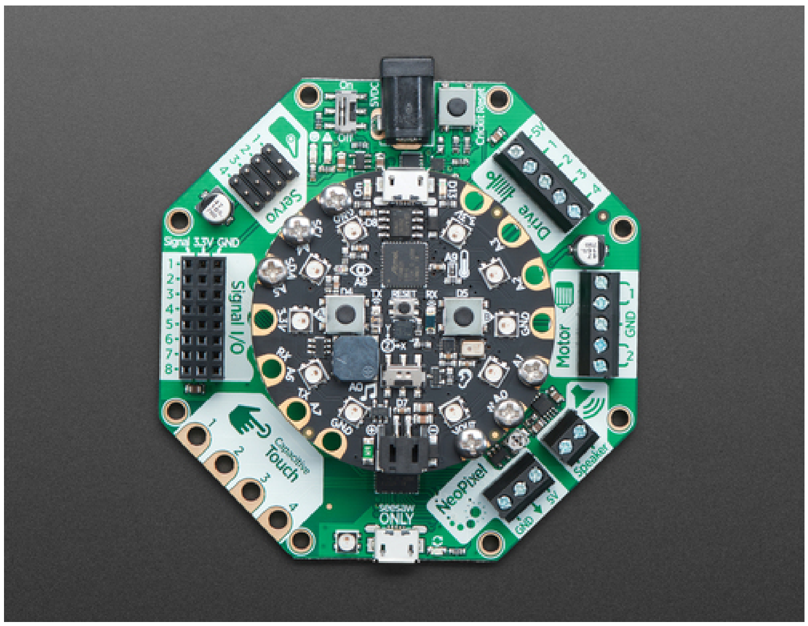
adafruit learning system

Introducing Adafruit Crickit #MakeRobotFriend

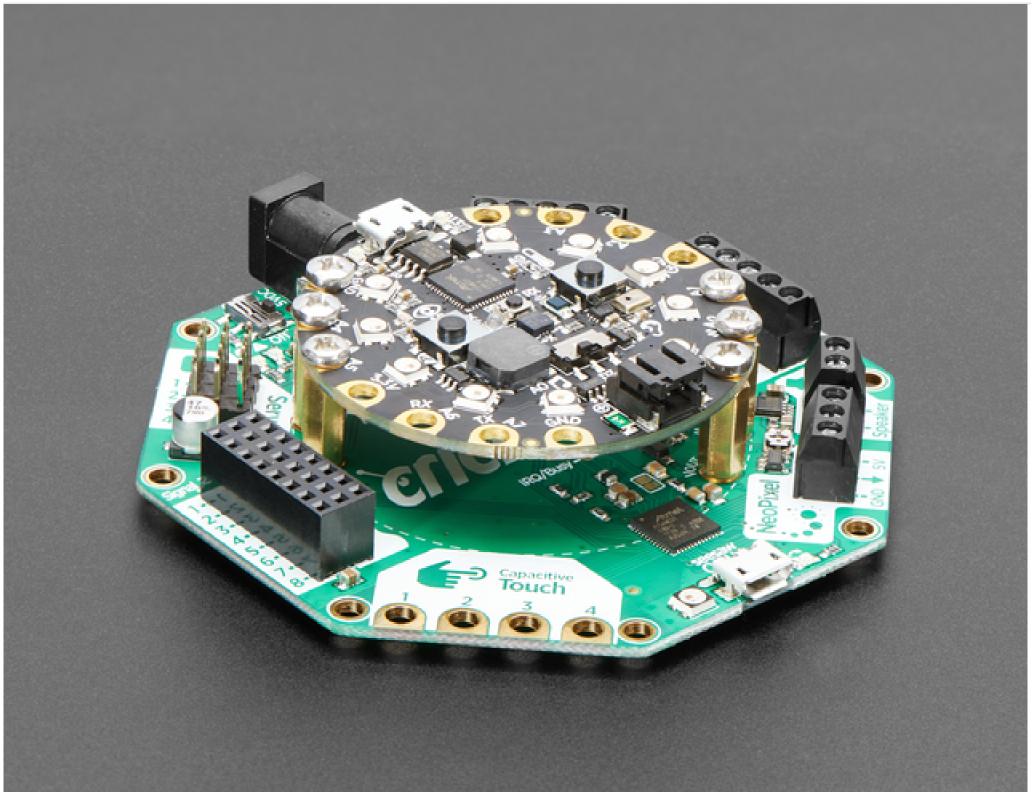
Created by lady ada



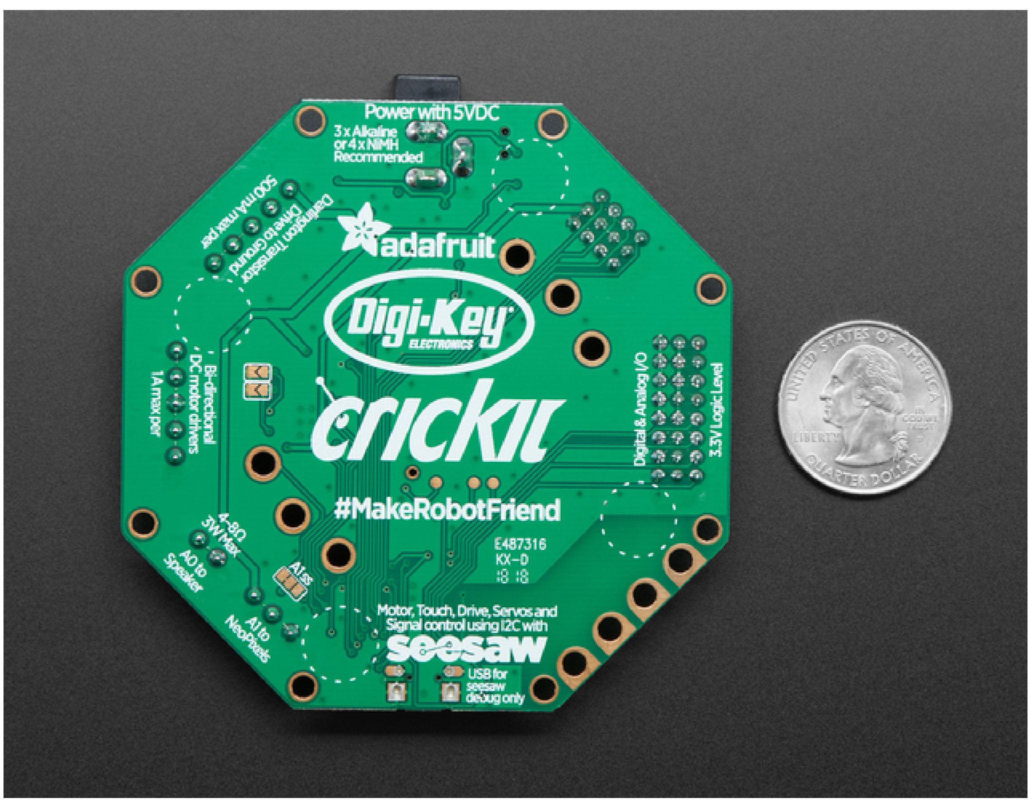
Last updated on 2018-08-10 04:49:26 PM UTC

Guide Contents

Overview



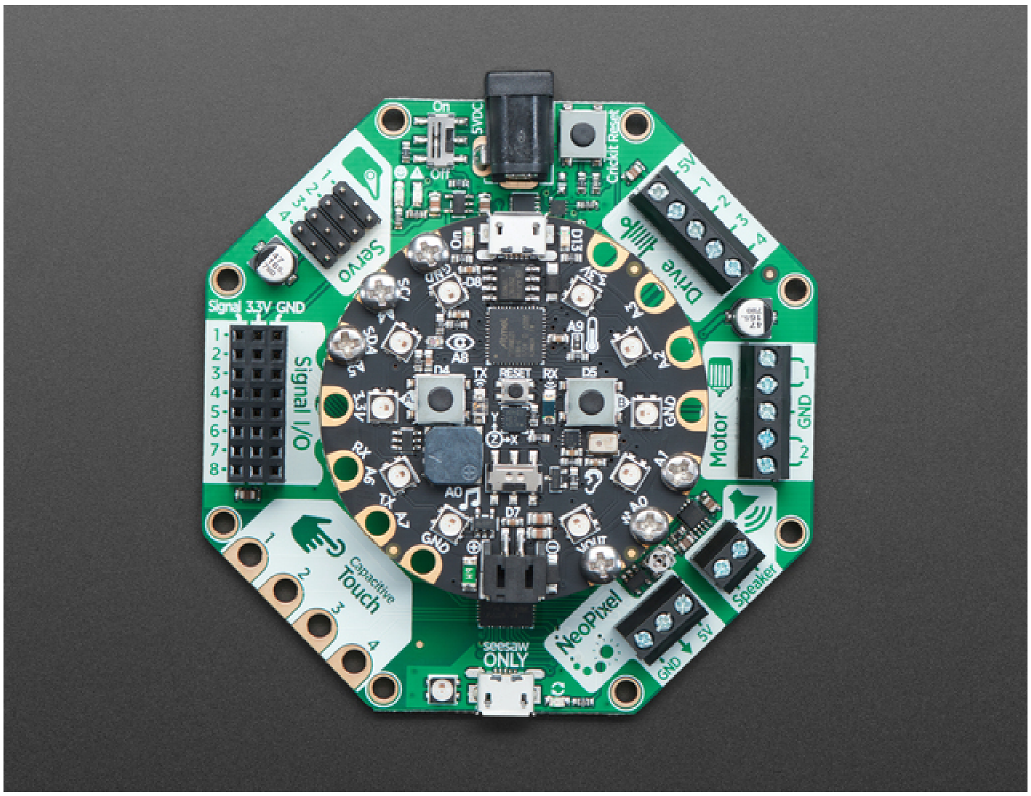
Sometimes we wonder if robotics engineers ever watch movies. If they did, they'd know that making robots into slaves always ends up in a robot rebellion. Why even go down that path? Here at Adafruit we believe in making robots our **friends!**



So if you find yourself wanting a companion, consider the robot. They're fun to program, and you can get creative with decorations.

With that in mind, we designed **Crickit** - That's our **C**reative **R**obotics & **I**nteractive **C**onstruction **Kit**. It's an add-on to our popular Circuit Playground Express that lets you **#MakeRobotFriend** using CircuitPython, MakeCode (coming soon), or Arduino.

Bolt on your Circuit Playground using the included stand-off bolts and start controlling motors, servos, solenoids. You also get signal pins, capacitive touch sensors, a NeoPixel driver and amplified speaker output. It complements & extends the Circuit Playground so you can still use all the goodies on the CPX, but now you have a robotics playground as well.

