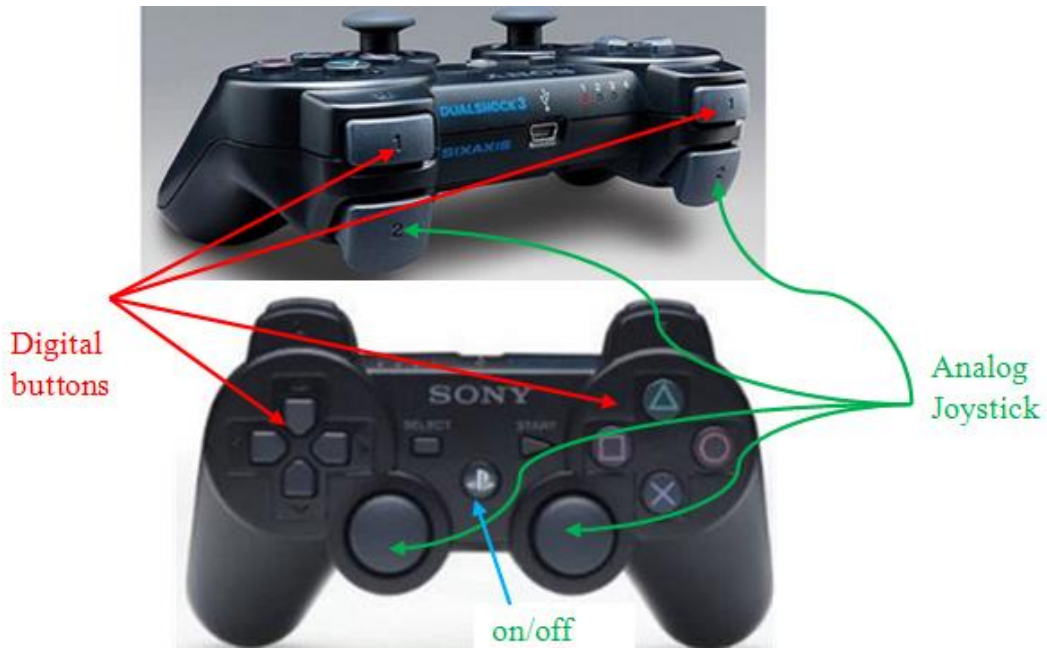


Figure 4-12. Analog and digital inputs provided by the Sony PS3 controller.

4.3.5 Using the SparkFun USB Host Shield, Bluetooth Dongle, and Sony PS3 Controller

We will learn how to use the SparkFun USB Host Shield, Bluetooth Dongle, and Sony PS3 controller combination by considering a simple example.

EXAMPLE 4.3

Sending analog and digital signals from the Sony PS3 controller to the Arduino using a Bluetooth connection.

The following example will show the procedure to use the SparkFun USB Host Shield, Bluetooth dongle, and Sony PS3 controller together to make a Bluetooth wireless connection between the Sony PS3 controller and the SparkFun USB Host Shield that can send analog and digital signals from the Sony PS3 controller to the Arduino.

Before we begin:

- We only need the Arduino UNO, the SparkFun USB Host Shield, the Bluetooth dongle, and the Sony PS3 controller for this example.
- If you have other shields stacked on top of the Arduino, you don't need to remove those shields. However, when several shields are stacked on top of the Arduino, you need to make sure that the SparkFun USB Host Shield is at the top of the stack.

- Complete all the steps listed below.
  1. Download and install the "**SixaxisPairTool**" program from: http://dancingpixelstudios.com/sixaxis-controller/sixaxispairtool/
  2. Go to the Windows "**Start**" menu and run the "**SixaxisPairTool**" program.
  3. Connect the Sony PS3 controller to your computer using a USB cable. The "window" of the "**SixaxisPairTool**" program should look like Figure 4-13.

     Note: In some cases, no MAC address is shown for the "Current Master". This is not a problem since our goal is to replace whatever MAC address the Sony PS3 controller has (if any) with a new one.

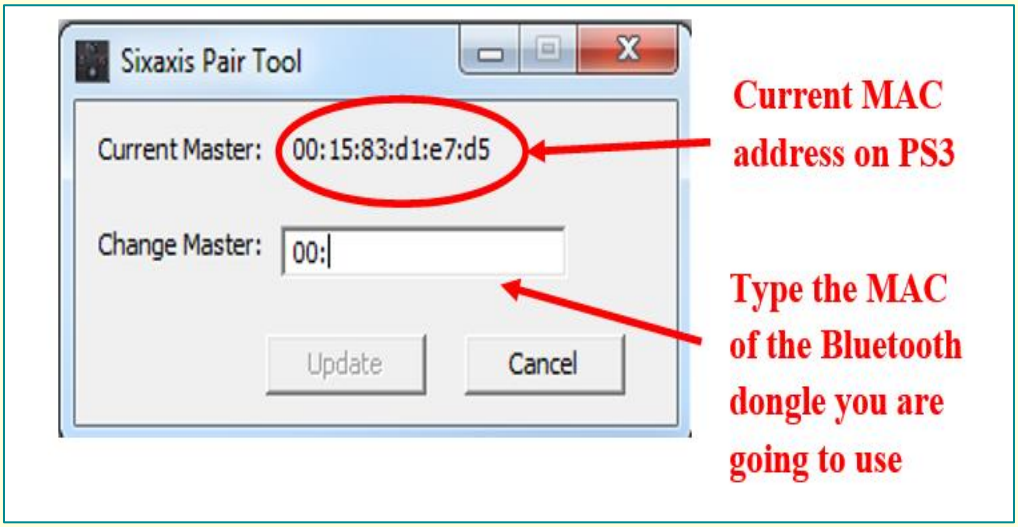


Figure 4-13. Changing the MAC address on the Sony PS3 controller.

  4. Type the MAC address of the Bluetooth dongle you are going to use in the "Change Master" input area and click "**Update**".
  5. The "Current Master" MAC address of the Sony PS3 controller will change to the new MAC address that you provided.

Follow the steps below to make the required hardware connections and upload the example code to the Arduino.

1. Place the SparkFun USB Host Shield on top of the last shield stacked on top of the Arduino. If there are no shields stacked on top of the Arduino, place the SparkFun USB Host Shield on top of the Arduino.
2. Insert the Bluetooth dongle in the USB port of the SparkFun USB Host Shield as shown in Figure 4-14.

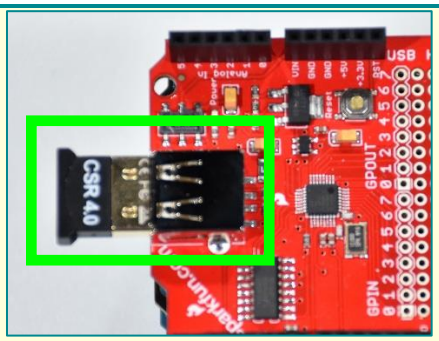


Figure 4-14. Inserting the Bluetooth dongle in the USB port of the SparkFun USB Host Shield.

3. Power the Arduino so that the SparkFun USB Host Shield can receive power.

   Note: Remember that the SparkFun USB Host Shield takes its power from the "Vin" pin of the Arduino.

4. Make sure that the **on/off** switch of the SparkFun USB Host shield is set to the "**on"** position as shown in Figure 4-15.

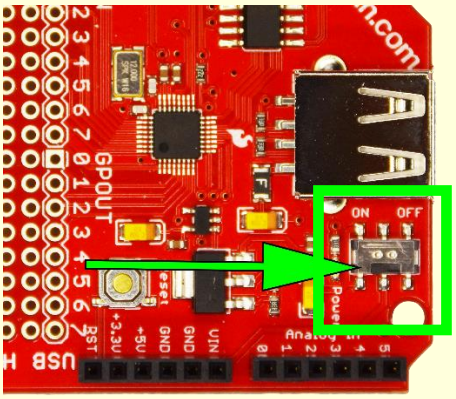


Figure 4-15. Location of the on/off switch of the SparkFun USB Host Shield.

5. Copy and paste the code below in the Arduino IDE and save it as ***BluetoothDongle.ino***.
6. Upload the code to the Arduino.

```
/* BluetoothDongle.ino */

#include<PS3BT.h>
#include<usbhub.h>

/* Satisfy the Arduino IDE, which needs to see the include statements
below in the ino file too. */
#ifdefdobogusinclude
#include<spi4teensy3.h>
#include<SPI.h>
```

```
#endif

USB Usb;
//USBHub Hub1(&Usb); /* Some dongles have a hub inside */

BTD Btd(&Usb); /* You must create the Bluetooth Dongle instance */

/* Only set the MAC address of the Bluetooth dongle if needed. */
/* The following statement only creates the PS3 controller instance.
Keep commented if you use an option that includes the specific MAC
address */
PS3BT PS3(&Btd);

/*Declare variables to store readings; */
int LeftX, LeftY, RightX, RightY;
int triangle, circle, cross, squa;
int up, down, right, left;
int l2, r2, l1, r1;
int select, start;

void setup() {
Serial.begin(115200);
if (Usb.Init() ==-1) {
Serial.print(F("\r\nOSC did not start"));
while (1); //halt (infinite loop since while (1) is always true)
}
Serial.print(F("\r\nPS3 Bluetooth Library Started"));
}

void loop() {
Usb.Task();

// Read the two joysticks on the PS3 Controller
if (PS3.PS3Connected || PS3.PS3NavigationConnected) {
LeftX= PS3.getAnalogHat(LeftHatX);
LeftY= PS3.getAnalogHat(LeftHatY);
RightX= PS3.getAnalogHat(RightHatX);
RightY= PS3.getAnalogHat(RightHatY);
Serial.print(LeftX); Serial.print("\t");
Serial.print(LeftY); Serial.print("\t");
```

```
Serial.print(RightX); Serial.print("\t");
Serial.println(RightY);
}

// Read Buttons (Both Left and Right Buttons)
triangle = PS3.getButtonClick(TRIANGLE);
circle = PS3.getButtonClick(CIRCLE);
cross = PS3.getButtonClick(CROSS);
squa = PS3.getButtonClick(SQUARE);
up = PS3.getButtonClick(UP);
down = PS3.getButtonClick(DOWN);
right = PS3.getButtonClick(RIGHT);
left = PS3.getButtonClick(LEFT);
//  Serial.print(triangle); Serial.print("\t");
//  Serial.print(circle); Serial.print("\t");
//  Serial.print(cross); Serial.print("\t");
//  Serial.print(squa); Serial.print("\t");
//  Serial.print(up); Serial.print("\t");
//  Serial.print(down); Serial.print("\t");
//  Serial.print(right); Serial.print("\t");
//  Serial.println(left);

// Reading Analog button values
l2 = PS3.getAnalogButton(L2);
r2 = PS3.getAnalogButton(R2);
//  Serial.print(l2); Serial.print("\t");
//  Serial.println(r2);

// digital left and right front buttons
l1 = PS3.getButtonClick(L1);
r1 = PS3.getButtonClick(R1);
select = PS3.getButtonClick(SELECT);
start = PS3.getButtonClick(START);
//  Serial.print(l1); Serial.print("\t");
//  Serial.print(r1); Serial.print("\t ");
//  Serial.print(select); Serial.print("\t");
//  Serial.println(start);
}
```

