Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Mecànica

Títol: Disseny i fabricació del prototip d'una pròtesis biònica de mà

Document: Plec de condicions

Alumne: Pau Quintana Casanovas

Tutor: Jordi Bayer Resplandis

Departament: Enginyeria Química, agrària i tecnologia agroalimentària

Àrea: Enginyeria Química

Convocatòria (mes/any) FEBRER 2022

ÍNDEX

1 INTRODUCCIÓ

1.1 Objecte i abast

El plec de condicions és el document on es recullen cadascuna de les especificacions dels materials i dels equips de fabricació a emprar, així com el procés a seguir en el seu muntatge.

Així doncs, aquest document assenyala les obligacions i responsabilitats existents a l'hora de fabricar.

Per altra banda, amb aquest document el projectista queda exempt de culpabilitat en cas de trencament, fallida o error, degut a l'incorrecte seguiment de les instruccions descrites a continuació.

2 PRESCRIPCIONS TÈCNIQUES

2.1 Condicions dels materials

El prototip de la pròtesi ha estat dissenyat i comprovat per a ser fabricat amb uns materials de característiques concrets per a cada peça. Aquests materials estan especificats tant en el document *Plànols*, com en el document ***Estat d'amidaments***, i serà obligatòria la seva utilització per obtenir un disseny vàlid.

En cas que algun material o element dels indicats no es trobi en estoc i es necessités amb urgència, la decisió de canvi per un que presenti les mateixes característiques o semblants que els originals requereix l'aprovació del projectista.

2.2 Condicions de fabricació

Per tal d'implementar el procés de fabricació adient, cal seguir els passos indicats a l'*Annex C. Fabricació*. És important que els elements rotatius quedin ben impresos per tal d'assegurar un bon funcionament del mecanisme i evitar problemes en la seva utilització i possibles desajustatges.

El disseny de la pròtesi implementa una sèrie de peces que requereixen unes característiques específiques, tan dimensionals com superficials. Aquestes es veuen representades en els plànols de cadascuna de les peces i s'han de seguir de manera estricta durant la seva fabricació. Així mateix i una vegada acabada la fabricació s'ha de realitzar un control dimensional i visual per assegurar que les peces compleixen amb les característiques especificades.

Per altra banda, és obligatori utilitzar eines adequades i perfectament calibrades per a la fabricació de totes aquestes peces si es vol aconseguir les dimensions i els acabats marcats.

2.3 Condicions de muntatge

Durant el procés de muntatge és imprescindible seguir l'ordre indicat en els plànols per tal de garantir el correcte funcionament del prototip i evitar futurs problemes.

En aquest procés s'ha de fer especial èmfasi en la correcta col·locació i la fixació de tots els elements, així com la correcta instal·lació dels elements de retorn juntament amb la correcta col·locació de la posició dels braços dels servomotors.

Per a la instal·lació electrònica, cal seguir el diagrama i la programació de l'Annex *A.3. Electrònica i control.*

Finalment i una vegada acabat el muntatge es realitza un control funcional i visual per assegurar que el conjunt compleix amb les característiques especificades.

2.4 Condicions per l'usuari

En el moment d'utilitzar el prototip, aquest haurà de seguir les indicacions marcades per la seva utilització a l'apartat *Annex B: Manual d'usuari i manteniment* del document *Memòria i annexos*.

Així mateix, l'usuari haurà de procurar no utilitzar elements que puguin malmetre el prototip.

FITXES TÈCNIQUES

PLA

Ficha de datos técnicos PLA

Ultimaker

Denominación química	Ácido poliláctico
Descripción	El filamento de PLA Ultimaker ofrece una experiencia de impresión 3D sencilla gracias a su fiabilidad y buena calidad superficial. Nuestro PLA está fabricado con materiales orgánicos y renovables. Es seguro, fácil de utilizar en la impresión y se adecua a una amplia gama de aplicaciones para usuarios nuevos y experimentados.
Características principales	El PLA ofrece una buena resistencia a la tracción y calidad superficial, facilita el trabajo a altas velocidades de impresión, simplifica el uso en entornos domésticos y de oficina y permite la creación de piezas de alta resolución. Existe una amplia gama de opciones de color disponibles.
Aplicaciones	Herramientas domésticas, juguetes, proyectos educativos, objetos de exposición, prototipado, modelos arquitectónicos y también métodos de fundición a la cera perdida para crear piezas de metal.
No adecuado para	Aplicaciones en contacto con alimentos e in vivo. Uso prolongado en exteriores o aplicaciones en las cuales la parte impresa está expuesta a temperaturas superiores a 50 ˚C.

Especificaciones del filamento	Valor	Método
Diámetro	2,85 ± 0,10 mm	-
Desviación de redondez máxima	0,10 mm	-
Peso neto del filamento	350 g / 750 g	-
Longitud del filamento	~44 m / ~95 m	-

Información sobre el color	Color	Código de color
	PLA verde	RAL 6018
	PLA negro	RAL 9005
	PLA plata metalizado	RAL 9006
	PLA blanco	RAL 9010
	PLA transparente	n.p.
	PLA naranja	RAL 2008
	PLA azul	RAL 5002
	PLA magenta	RAL 4010
	PLA rojo	RAL 3020
	PLA amarillo	RAL 1003
	PLA blanco nacarado	RAL 1013

Propiedades mecánicas (*)	Moldeo por inyección		Impresión 3D	
	Valor típico	**Método de ensayo**	**Valor típico**	**Método de ensayo**
Módulo de elasticidad a la tracción	-	-	2346,5 MPa	ISO 527 (1 mm/min)
Esfuerzo de tracción a la deformación	-	-	49,5 MPa	ISO 527 (50 mm/min)
Esfuerzo de tracción a la rotura	-	-	45,6 MPa	ISO 527 (50 mm/min)
Alargamiento a la deformación	-	-	3,3 %	ISO 527 (50 mm/min)
Alargamiento a la rotura	-	-	5,2 %	ISO 527 (50 mm/min)
Resistencia a la flexión	-	-	103,0 MPa	ISO 178
Módulo de flexión	-	-	3150,0 MPa	ISO 178
Resistencia a la prueba de impacto Izod, con mella (a 23 ˚C)	-	-	5,1 kJ/m^2	ISO 180
Resistencia a la prueba de impacto Charpy (a 23 ˚C)	-	-	-	-
Dureza	-	-	83 (Shore D)	Durómetro

Propiedades térmicas	Valor típico	Método de ensayo
Índice de fluidez (MFR)	6,09 g/10 min	ISO 1133 (210 ˚C, 2,16 kg)
Deformación térmica (HDT) a 0,455 MPa	-	-
Deformación térmica (HDT) a 1,82 MPa	-	-
Transición vítrea	~60 ˚C	ISO 11357
Coeficiente de expansión térmica	-	-
Temperatura de fusión	145-160 ˚C	ISO 11357
Contracción térmica	-	-

Otras propiedades	Valor típico	Método de ensayo
Gravedad específica	1,24	ASTM D1505
Clasificación de llama	-	-

(*) Ver las notas.

Notas

Las propiedades indicadas corresponden a los valores promedio de un lote típico. Las muestras de prueba impresas en 3D se imprimieron en el plano XY, utilizando el perfil de calidad normal en Cura 2.1, una Ultimaker 2+, una tobera de 0,4 mm, relleno del 90 %, una temperatura de tobera de 210 ˚C y una temperatura de la placa de impresión de 60 ˚C. Los valores son la media de 5 muestras blancas y 5 negras para los ensayos de tracción, flexión e impacto. La dureza Shore D se midió en un recuadro de 7 mm de grosor impreso en el plano XY, utilizando el perfil de calidad normal en Cura 2.5, una Ultimaker 3, un núcleo de impresión de 0,4 mm y relleno del 100 %. Ultimaker trabaja constantemente para ampliar la información de las fichas de datos técnicos.

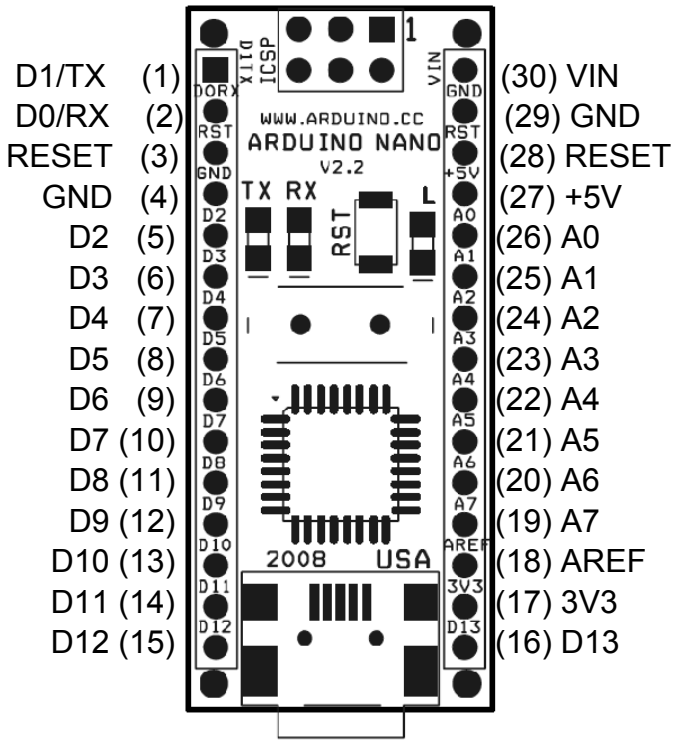
Descargo de responsabilidad

Versión Versión 3.011

Fecha 16/05/2017

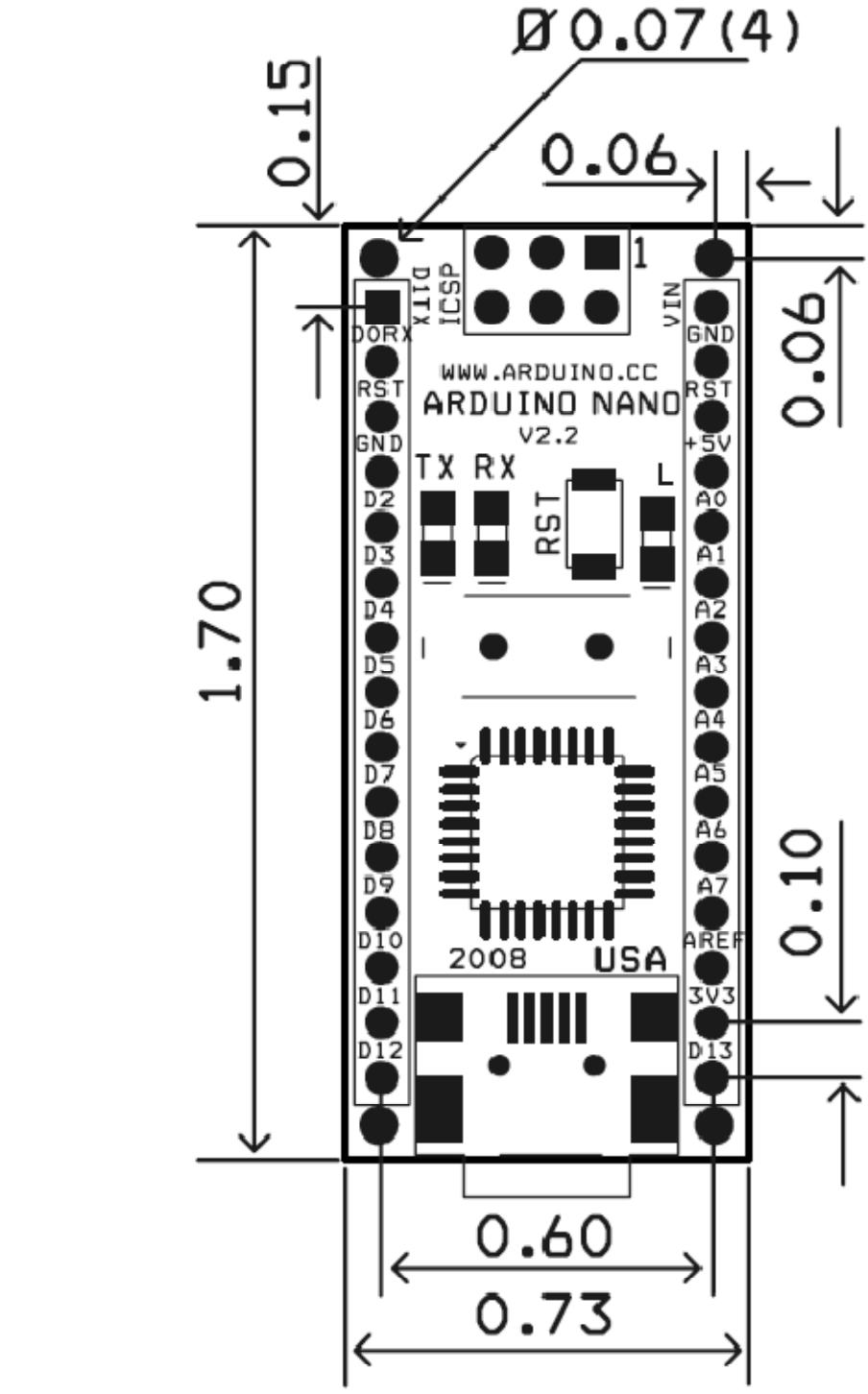
Arduino Nano

Arduino Nano Pin Layout



Pin No.	Name	Type	Description
1-2, 5-16	D0-D13	I/O	Digital input/output port 0 to 13
3, 28	RESET	Input	Reset (active low)
4, 29	GND	PWR	Supply ground
17	3V3	Output	+3.3V output (from FTDI)
18	AREF	Input	ADC reference
19-26	A7-A0	Input	Analog input channel 0 to 7
27	+5V	Output or Input	+5V output (from on-board regulator) or +5V (input from external power supply)
30	VIN	PWR	Supply voltage

