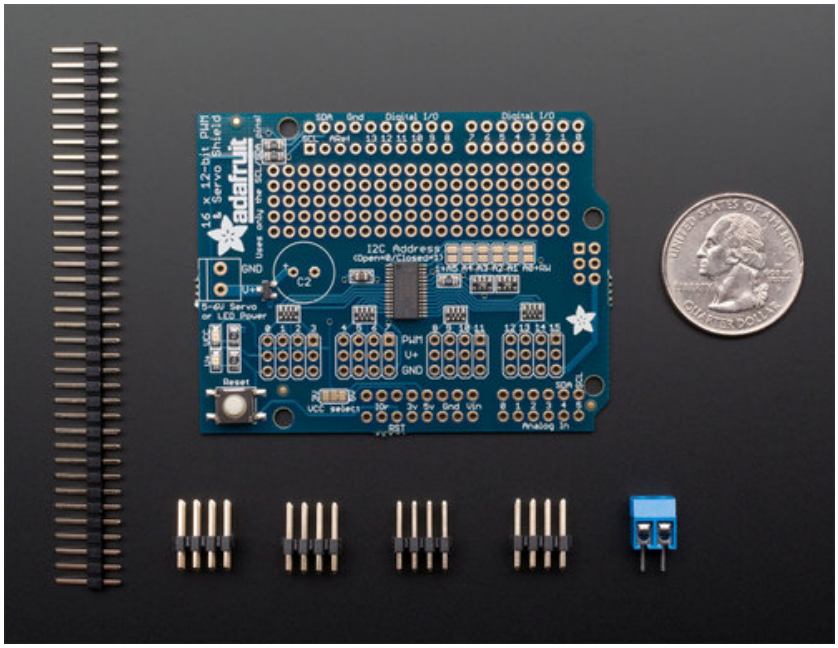
adafruit learning system

Adafruit 16-channel PWM/Servo Shield

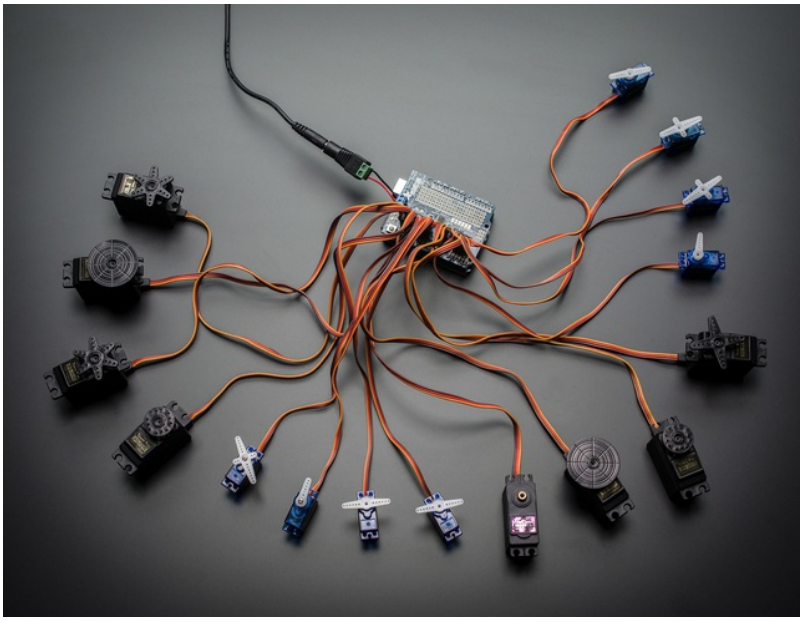
Created by Ladyada



Last updated on 2013-06-20 04:30:24 PM EDT

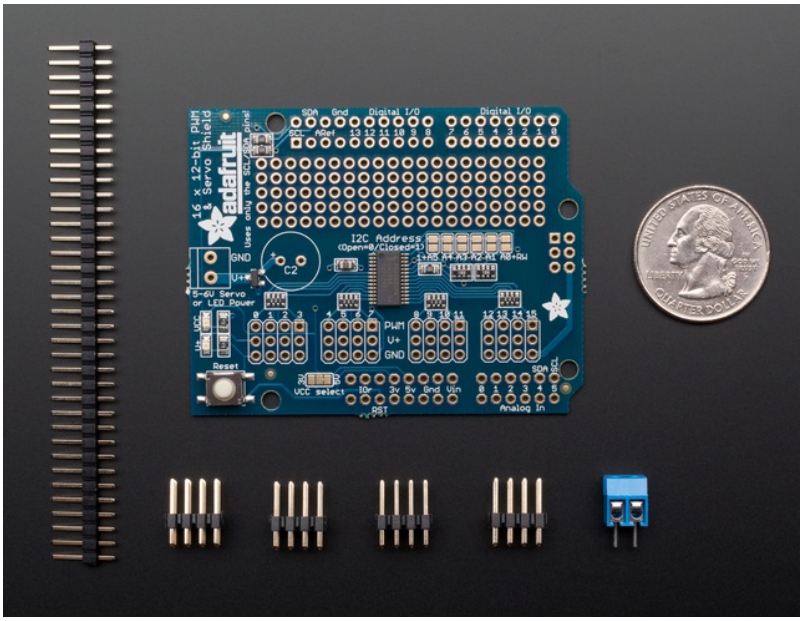
Guide Contents

Overview

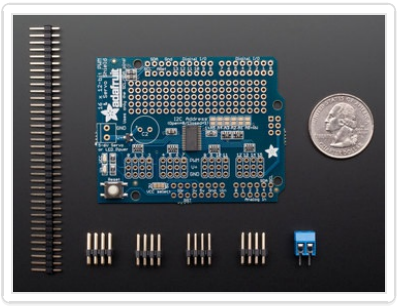


Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power. The Adafruit 16-Channel 12-bit PWM/Servo Driver Shield will drive up to 16 servos over I2C with only 2 pins. The on-board PWM controller will drive all 16 