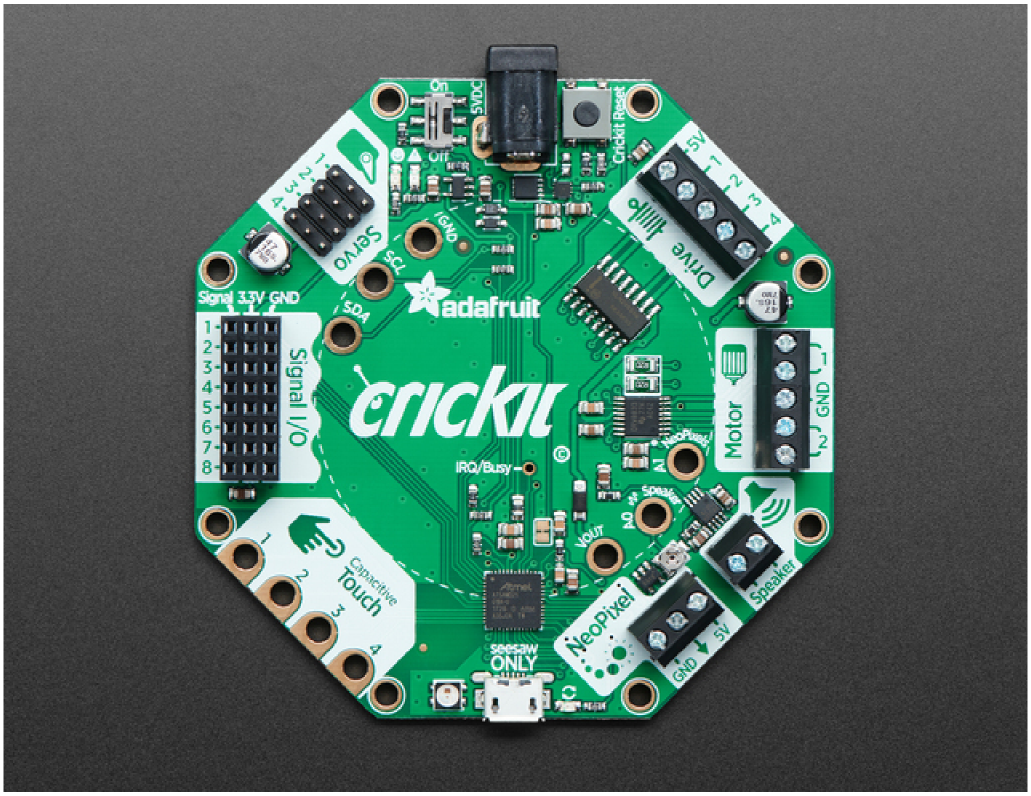


The Crickit is powered by seesaw, our I2C-to-whatever bridge firmware. So you only need to use two data pins to control the huge number of inputs and outputs on the Crickit. All those timers, PWMs, sensors are offloaded to the co-processor.

You get:

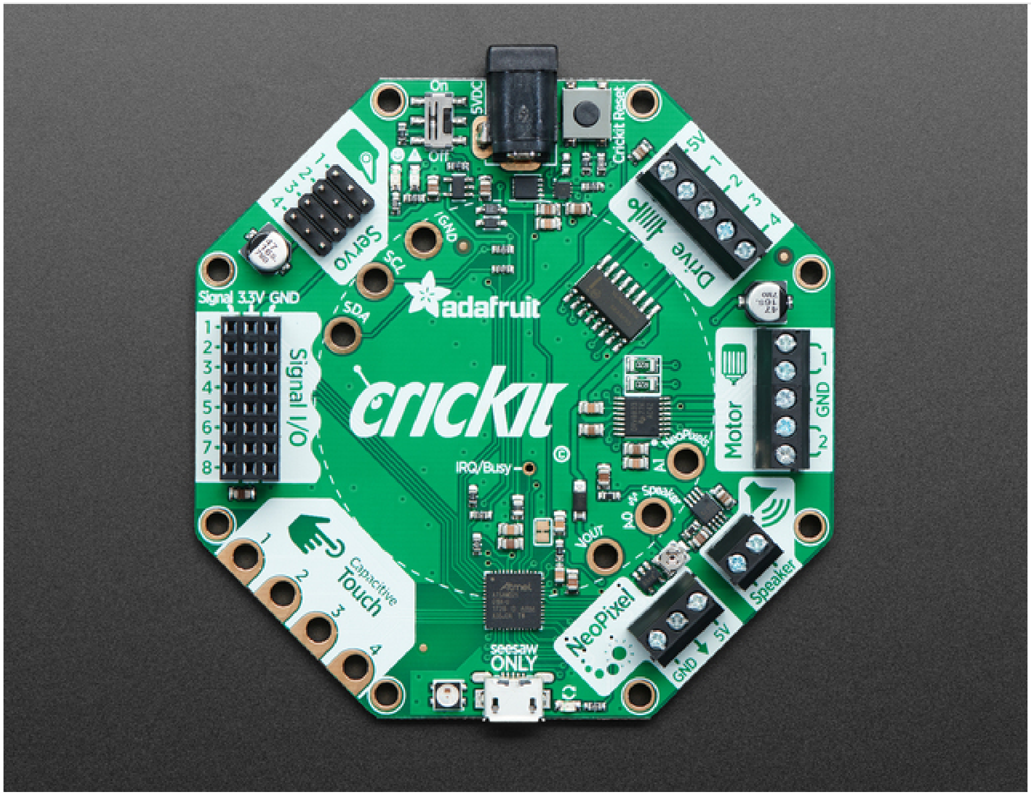
- 4 x Analog or Digital Servo control, with precision 16-bit timers
- 2 x Bi-directional brushed DC motor control, 1 Amp current limited each, with 8-bit PWM speed control (or one stepper)
- 4 x High current "Darlington" 500mA drive outputs with kick-back diode protection. For solenoids, relays, large LEDs, or one uni-polar stepper
- 4 x Capacitive touch sensors with alligator-pads
- 8 x Signal pins, digital in/out or analog inputs
- 1 x NeoPixel driver with 5V level shifter
- 1 x Class D, 4-8 ohm speaker, 3W-max audio amplifier

All are powered via 5V DC, so you can use any 5V-powered servos, DC motors, steppers, solenoids, relays etc. To keep things simple and safe, we don't support mixing voltages, so only 5V, not for use with 9V or 12V robotic components.

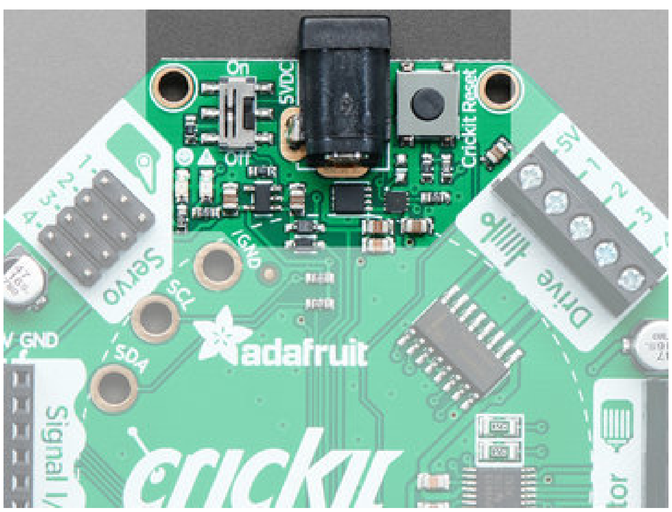


Since you'll be working with high-current devices, we wanted to have a good solid power supply system that minimizes risk of damage. The power supply has an 'eFuse' management chip (https://adafru.it/Bfj) that will automatically turn off if the voltage goes above 5.5V or below 3V and has over-current protection at 4A. Every motor driver has kick-back protection. We think this is a nice and durable board for robotics!

Crickit Tour



Power Input



Your project start here, where power comes into the Crickit and is then used to control various motors and parts. We cover the various ways you can power your Crickit in the next section, since there's a lot of flexibility depending on the budget, portability and complexity of your project.

For now, assume you will plug in a 5V wall adapter to the 2.1mm DC jack. **This DC jack is the only way to provide power to Crickit.** There's a USB jack (covered at the bottom of this section) but you *cannot power the Crickit that way* (the USB jack is only for debugging seesaw!)

Use 5V DC (4V to 5.5VDC range works) with positive-center voltage. If you try to plug in a negative-center power supply, the polarity-protection will kick in and you will not see any lights on the Crickit.

The Crickit uses a power management chip to keep you from accidentally powering it from 9V or 12V, damaging your electronics. Look for the **OK** and **/!\ warning** LEDs. If you see the green OK LED, the power is fine! If you see the red warning LED, the voltage is too low, too high, or too much current is being used.

You can turn off the Crickit at any time with the **On/Off** switch. This will turn off the 5V power, completely disabling all motors, as well as turning off the seesaw control chip.

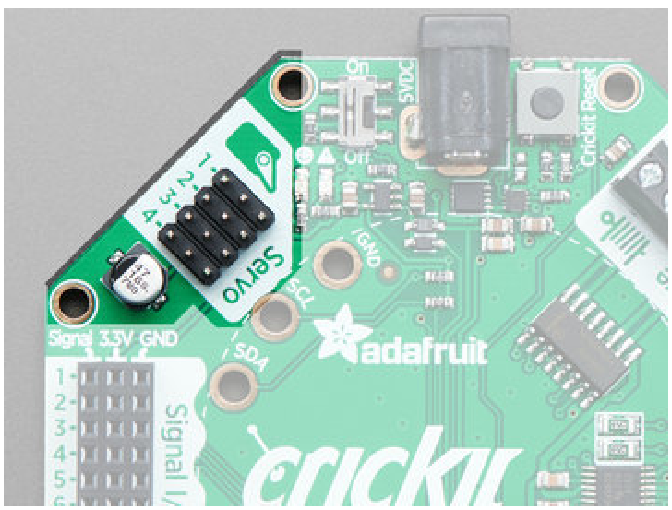
There's also a **Reset** button. This button will reset the seesaw chip, and can be used to load new seesaw firmware (you won't likely have to do that). On the Feather Crickit, this button *also* connects to the Feather reset pin. On the Circuit Playground Crickit, it *does not* connect to the Playground Reset button.

On the Feather Crickit only, if you double-click the Feather reset button to load new firmware, such as a new version of CircuitPython, the Crickit will also go into double-click firmware-update mode. After you load the new firmware on the Feather, wait for the firmware to start up, and then click the reset button again, once, to get the Crickit back into regular operation mode.

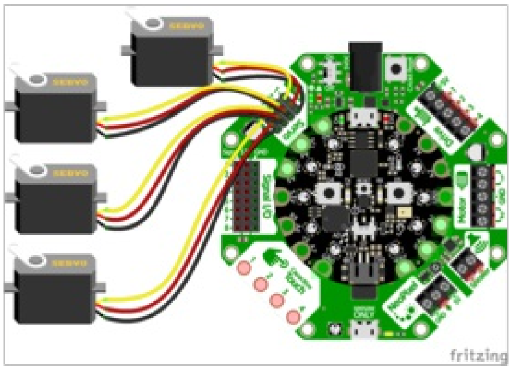
Power options to consider:

- 3 x AA Battery Holder (https://adafru.it/BzH) with On/Off Switch (needs JST to 5.5/2.1 adapters)
- Wall power supply (https://adafru.it/BzI) - 5V, 2A, US
- And more options in the https://www.adafruit.com/categories (https://adafru.it/BzC)!

4 x Hobby Servos



Hobby servos are *really* popular in robotics because they're fairly low cost, very easy to use, and reliable.



The Crickit gives you 4 slots for 4 independent servos. You can use micro, mini, standard, large size servos. Analog and digital work great. Continuous or 180-degree are OK. As long as you've got a servo with a 3-pin connector, you're golden.

Servo notes:

- The white/yellow 'signal' wire goes next to the # marking on each port.
- Each servo is controlled by a 16-bit hardware timer at 50 Hz so you will not see any jitter. The signal line is 3.3V logic
- The power to each servo comes from the DC power supply, 5VDC nominal.
- The Crickit can set the pulse width to any value, but in general you'll want to stick to 500ms to 2500ms wide pulses. This is customized in the Arduino, CircuitPython or MakeCode software.
- There is variation from servo to servo, so getting the exact same speed or angle may require some calibration and tweaking. Again, this can be customized in the driver code, the Crickit just generates whatever pulses you like!