Arduino Nano Mechanical Drawing

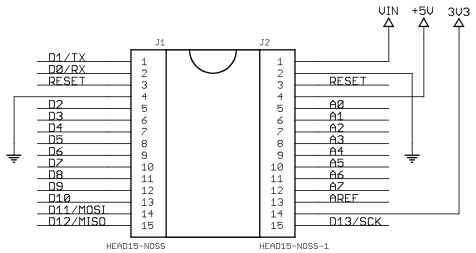


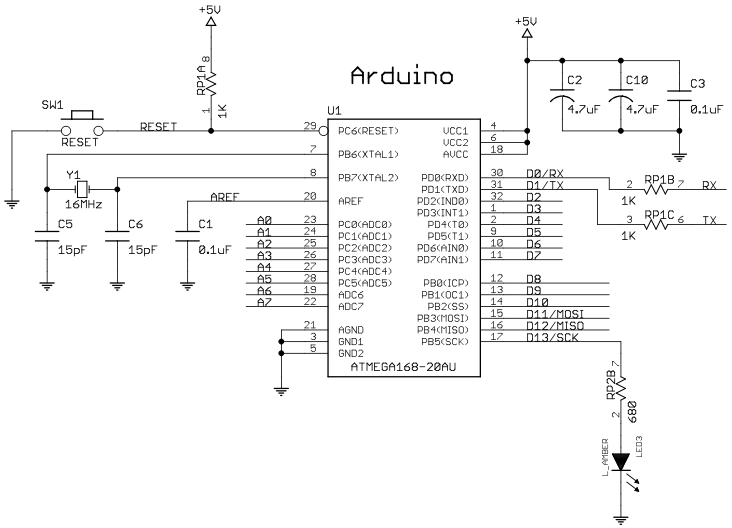
ALL DIMENTIONS ARE IN INCHES

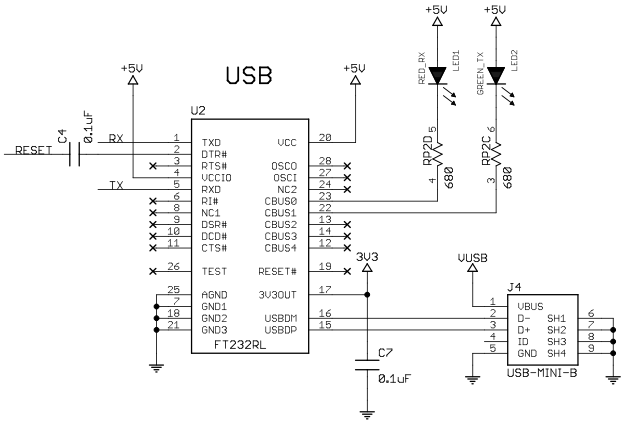
Arduino Nano Bill of Material

Item Number	Qty.	Ref. Dest.	Description	Mfg. P/N	MFG	Vendor P/N	Vendor
1	5	C1,C3,C4,C7,C9	Capacitor, 0.1uF 50V 10% Ceramic X7R 0805	C0805C104K5RACTU	Kemet	80-C0805C104K5R	Mouser
2	3	C2,C8,C10	Capacitor, 4.7uF 10V 10% Tantalum Case A	T491A475K010AT	Kemet	80-T491A475K010	Mouser
3	2	C5,C6	Capacitor, 18pF 50V 5% Ceramic NOP/COG 0805	C0805C180J5GACTU	Kemet	80-C0805C180J5G	Mouser
4	1	D1	Diode, Schottky 0.5A 20V	MBR0520LT1G	ONSemi	863-MBR0520LT1G	Mouser
5	1	J1,J2	Headers, 36PS 1 Row	68000-136HLF	FCI	649-68000-136HLF	Mouser
6	1	J4	Connector, Mini-B Recept Rt. Angle	67503-1020	Molex	538-67503-1020	Mouser
7	1	J5	Headers, 72PS 2 Rows	67996-272HLF	FCI	649-67996-272HLF	Mouser
8	1	LD1	LED, Super Bright RED 100mcd 640nm 120degree 0805	APT2012SRCPRV	Kingbright	604-APT2012SRCPRV	Mouser
9	1	LD2	LED, Super Bright GREEN 50mcd 570nm 110degree 0805	APHCM2012CGCK-F01	Kingbright	604-APHCM2012CGCK	Mouser
10	1	LD3	LED, Super Bright ORANGE 160mcd 601nm 110degree 0805	APHCM2012SECK-F01	Kingbright	04-APHCM2012SECK	Mouser
11	1	LD4	LED, Super Bright BLUE 80mcd 470nm 110degree 0805	LTST-C170TBKT	Lite-On Inc	160-1579-1-ND	Digikey
12	1	R1	Resistor Pack, 1K +/-5% 62.5mW 4RES SMD	YC164-JR-071KL	Yageo	YC164J-1.0KCT-ND	Digikey
13	1	R2	Resistor Pack, 680 +/-5% 62.5mW 4RES SMD	YC164-JR-07680RL	Yageo	YC164J-680CT-ND	Digikey
14	1	SW1	Switch, Momentary Tact SPST 150gf 3.0x2.5mm	B3U-1000P	Omron	SW1020CT-ND	Digikey
15	1	U1	IC, Microcontroller RISC 16kB Flash, 0.5kB EEPROM, 23 I/O Pins	ATmega168-20AU	Atmel	556-ATMEGA168-20AU	Mouser
16	1	U2	IC, USB to SERIAL UART 28 Pins SSOP	FT232RL	FTDI	895-FT232RL	Mouser
17	1	U3	IC, Voltage regulator 5V, 500mA SOT-223	UA78M05CDCYRG3	TI	595-UA78M05CDCYRG3	Mouser
18	1	Y1	Cystal, 16MHz +/-20ppm HC-49/US Low Profile	ABL-16.000MHZ-B2	Abracon	815-ABL-16-B2	Mouser

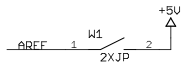
Arduino Nano Schematic



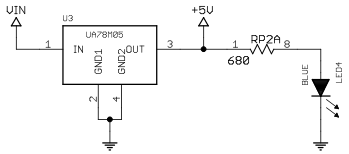




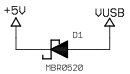
+5V AREF OPTION



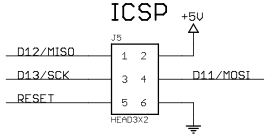
+5V REG



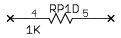
+5V AUTO SELECTOR

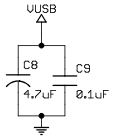


ICSP



NOT USED





v2.3 - Modify FT232RL to use +5V	
TITLE: Arduino Nano	
Document Number:	REV: 2.3
Date: 6/26/2008 8:35:54 PM	Sheet: 1/1

MyoWare

3-lead Muscle / Electromyography Sensor for Microcontroller Applications

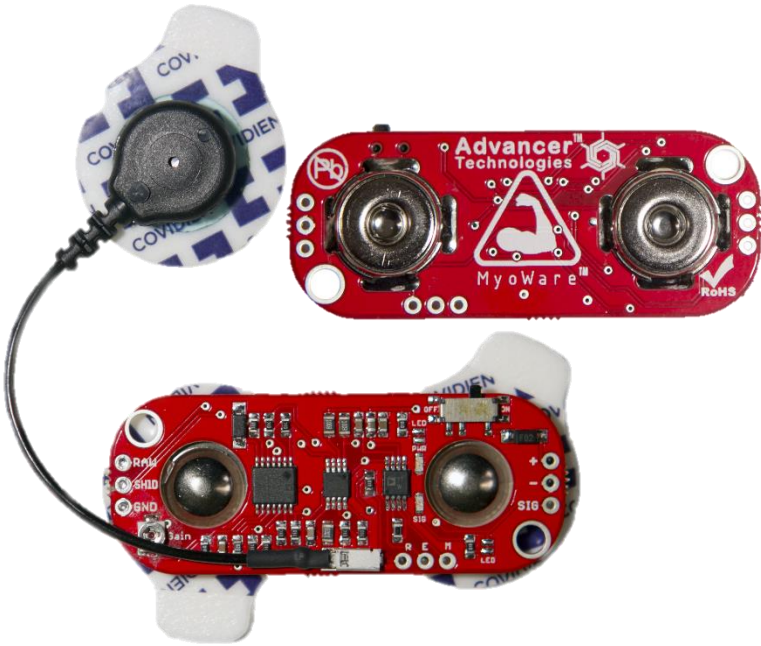
MyoWare™ Muscle Sensor (AT-04-001) DATASHEET

FEATURES

- *NEW* - Wearable Design
- *NEW* - Single Supply
  - +2.9V to +5.7V
  - Polarity reversal protection
- *NEW* - Two Output Modes
  - EMG Envelope
  - Raw EMG
- *NEW* - Expandable via Shields
- *NEW* - LED Indicators
- Specially Designed For Microcontrollers
- Adjustable Gain

APPLICATIONS

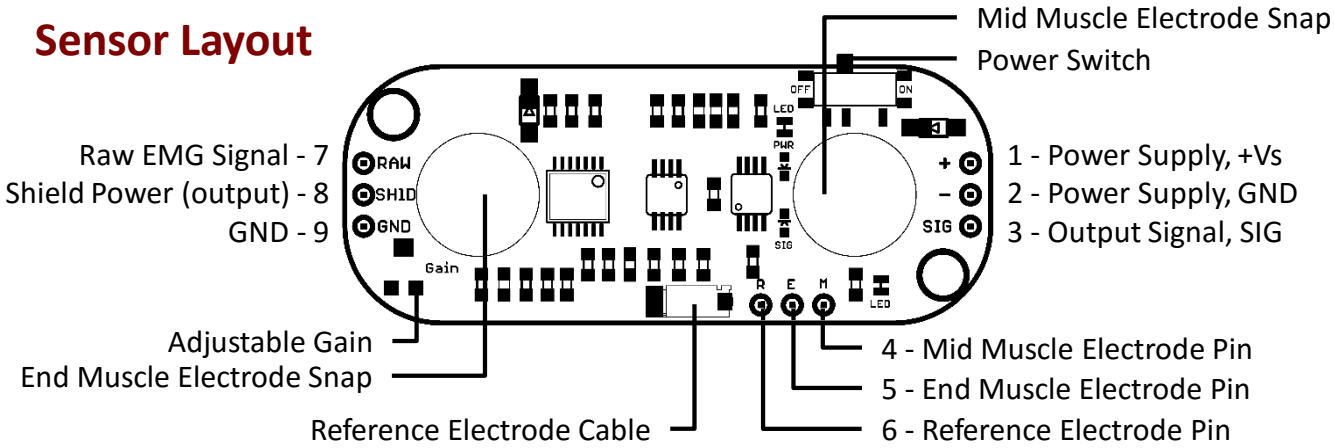
- Video games
- Robotics
- Medical Devices
- Wearable/Mobile Electronics
- Prosthetics/Orthotics



What is electromyography?

Measuring muscle activation via electric potential, referred to as electromyography (EMG), has traditionally been used for medical research and diagnosis of neuromuscular disorders. However, with the advent of ever shrinking yet more powerful microcontrollers and integrated circuits, EMG circuits and sensors have found their way into prosthetics, robotics and other control systems.