7. Open the **Serial Monitor** in the Arduino IDE and make sure that the correct baud rate is selected.
8. Observe the message displayed in the **Serial Monitor**. It should indicate that the USB host shield and the Bluetooth Library have started. If that is not the case, press the reset button on the USB host shield to see if the Library starts. If it still doesn't start, make sure that the Bluetooth dongle is properly connected to the USB host shield and that the Sony PS3 controller has the MAC address of the Bluetooth dongle that you are using.
9. Once the Bluetooth Library starts, press the **on/off** button of the Sony PS3 controller. At first, all the LEDs in the front of the Sony PS3 controller, which are labeled from 1 to 4, will be blinking. Wait until only the LED labeled as 1 is steadily **on**, which indicates that the Bluetooth connection between the Sony PS3 controller and the Bluetooth dongle has been made.

   Notes:

   - If the LED labeled as 1 is not steadily **on**, no connection has been made. To solve this problem, reset the USB host shield to see if you can make the connection. If that fails, make sure that the Bluetooth dongle is properly connected to the USB host shield and that the Sony PS3 controller has the MAC address of the Bluetooth dongle that you are using and repeat this step.
   - If you close and open the **Serial Monitor** the wireless connection between the Sony PS3 controller and the Bluetooth dongle may be lost. Repeat this step to reestablish the wireless connection.
   - If the Sony PS3 controller gets disconnected from the Bluetooth dongle due to power loss or any other reason, press the reset button of the USB host shield before trying to connect the Sony PS3 controller again.
10. Look at the **Serial Monitor** and observe all the readings that are displayed. Move the two joysticks to see how the values that are displayed change between **0** and **255** depending on the position of each joystick.

EXAMPLE 4.3 DISCUSSION

As mentioned earlier, with the Bluetooth connection to the Sony PS3 controller you have analog inputs via two two-axis joysticks and two "trigger" buttons as well as digital inputs via ten different buttons. You can use those inputs for any application in your project.

Each Bluetooth dongle has a different **MAC** address. Thus, you need to make sure that the Sony PS3 controller is set to the MAC address of the Bluetooth dongle that you are using.

Although the code in the example is written to read and assign all the digital and analog signals to variables, only the variables corresponding to the analog signals from the two joysticks are printed to the **Serial Monitor**. You can observe the value of the other variables in the **Serial Monitor** by "un-commenting" (removing the "//" at the beginning of the line of code) and "commenting" (adding a "//" at the beginning of the line of code) different parts of the code as needed.

4.4 Hardware Assembly Process

The hardware assembly for the complete kit requires stacking all the following shields on top of the Arduino UNO: the Adafruit Motor/Stepper/Servo Shield, the Adafruit 16-Channel 12-Bit PWM/Servo Shield, the Grove Base Shield, and the SparkFun USB Host Shield. Follow the steps shown in Figure 4-16 to Figure 4-20 to stack the shields in the kit.

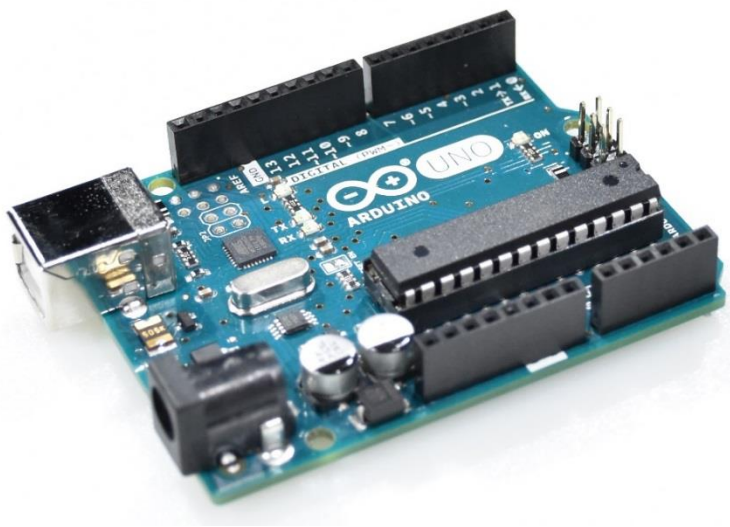


Figure 4-16. The base of the complete stack is the Arduino UNO.

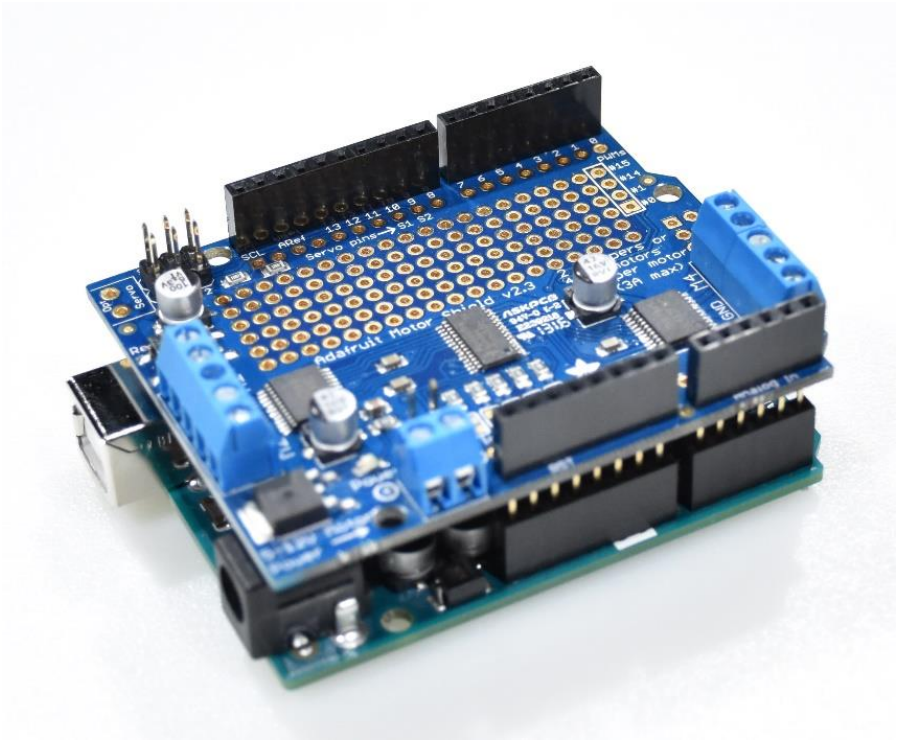


Figure 4-17. Connect the Adafruit Motor/Stepper/Servo Shield on top of the Arduino UNO. Make sure that the connecting pins are properly aligned with the headers of the Arduino UNO.

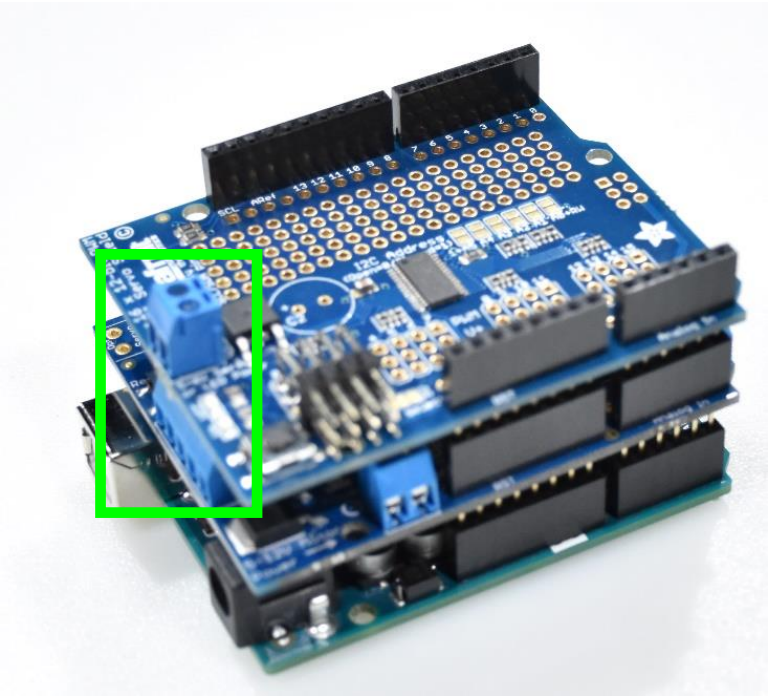


Figure 4-18. Connect the Adafruit 16-Channel 12-Bit PWM/Servo Shield on top of the Adafruit Motor/Stepper/Servo Shield. Make sure that the extensions from the power connector of the Adafruit PWM/Servo Shield are not touching the connectors of the Adafruit Motor/Stepper/Servo Shield.