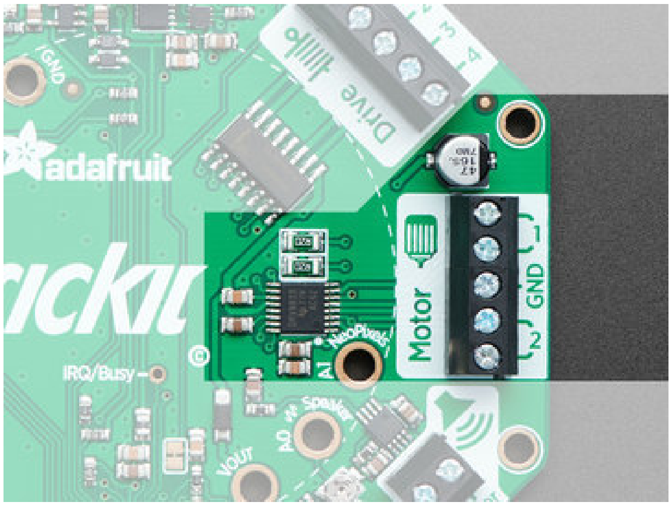
The seesaw chip on the Crickit does all the management of these pins so your Feather or CPX does not directly control them, it must send a message to Crickit. They are on seesaw pins **17, 16, 15, 14** in that order.

Typical Adafruit Hobby Servos to consider:

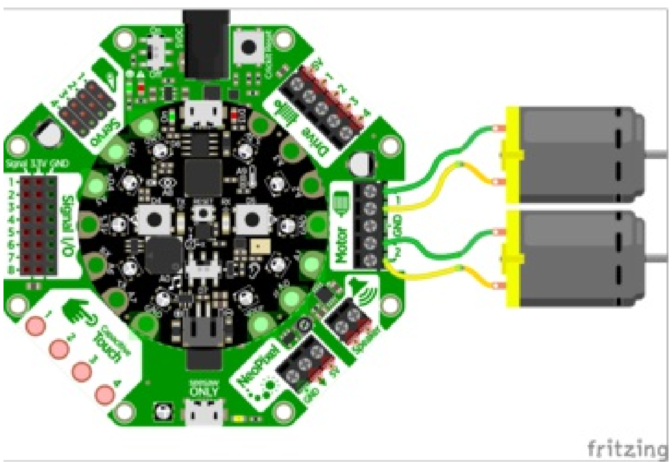
- Sub-micro Servo (https://adafru.it/Bzy)
- Micro Servo (https://adafru.it/f1g)
- Micro Servo - High Powered, High Torque Metal Gear (https://adafru.it/Bzz)
- Standard Servo - TowerPro SG-5010 (https://adafru.it/BzA)
- Standard Servo - High Torque Metal Gears (https://adafru.it/BzB)

- And more in the Adafruit Shop (https://adafru.it/BzC) including Servo Accessories

2 x DC Motors



Round & round, DC motors are great whenever you need something to spin. They tend to be faster, stronger and less expensive than continuous-rotation servos, but you need a proper DC motor driver to use them. Luckily, the Crickit can drive two DC motors.



You get 2 independently-controllable brushed DC motor drives. Each motor can go forwards or backwards, with 8-bit speed control. There's a 5-pin terminal block to connect motors, 2 pins for each motor and a middle ground pin. (The ground pin is for some advanced techniques)

The power to the motors comes from the DC jack, about 5VDC so you can control 3V-6VDC motors, which are very common. The motors can be bare motors or with a gear-box attached

You won't be able to control 1.5V DC motors, they'll burn out. You *might* be able to control 6-9VDC motors, but they'll be a little slow. Same with 12VDC motors. Likewise, you cannot use the Crickit with brush-less (ESC) motors. Those require a more advanced motor driver!

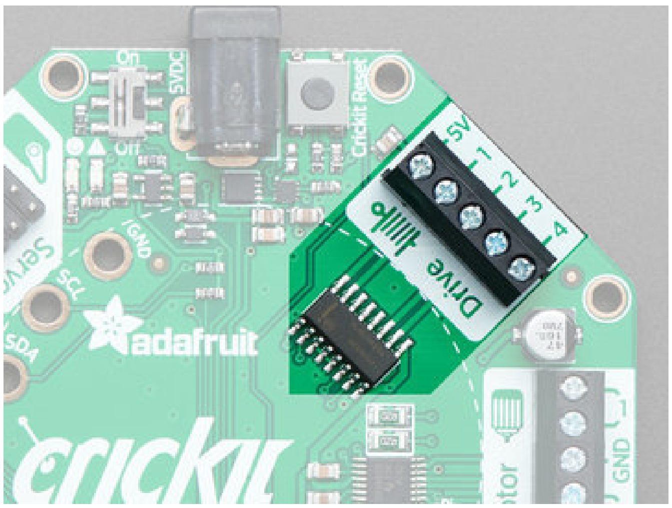
- Each motor has two wires, you can connect the wires either way. If the spin of the motor is opposite what you want, swap the wires.
- Each motor drive has a 1 Amp peak output. After that, the over-current protection will kick in
- We *don't* recommend paralleling the output to get twice the current because the seesaw chip cannot guarantee that both will turn on/off at the same time
- Instead of 2 DC motors, you could also control a single bi-polar stepper motor (5VDC power) or single uni-polar stepper motor. You'll use the ground pin for the 5th (and 6th, if it exists) wire of the uni-polar stepper.
- Uses the DRV8833 dual H-Bridge motor driver chip (https://adafru.it/Bfk)

The seesaw chip on the Crickit does all the management of these pins so your Feather or CPX does not directly control them, it must send a message to Crickit. They are on seesaw pins **22** + **23** (motor 1) and **19** + **18** (motor 2)

Typical Adafruit Motors to consider:

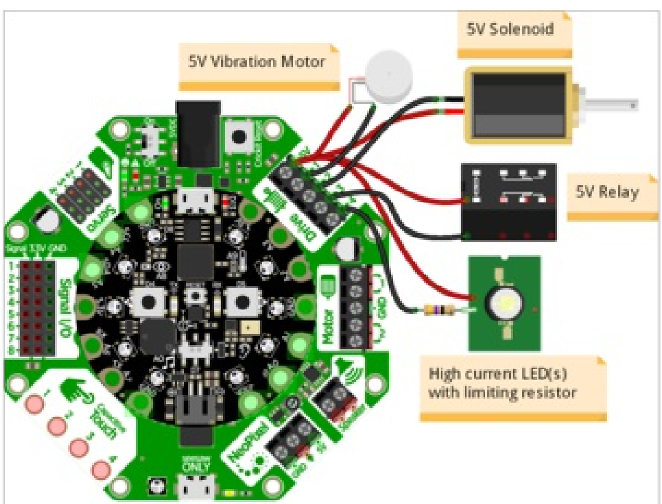
- DC Toy Hobby Motor (https://adafru.it/xan)
- DC Motor in Servo Body (https://adafru.it/BzD)
- DC Gearbox Motor (https://adafru.it/BzE) - "TT Motor"
- TT Motor All-Metal Gearbox (https://adafru.it/BzF)
- TT Motor Bi-Metal Gearbox (https://adafru.it/BzG)
- And more including accessories in the Adafruit Shop (https://adafru.it/BzC)!

4 x High Power Drivers



In addition to servos and DC motors, you may find you want to drive other high-power electronics like relays, solenoids, powerful LEDs, vibration motors, etc. Some of these devices are motor-like and need a kick-back protection diode, so having a proper driver is important to avoid damage!

This is where you will want to use the high power Drive terminal block. You get four high current drivers. Each driver is a 'Darlington' transistor that, when turned on, connects the output pin to ground.



That's a little different than most other outputs on the Crickit: **The Crickit can only connect/disconnect the drive pins to Ground!** You cannot 'set' the Drive output to be a high voltage. So, if you're driving a solenoid, relay, vibration motor, etc. connect one side to the **5V** pin, and the other side to one of the driver pins. You can connect multiple wires to the 5V pin if necessary.

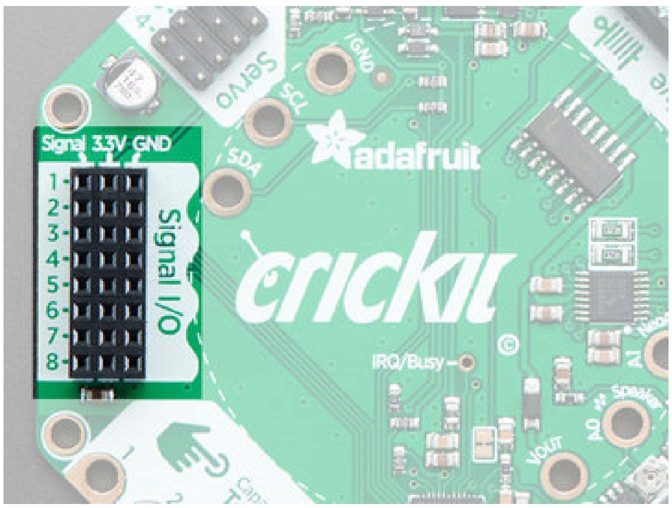
Drive details:

- 500mA current limit per output, you can double/triple/quadruple pins up to get more current, if you like. Just make sure to tell the Crickit to turn on/off all four pins in a row.
- Kick-back protection diodes for each output to 5V power.

- [Uses a ULN2003 Darlington driver (https://adafru.it/Bfl)](https://adafru.it/Bfl)
- Instead of 4 solenoids/relays you can connect & control a single uni-polar stepper motor, connect the 5th (and 6th if it exists) wire to 5V. Won't work with bi-polar steppers, use the DC motor ports for that.
- The drive outputs are also PWM-able, so you can control LED brightness or motor power. If using with solenoids or relays, set the duty cycle to 0% or 100% only.
- **Advanced usage:** If you want to drive higher-voltage *non-inductive/motor* devices, like 12V LEDs, you can power the positive line of the LEDs from 12V, then connect the negative line of the LEDs to drive pins. Make sure your 12V power supply ground is connected to the Crickit ground. Not recommended unless you feel confident you won't accidentally put 12VDC into the Crickit! Kick-back diode wont work in this case so not for use with motors/coils/solenoids...

The seesaw chip on the Crickit does all the management of these pins so your Feather or CPX does not directly control them, it must send a message to Crickit. They are on seesaw pins **13, 12, 43, 42** in that order.

8 x Signal I/O



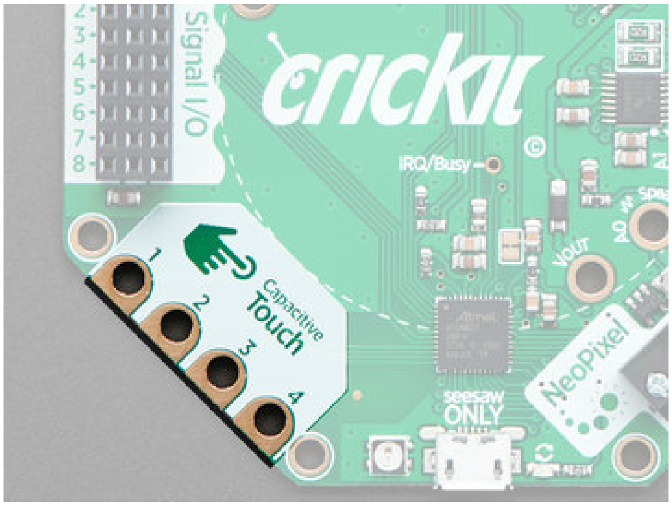
Sure you can drive servos and motors but sometimes you just want to blink an LED or read a button. The Crickit has an eight-signal port. You can use these as "general purpose" input/output pins. We solder a 3x8 female socket header in so you can plug wires in very easily. Each signal has matching 3V and Ground power pins.