Sensor Layout



EMAIL: support@advancer.co

Advancer Technologies™

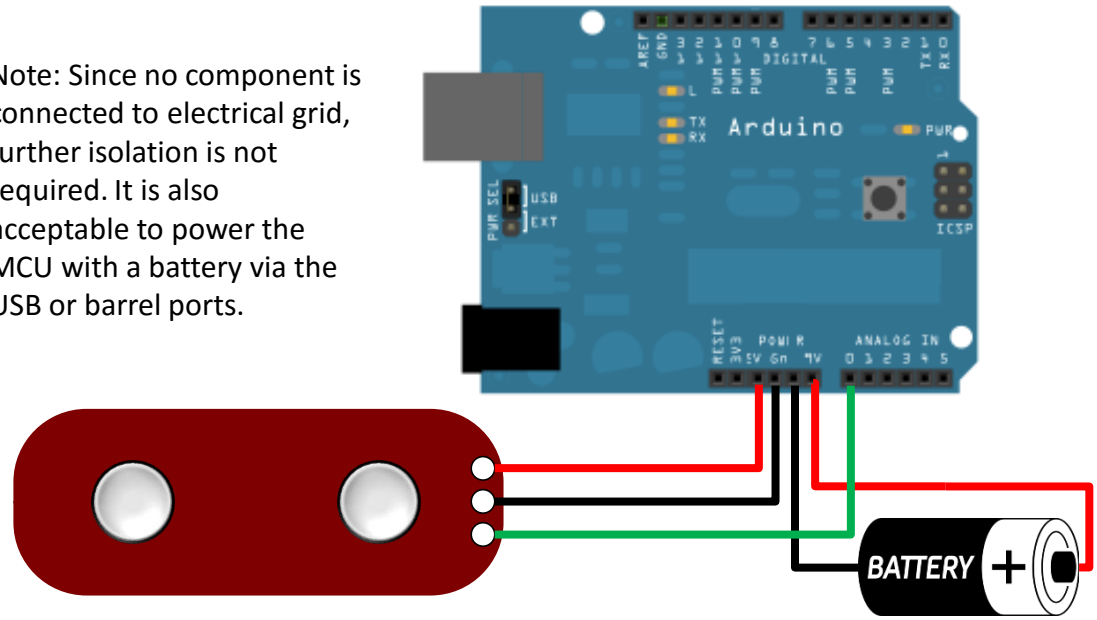
www.AdvancerTechnologies.com

Setup Configurations *(Arduino is shown but MyoWare is compatible with most development boards)*

a) Battery powered with isolation via no direct external connections

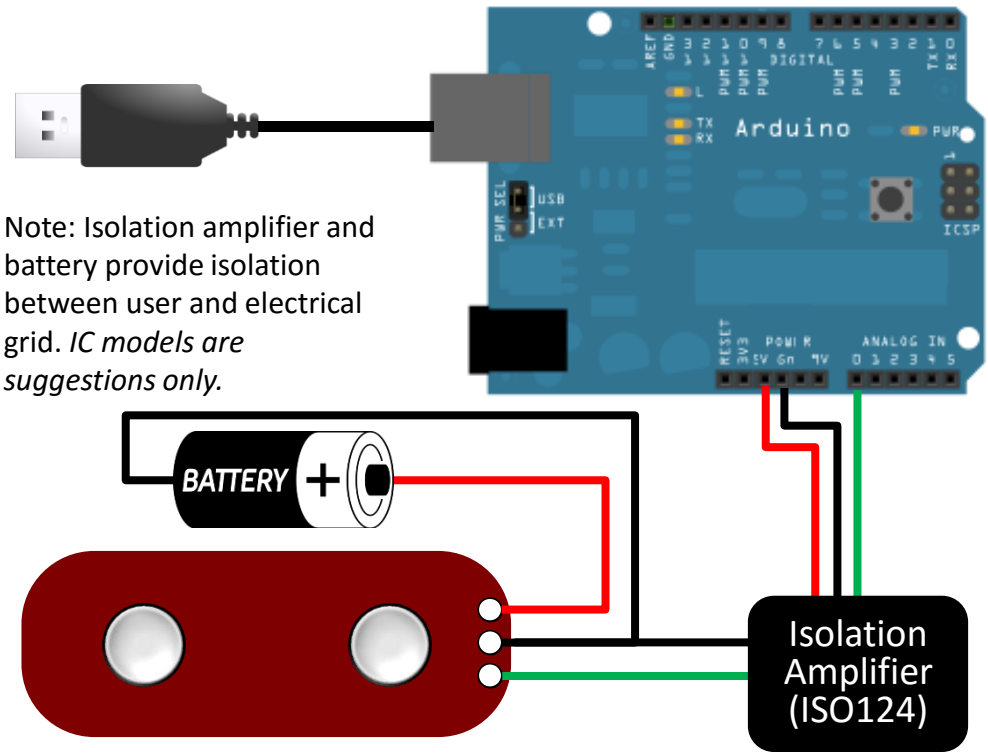
RECOMMENDED

Note: Since no component is connected to electrical grid, further isolation is not required. It is also acceptable to power the MCU with a battery via the USB or barrel ports.



b) Battery powered sensor, Grid powered MCU with output isolation

Note: Isolation amplifier and battery provide isolation between user and electrical grid. *IC models are suggestions only.*



(Note: Arduino and batteries not included. Arduino setup is only an example; sensor will work with numerous other devices.)

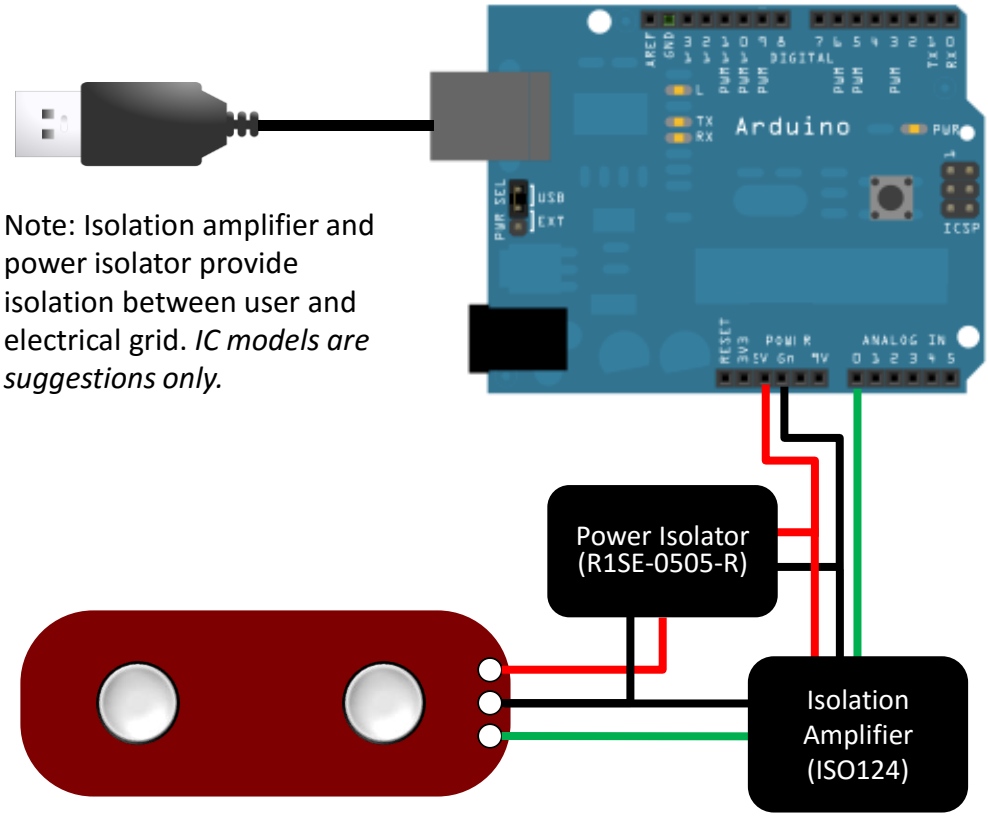
EMAIL: support@advancer.co

Advancer Technologies™

www.AdvancerTechnologies.com

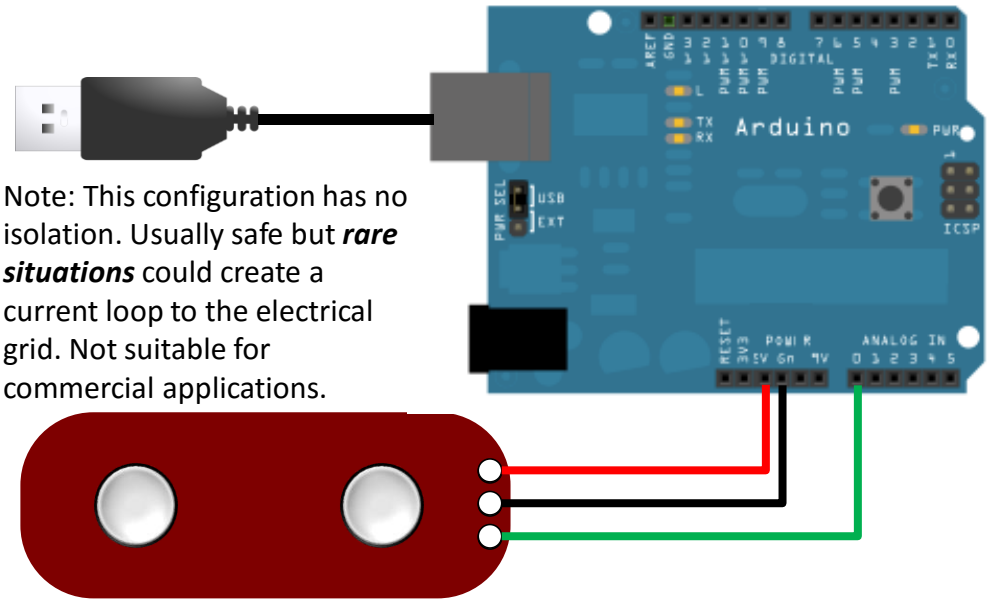
Setup Configurations (cont'd)

c) Grid powered with power and output isolation



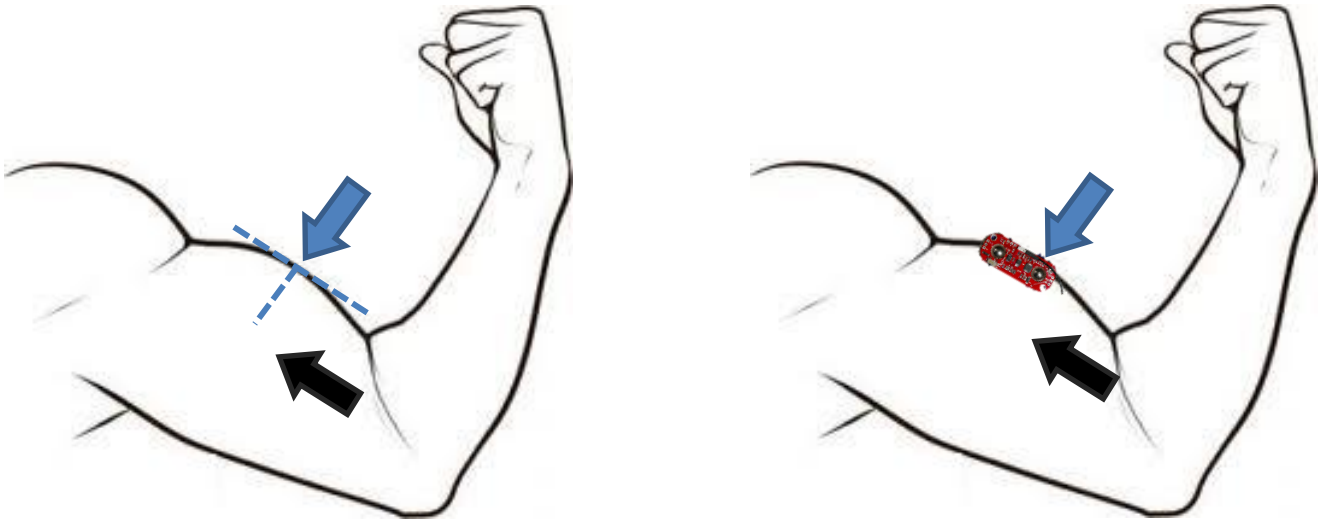
Note: Isolation amplifier and power isolator provide isolation between user and electrical grid. *IC models are suggestions only.*

d) Grid powered. ***Warning: No isolation.***



Note: This configuration has no isolation. Usually safe but ***rare situations*** could create a current loop to the electrical grid. Not suitable for commercial applications.