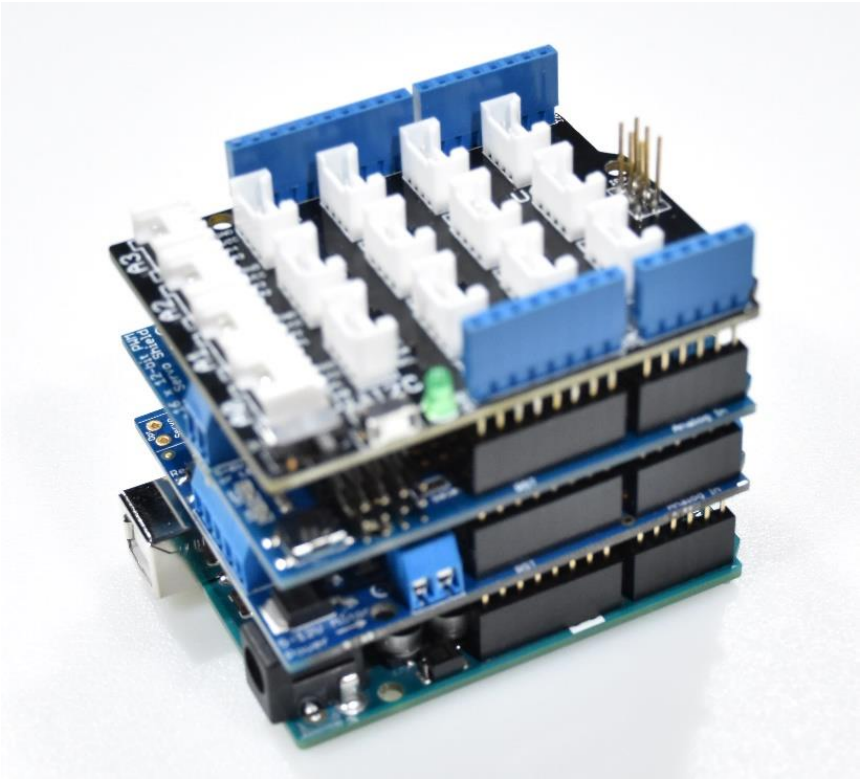


Figure 4-19. Connect the Grove Base Shield on top of the Adafruit 16-Channel 12-Bit PWM/Servo Shield.

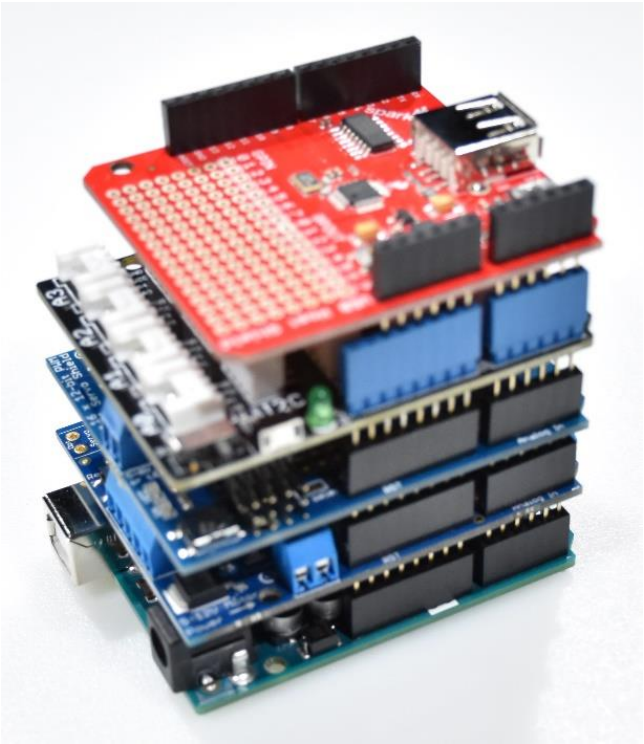


Figure 4-20. Connect the SparkFun USB Host Shield on top of the the Grove Base Shield.

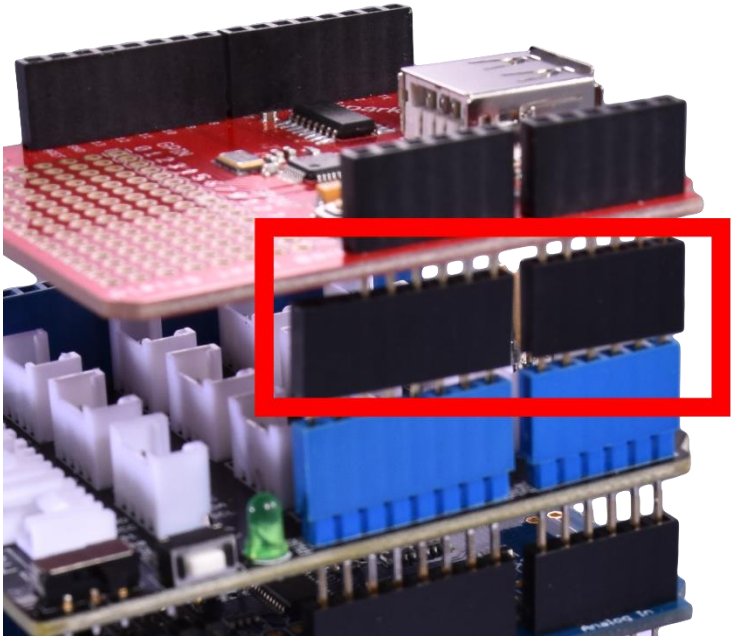


Figure 4-21. To facilitate access to the analog and digital sockets of the Grove Base Shield, one can place headers as spacers between the Grove Base Shield and the SparkFun USB Host Shield.

After stacking all the shields together, some of the Arduino's digital pins will be taken by the SparkFun USB Host Shield. The digital pins that remain available are **D0** to **D6**. All the analog pins (i.e., pins **A0**

to **A5**) remain available. Figure 4-22 indicates all the digital and analog pins that remain available. You can use those pins to connect sensors or actuators.

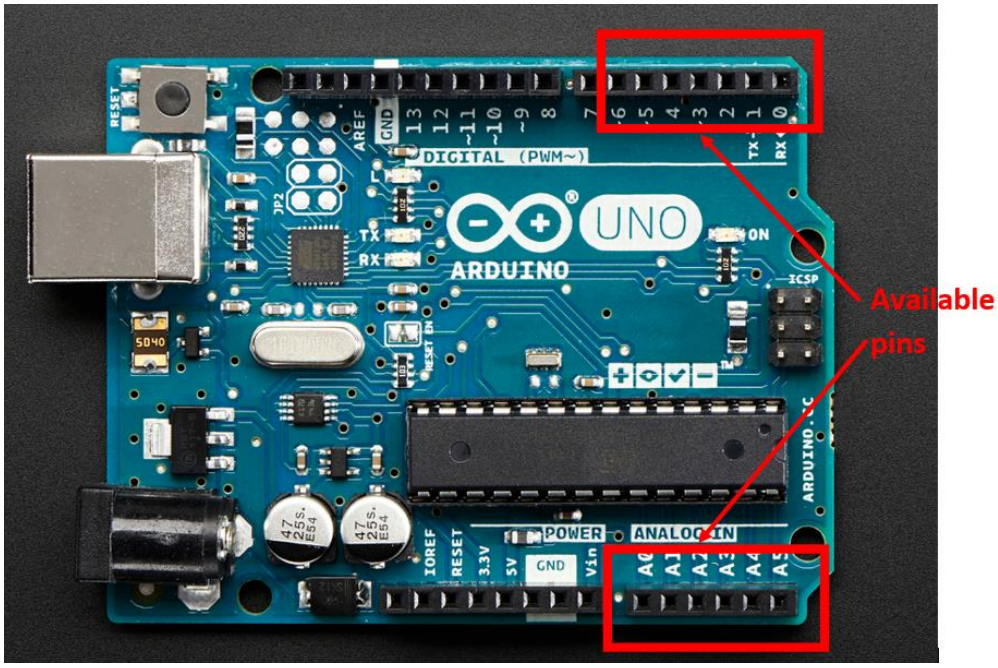


Figure 4-22. Arduino digital and analog pins that remain available after stacking all the shields.

APPENDIX A – Pictures Showing the Layout of the Arduino Uno and the Shields in the Kit

When shields are stacked, it is not possible to see the information printed on the boards. This section has pictures showing the layout of the Arduino Uno as well as the other shields in the kit.

A.1 Arduino UNO

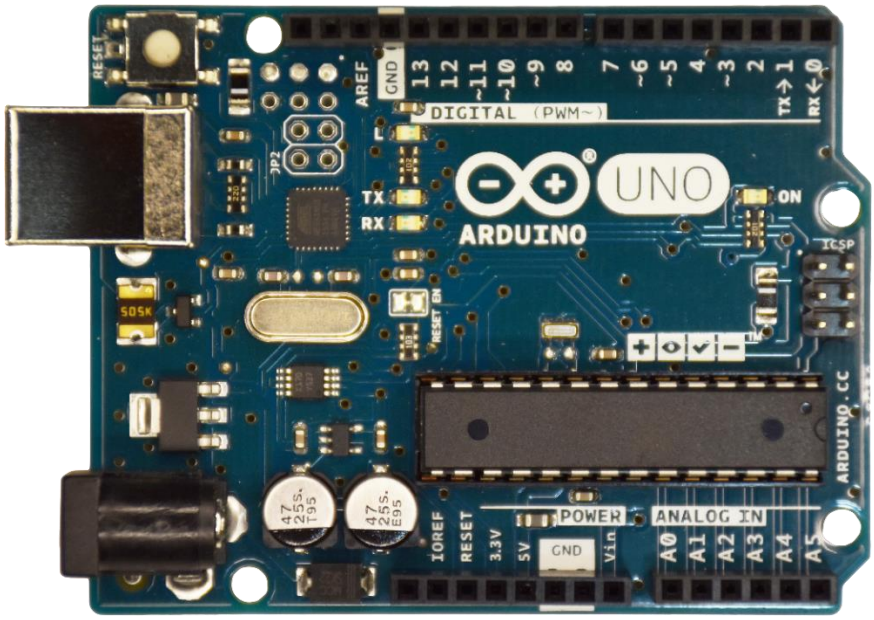


Figure A-1. Top of the Arduino UNO.