channels simultaneously with no additional Arduino processing overhead. What's more, you can stack up to 62 of them to control up to 992 servos - all with the same 2 pins!

The Adafruit PWM/Servo Driver is the perfect solution for any project that requires a lot of servos!

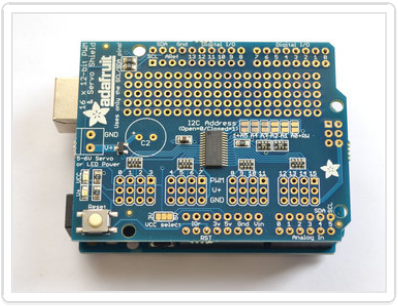


Assembly

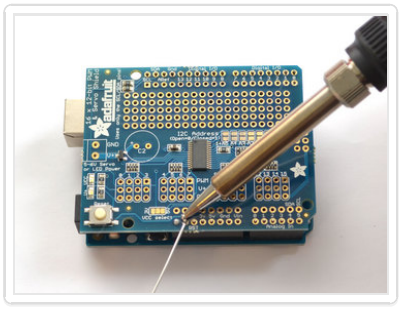


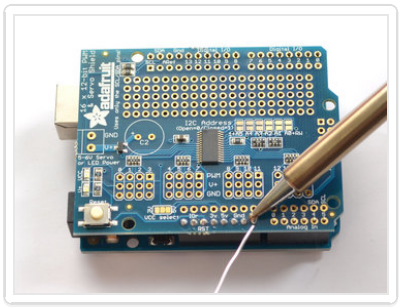
Check you have everything you need: assembled shield PCB, 0.1" male header, 4 of 3x4 male header, and a 2 pin terminal block