Setup Instructions



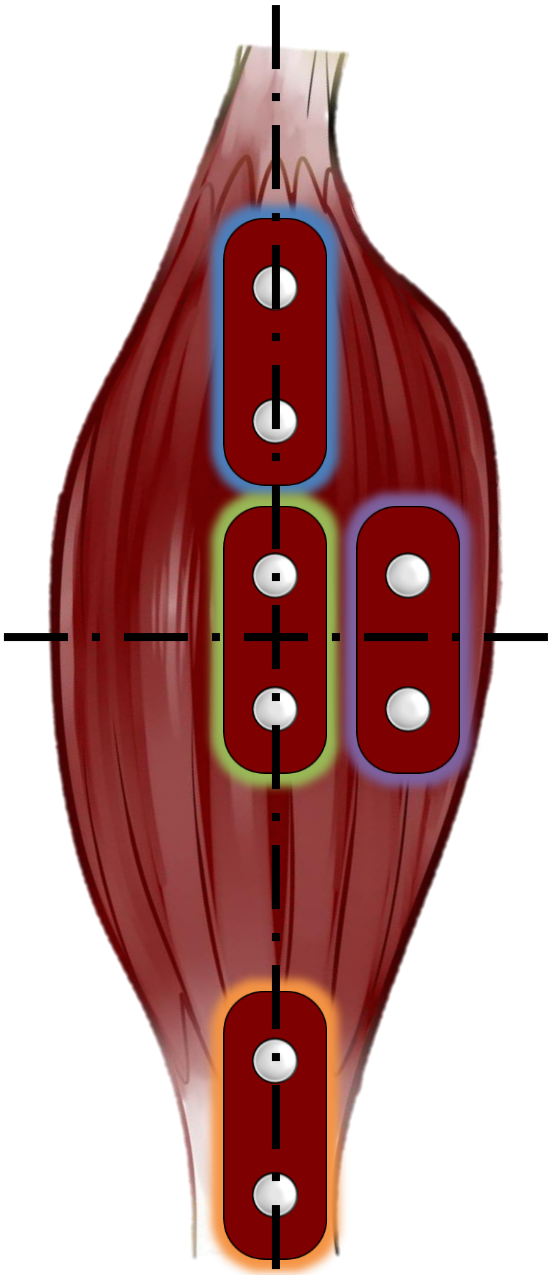
Note: Not To Scale

Example Sensor Location for Bicep

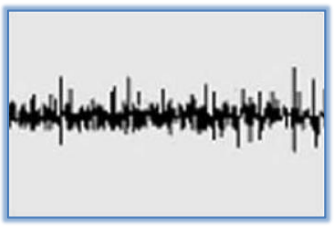
1) Thoroughly clean the intended area with soap to remove dirt and oil
2) Snap electrodes to the sensor's snap connectors

   (Note: While you can snap the sensor to the electrodes after they've been placed on the muscle, we do not recommend doing so due to the possibility of excessive force being applied and bruising the skin.)
3) Place the sensor on the desired muscle
   a. After determining which muscle group you want to target (e.g. bicep, forearm, calf), clean the skin thoroughly
   b. Place the sensor so one of the connected electrodes is in the middle of the muscle body. The other electrode should line up in the direction of the muscle length
   c. Peel off the backs of the electrodes to expose the adhesive and apply them to the skin
   d. Place the reference electrode on a bony or nonadjacent muscular part of your body near the targeted muscle
4) Connect to a development board (e.g. Arduino, RaspberryPi), microcontroller, or ADC
   a. See configurations previously shown

Why is electrode placement important?



Raw EMG output

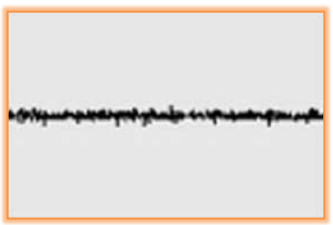


Innervation Zone

Correct Placement

Midline of the muscle belly between an innervation zone and a myotendon junction

Midline Offset



Myotendon Junction

Position and orientation of the muscle sensor electrodes has a vast effect on the strength of the signal. The electrodes should be place in the middle of the muscle body and should be aligned with the orientation of the muscle fibers. Placing the sensor in other locations will reduce the strength and quality of the sensor's signal due to a reduction of the number of motor units measured and interference attributed to crosstalk.

EMAIL: support@advancer.co

Advancer™ Technologies

www.AdvancerTechnologies.com

RAW EMG vs EMG Envelope

Our Muscle Sensors are designed to be used directly with a microcontroller. Therefore, our sensors primary output is not a RAW EMG signal but rather an amplified, rectified, and integrated signal (AKA the EMG's envelope) that will work well with a microcontroller's analog-to-digital converter (ADC). This difference is illustrated below using a representative EMG signal. *Note: Actual sensor output not shown.*

RAW EMG Signal

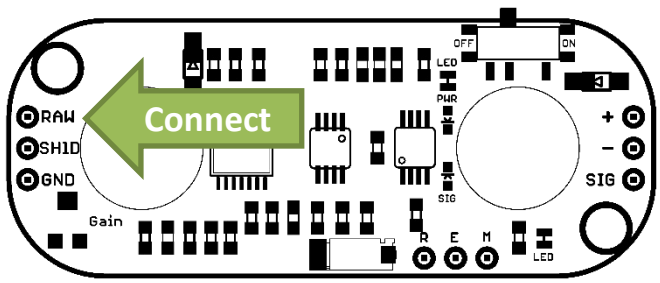
Rectified EMG Signal

Rectified & Integrated EMG Signal

Reconfigure for Raw EMG Output

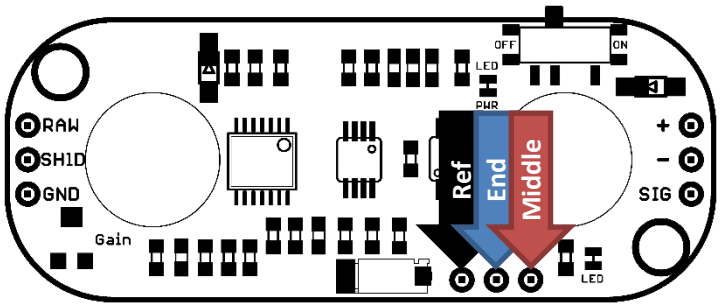
This new version has the ability to output an amplified raw EMG signal.
To output the raw EMG signal, simply connect the raw EMG signal pin to your measuring device instead of the SIG pin.

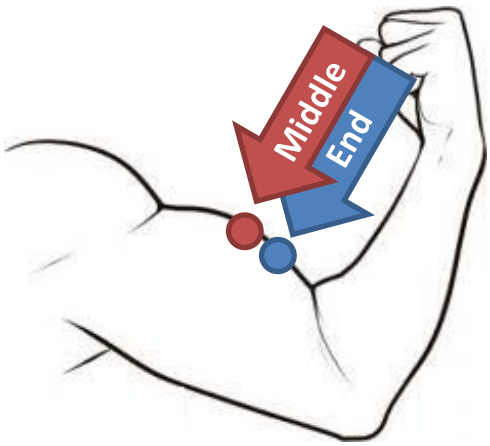
Note: This output is centered about an offset voltage of +Vs/2, see above. It is important to ensure +Vs is the max voltage of the MCU's analog to digital converter. This will assure that you completely see both positive and negative portions of the waveform.



Connecting external electrode cables

This new version has embedded electrode snaps right on the sensor board itself, replacing the need for a cable. However, if the on board snaps do not fit a user's specific application, an external cable can be connected to the board through three through hole pads shown above.





Middle

Connect this pad to the cable leading to an electrode placed in the middle of the muscle body.

End

Connect this to the cable leading to an electrode placed adjacent to the middle electrode towards the end of the muscle body.

Ref

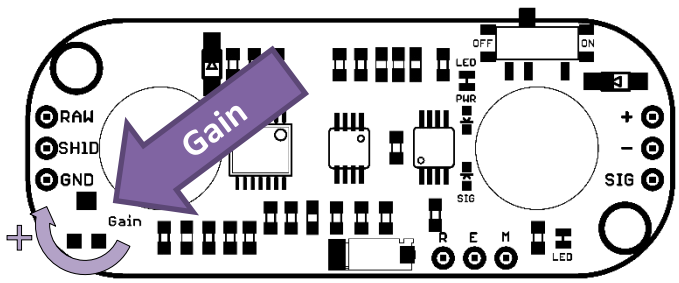
Connect this to the reference electrode. The reference electrode should be placed on an separate section of the body, such as the bony portion of the elbow or a nonadjacent muscle

Adjusting the gain

We recommend for users to get their sensor setup working reliably prior to adjusting the gain. The default gain setting should be appropriate for most applications.

To adjust the gain, locate the gain potentiometer in the lower left corner of the sensor (marked as "GAIN"). Using a Phillips screwdriver, turn the potentiometer counterclockwise to increase the output gain; turn the potentiometer clockwise to reduce the gain.

Note: In order to reduce the required voltage for the sensor, the redesign switch out a JFET amplifier for a CMOS amplifier. However CMOS amplifiers tend to have slower recovery times when saturated. Therefore, we advise users to adjust the gain such that the output signal will not saturate the amplifier.



EMAIL: support@advancer.co

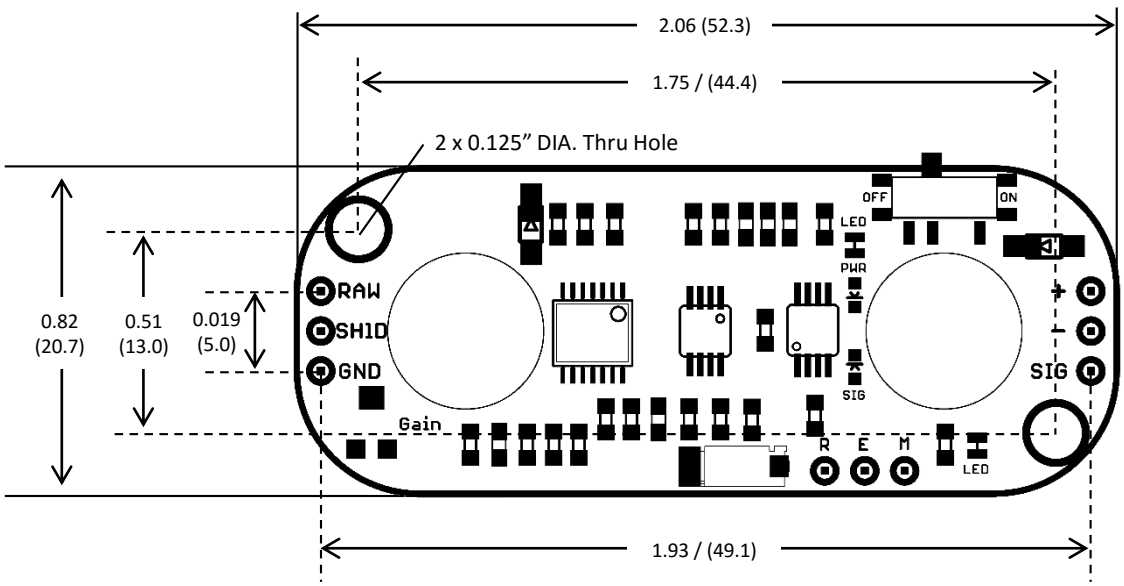
Advancer™ Technologies

www.AdvancerTechnologies.com

Electrical Specifications

Parameter	Min	TYP	Max
Supply Voltage	+2.9V	+3.3V or +5V	+5.7V
Adjustable Gain Potentiometer	0.01 Ω	50 kΩ	100 kΩ
Output Signal Voltage EMG Envelope Raw EMG (centered about +Vs/2)	 0V 0V	 -- --	 +Vs +Vs
Input Impedance	--	110 GΩ	--
Supply Current	--	9 mA	14 mA
Common Mode Rejection Ratio (CMRR)	--	110	--
Input Bias	--	1 pA	--