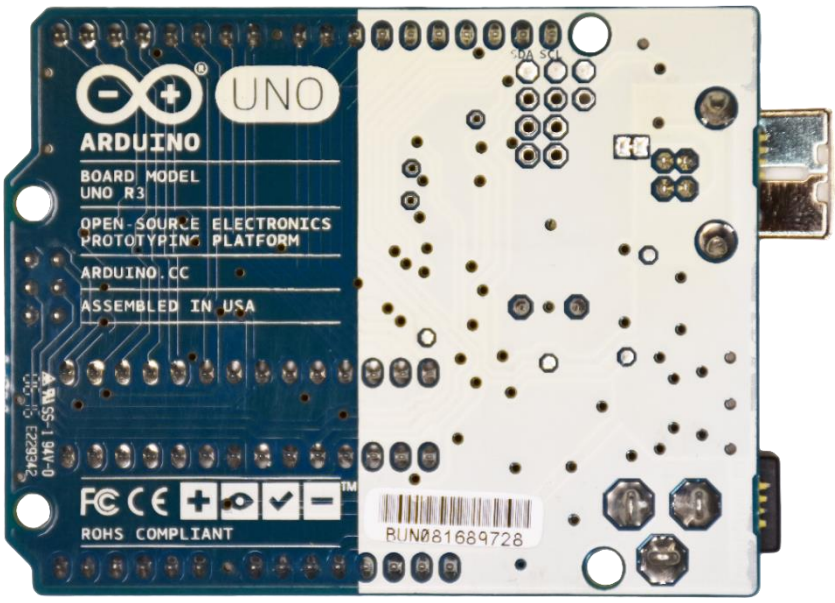


Figure A-2. Bottom of the Arduino UNO.

A.2 Grove Base Shield

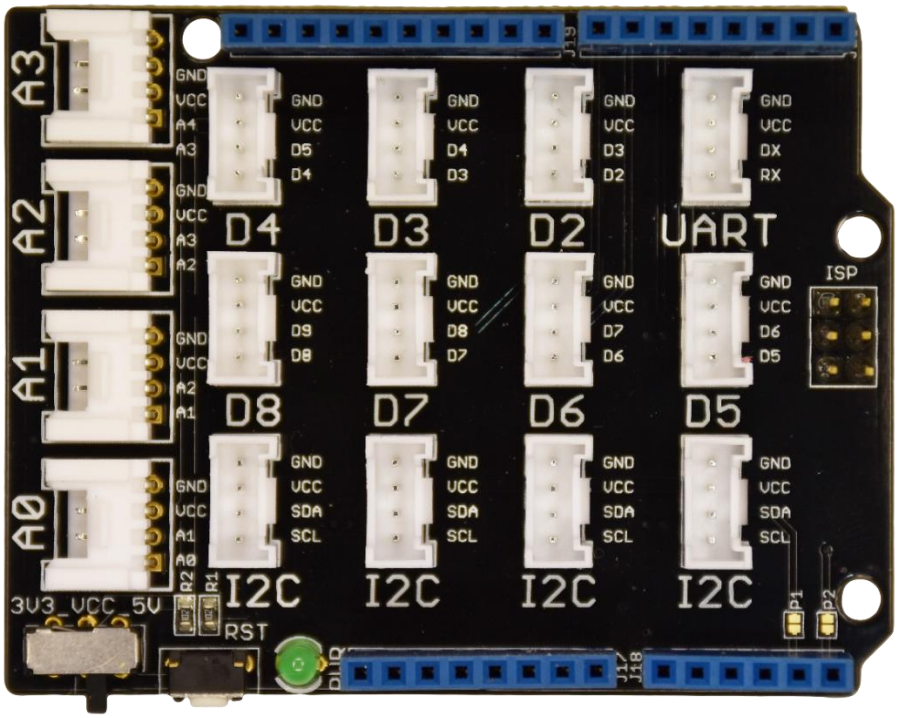


Figure A-3. Top of the Grove Base Shield.

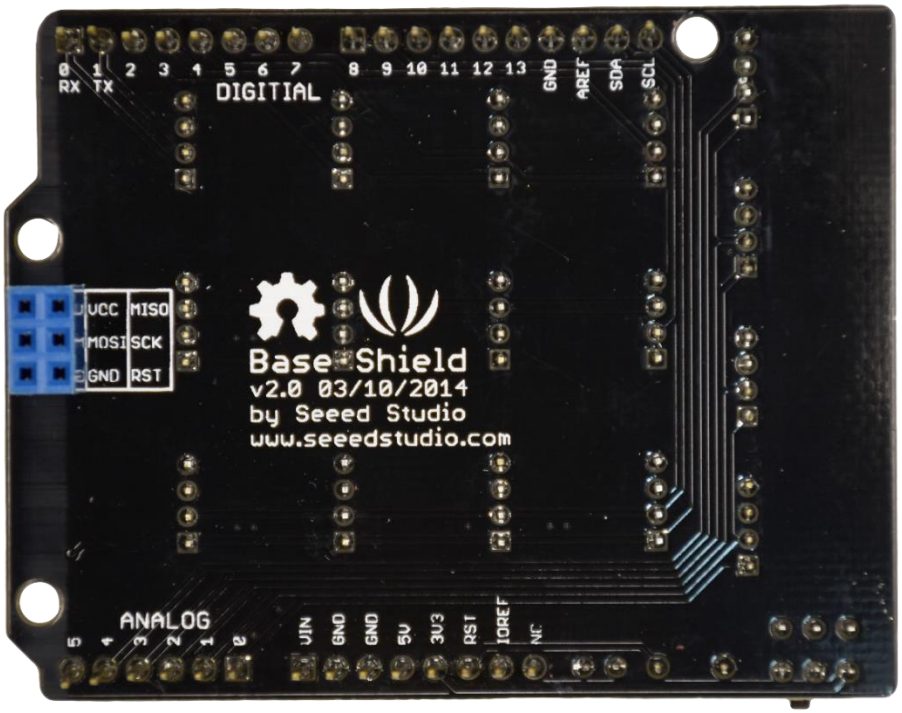


Figure A-4. Bottom of the Grove Base Shield.

A.3 Adafruit Motor/Stepper/Servo Shield

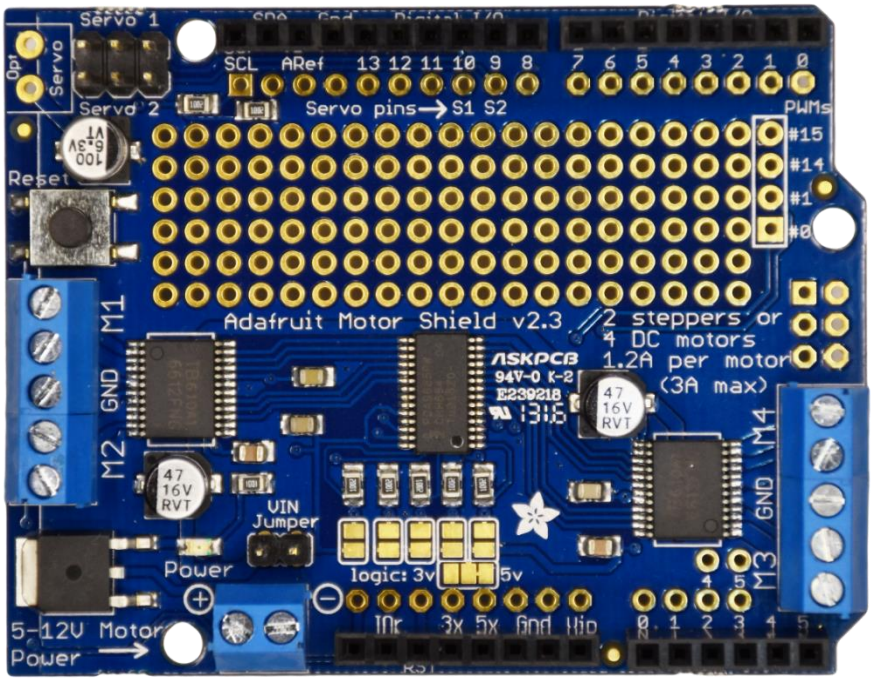


Figure A-5. Top of the Adafruit Motor Shield.

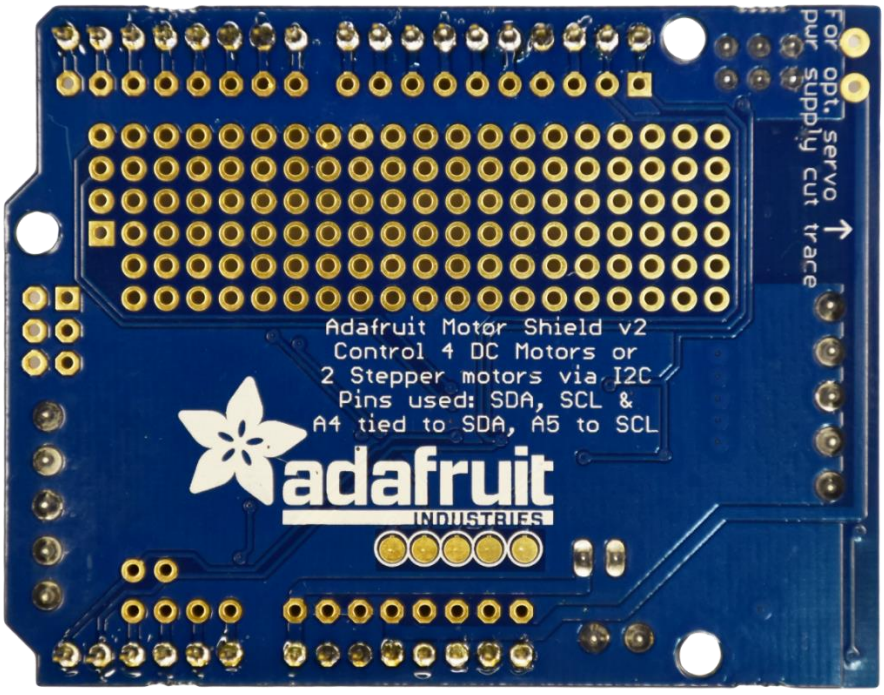


Figure A-6. Bottom of the Adafruit Motor Shield.