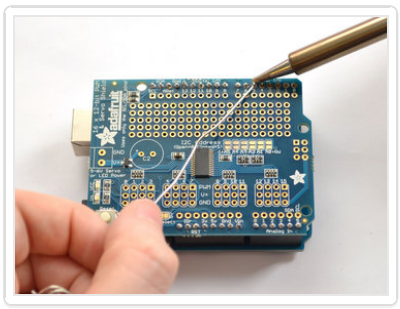
Break apart the 0.1" header into 6, 8 and/or 10-pin long pieces and slip the long ends into the headers of your Arduino

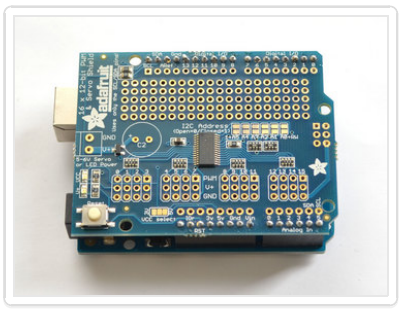


Place the shield on top of the header pins, they should fit into each of the holes along the edge

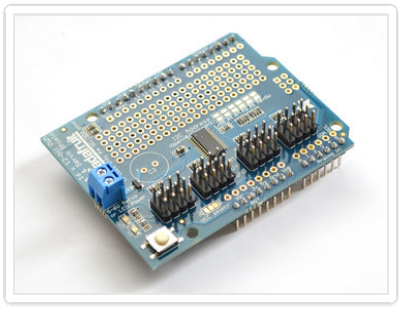


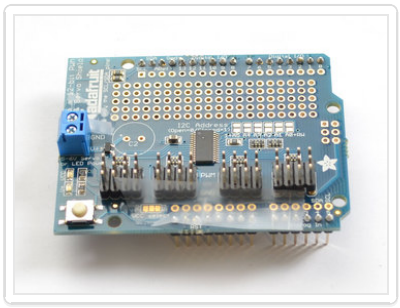






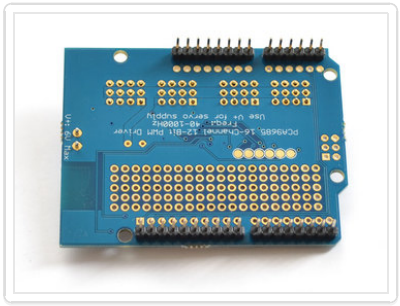
Solder each of the pins to secure the shield to the headers

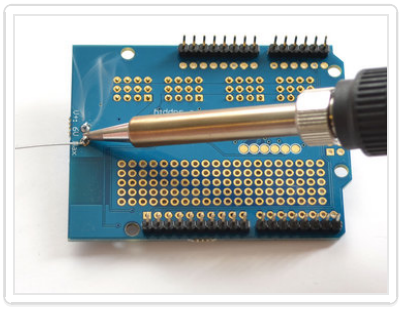




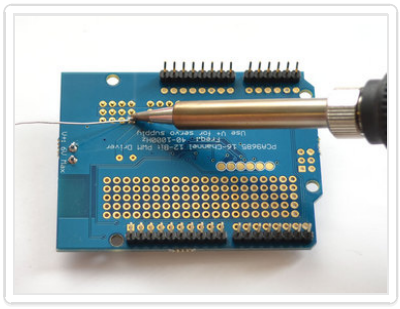
Place the 2 pin terminal block so it faces out. Also place the 3x4 headers so the short pins are plugged into the shield and the long pins are sticking up

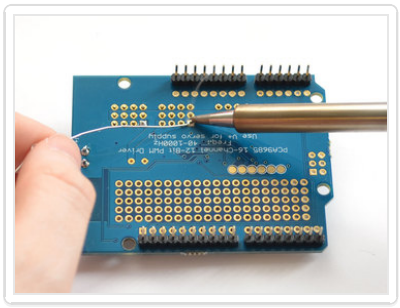
To keep them in place, a few pieces of tape will hold them for when you flip the board over.