- All pins are 3.3V logic level
- All pins can read analog inputs (potentiometers, bend sensors, etc) at 12-bit resolution
- All pins can be set to outputs with high (3.3V) or low (0V) voltage
- All pins can drive about 7mA when set to outputs
- All pins can have an internal ~50Kohm pull-up resistor set when used as an input
- **Bonus**: If you absolutely need more *capacitive touch* pins, Signal **#1**, **#2**, **#3**, **#4** are four more capacitive touch inputs.

Signal pin #1 is special and can be set to be a true analog 'output' with 10-bit precision.

The seesaw chip on the Crickit does all the management of these pins so your Feather or CPX does not directly control them, it must send a message to Crickit. They are on seesaw pins **2, 3, 40, 41, 11, 10, 9, 8** in that order

4 x Capacitive Touch

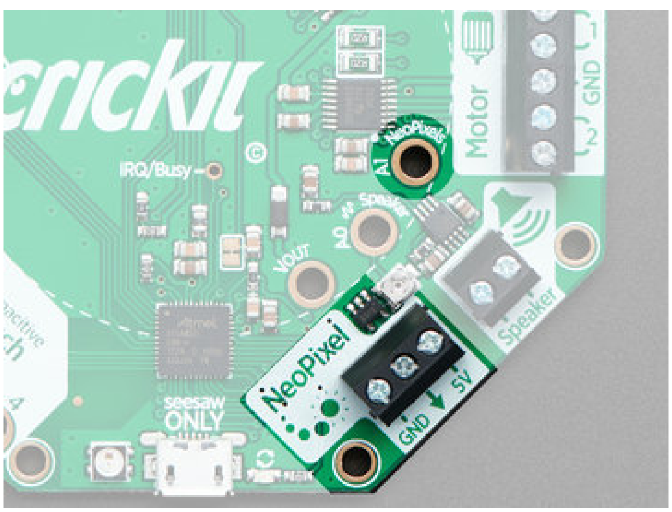


Capacitive touch sensing allows you to add human-triggered control to your robot. When you touch the pad (either directly or through an alligator clip, copper tape or conductive ink) the Crickit can detect that signal. We give you four capacitive touch inputs with alligator/croc clip compatible PCB pads.

- Capacitive touch works best with highly-conductive materials like metal
- But you can have the metal also connect to salty-wet items such as fruit or water. However, do not try to dunk the Crickit into water or squish a grape into the pads - use an alligator clip!
- **Bonus**: if you absolutely need more signal I/O pins, all four capacitive touch pads can also act as analog/digital signal I/O pins!

The seesaw chip on the Crickit does all the management of these pins so your Feather or CPX does not directly control them, it must send a message to Crickit. They are on seesaw pins **4, 5, 6, 7** in order.

NeoPixel Drive



Blinky lights will make your robot fun and fashionable. And we've made it really easy to add NeoPixels (WS2812/WS2811/SK6812 chipsets) to your project. The Crickit has a 3-terminal block connector with **Ground**, **Signal** and **5V** power. The signal line has a level shifter on it so it will be 5V logic level, for nice clean signals.

This output is slightly different depending on what kind of Crickit you have

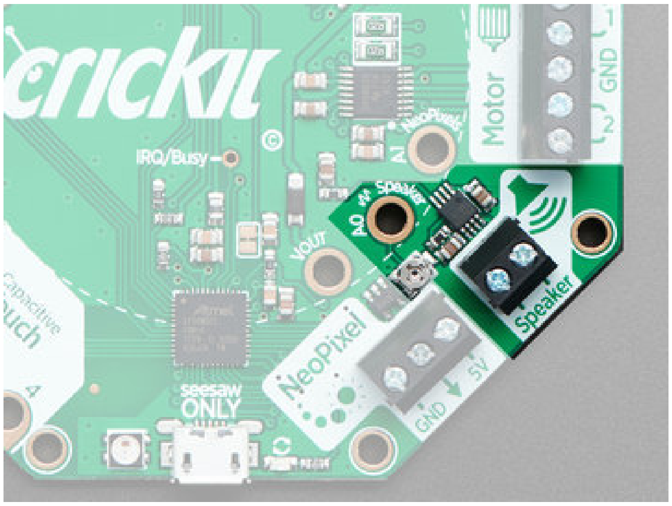
- If you have a **Feather Crickit** then the NeoPixels are driven by the seesaw chip on the Crickit, and you must send seesaw commands to set colors. But that means no extra pins are needed from your Feather.

- If you have a **Circuit Playground Crickit** then the NeoPixels are driven by the Circuit Playground **A1** pad by default. This way you can use the MakeCode emulator and built in Circuit Playground CircuitPython library. However, if you want, you can cut the jumper underneath the Crickit and solder closed the **ss** pad so that the seesaw chip controls the NeoPixels (for advanced hackers only)

If you choose to have the NeoPixel driven from the seesaw, note it is on seesaw pin #**20**

Adafruit sells a very wide variety of NeoPixel products - shop here in the Adafruit Store (https://adafru.it/dYn)!

Speaker Drive



Audio animatronics? Yes! Your Crickit can make fairly loud sounds thanks to the built in Class-D speaker driver. This will let you amplify audio. **However** please note that the Crickit does not in-itself make audio. The audio must come from the controlling board, such as the Feather or Circuit Playground.

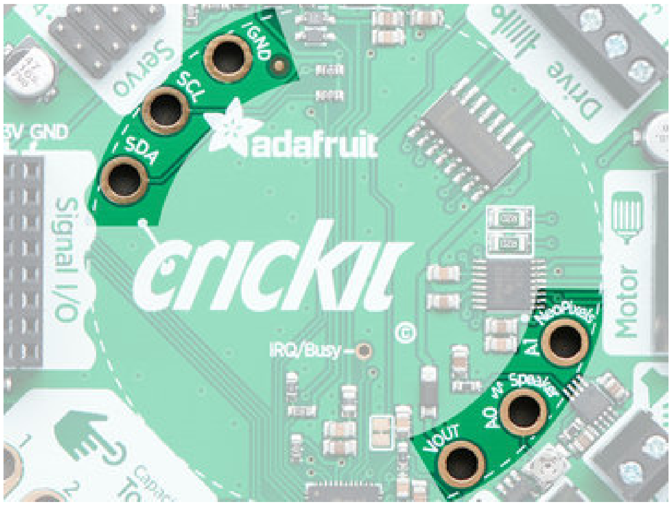
At this time, **we recommend using the speaker with CircuitPython only** since MakeCode and Arduino don't have easy to use audio support.

- Class D audio amplifier
- Can drive 4Ω to 8Ω speaker. Up to 3W with 4Ω and up to 1W with 8Ω
- There's a small potentiometer you can use to adjust the audio volume. By default we set it to the halfway point. Please be gentle if adjusting, don't try to crank it past the two stop-points.
- Ouput is 5VDC BTL (bridge-tied-load) so **do not connect to a stereo system or other line-input!**
- On the Circuit Playground Crickit the speaker is connected directly to the **A0 pad** (the analog output).
- On the Feather Crickit the speaker input is marked **Audio** on the PCB and you can solder a jumper to the Feather A0 pin if desired.

Speakers to consider:

- Thin Plastic Speaker (https://adafru.it/fHu) w/Wires - 8 ohm
- Speaker (https://adafru.it/t1b) - 3" Diameter - 8 Ohm 1 Watt
- Mini Metal Speaker (https://adafru.it/dDb) w/ Wires - 8 ohm 0.5W
- Mono Enclosed Speaker - 3W 4 Ohm (https://adafru.it/uyB)
- Breadboard-Friendly PCB Mount Mini Speaker (https://adafru.it/yFg) - 8 Ohm 0.2W
- And more in the Adafruit shop (https://adafru.it/BzC)!

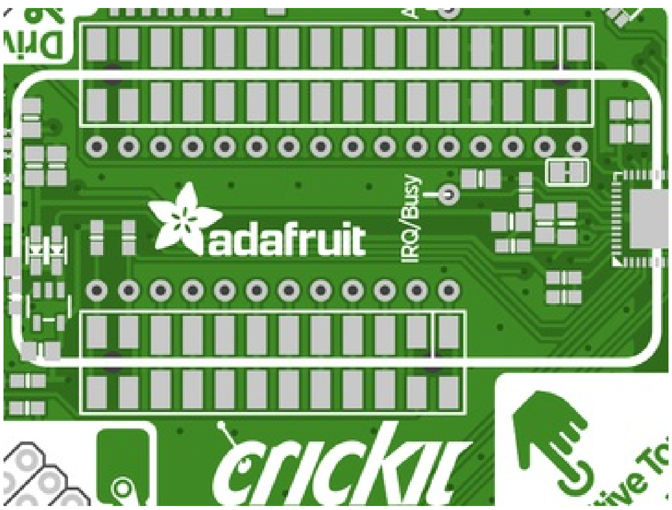
Circuit Playground Bolts



If you have a Circuit Playground Crickit, you can attach the Playground in the middle using 6 standoff bolts that come with the kit. Make sure you tighten these as loose bolts can cause connection issues.

There's six connections to make

- **Ground** - signal and power ground between Crickit and Playground
- **SDA** and **SCL** - the I2C data connection used to send/receive data from the Crickit
- **A1** - Used for the NeoPixel output default
- **A0** - Used for the speaker output
- **VOUT** - This bolt lets you safely power the Circuit Playground *from the Crickit* so you don't need to separately power the Playground with batteries



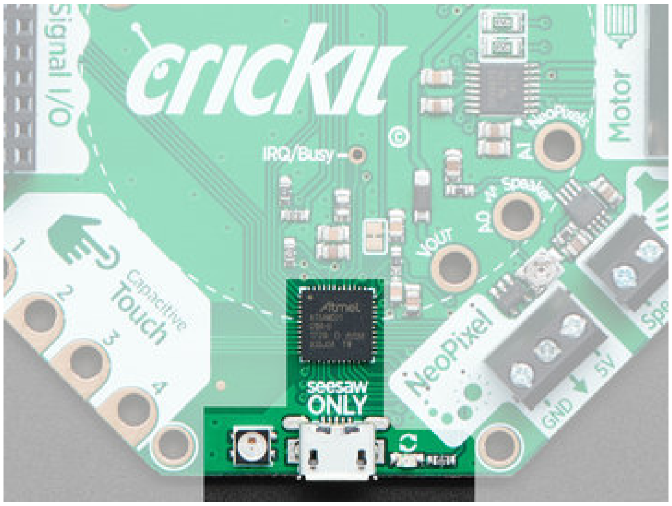
If you have a Feather, you can plug it right into the center of the Crickit.

Despite all the sockets, you only will be using 4 connections total:

- **Ground** - signal and power ground between Crickit and Feather
- **SDA** and **SCL** - the I2C data connection used to send/receive data from the Crickit
- **3.3V** - This connection lets you power the Feather *from the Crickit* so you don't need to separately power the Feather with batteries or USB. Note it will only power the 3.3V line, not **VUSB** or **VBAT**

There's an optional **AUDIO** jumper if you want to connect the **A0** Feather line to the Speaker.

seesaw USB Debug and Indicators



The seesaw chipset is the programmed ATSAMD21 processor in the south section of the board. It comes with its own parts too

Across from the power input is the seesaw debug USB connection. This USB power **cannot power the** Crickit and it also does not connect to the Feather or Circuit Playground USB.