A.4 Adafruit Servo Shield

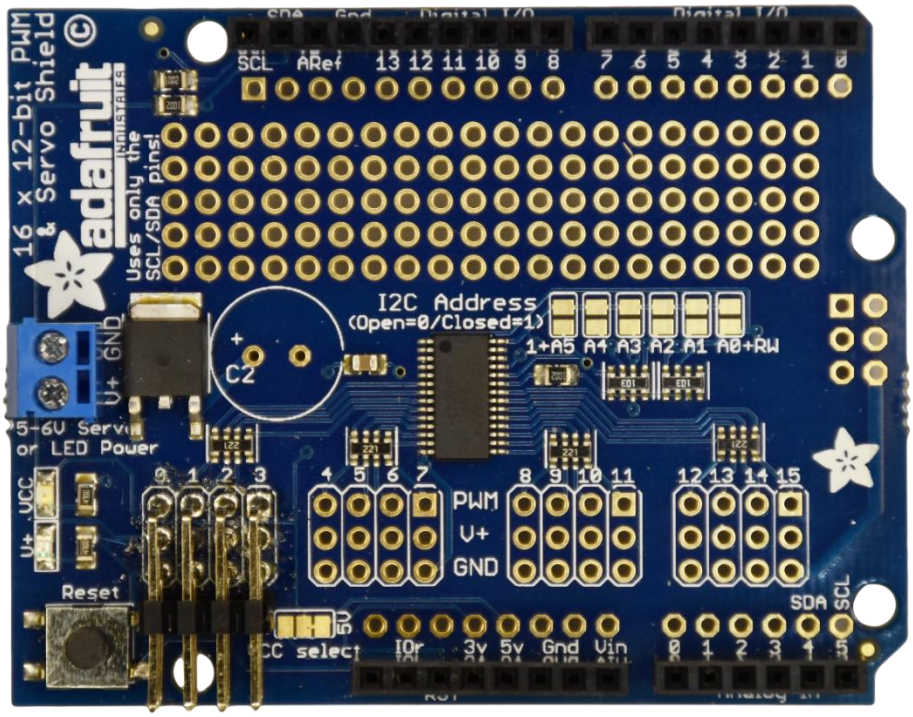


Figure A-7. Top of the Adafruit Servo Shield.

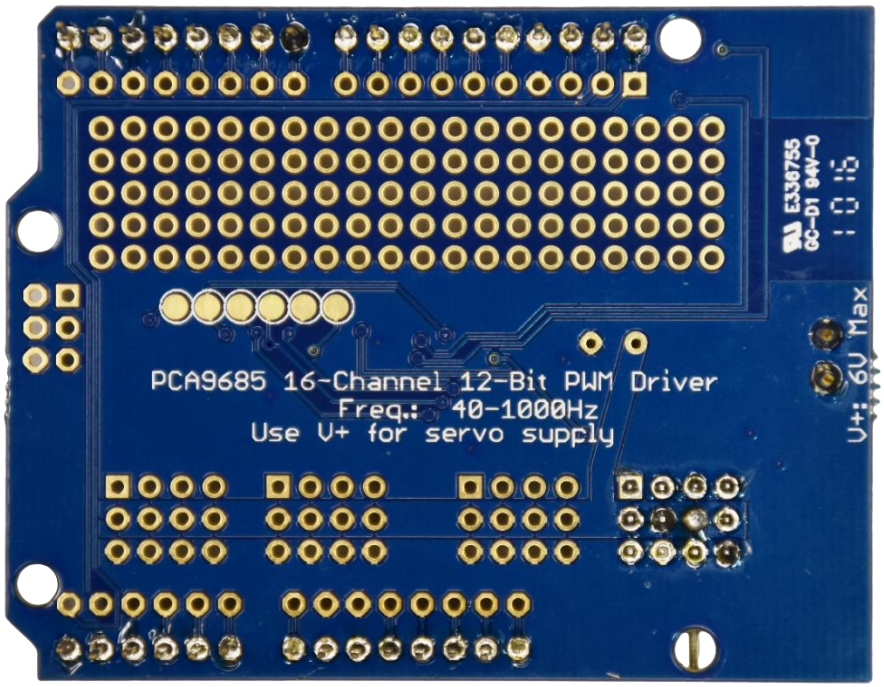


Figure A-8. Bottom of the Adafruit Servo Shield.

A.5 SparkFun USB Host Shield

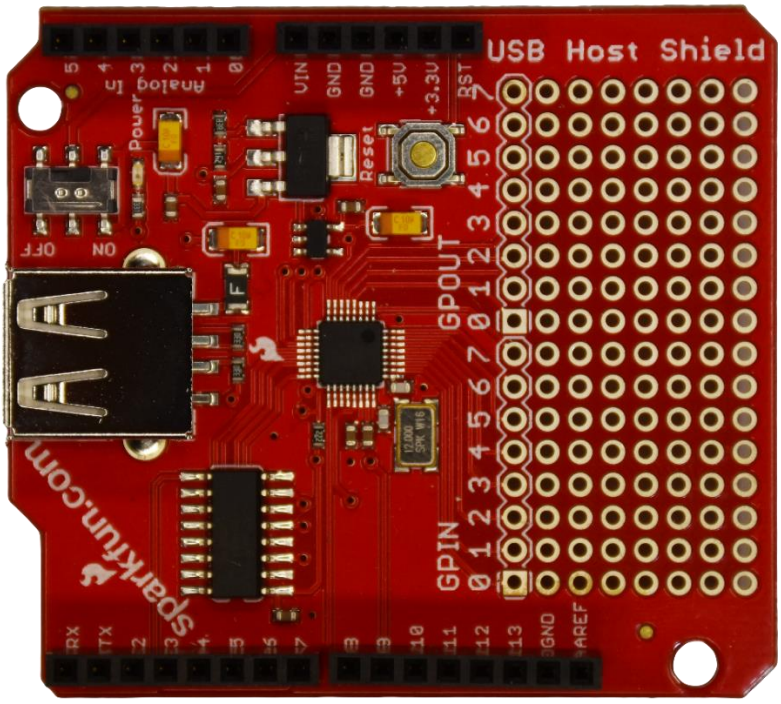


Figure A-9. Top of the USB Host Shield.

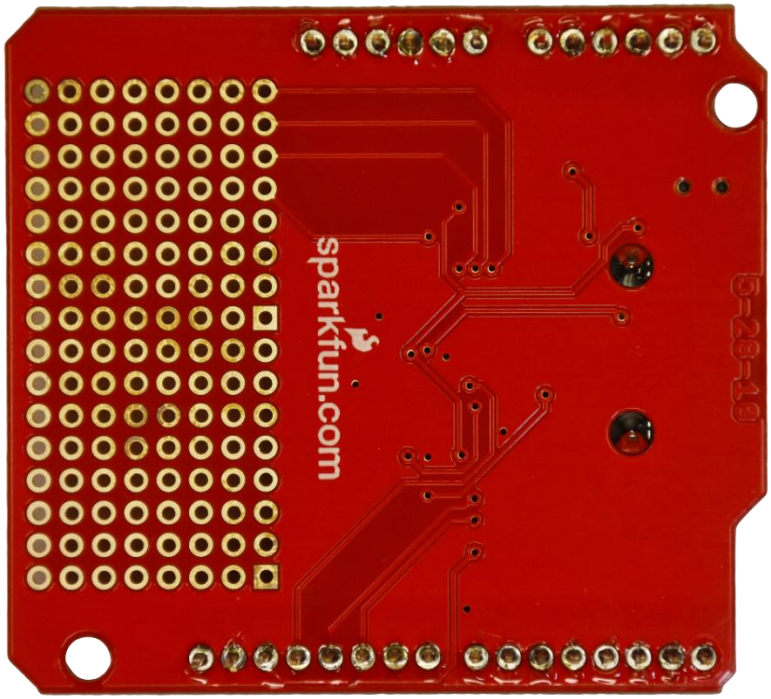


Figure A-10. Bottom of the USB Host Shield.

APPENDIX B - Possible Suppliers

The following table lists possible suppliers for the different components of the hardware kit. The part numbers used by the possible suppliers and the estimated unit cost for each part are provided for reference purposes.