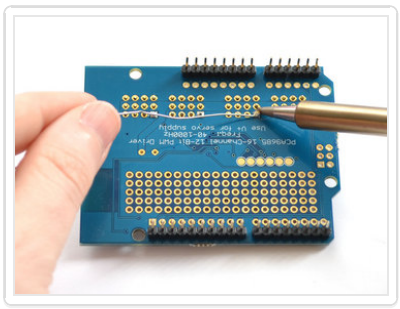


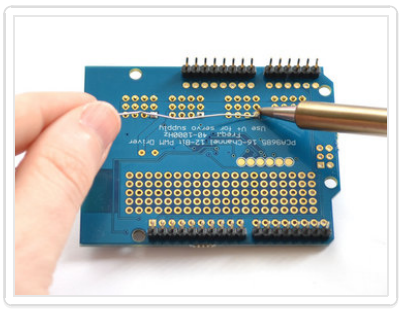


Flip the board over and solder the terminal block

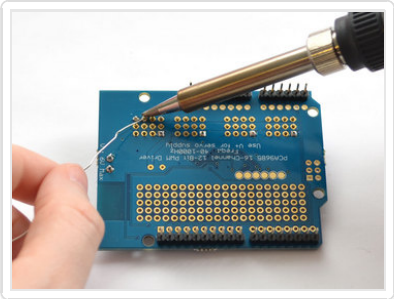


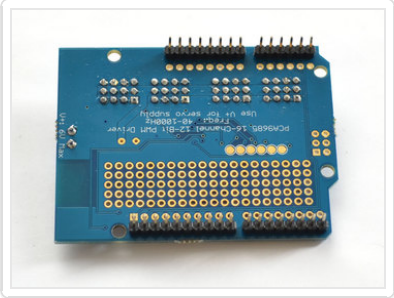




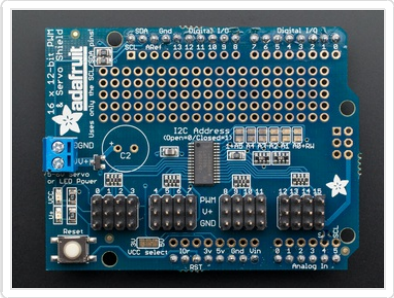


Then solder one pin of each 3x4 header to tack them in place.





Finally...solder every pin of the 3x4 headers



You're done! Go onto the next sections to learn how to use your servo/pwm shield

---

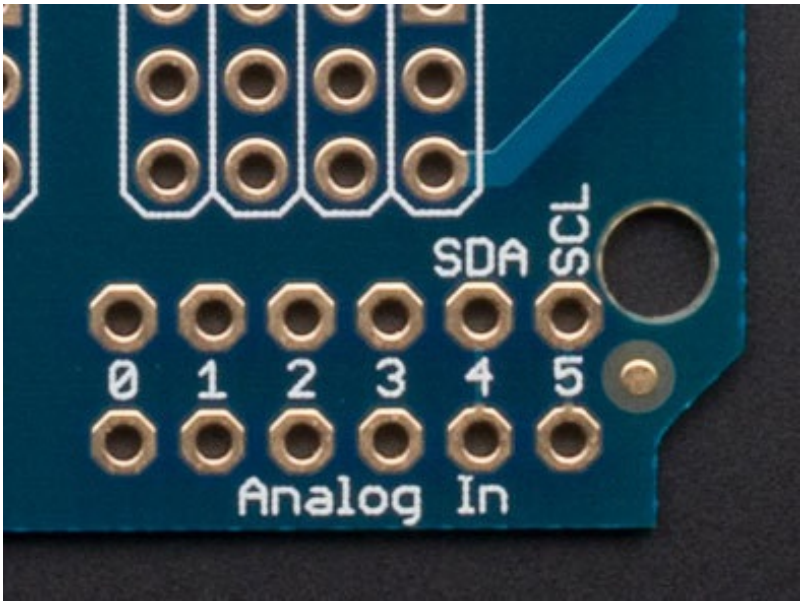
Shield Connections

Pins Used

The shield plugs in directly into any shield-compatible Arduino such as Duemilanove, Diecimila, UNO, Leonardo, Mega R3+, ADK R3+. The only pins required to run are the **Ground**, **5V** and **SDA** + **SCL** I2C control pins.