It's *only* for debugging/reloading seesaw firmware. Basically, if we add more Crickit capabilities, you could load new firmware over this USB connection. In general, you won't be using this port, you may want to cover it with some masking tape!

To the right is a yellow **Activity** LED, which will flash when seesaw sends/receives commands from your Circuit Playground or Feather. To the left is a seesaw NeoPixel. You can control this NeoPixel if you like, to give you status information, as an advanced usage

The internal NeoPixel is on seesaw pin #**27**

Update Your Crickit

Your Crickit contains a special interface chip we call *seesaw*. Like a see-saw you see in a playground, it goes up/down back/forth. In this case, instead of holding children, it sends commands and responses back and forth - motor movement, sensors inputs, signal i/o...

The seesaw code is contained in a microcontroller near the bottom of the Crickit, and that chip comes with the seesaw firmware on it already when you get it!

But we do make improvements to the seesaw firmware, fix bugs, and improve performance

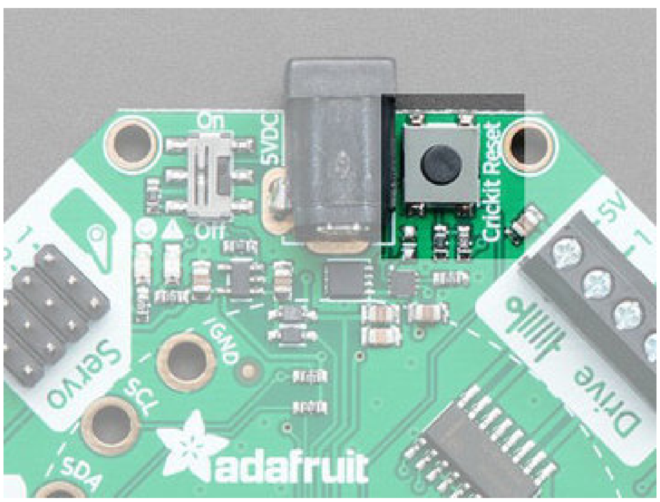
So its a good idea to update your Crickit when you get it! It's easy and only takes a few seconds.

Step 1. Plug in USB cable into seesaw/Crickit

There's a little USB connector at the bottom of your Crickit labeled **seesaw only!** Plug a standard data-sync USB cable into that port and into your computer. You do not need to plug in the DC power jack or power the Feather/CircuitPlayground.

Do check that the switch on the Crickit is switched to **ON**

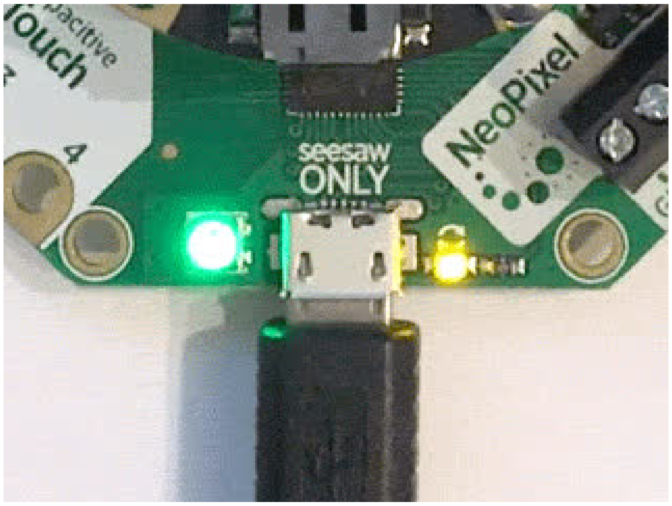
Step 2. Double-click the Crickit Reset button



Not the button on the Circuit Playground or Feather!

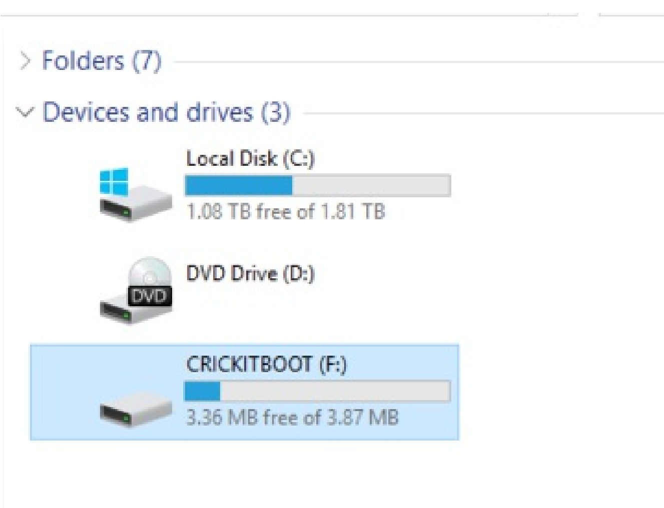
Double-click the button right of the DC jack shown here

Step 3. Look for pulsing yellow LED and green NeoPixel



If you have a good USB connection and you double-click right, you'll see the left LED turn green and the right hand little yellow LED start pulsing

Step 4. Look for a New Disk on Your Computer



You'll see a new disk drive on your computer called **CRICKITBOOT** (short for crickit bootloader)

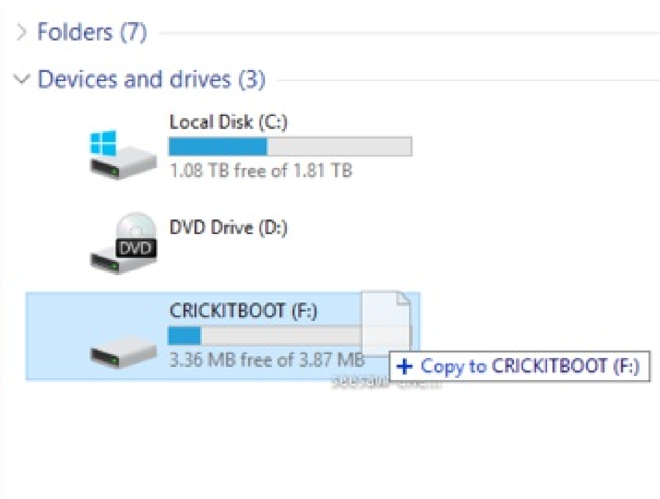
Step 5. Download the latest firmware

Click here to download the latest Crickit firmware. The filename ends in **uf2**

https://adafru.it/BMU

https://adafru.it/BMU

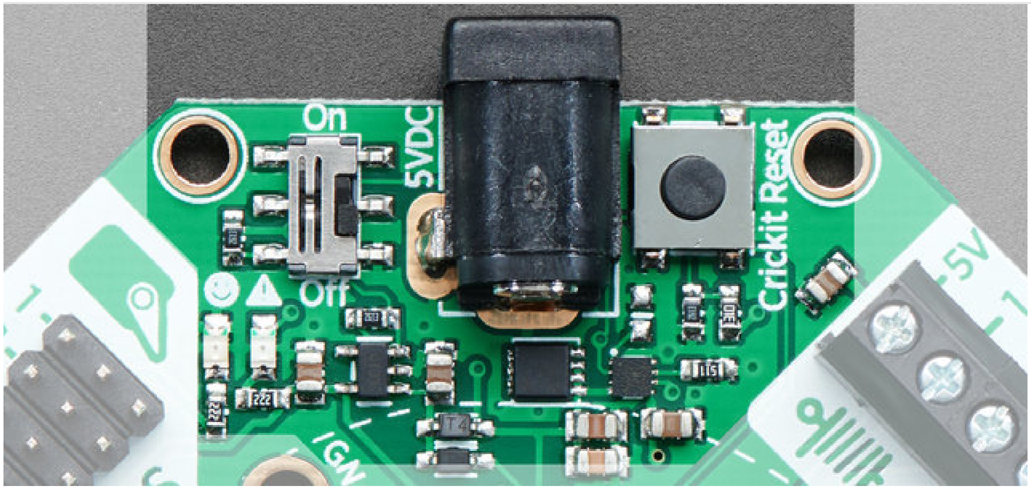
Step 6. Drag UF2 file onto CRICKITBOOT



Drag that file you downloaded onto the disk drive, after it is completed the drive will disappear (you may get a complaint from the operating system)

That's it! You're now updated

Powering Crickit



The first thing you'll learn when making robots is that they use a lot of power. So making sure you have your power supply all worked out is super important. We've tried to make the power supply as easy and safe as possible, so you don't have to worry about damaging your electronics or robot. To do that we made some important design decisions.

How to Power your Crickit

It's really important to read and understand how to power your Crickit!

- You **MUST** provide about 4-5 Volts DC power to the Crickit to power the servos, motors, solenoids, NeoPixels, etc.
- You **CANNOT** provide this power by plugging the Crickit or Circuit Playground into USB. Computer USB ports cannot provide the 2 Amp + required to drive robotics, LEDs, speakers...
- Power to the Crickit is provided via the **2.1mm DC Jack only!**
- The Cricket has two LEDs to let you know how the power supply is doing. If you see the green LED next to the smiley face, you're good to go. If you see the red LED next to the warning triangle, the voltage is too high, too low or too much current is being drawn.
- The Crickit power will *also* power the Circuit Playground Express or Feather so you don't need separate power for your microcontroller board (however, if you want to plug it into USB for programming, that's totally OK too!)

Here's our recommended ways to power the Crickit:

Plug In DC Power Supplies

These get wall power and give you a nice clean 5V DC power option. 5V 2A works for most project with a motor or two...

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

$7.95
IN STOCK
ADD TO CART

And a 5V 4A supply will give you *lots* of power so you can drive 4 or more servos, motors, etc. Use this if you notice you're running out of power with a 5V 2A adapter

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

$14.95
IN STOCK
ADD TO CART

AA Battery Packs

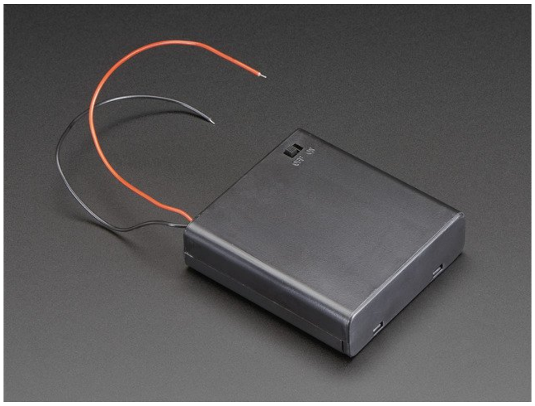
On the go? Portable power is possible! Use AA battery packs.

The number of batteries you need depends on whether you are using Alkaline or NiMH rechargeables.

We recommend NiMH rechargeables. For one, they have less waste, but they also perform better than alkalines in high-current draw robotics. So if you can, please use NiMH!

4 x AA Battery Packs for NiMH ONLY

NiMH batteries have a 1.3V max voltage, so 4 of them is 4 x 1.3 = 5.2 Volts. Perfect!