ITEM	SUPPLIER	PART NUMBER	QTY	UNIT PRICE
Microcontroller/Shields				
Arduino UNO Rev3	Adafruit	50	1	$24.95
Adafruit Motor/Stepper/Servo Shield	Adafruit	1438	1	$19.95
Adafruit 16-Channel 12-bit PWM/Servo Shield - I2C interface	Adafruit	1411	1	$17.50
SparkFun USB Host Shield	Sparkfun	DEV-09947	1	$24.95
Grove Components				
Grove Starter Kit for Arduino	Seeed Studio	110060024	1	$49.90
Extra Buttons	Seeed Studio	101020003	2	$1.90
Extra LED Modules	Seeed Studio	104030009	3	$1.90
Other Components				
USB Cable A to B - 6 Foot	Sparkfun	CAB-00512	1	$3.95
SparkFun USB Mini-B Cable - 6 Foot	Sparkfun	CAB-11301	1	$3.95
Arduino Stackable Header Kit	Sparkfun	10007	3	$1.50
Arduino Uno Enclosure	Sparkfun	12839	1	$7.95
Battery Holder 4xAA with Cover and Switch	Sparkfun	12083	1	$1.95
Holder 8 AA Cell W/Cover and switch	Digi-Key	BK-6049-ND	1	$6.53
Bluetooth 4.0 USB Module (v2.1 Back-Compatible)	Adafruit	1327	1	$11.95
Shield stacking headers for Arduino (R3 Compatible)	Adafruit	85	1	$1.95
8 x AA battery holder with 5.5mm/2.1mm Plug and On/Off Switch	Adafruit	875	1	$5.95
3x4 Right Angle Male Header - 4 pack	Adafruit	816	1	$2.95
Sony Dual Shock 3 - Black (PS3)	WalMart	000599384	1	$49.99

APPENDIX C - Required Hardware Preparations

Before putting the boards in the kit together, you need to assemble the Adafruit Motor/Stepper/Servo Shield, the Adafruit 16-Channel 12-bit PWM/Servo Shield, and the SparkFun USB Host Shield. Once you have assembled those shields, you can proceed to stack all the shields on top of the Arduino UNO in the order indicated in Chapter 4.

C.1 Adafruit Motor/Stepper/Servo Shield Assembly

Before you can start using the Adafruit Motor/Stepper/Servo Shield, you need to assemble the parts. This requires some soldering. If you don't know how to solder, please refer to the steps given in the following website: https://learn.sparkfun.com/tutorials/how-to-solder---through-hole-soldering.

Follow the steps in Figure C-1 to Figure C-4 to assemble the motor shield.

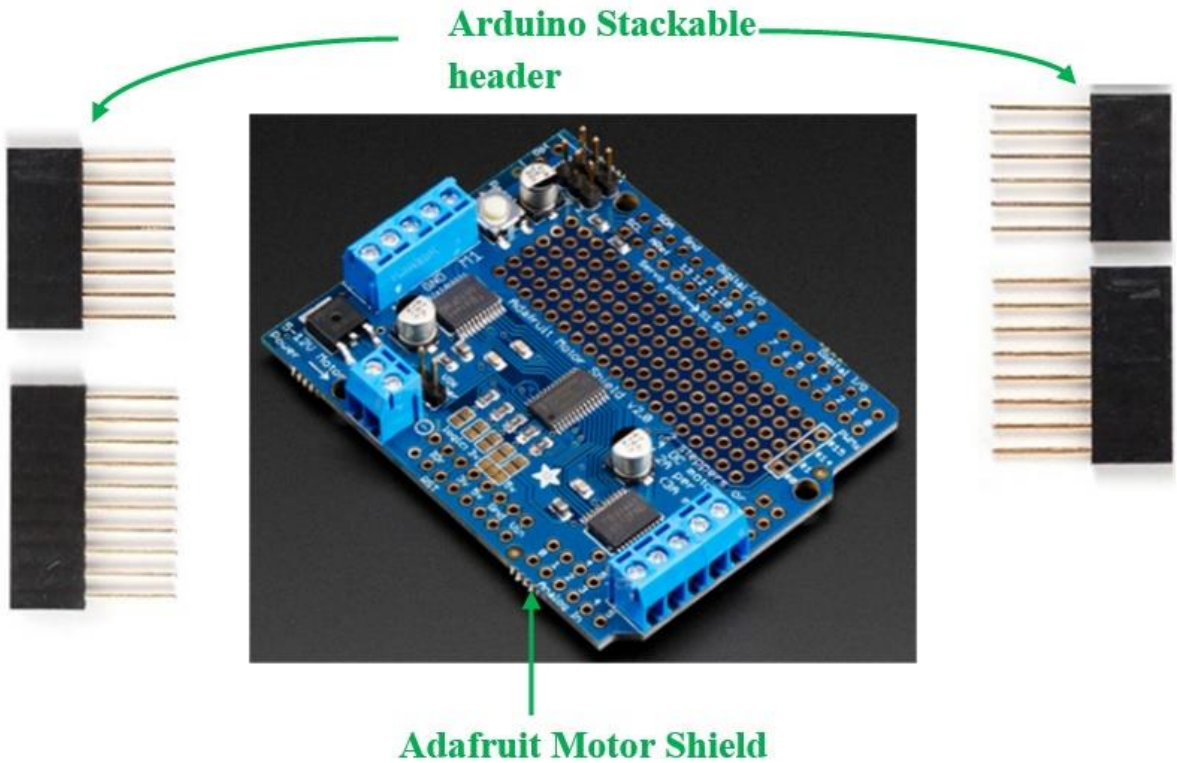


Figure C-1. Components of the Adafruit Motor shield.

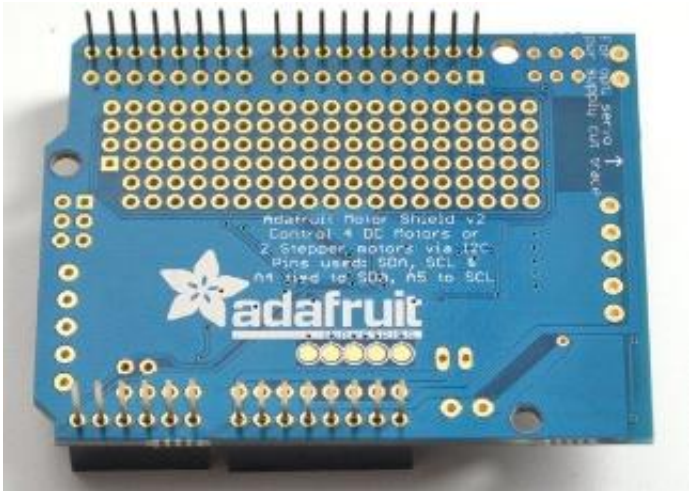


Figure C-2. Start by sliding the 10-pin, the two 8-pin and the 6-pin stacking headers into the outer rows of the top side of the shield. Then flip the board over so that it is resting on the four headers.

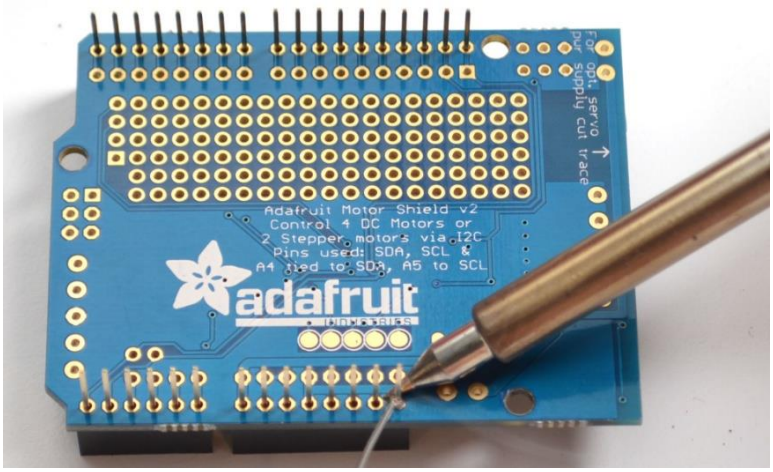


Figure C-3. Tack one pin of each header to set the headers in place before doing more soldering.

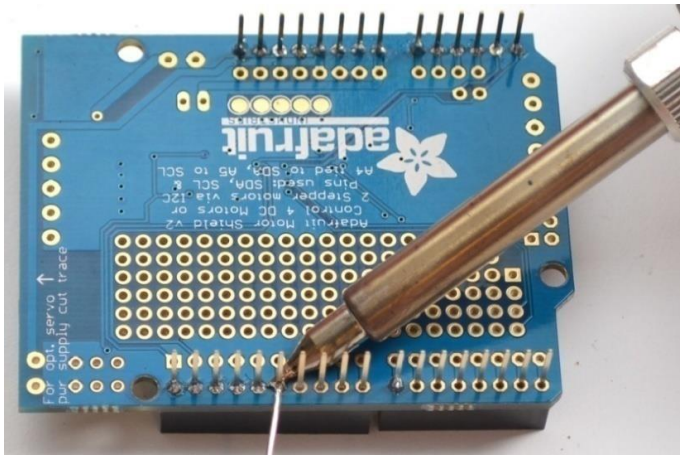


Figure C-4. Once you have tacked and straightened all the headers, solder the remaining pins of each header.

C.2 Adafruit 16-Channel 12-bit PWM/Servo Shield Assembly

The Adafruit 16-channel 12-bit PWM/Servo Shield requires the following parts: Arduino stackable headers, one power connector, and one 3x4 right angle male header. Follow the steps shown in Figure C-5 to Figure C-9 to completely assemble the 16-Channel 12-bit PWM/Servo shield before using it.