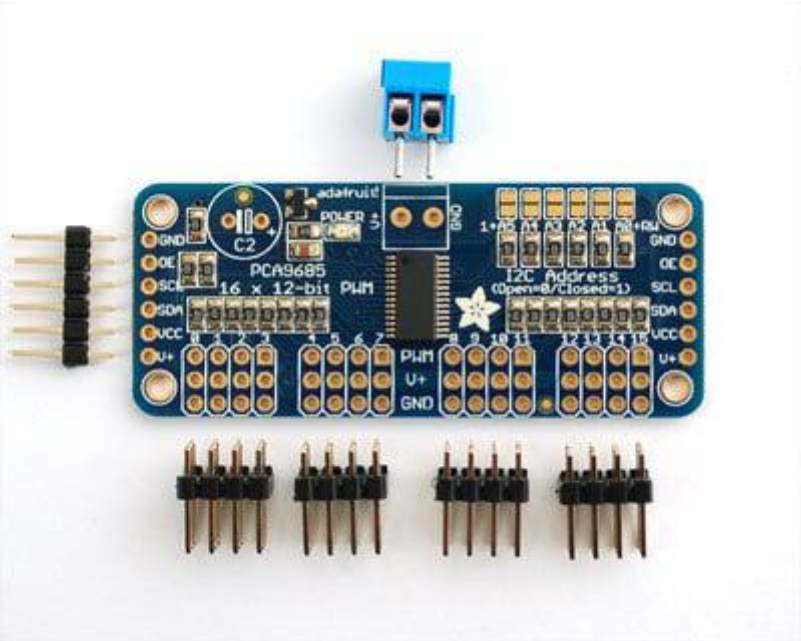
Dimensions



PCA9685

Adafruit PCA9685 16-Channel Servo Driver

Created by Bill Earl

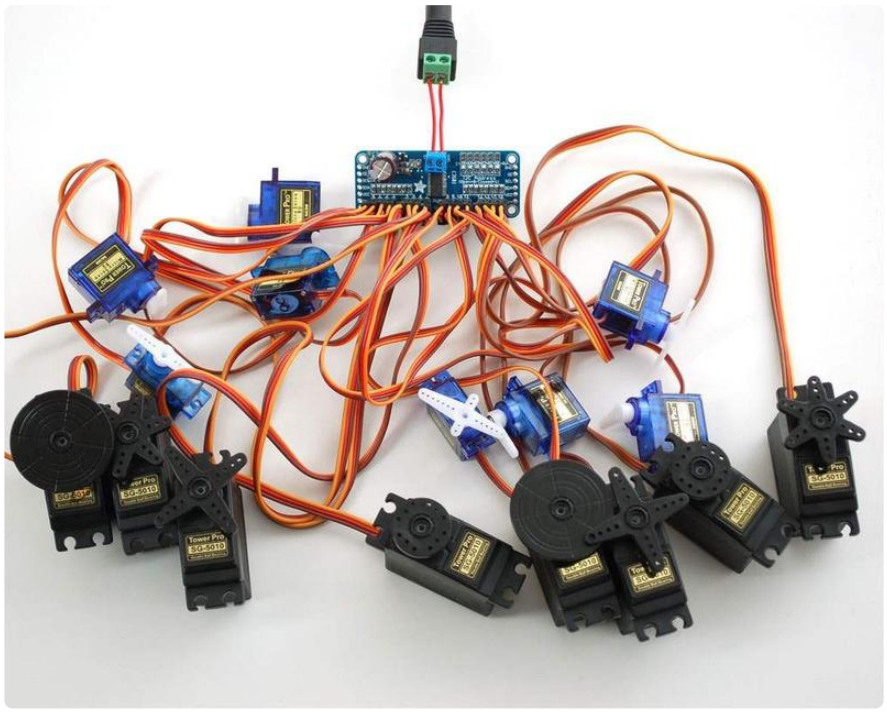


https://learn.adafruit.com/16-channel-pwm-servo-driver

Last updated on 2021-11-15 05:52:14 PM EST

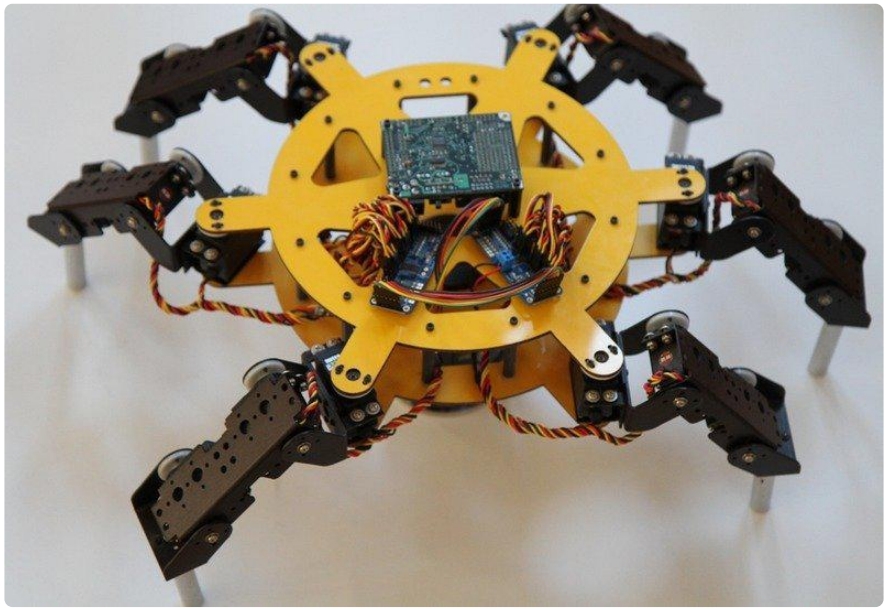
Table of Contents

Overview

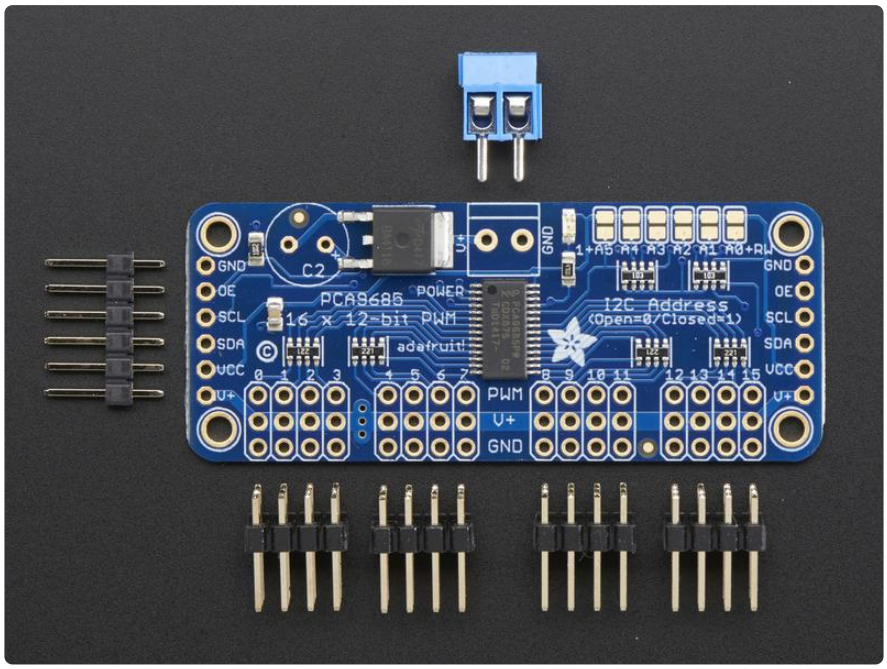


Driving servo motors with the Arduino Servo library is pretty easy, but each one consumes a precious pin - not to mention some Arduino processing power. The Adafruit 16-Channel 12-bit PWM/Servo Driver will drive up to 16 servos over I2C with only 2 pins. The on-board PWM controller will drive all 16 channels simultaneously with no additional Arduino processing overhead. What's more, you can chain up to 6 2 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.



Pinouts



There are two sets of control input pins on either side. Both sides of the pins are identical! Use whichever side you like, you can also easily chain by connecting up two side-by-side

Power Pins

- GND - This is the power and signal ground pin, must be connected
- VCC - This is the logic power pin, connect this to the logic level you want to use for the PCA9685 output, should be 3 - 5V max! It's also used for the 10K pullups on SCL/SDA so unless you have your own pullups, have it match the microcontroller's logic level too!
- V+ - This is an optional power pin that will supply distributed power to the servos. If you are not using for servos you can leave disconnected. It is not used at all by the chip. You can also inject power from the 2-pin terminal block at the top of the board. You should provide 5-6VDC if you are using servos. If you have to, you can go higher to 12VDC, but if you mess up and connect VCC to V+ you could damage your board!

Control Pins

- SCL - I2C clock pin, connect to your microcontrollers I2C clock line. Can use 3V or 5V logic, and has a weak pullup to VCC

- SDA - I2C data pin, connect to your microcontrollers I2C data line. Can use 3V or 5V logic, and has a weak pullup to VCC
- OE - Output enable. Can be used to quickly disable all outputs. When this pin is I ow all pins are enabled. When the pin is high the outputs are disabled. Pulled low by default so it's an optional pin!

Output Ports

There are 16 output ports. Each port has 3 pins: V+, GND and the PWM output. Each PWM runs completely independently but they must all have the same PWM frequency. That is, for LEDs you probably want 1.0 KHz but servos need 60 Hz - so you cannot use half for LEDs @ 1.0 KHz and half @ 60 Hz.

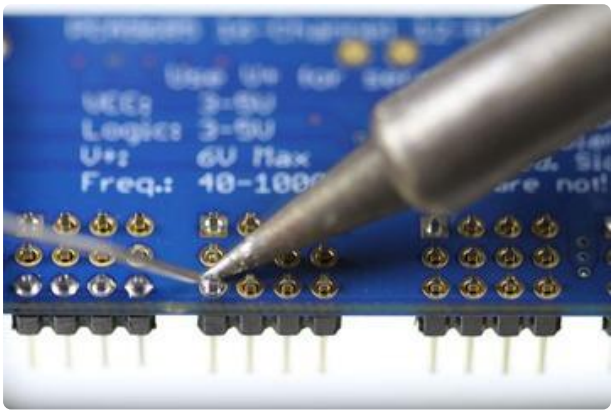
They're set up for servos but you can use them for LEDs! Max current per pin is 25mA.

There are 220 ohm resistors in series with all PWM Pins and the output logic is the same as VCC so keep that in mind if using LEDs.

Assembly

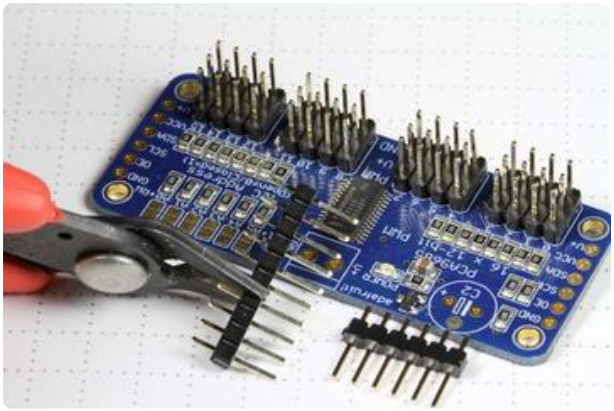
Install the Servo Headers

Install 4 3x4 pin male headers into the marked positions along the edge of the board.



Solder all pins

There are a lot of them!

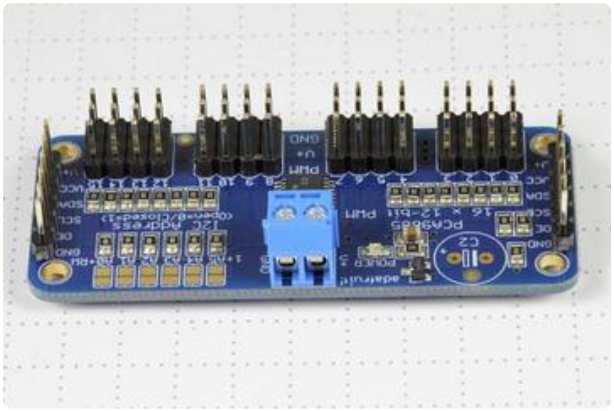


Add Headers for Control

A strip of male header is included. Where you want to install headers and on what side depends a little on use:

- For breadboard (http://adafru.it/239) use, install headers on the bottom of the board.
- For use with jumper wires (http://adafru.it/758), install the headers on top of the board.
- For use with our 6-pin cable (http://adafru.it/206), install the headers on top of the board.

If you are chaining multiple driver boards, you will want headers on both ends.



Install Power Terminals

If you are chaining multiple driver boards, you only need a power terminal on the first one.