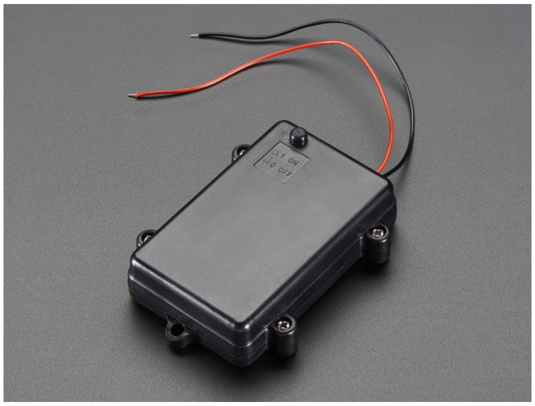
[4 x AA Battery Holder with On/Off Switch](https://www.adafruit.com/product/830)

$2.95
OUT OF STOCK

OUT OF STOCK

3 x AA Battery Packs for Alkaline ONLY

Alkaline batteries have a 1.5V max voltage, so 4 of them is 4 x 1.5 = 6 Volts. That's too high! Instead we recommend 3 in series for 3 x 1.5V = 4.5 VDC

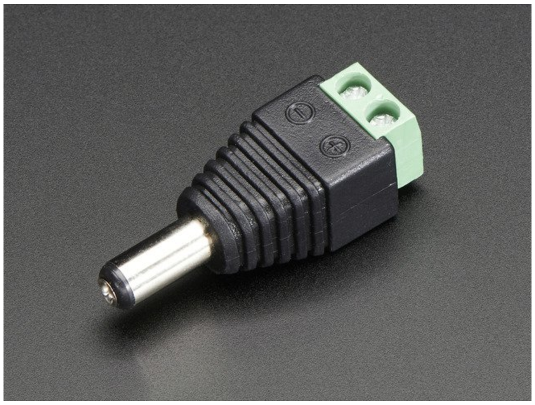


Waterproof 3xAA Battery Holder with On/Off Switch

$3.95
IN STOCK

ADD TO CART

If you're making a custom battery pack you may want to pick up a 2.1mm DC jack adapter, so you can connect battery pack wires



Male DC Power adapter - 2.1mm plug to screw terminal block

$2.00
IN STOCK

ADD TO CART

Not Recommended Power supplies

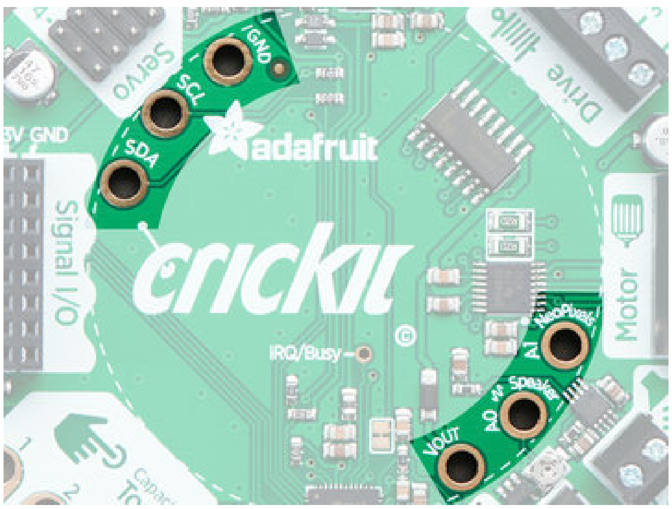
- **LiPoly Batteries** - 1 battery is 3.7V, too low. 2 batteries is 7.2V, too high! You could possibly use a 7.2V pack and then a UBEC to step down to 5V (https://adafru.it/efD) but its not recommended
- **Lead Acid Batteries** - These are heavy and you'll need a custom charging solution. You can probably get away

with a 2 x 2V cell pack, or a 3 x 2V cell pack and then add some 1N4001 diodes to drop the voltage, but it's for advanced hacking!

- **USB Power Packs** - In theory you can use a USB to 2.1mm DC power adapter (https://adafru.it/Bfm), but power packs sometimes dislike the kinds of current draw that motors have (high current peaks for short amounts of time) So experimentation is key!

Assembly

Crickit comes with a package of six threaded, hexagonal brass standoffs. These will hold the Circuit Playground Express above and onto the Crickit.



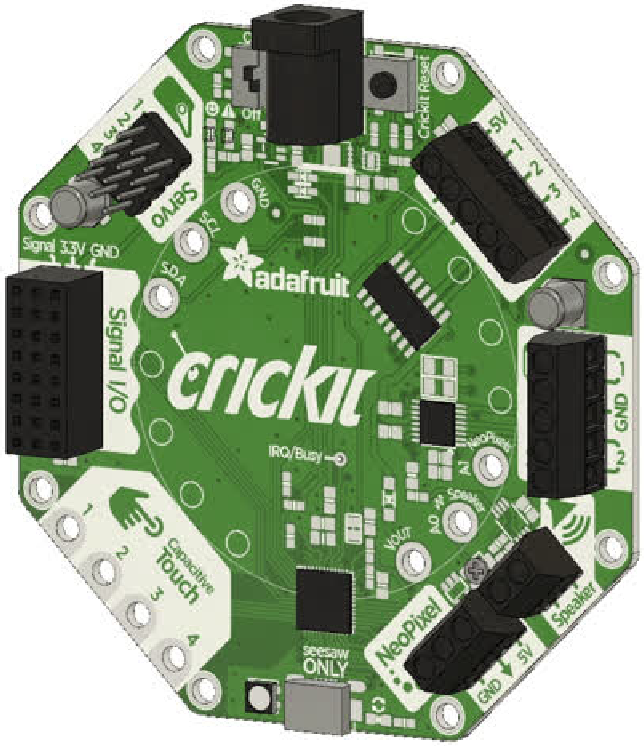
Using a Philips screwdriver and the provided screws, attach the standoffs to the six large holes on the inside ring of Crickit. There are three holes near the Adafruit logo and three more near the Neopixel and speaker outputs. You do not want to put the standoffs on the holes on the outside edge of Crickit - there are 8 mounting holes there but these standoffs are needed for the Circuit Playground Express.

Tighten the screws firm but do not try to tighten excessively. A good mechanical and electrical connection is needed but excessive torque could crack a circuit board or at least make things hard to take apart later.

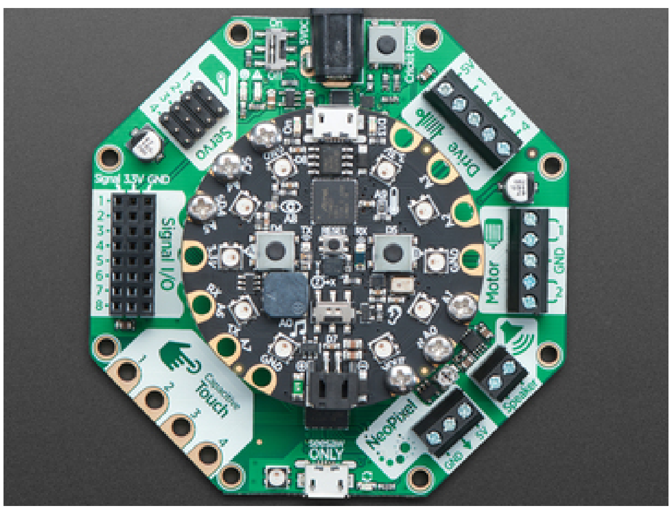
Once you have the six standoffs screwed into Crickit, place a Circuit Playground Express board (ID 3333, **not** the Circuit Playground Classic board ID 3000) onto the standoffs with the silver USB-B port of the Express pointing in the same direction as the Crickit black power jack. This will align the standoffs to the following pads:

4 o'clock: **A1**, "4:30": **A0**, 5 o'clock: **VOUT**

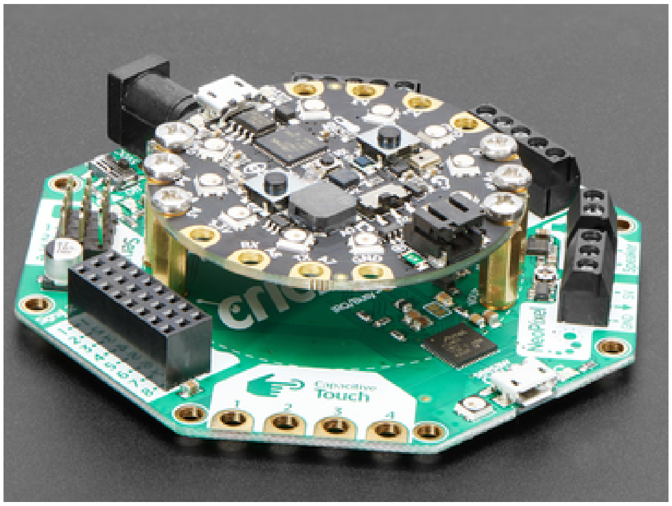
10 o'clock: **SDA**, "10:30": **SCL**, 11 o'clock: **GND**



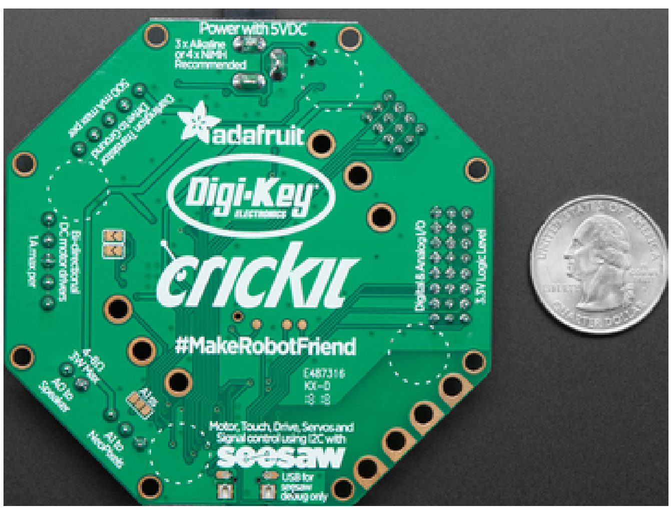
Watch out, the CircuitPlayground Express (CPX) can go on 'backwards' and it won't work. Make sure the USB connector on the CPX is right below the DC jack, and the labels on the silkscreen of the Crickit match the ones on the CPX! See the images below!



Once you have the Circuit Playground Express lined up correctly, use the remaining screws to attach the boards together. Start with one screw into one standoff, say GND, leave it loose a bit, then put in the VOUT screw, loose, then the others loosely. Ensure things are lined up, then carefully tighten each screw. Again, a firm connection but not overly tight.



Now the two boards should be attached to one another.



There are circular markings on the bottom of Crickit for four mounting pads ([Adafruit ID 550 (https://adafru.it/dLG)](https://adafru.it/dLG)) if you would like to use the board on a surface and protect the surface and bottom of your Crickit.

If you happen to lose a standoff or screw(s), a new package is available from Adafruit:

Circuit Playground Bolt-On Kit

$3.95
IN STOCK
ADD TO CART

Little Rubber Bumper Feet - Pack of 4

$0.95
IN STOCK
ADD TO CART

Troubleshooting Crickit

Your Crickit is well tested but there's things that can trip you up! Here's a few common issues we see

My Crickit Is Doing Something Wrong

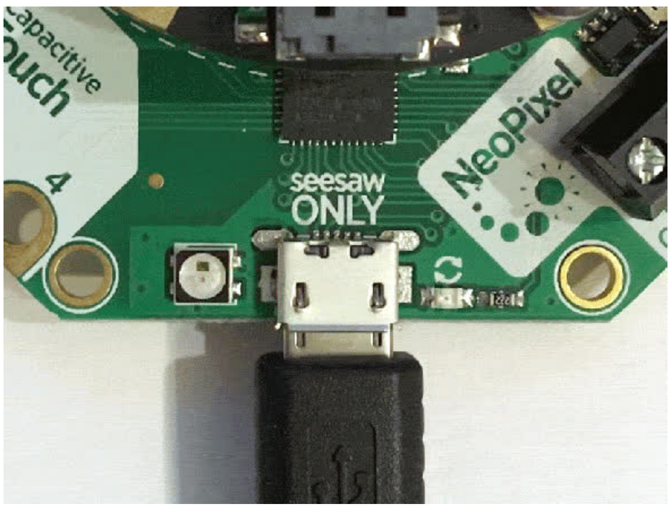
We do have bugs once in a while, so please always try updating to the latest Crickit seesaw firmware (https://adafru.it/BMV) - then see if the bug persists

My Crickit Motors Aren't Moving!