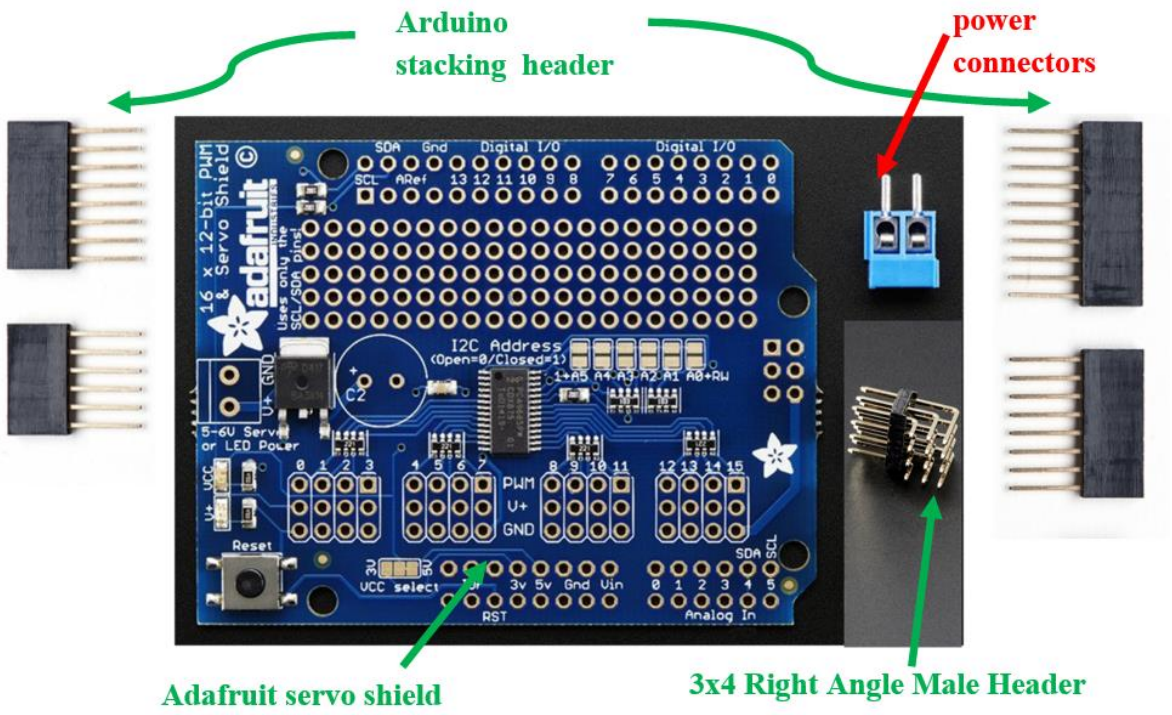


Figure C-5. Components of Adafruit 16-Channel 12-bit PWM/Servo shield

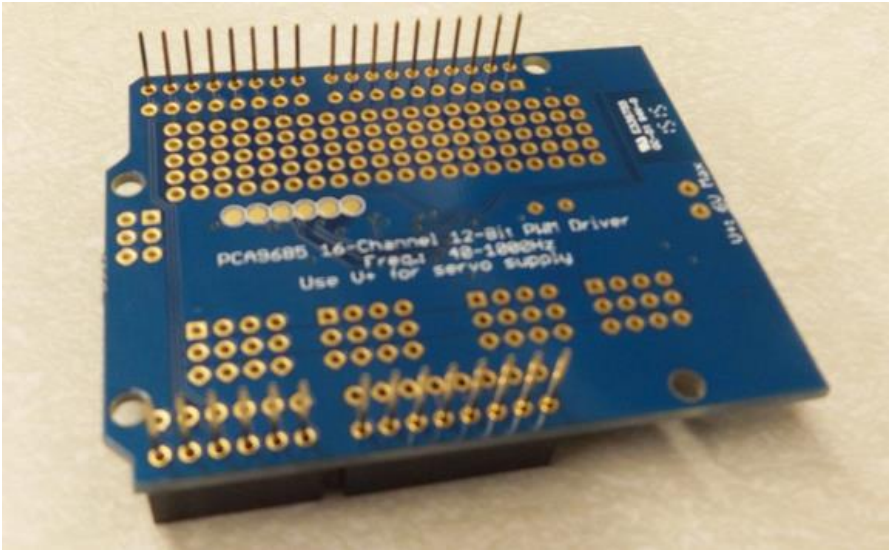


Figure C-6. Start by sliding the 10-pin, the two 8-pin and the 6-pin stacking headers into the outer rows of the top side of the shield. Then flip the board over so that it is resting on the four headers.

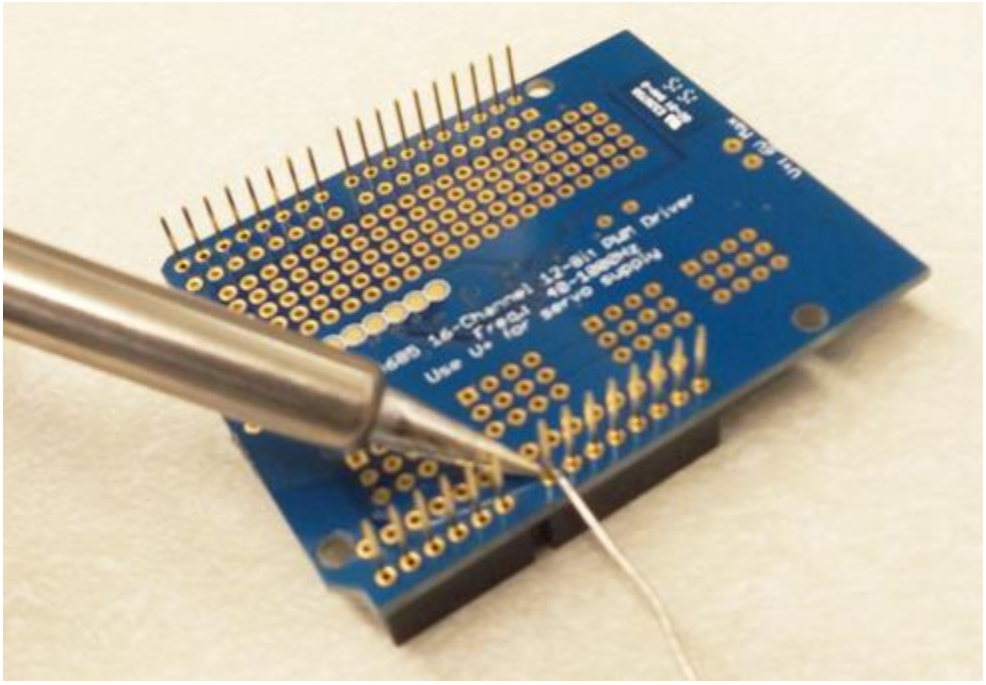


Figure C-7. Tack one pin of each header to set the headers in place. Once you have tacked and straightened all the headers, solder the remaining pins of each header.

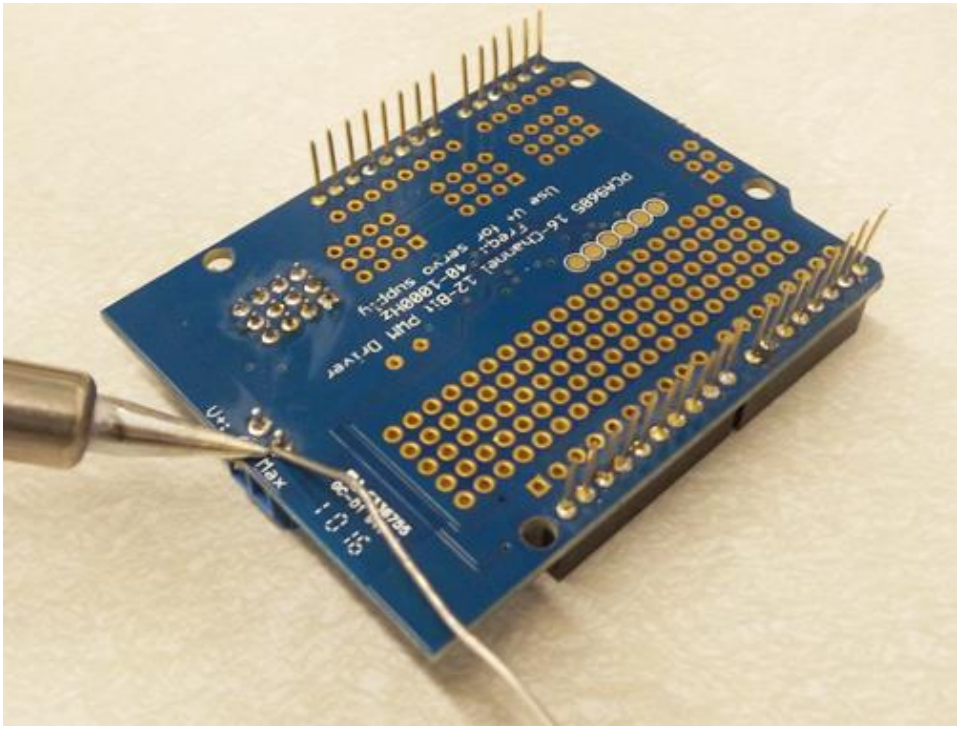


Figure C-8. Insert the power conector terminals and solder them as shown.

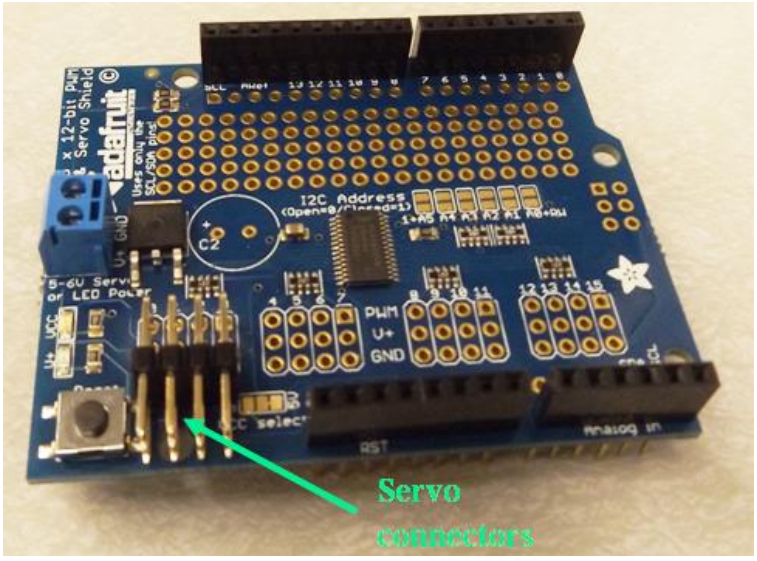


Figure C-9. Insert the 3x4 right angle male header and solder its pins on the bottom side of the shield. Note: As shown in the picture, we are going to place the 3x4 right angle male header in a position that does not interfere with the stackable header for the Arduino pins.

C.3 SparkFun USB Host Shield Assembly

The SparkFun USB Host Shield and the Arduino stackable headers come as separate parts. To use the shield with the Arduino UNO, we need to solder the headers pins first. Follow the steps shown in Figure C-10 to Figure C-12 to completely assemble the USB host shield.