Hooking it Up

Connecting to the Arduino

The PWM/Servo Driver uses I2C so it take only 4 wires to connect to your Arduino:

"Classic" Arduino wiring:

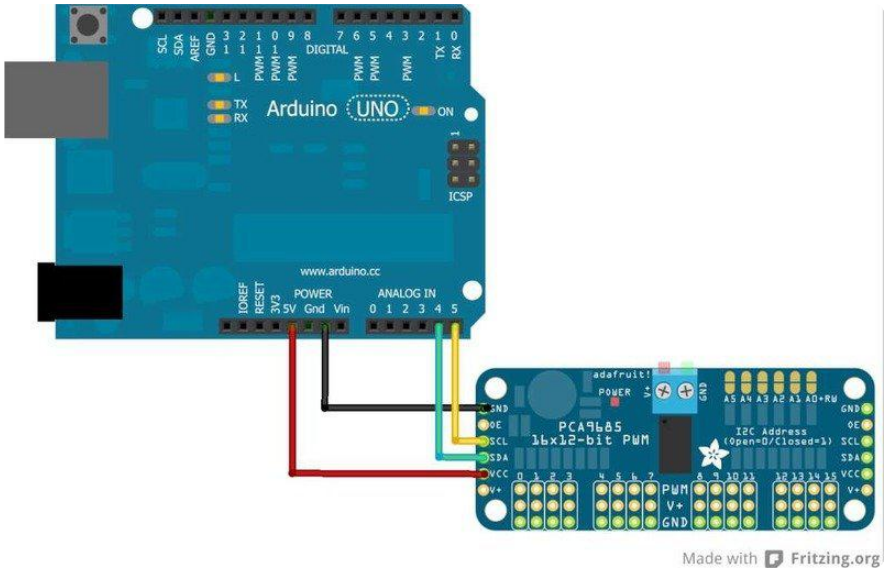
- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Analog 4 -> SDA
- Analog 5 -> SCL

Older Mega wiring:

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- Digital 20 -> SDA
- Digital 21 -> SCL

R3 and later Arduino wiring (Uno, Mega & Leonardo):
(These boards have dedicated SDA & SCL pins on the header nearest the USB connector)

- +5v -> VCC (this is power for the BREAKOUT only, NOT the servo power!)
- GND -> GND
- SDA -> SDA
- SCL -> SCL



The VCC pin is just power for the chip itself. If you want to connect servos or LEDs that use the V+ pins, you MUST connect the V+ pin as well. The V+ pin can be as high as 6V even if VCC is 3.3V (the chip is 5V safe). We suggest connecting power through the blue terminal block since it is polarity protected.

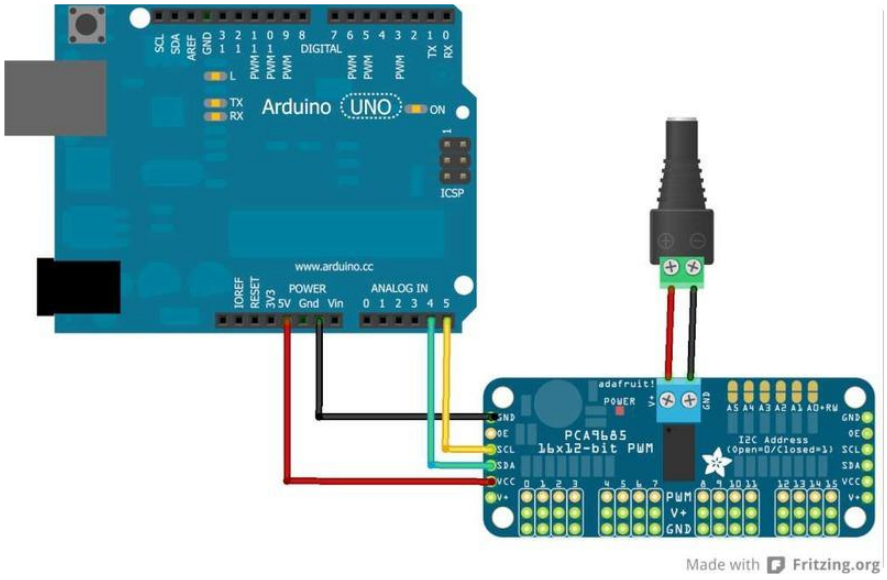
Power for the Servos

Most 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.



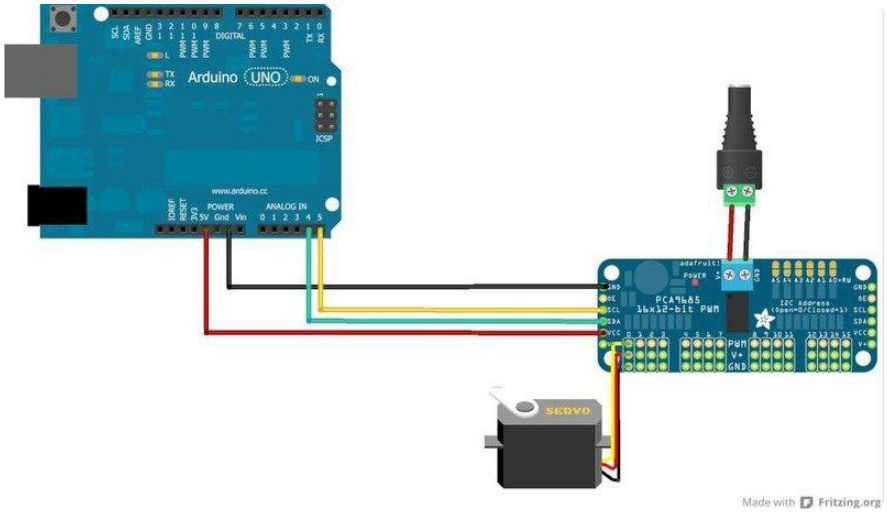
Adding a Capacitor to the thru-hole capacitor slot

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.

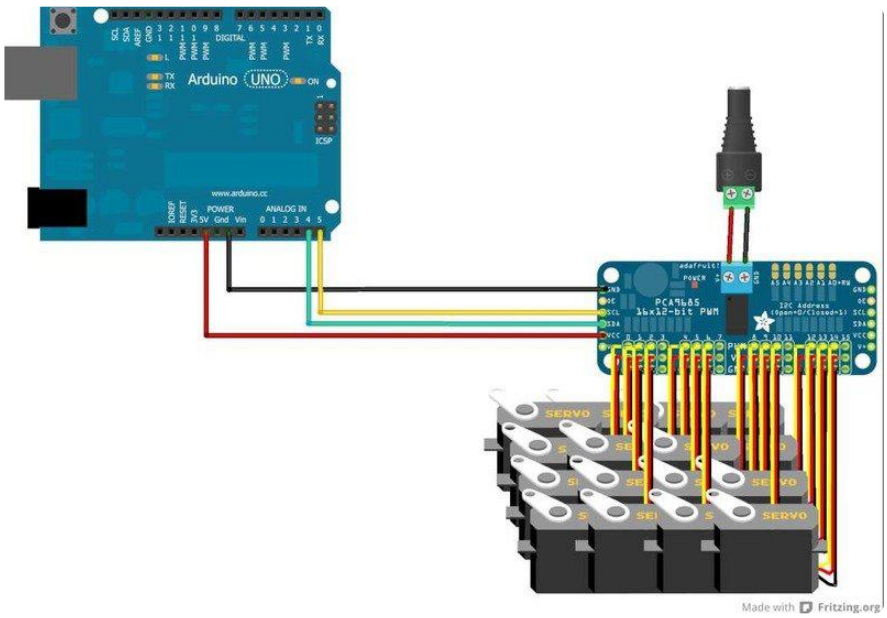
Connecting a Servo

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.



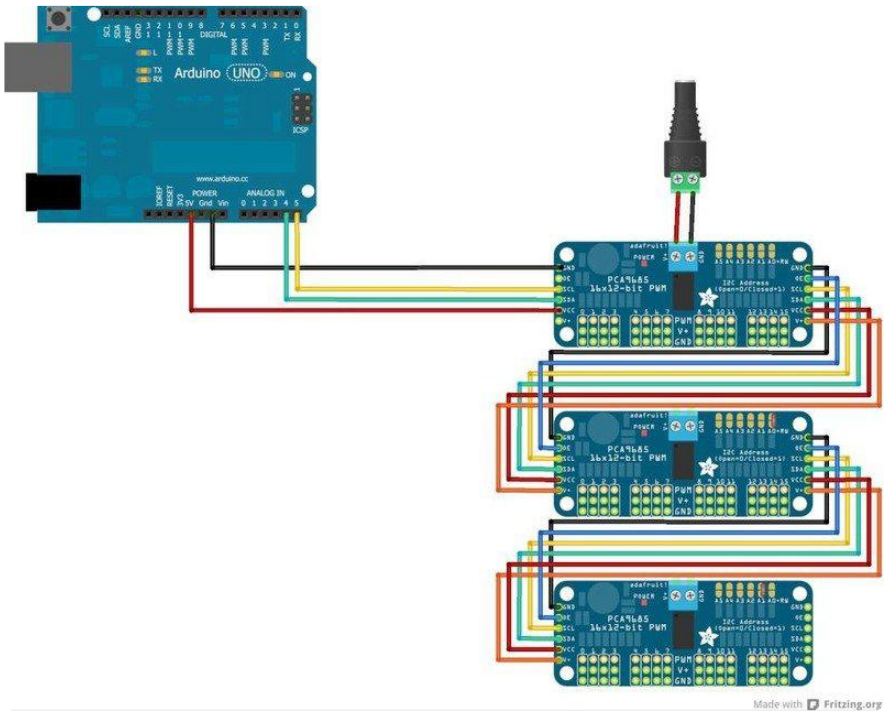
Adding More Servos

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



Chaining Drivers

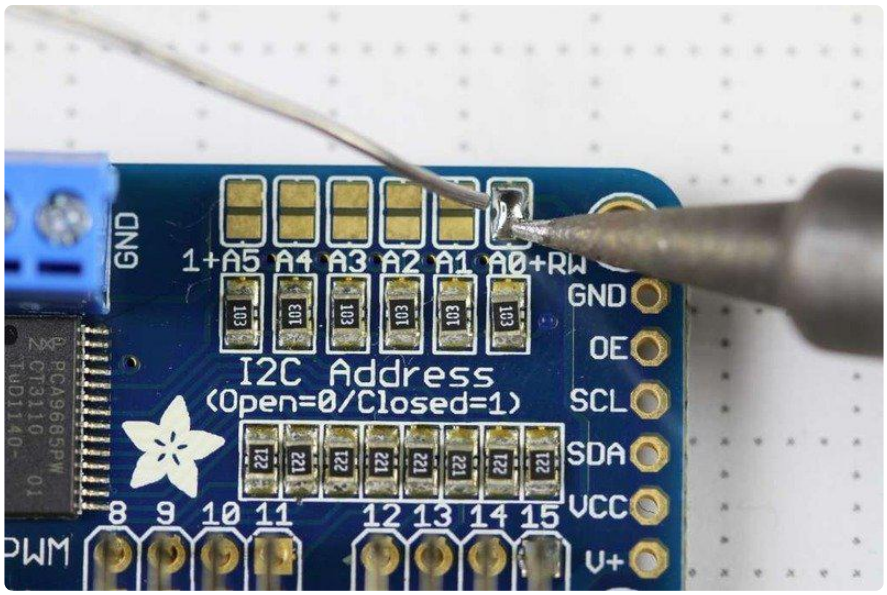
Multiple Drivers (up to 62) can be chained to control still more servos. With headers at both ends of the board, the wiring is as simple as connecting a 6-pin parallel cable (http://adafru.it/206) from one board to the next.



Addressing the Boards

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.