For backwards compatibility with old Ardunos, **SCL** is connected to **A5** and SDA is connected to **A4**. UNOs already have this connection on board. If you are using a Leonardo or Mega and want to use the A4/A5 pins, cut the traces on the top of the board between A4 and A5 and the two pins next to them labeled SCL/SDA.

If you are using a Mega or ADK R2 or earlier, you will have to solder a wire from **SCL** to **D21** and **SDA** to **D20**



Connecting other I2C devices

Since I2C is a 'shared bus' you can still connect other I2C devices to the SDA/SCL pins as long as they do not have a conflicting address. The default address for the shield is address 0x40

Powering Servos / PWM

This shield has **two** power supplies. One is VCC - that is the 5V power from the Arduino, it is

used to power the PWM chip and determines the I2C logic level and the PWM signal logic level. When this power supply is working you will see a red LED. **The red LED must be lit for the Arduino & shield to work!** Plug in the Arduino to USB or a wall adapter to provide it.

To power servos you will need to also connect the V+ power supply - this is the power supply for the servos. (If you are lighting up single LEDs you may not need this power supply.) This power supply should be 5 or 6VDC. You can connect this power through the blue terminal block. There is reverse-polarity protection in case you hook up power backwards.

Nearly all servos are designed to run on about 5 or 6v. Keep in mind that a lot of servos moving at the same time (particularly large powerful ones) will need a lot of current. Even micro servos will draw several hundred mA when moving. Some High-torque servos will draw more than 1A each under load.

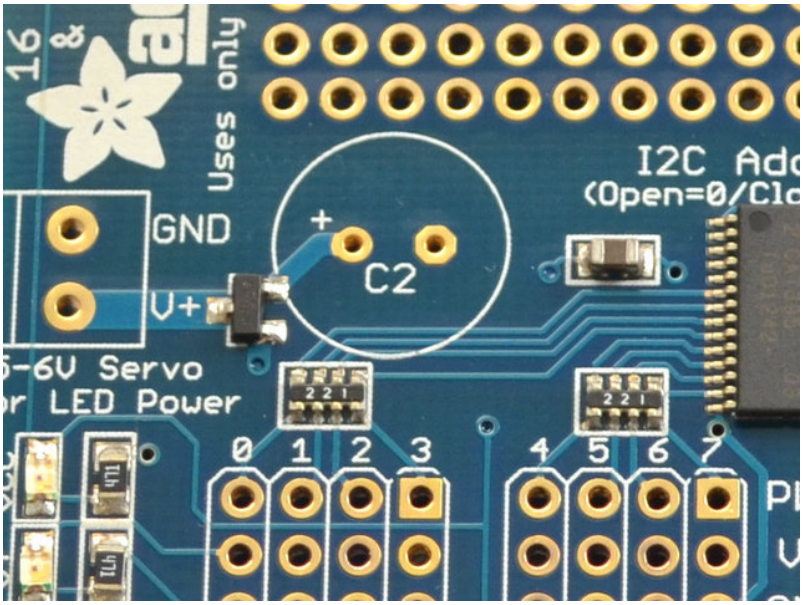
Good power choices are:

- 5v 2A switching power supply (http://adafru.it/276)
- 5v 10A switching power supply (http://adafru.it/658)
- 4xAA Battery Holder (http://adafru.it/830) - 6v with Alkaline cells. 4.8v with NiMH rechargeable cells.
- 4.8 or 6v Rechargeable RC battery packs from a hobby store.