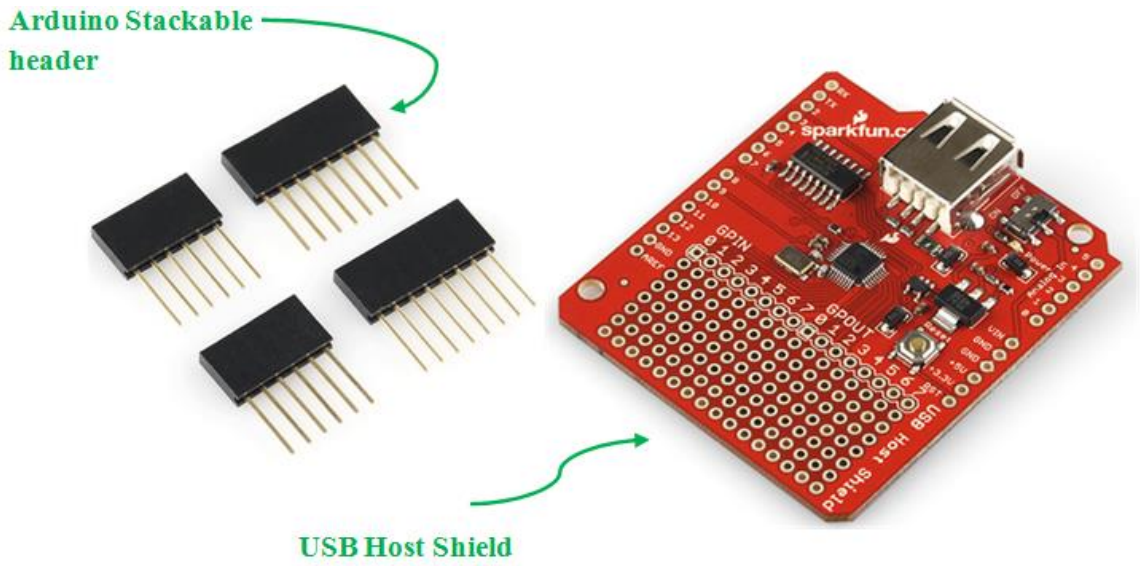


Figure C-10. Components of the SparkFun USB Host Shield.

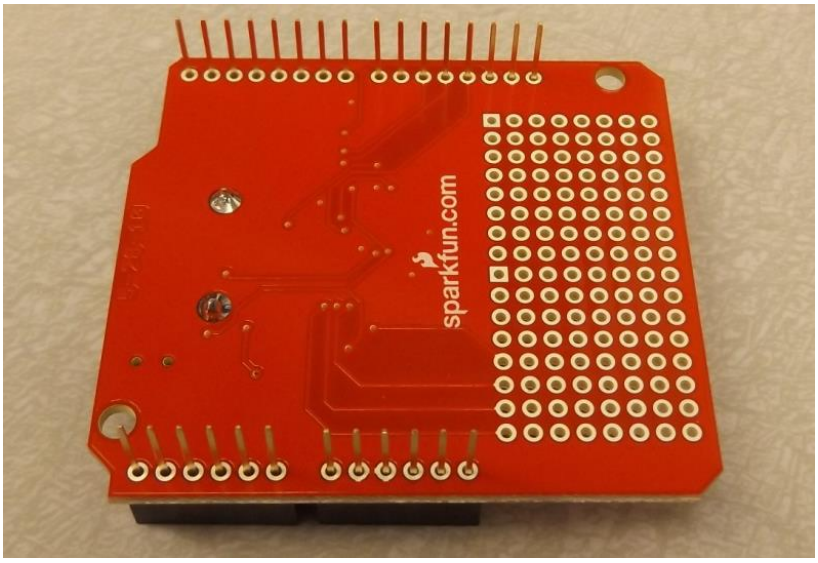


Figure C-11. Insert the two 8-pin and the two 6-pin Arduino stackable headers as shown. Then flip the board over so it is resting on the four headers.

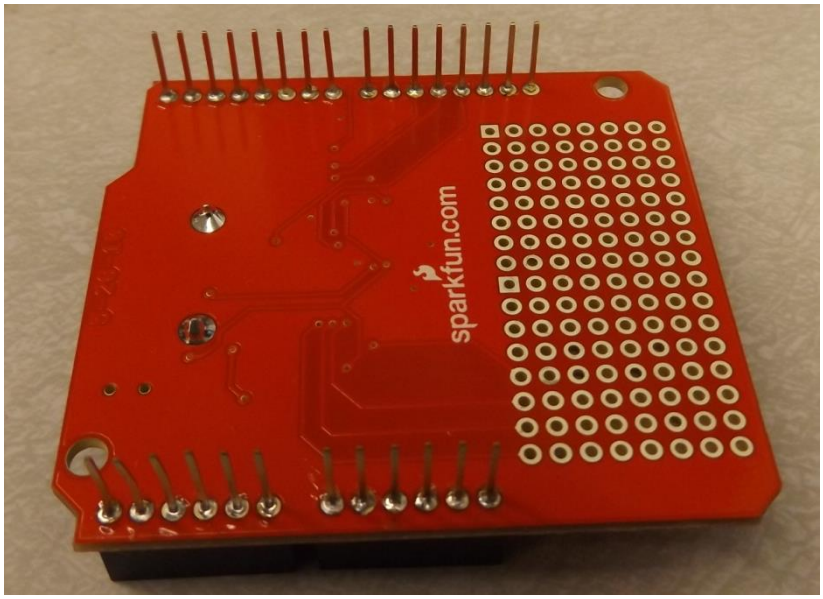


Figure C-12. Tack one pin of each header to set the headers in place. Once you have tacked and straightened all the headers, solder the remaining pins of each header.

Now the Adafruit Motor/Stepper/Servo Shield, the Adafruit 16-Channel 12-bit PWM/Servo Shield, and the SparkFun USB Host Shield are fully assembled and ready for use. These shields will be stacked together with the Arduino UNO Rev3 and the Grove Base Shield and used as a complete hardware kit (see Chapter 4).

APPENDIX D - PWM Concept

Pulse Width Modulation (PWM) is a technique for getting analog results by digital means. Digital control is used to create a square wave: a signal switched between **ON** and **OFF**. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal is **ON** versus the time that the signal is **OFF**. The duration of the "**ON** time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0V and 5V controlling the brightness of the LED.

In Figure D-1 the separation between the vertical green lines represents a specified time period. This time duration or period is the inverse of the PWM frequency. With Arduino's PWM frequency at about 500Hz, the time between two consecutive green lines would be about 2 milliseconds. The PWM value ranges between 0 and 255. The command ***analogWrite(255)*** induces a 100% duty cycle (always **ON**), and ***analogWrite***(127) is a 50% duty cycle (**ON** half of the time). Likewise ***analogWrite(64)*** is a 25% duty cycle.

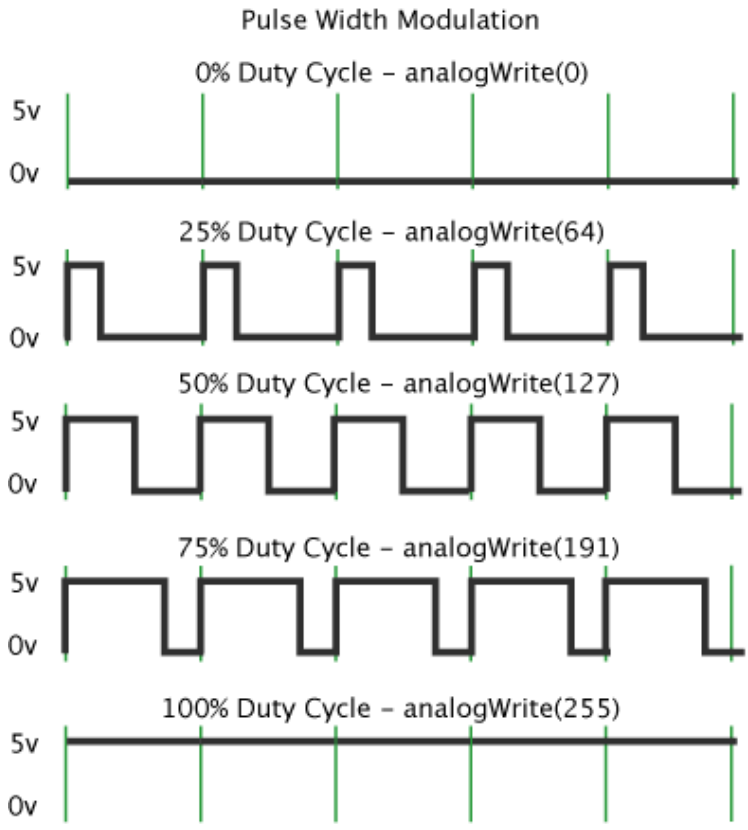


Figure D-1. Illustration of Pulse Width Modulation (PWM)

For more detail explanations and examples on PWM see the following web sites:

- https://learn.sparkfun.com/tutorials/pulse-width-modulation
- https://en.wikipedia.org/wiki/Pulse-width_modulation

REFERENCES

1. Blum, Richard. Sams Teach Yourself Arduino Programming in 24 Hours. Pearson Education, Inc. 2015.
2. https://en.wikipedia.org/wiki/Arduino
3. https://www.arduino.cc/
4. https://www.seeedstudio.com/Grove-Starter-Kit-for-Arduino-p-1855.html
5. http://www.seeedstudio.com/wiki/Base_shield_v2.
6. https://www.adafruit.com/product/1438
7. https://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield/overview?view=all
8. https://www.sparkfun.com/products/9947
9. https://www.adafruit.com/products/1327
10. http://www.vexrobotics.com/motors.html
11. http://curriculum.vexrobotics.co.uk/curriculum/speed-power-torque-and-dc-motors/dc-motors