In your sketch, you'll need to declare a separate pobject for each board. Call begin on each object, and control each servo through the object it's attached to. For example:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_PWMServoDriver.h&gt;

Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);
Adafruit_PWMServoDriver pwm2 = Adafruit_PWMServoDriver(0x41);

void setup() {
  Serial.begin(9600);
  Serial.println("16 channel PWM test!");

  pwm1.begin();
  pwm1.setPWMFreq(1600);  // This is the maximum PWM frequency

  pwm2.begin();
  pwm2.setPWMFreq(1600);  // This is the maximum PWM frequency
}
```

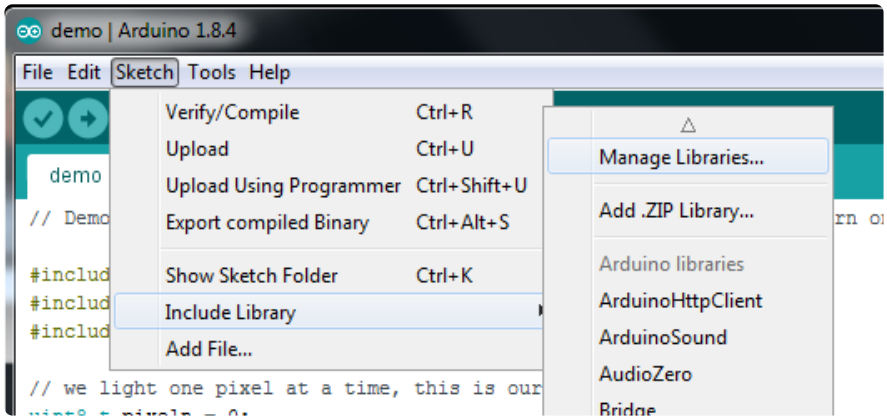
Using the Adafruit Library

Since the PWM Servo Driver is controlled over I2C, its super easy to use with any microcontroller or microcomputer. In this demo we'll show using it with the Arduino IDE but the C++ code can be ported easily

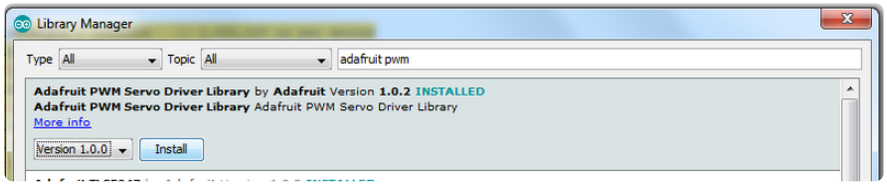
Install Adafruit PCA9685 library

To begin reading sensor data, you will need to install the Adafruit_PWMServo library (code on our github repository) (https://adafru.it/aQI). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...



And type in adafruit pwm to locate the library. Click Install

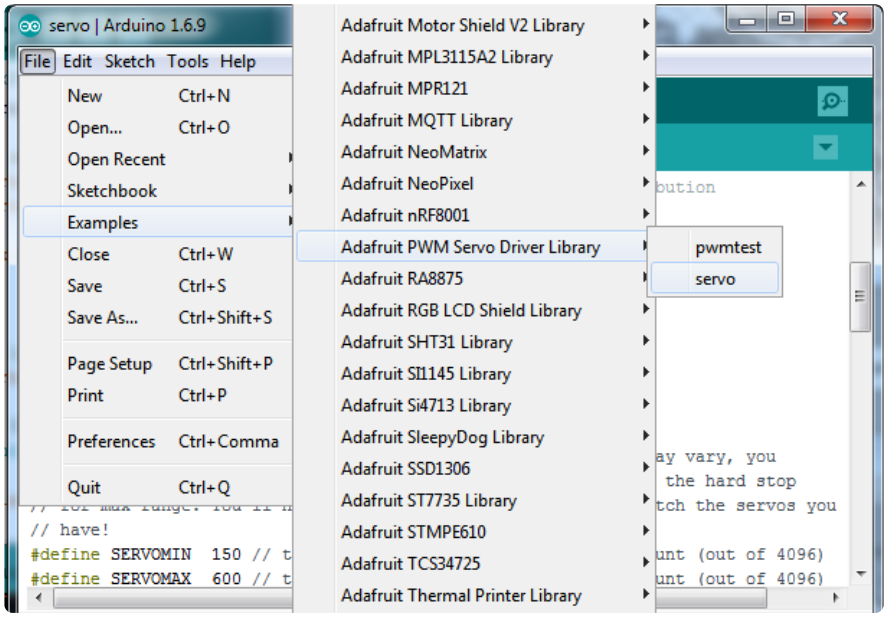


We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

Test with the Example Code:

First make sure all copies of the Arduino IDE are closed.

Next open the Arduino IDE and select File->Examples->Adafruit_PWMServoDriver->Servo. This will open the example file in an IDE window.



If using a Breakout:

Connect the driver board and servo as shown on the previous page. Don't forget to provide power to both Vin (3-5V logic level) and V+ (5V servo power). Check the green LED is lit!

If using a Shield:

Plug the shield into your Arduino. Don't forget you will also have to provide 5V to the V+ terminal block. Both red and green LEDs must be lit.

If using a FeatherWing:

Plug the FeatherWing into your Feather. Don't forget you will also have to provide 5V to the V+ terminal block. Check the green LED is lit!

Connect a Servo

A single servo should be plugged into the PWM #0 port, the first port. 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 (https://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:

```
pulselength = map(degrees, 0, 180, SERVOMIN, SERVOMAX);
```

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 1600

Example

The following code will set the PWM frequency to 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 off. 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)
```

Using as GPIO

There's also some special settings for turning the pins fully on or fully off

You can set the pin to be fully on with

```
pwm.setPWM(pin, 4096, 0);
```

You can set the pin to be fully off with

```
pwm.setPWM(pin, 0, 4096);
```

Arduino Library Docs

Arduino Library Docs (https://adafru.it/Au7)

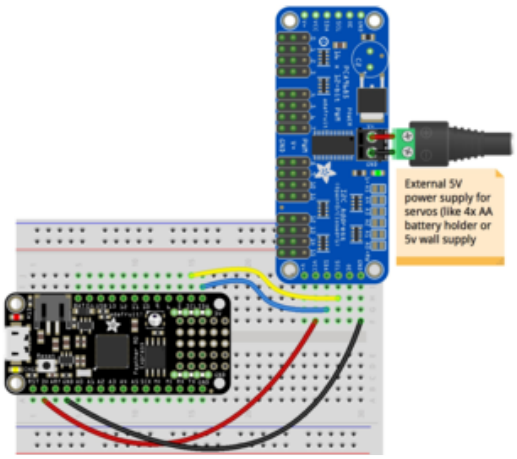
Python & CircuitPython

It's easy to use the PCA9685 sensor with Python or CircuitPython and the Adafruit CircuitPython PCA9685 (https://adafru.it/tZF) module. This module allows you to easily write Python code that control servos and PWM with this breakout.

You can use this sensor with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

CircuitPython Microcontroller Wiring

First wire up a PCA9685 to your board exactly as shown on the previous pages for Arduino. Here's an example of wiring a Feather M0 to the sensor with I2C:

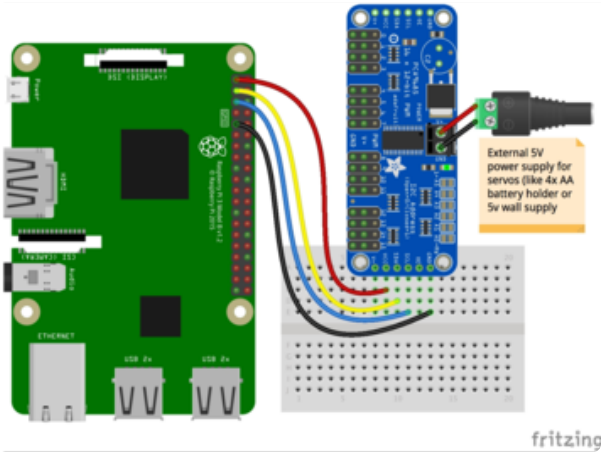


- Board 3V to sensor VCC
- Board GND to sensor GND
- Board SCL to sensor SCL
- Board SDA to sensor SDA

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here's the Raspberry Pi wired with I2C:



- Pi 3V3 to sensor VCC
- Pi GND to sensor GND
- Pi SCL to sensor SCL
- Pi SDA to sensor SDA

Don't try to power your servos from the RasPi or Linux board's 5V power, you can easily cause a power supply brown-out and mess up your Pi! Use a separate 5v 2A or 4A adapter

5V 2A (2000mA) switching power supply - UL Listed

This is an FCC/CE certified and UL listed power supply. Need a lot of 5V power? This switching supply gives a clean regulated 5V output at up to 2000mA. 110 or 240 input, so it works...
https://www.adafruit.com/product/276

5V 4A (4000mA) switching power supply - UL Listed

Need a lot of 5V power? This switching supply gives a clean regulated 5V output at up to 4 Amps (4000mA). 110 or 240 input, so it works in any country. The plugs are "US...
https://www.adafruit.com/product/1466

CircuitPython Installation of PCA9685 and ServoKit Libraries

You'll need to install the Adafruit CircuitPython PCA9685 (https://adafru.it/tZF) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

For non-express boards like the Trinket M0 or Gemma M0, you'll need to manually install the necessary libraries from the bundle:

- adafruit_pca9685.mpy
- adafruit_bus_device

- adafruit_register
- adafruit_motor
- adafruit_servokit

Before continuing make sure your board's lib folder or root filesystem has the adafruit_pca9685.mpy, adafruit_register, adafruit_servokit, adafruit_motor and adafruit_bus_device files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

Python Installation of PCA9685 and ServoKit Libraries

You'll need to install the Adafruit_Blinka library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following commands:

- `sudo pip3 install adafruit-circuitpython-pca9685`
- `sudo pip3 install adafruit-circuitpython-servokit`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

CircuitPython & Python Usage

The following section will show how to control the PCA9685 from the board's Python prompt / REPL. You'll learn how to interactively control servos and dim LEDs by typing in the code below.

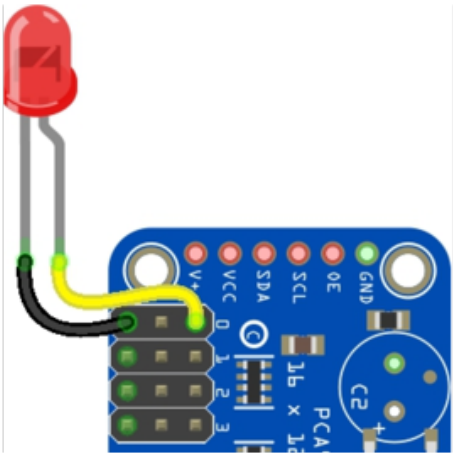
Dimming LEDs

Run the following code to import the necessary modules and initialize the I2C connection with the sensor:

```
import board
import busio
import adafruit_pca9685
i2c = busio.I2C(board.SCL, board.SDA)
pca = adafruit_pca9685.PCA9685(i2c)
```

Each channel of the PCA9685 can be used to control the brightness of an LED. The PCA9685 generates a high-speed PWM signal which turns the LED on and off very quickly. If the LED is turned on longer than turned off it will appear brighter to your eyes.

First wire a LED to the board as follows. Note you don't need to use a resistor to limit current through the LED as the PCA9685 will limit the current to around 10mA:



- LED cathode / shorter leg to PCA9685 channel GND / ground.
- LED anode / longer leg to PCA9685 channel PWM.

The PCA9685 class provides control of the PWM frequency and each channel's duty cycle. Check out the PCA9685 class documentation (https://adafru.it/C5n) for more details.

For dimming LEDs you typically don't need to use a fast PWM signal frequency and can set the board's PWM frequency to 60hz by setting the frequency attribute:

```
pca.frequency = 60
```

The PCA9685 supports 16 separate channels that share a frequency but can have independent duty cycles. That way you could dim 16 LEDs separately!

The PCA9685 object has a channels attribute which has an object for each channel that can control the duty cycle. To get the individual channel use the [] to index into ch annels.

```
led_channel = pca.channels[0]
```

Now control the LED brightness by controlling the duty cycle of the channel connected to the LED. The duty cycle value should be a 16-bit value, i.e. 0 to 0xffff, which represents what percent of the time the signal is on vs. off. A value of 0xffff is 100% brightness, 0 is 0% brightness, and in-between values go from 0% to 100% brightness.

For example set the LED completely on with a duty cycle of 0xffff:

```
led_channel.duty_cycle = 0xffff
```

After running the command above you should see the LED light up at full brightness!

Now turn the LED off with a duty cycle of 0:

```
led_channel.duty_cycle = 0
```

Try an in-between value like 1000:

```
led_channel.duty_cycle = 1000
```

You should see the LED dimly lit. Try experimenting with other duty cycle values to see how the LED changes brightness!

For example make the LED glow on and off by setting duty_cycle in a loop:

```
# Increase brightness:
for i in range(0xffff):
    led_channel.duty_cycle = i

# Decrease brightness:
for i in range(0xffff, 0, -1):
    led_channel.duty_cycle = i
```

These for loops take a while because 16-bits is a lot of numbers. CTRL-C to stop the loop from running and return to the REPL.