> It is not a good idea to use the Arduino 5v pin to power your servos. Electrical noise and 'brownouts' from excess current draw can cause your Arduino to act erratically, reset and/or overheat.

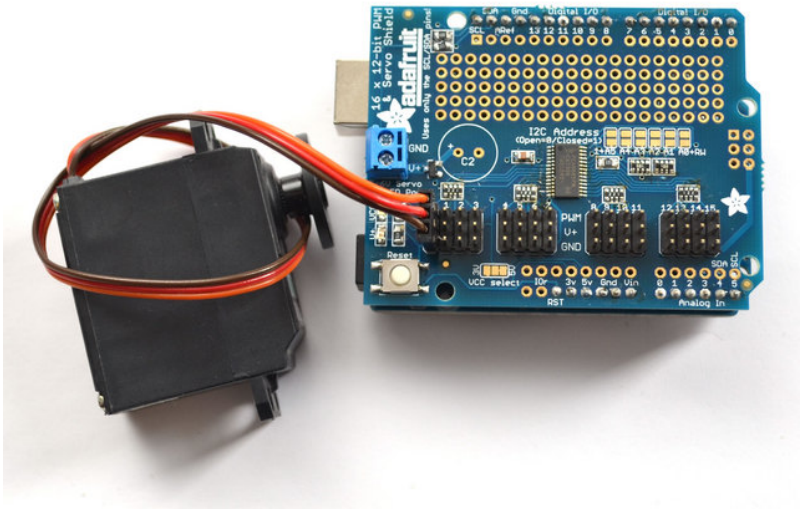
Adding a Capacitor to the thru-hole capacitor slot

We have a spot on the PCB for soldering in an electrolytic capacitor. Based on your usage, you may or may not need a capacitor. If you are driving a lot of servos from a power supply that dips a lot when the servos move, **n * 100uF** where **n** is the number of servos is a good place to start - eg **470uF** or more for 5 servos. Since its so dependent on servo current draw, the torque on each motor, and what power supply, there is no "one magic capacitor value" we can suggest which is why we don't include a capacitor in the kit.



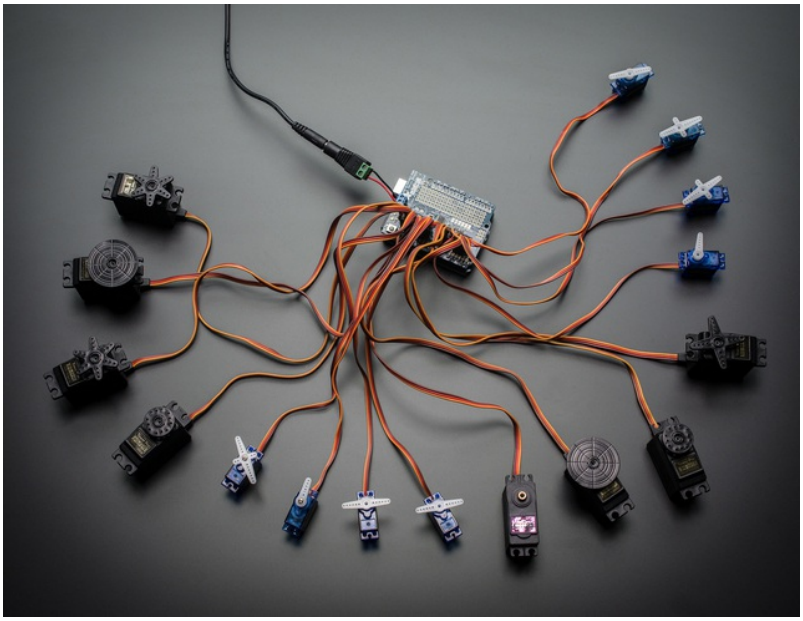
Connecting a Servo

Most servos come with a standard 3-pin female connector that will plug directly into the headers on the Servo Driver. Be sure to align the plug with the ground wire (usually black or brown) with the bottom row and the signal wire (usually yellow or white) on the top.