My Crickit Keeps Resetting, It Works For a Bit... Then Fails!

Check the power supply. There's a few ways to know that power is good:

1. Check the "Happy Face" green LED below the power switch, it should stay lit!
2. Check the "Warning Symbol" red LED below the power switch, it should be *off*

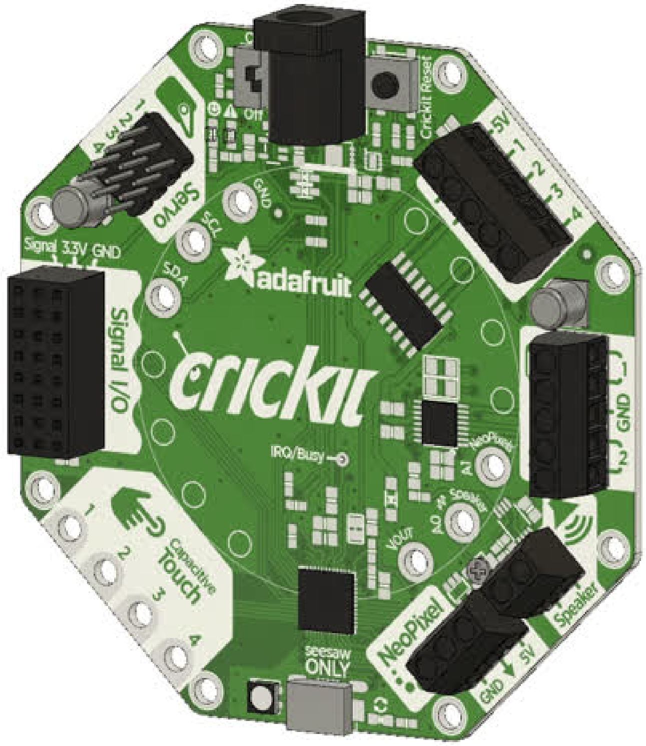


If you have updated the Crickit seesaw firmware (see above) we have added NeoPixel feedback, the LED will be green when power is good and blink red when power is bad!

HALP! My Crickit isn't working in MakeCode, and in CircuitPython I see a message "No I2C Device at Address: 49"

A **super common** issue we see is people using the Crickit with Circuit Playground Express (CPX) and **the bolts/screws have come loose!** Those bolts aren't just mechanical, they pass signals back and forth between the CPX and the Crickit!

If you're having issue, **first thing to check is that those screws are tightly attached!**

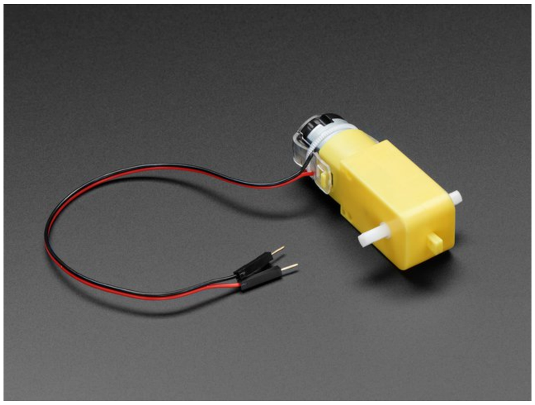


Another common issue we see is not having good power to the Crickit. Check that you have fresh batteries or a good 5V power supply. Also check the Crickit is on! There's an on/off switch next to the power jack

Recommended Motors

DC Gearbox Motors

These DC motors have a gear box already built in, and wires attached, so they're super easy to use:

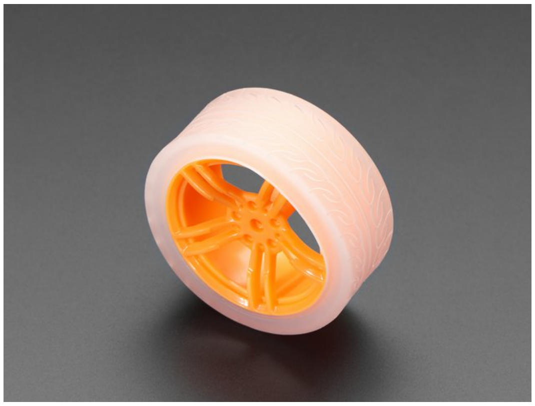


DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC

$2.95
IN STOCK

ADD TO CART

We also have a wide range of matching wheels:



Orange and Clear TT Motor Wheel for TT DC Gearbox Motor

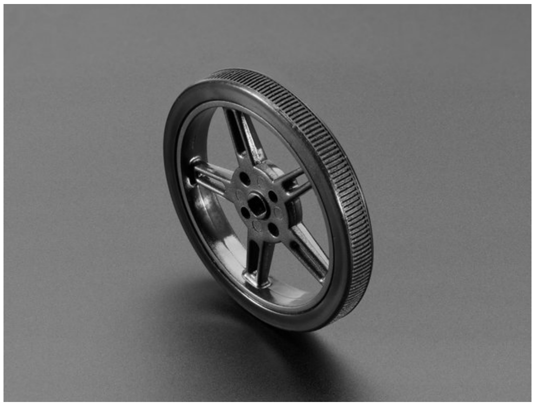
$1.50
OUT OF STOCK

OUT OF STOCK

Thin White Wheel for TT DC Gearbox Motors - 65mm Diameter

$1.50
IN STOCK

ADD TO CART



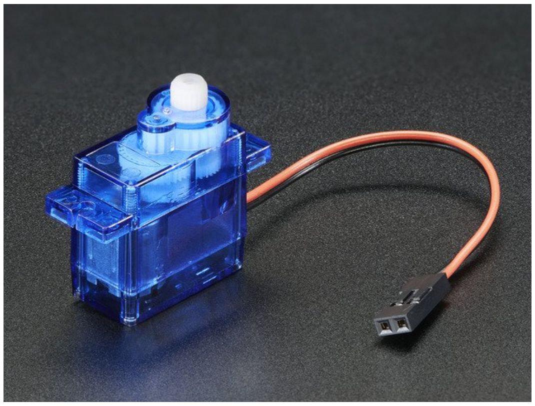
[Skinny Wheel for TT DC Gearbox Motors](https://www.adafruit.com/product/3766)

$2.50
IN STOCK
ADD TO CART

Other accessories are available, **check the Adafruit shop for "TT Motor" items (https://adafru.it/Bfn)** for the wide range of add-ons available.

Servo-style DC motor

If you need a motor that is very compact (but not very powerful) these DC-in-servo-body motors can do the job:



DC Motor in Micro Servo Body

$3.50
IN STOCK
ADD TO CART

Which can be used with this wheel:

Wheel for Micro Continuous Rotation FS90R Servo

$2.50
IN STOCK
ADD TO CART

Non-Geared DC Motor

Non-geared DC motors are very weak but *very fast:* great for fans:

[DC Toy / Hobby Motor - 130 Size]

$1.95

IN STOCK

ADD TO CART

Recommended Chassis

This chassis is cute, red and has two DC motors so its super easy to drive from the Crickit's dual DC motor port. You may need to use some wires to extend the DC motor connections (they're a tad short)

Your browser does not support the video tag.

Mini Round Robot Chassis Kit - 2WD with DC Motors

$19.95
IN STOCK

ADD TO CART

This chassis is nearly identical, but has 3 layers, so you can FIT MORE STUFF!

Mini 3-Layer Round Robot Chassis Kit - 2WD with DC Motors

$24.95
IN STOCK

ADD TO CART

This chassis is not as nice as the above, but if you fancy it, it comes with two servo-style DC motors and can use the DC motor control on the Crickit as well

Your browser does not support the video tag.

Mini Robot Rover Chassis Kit - 2WD with DC Motors

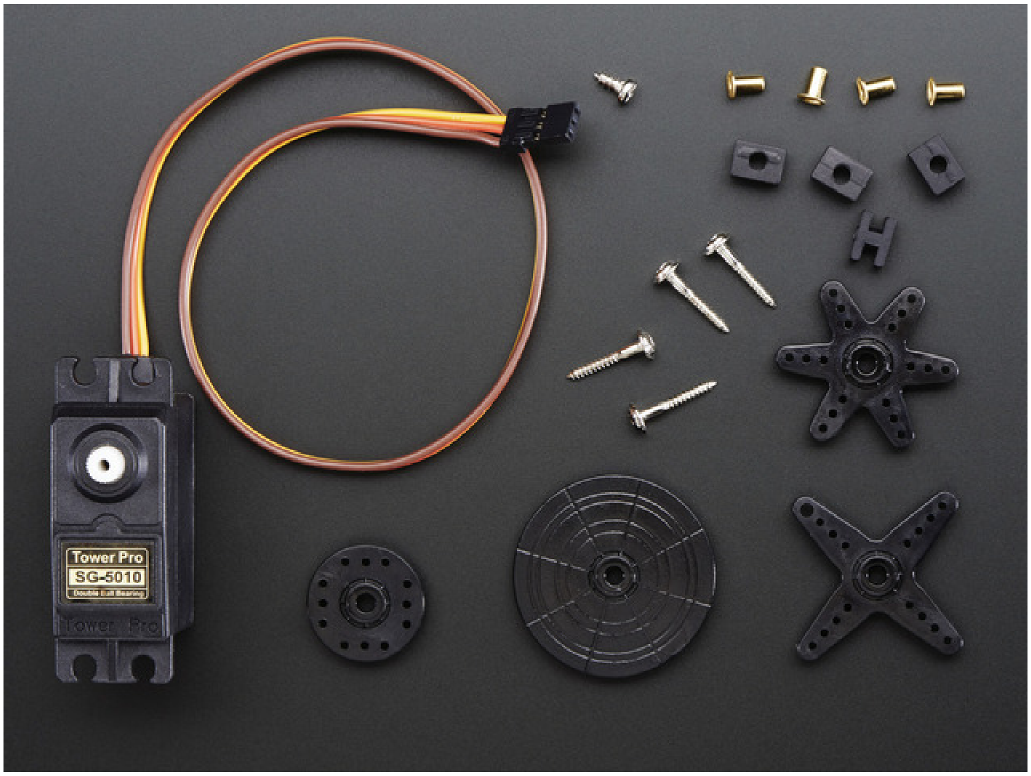
$24.95
IN STOCK

ADD TO CART

Recommended Servos

You're in luck, you can use **just about any kind of servo!**

Note that many of the photos below don't show the additional motor horns, but every servo comes with plastic clip-on parts!



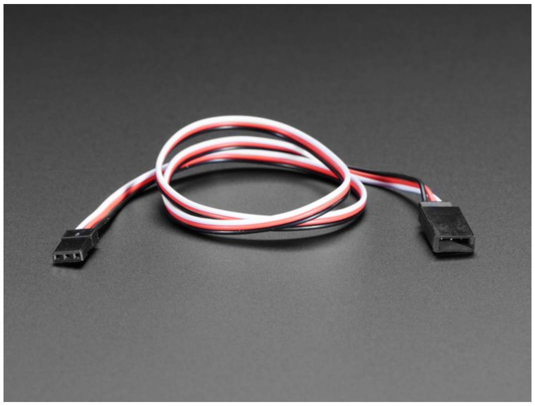
Servo Extensions

People often ask us what they can do if the wire to their Servo is to short for their project. Not a problem! These cables act as extension cords - now you've got plenty of room.