Full Example Code

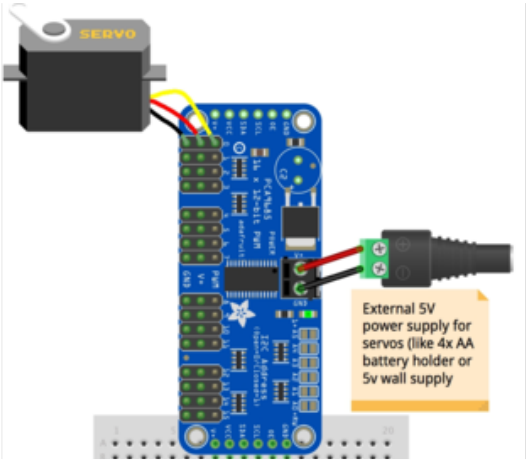
```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# This simple test outputs a 50% duty cycle PWM single on the 0th channel. Connect
an LED and
# resistor in series to the pin to visualize duty cycle changes and its impact on
brightness.

from board import SCL, SDA
import busio

# Import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# Create the I2C bus interface.
i2c_bus = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c_bus)

# Set the PWM frequency to 60hz.
pca.frequency = 60

# Set the PWM duty cycle for channel zero to 50%. duty_cycle is 16 bits to match
other PWM objects
# but the PCA9685 will only actually give 12 bits of resolution.
pca.channels[0].duty_cycle = 0x7FFF
```

Controlling Servos

We've written a handy CircuitPython library for the various PWM/Servo kits called Adafruit CircuitPython ServoKit (https://adafru.it/Dpu) that handles all the complicated setup for you. All you need to do is import the appropriate class from the library, and then all the features of that class are available for use. We're going to show you how to import the `ServoKit` class and use it to control servo motors with the Adafruit 16-channel breakout.

If you aren't familiar with servos be sure to first read this intro to servos page (https://adafru.it/scW) and this in-depth servo guide page (https://adafru.it/scS).

First connect the servo to channel 0 on the PCA9685. Here is an example of a servo connected to channel 0:



- Servo orange wire to breakout PWM on channel 0
- Servo red wire to breakout V+ on channel 0
- Servo brown wire to breakout Gnd on channel 0

Check your servo data sheet to verify how to connect it!

Be sure you've turned on or plugged in the external 5V power supply to the PCA9685 board too!

First you'll need to import and initialize the `ServoKit` class. You must specify the number of channels available on your board. The breakout has 16 channels, so when you create the class object, you will specify `16`.

```
from adafruit_servokit import ServoKit
kit = ServoKit(channels=16)
```

Now you're ready to control both standard and continuous rotation servos.

Standard Servos

To control a standard servo, you need to specify the channel the servo is connected to. You can then control movement by setting `angle` to the number of degrees.

For example to move the servo connected to channel `0` to `180` degrees:

```
pca.frequency = 50
```

Now that the PCA9685 is set up for servos lets make a Servo object so that we can adjust the servo based on angle instead of duty_cycle.

By default the Servo class will use actuation range, minimum pulse-width, and maximum pulse-width values that should work for most servos. However check the

Servo class documentation (https://adafru.it/BNE) for more details on extra parameters to customize the signal generated for your servos.

```
import adafruit_motor.servo
servo = adafruit_motor.servo.Servo(servo_channel)
```

With Servo, you specify a position as an angle. The angle will always be between 0 and the actuation range given when Servo was created. The default is 180 degrees but your servo might have a smaller sweep--change the total angle by specifying the actuation_angle parameter in the Servo class initializer above.

Now set the angle to 180, one extreme of the range:

```
    kit.servo[0].angle = 180
```

To return the servo to `0` degrees:

```
kit.servo[0].angle = 0
```

With a standard servo, you specify the position as an angle. The angle will always be between 0 and the actuation range. The default is 180 degrees but your servo may have a smaller sweep. You can change the total angle by setting `actuation_range`.

For example, to set the actuation range to 160 degrees:

```
servokit.servo[0].actuation_range = 160
```

Often the range an individual servo recognises varies a bit from other servos. If the servo didn't sweep the full expected range, then try adjusting the minimum and maximum pulse widths using `set_pulse_width_range(min_pulse, max_pulse)`.

To set the pulse width range to a minimum of 1000 and a maximum of 2000:

```
kit.servo[0].set_pulse_width_range(1000, 2000)
```

That's all there is to controlling standard servos with the PCA9685 breakout, Python and `ServoKit`!

Continuous Rotation Servos

To control a continuous rotation servo, you must specify the channel the servo is on. Then you can control movement using `throttle`.

For example, to start the continuous rotation servo connected to channel `1` to full throttle forwards:

```
kit.continuous_servo[1].throttle = 1
```

To start the continuous rotation servo connected to channel `1` to full reverse throttle:

```
kit.continuous_servo[1].throttle = -1
```

To set half throttle, use a decimal:

```
kit.continuous_servo[1].throttle = 0.5
```

And, to stop continuous rotation servo movement set `throttle` to `0`:

```
kit.continuous_servo[1].throttle = 0
```

That's all there is to controlling continuous rotation servos with the the PCA9685 16-channel breakout, Python and `ServoKit`!

Full Example Code

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

"""Simple test for a standard servo on channel 0 and a continuous rotation servo on
channel 1."""
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=8)

kit.servo[0].angle = 180
kit.continuous_servo[1].throttle = 1
time.sleep(1)
kit.continuous_servo[1].throttle = -1
time.sleep(1)
kit.servo[0].angle = 0
kit.continuous_servo[1].throttle = 0
```

Python Docs

Python Docs (https://adafru.it/C5p)

Python Docs: ServoKit

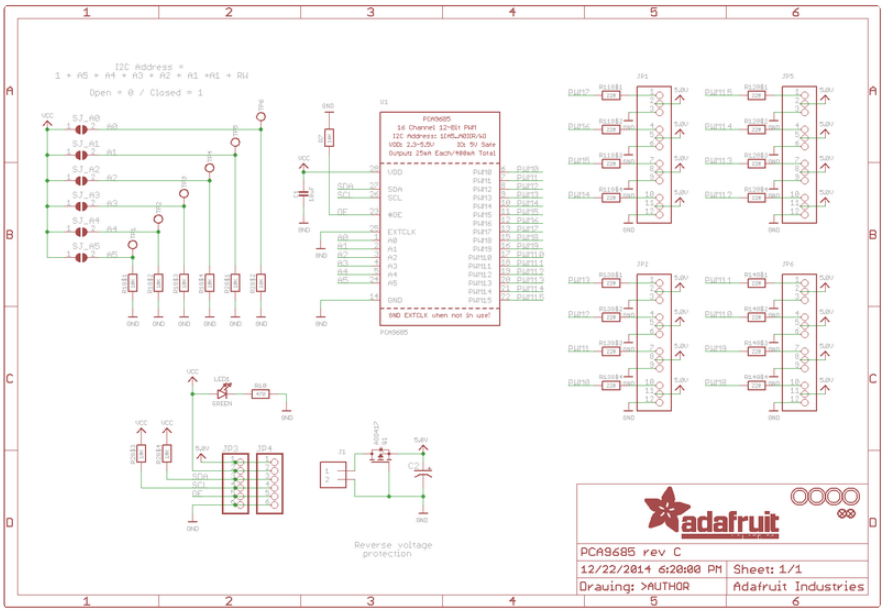
Python Docs: ServoKit (https://adafru.it/Dkx)

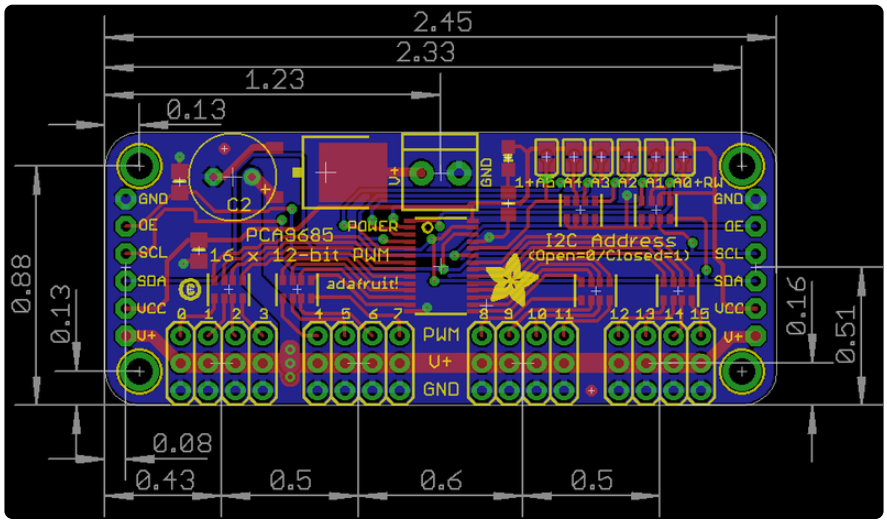
Downloads

Files

- PCA9685 datasheet (https://adafru.it/okB)
- Arduino driver library (https://adafru.it/aQl)
- EagleCAD PCB files on GitHub (https://adafru.it/rME)
- Fritzing object in the Adafruit Fritzing library (https://adafru.it/aP3)

Schematic & Fabrication Print





Holes are 2.5mm diameter

FAQ

Can this board be used for LEDs or just servos?

It can be used for LEDs as well as any other PWM-able device!

I am having strange problems when combining this shield with the Adafruit LED Matrix/7Seg Backpacks

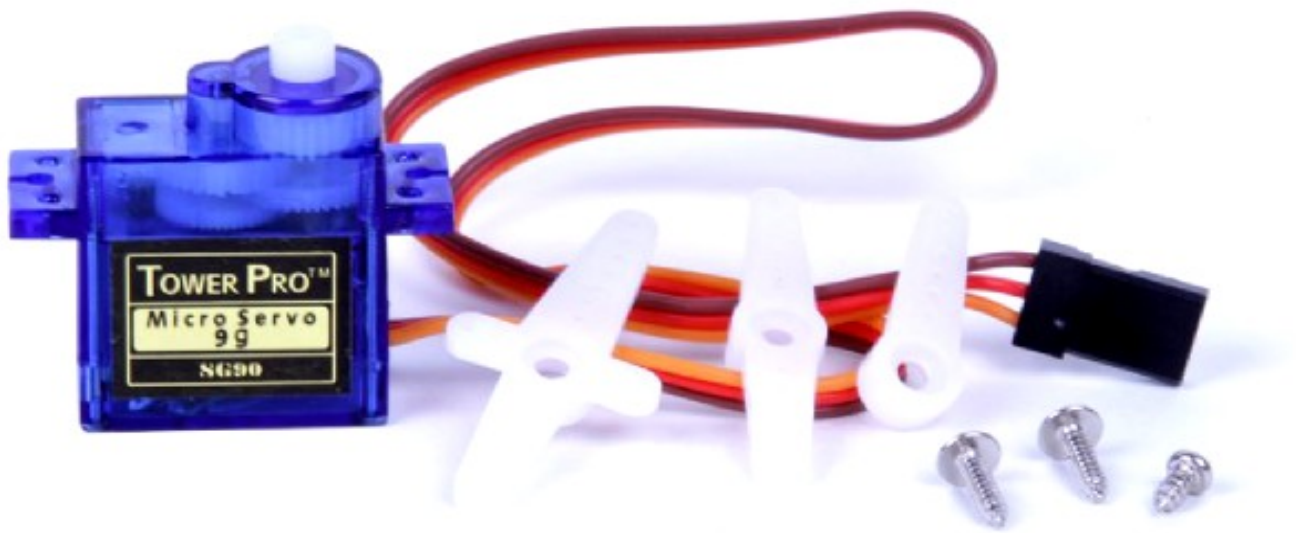
The PCA9865 chip has an "All Call" address of 0x70. This is in addition to the configured address. Set the backpacks to address 0x71 or anything other than the default 0x70 to make the issue go away.

With LEDs, how come I cant get the LEDs to turn completely off?

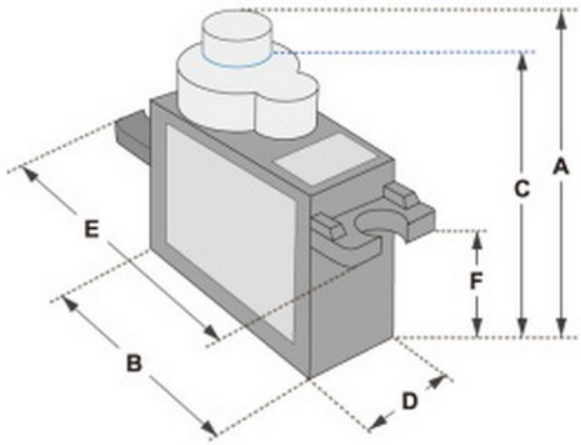
If you want to turn the LEDs totally off use setPWM(pin, 4096, 0); not setPWM(pin, 4095, 0);

Servomotor SG90

SERVO MOTOR SG90 DATA SHEET



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.



Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.