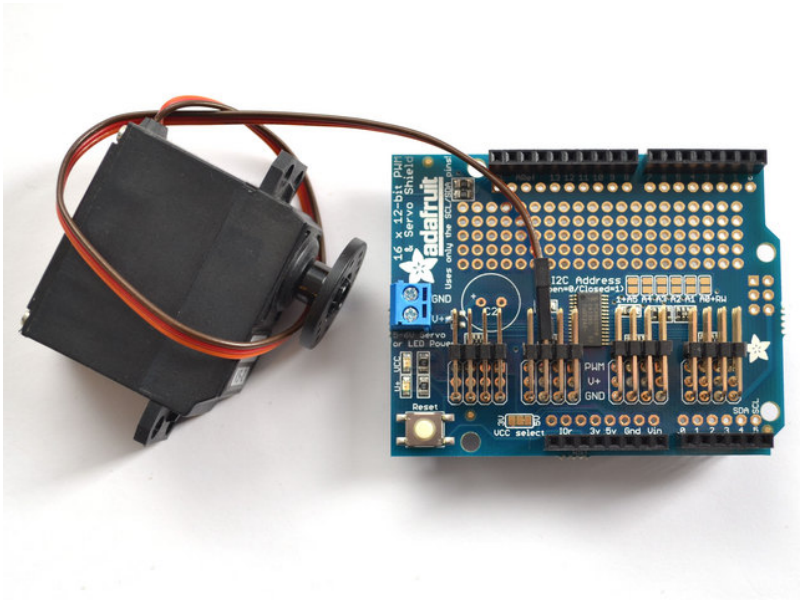
Adding More Servos

Up to 16 servos can be attached to one board. If you need to control more than 16 servos, additional boards can be stacked as described on the next page.



Stacking Shields

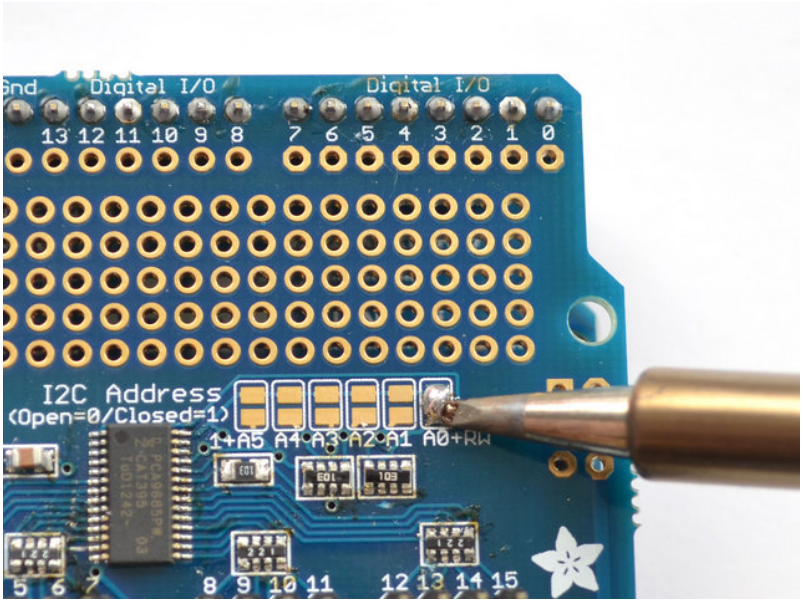
Since this shield only uses the I2C pins and the I2C bus is sharable, you can stack multiple shields on top of each other. You will need to have installed stacking headers & right angle 3x4 connections for it to physically connect. Multiple shields (up to 62!) can be stacked to control still more servos.



Addressing the Shields

Each board in the chain must be assigned a unique address. This is done with the address jumpers on the upper right edge of the board. The I2C base address for each board is 0x40. The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.



Board 0: Address = 0x40 Offset = binary 00000 (no jumpers required)
Board 1: Address = 0x41 Offset = binary 00001 (bridge A0 as in the photo above)
Board 2: Address = 0x42 Offset = binary 00010 (bridge A1)
Board 3: Address = 0x43 Offset = binary 00011 (bridge A0 & A1)
Board 4: Address = 0x44 Offset = binary 00100 (bridge A2)

etc.

Using the Adafruit Library

Download the library from Github

Start by downloading the library from https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library (http://adafru.it/aQI)- you can just click the button below

Copy the zip file to the Libraries folder inside your Arduino Sketchbook folder and re-name it to Adafruit_PWMServoDriver. For more details on how to install Arduino libraries, check out our detailed tutorial! (http://adafru.it/aYM)

Test with the Example Code:

First make sure all copies of the Arduino are closed.

Next open a new copy of the IDE and select **File->Examples->Adafruit_PWMServoDriver->Servo**. This will open the example file in an IDE window.

Plug the shield into your Arduino, and a servo into the left-most 'port' as shown on the previous page and upload the example code. Don't forget you will also have to provide 5V to the V+ terminal block. **Both red and green LEDs must be lit**.

You should see the servo sweep back and forth over approximately 180 degrees.

Calibrating your Servos

Servo pulse timing varies between different brands and models. Since it is an analog control circuit, there is often some variation between samples of the same brand and model. For precise position control, you will want to calibrate the minumum and maximum pulse-widths in your code to match known positions of the servo.

Find the MInimum:
Using the example code, edit SERVOMIN until the low-point of the sweep reaches the minimum range of travel. It is best to approach this gradually and stop before the physical limit of travel is reached.

Find the Maximum:
Again using the example code, edit SERVOMAX until the high-point of the sweep reaches the maximum range of travel. Again, is best to approach this gradually and stop before the physical limit of travel is reached.

> Use caution when adjusting SERVOMIN and SERVOMAX. Hitting the physical limits of travel can strip the gears and permanently damage your servo.

Converting from Degrees to Pulse Length

The Arduino "map()" function (http://adafru.it/aQm) is an easy way to convert between degrees of rotation and your calibrated SERVOMIN and SERVOMAX pulse lengths. Assuming a typical

servo with 180 degrees of rotation; once you have calibrated SERVOMIN to the 0-degree position and SERVOMAX to the 180 degree position, you can convert any angle between 0 and 180 degrees to the corresponding pulse length with the following line of code:

Library Reference

setPWMFreq(freq)

Description

This function can be used to adjust the PWM frequency, which determines how many full 'pulses' per second are generated by the IC. Stated differently, the frequency determines how 'long' each pulse is in duration from start to finish, taking into account both the high and low segments of the pulse.

Frequency is important in PWM, since setting the frequency too high with a very small duty cycle can cause problems, since the 'rise time' of the signal (the time it takes to go from 0V to VCC) may be longer than the time the signal is active, and the PWM output will appear smoothed out and may not even reach VCC, potentially causing a number of problems.

Arguments

- **freq**: A number representing the frequency in Hz, between 40 and 1000

Example

The following code will set the PWM frequency to the maximum value of 1000Hz:

```
pwm.setPWMFreq(1000)
```

setPWM(channel, on, off)

Description

This function sets the start (on) and end (off) of the high segment of the PWM pulse on a specific channel. You specify the 'tick' value between 0..4095 when the signal will turn on, and when it will turn of. Channel indicates which of the 16 PWM outputs should be updated with the new values.

Arguments

- **channel**: The channel that should be updated with the new values (0..15)
- **on**: The tick (between 0..4095) when the signal should transition from low to high
- **off**:the tick (between 0..4095) when the signal should transition from high to low

Example

The following example will cause channel 15 to start low, go high around 25% into the pulse (tick 1024 out of 4096), transition back to low 75% into the pulse (tick 3072), and remain low for

the last 25% of the pulse:

```
pwm.setPWM(15, 1024, 3072)
```