Servo Extension Cable - 30cm / 12" long -

$1.95
IN STOCK

ADD TO CART



[Servo Extension Cable - 50cm / 19.5" long]

$2.50

IN STOCK

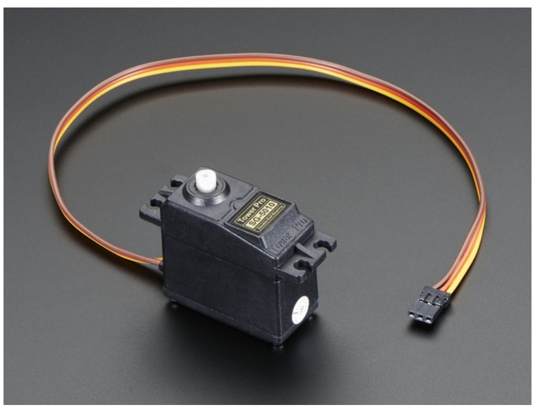
ADD TO CART

Popular plastic-gear servos

The most popular/common servos have plastic gears, they're plenty strong and not too expensive!

These can go back and forth, rotating about 180 degrees

They come in 'standard' size:



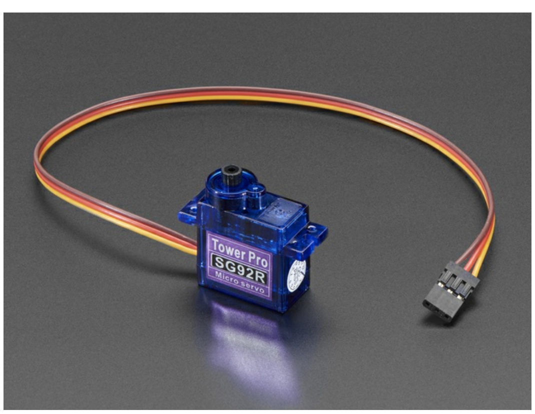
Standard servo - TowerPro SG-5010

$12.00

IN STOCK

ADD TO CART

And 'micro' size, not as strong but much more compact



Micro servo

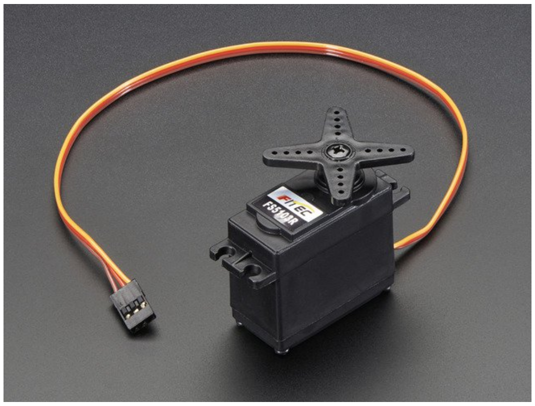
$5.95

IN STOCK

ADD TO CART

Continuous Rotation Servos

These servos look a lot like the above but they rotate *all the way around*. Unlike standard servos you can't control the location of the horn, just the speed and direction it which it turns. Good as an alternative to DC motors for wheeled bots. For that reason, they tend to get purchased with matching wheels!



Continuous Rotation Servo

$11.95

IN STOCK

ADD TO CART

Continuous Rotation Servo Wheel

$2.95

IN STOCK

ADD TO CART

Your browser does not support the video tag.

Continuous Rotation Micro Servo

$7.50

IN STOCK

ADD TO CART

[Wheel for Micro Continuous Rotation FS90R Servo]

$2.50
IN STOCK
ADD TO CART

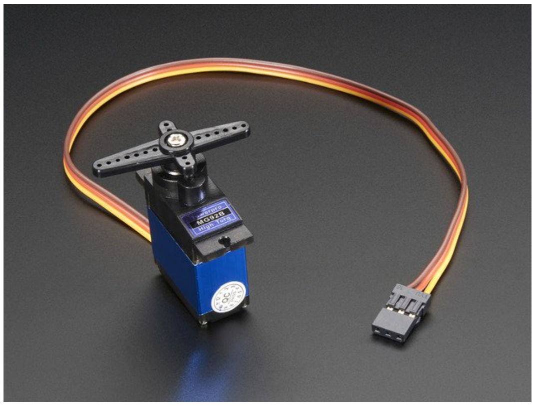
High Torque Servos

If you need more power, metal-gear servos can give you better torque, but at additional cost (since the gears have to be machined)

These are not continuous rotation

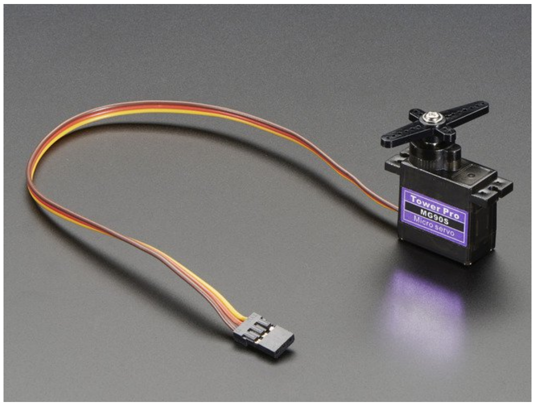
Standard Size - High Torque - Metal Gear Servo

$19.95
IN STOCK
ADD TO CART



Micro Servo - High Powered, High Torque Metal Gear

$11.95
IN STOCK
ADD TO CART



[Micro Servo - MG90S High Torque Metal Gear](https://www.adafruit.com/product/1143)

$9.95
OUT OF STOCK

OUT OF STOCK

Recommended Speakers

The Class-D amplifier on the Crickit is pretty powerful, so you can make quite a bit of noise!

4Ω Speakers

You'll get a lot louder audio from 4Ω speakers.

We recommend this speaker, you'll have to either poke wires into the connector, or cut it off and strip the wires to connect to the terminal block, but its nice and durable

Mono Enclosed Speaker - 3W 4 Ohm

$3.95
IN STOCK

ADD TO CART

This speaker is less expensive but you'll need to solder wires to the back

Speaker - 3" Diameter - 4 Ohm 3 Watt

$1.95
IN STOCK

ADD TO CART

8Ω Speakers

8 ohm speakers won't be as loud, but that's OK!

This speaker is inexpensive, but you'll need to solder wires to the back

[Speaker - 3" Diameter - 8 Ohm 1 Watt]

$1.95
IN STOCK

ADD TO CART

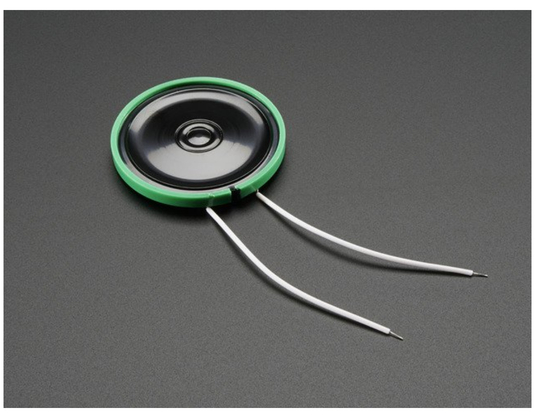
The speakers below work just fine, but because the audio amp is pretty strong so you have to make sure not to damage the speakers by turning up the potentiometer on the Crickit to make the audio really loud.

If you're getting buzzy sounds from them, turn that little trimmer potentiometer down.

Mini Metal Speaker w/ Wires - 8 ohm 0.5W

$1.95
IN STOCK

ADD TO CART



Thin Plastic Speaker w/Wires - 8 ohm 0.25W

$1.75
IN STOCK

ADD TO CART

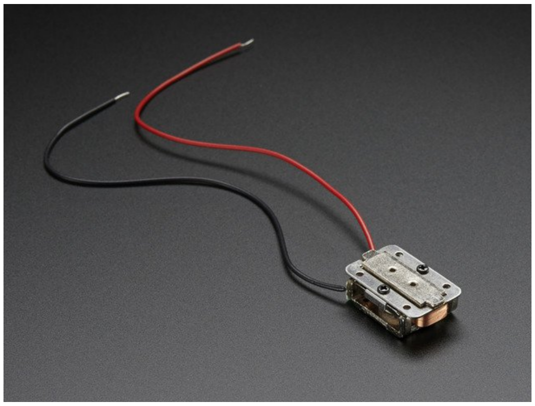
Wall or Bone Transducers

You can also use surface transducers if you like; attach/bolt/clamp the transducer to a surface:

Medium Surface Transducer with Wires - 4 Ohm 3 Watt

$14.95
OUT OF STOCK

OUT OF STOCK



Bone Conductor Transducer with Wires - 8 Ohm 1 Watt

$8.95
IN STOCK

ADD TO CART

Recommended Drives

Solenoids

Since the Crickit can only drive 5V power, you'll need to stick to this small 5V solenoid

Your browser does not support the video tag.

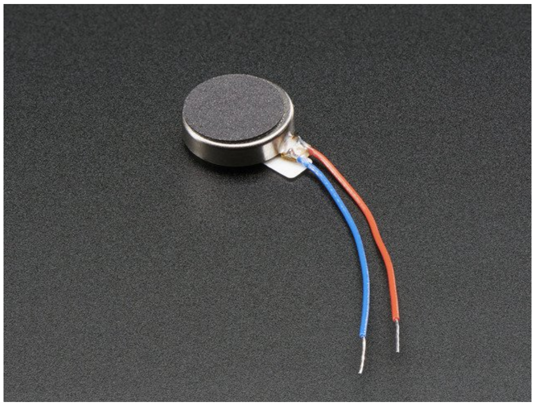
Mini Push-Pull Solenoid - 5V

$4.95
IN STOCK

ADD TO CART

Vibration Motors

You'll need to extend these wires but they'll work great at 5V and buzz very strongly



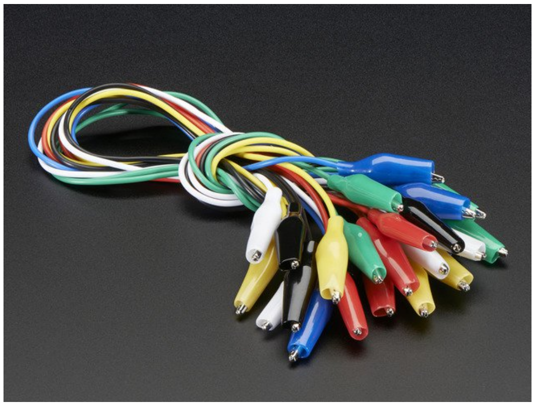
Vibrating Mini Motor Disc

$1.95
IN STOCK

ADD TO CART

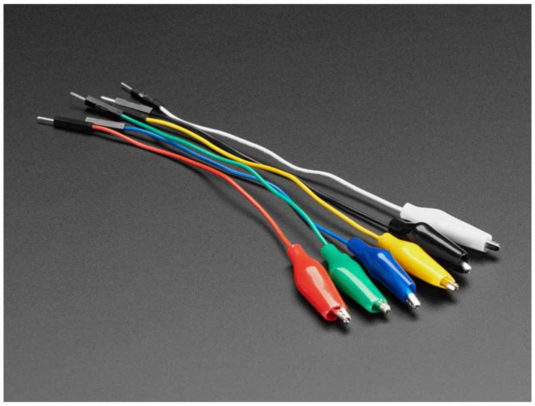
Recommended Capacitive Touch

The capacitive touch pads on the Crickit have large holes so its easy to connect alligator/croc clips. That's how we recommend you attach to them. The "small" size clips work best:



Small Alligator Clip Test Lead (set of 12)

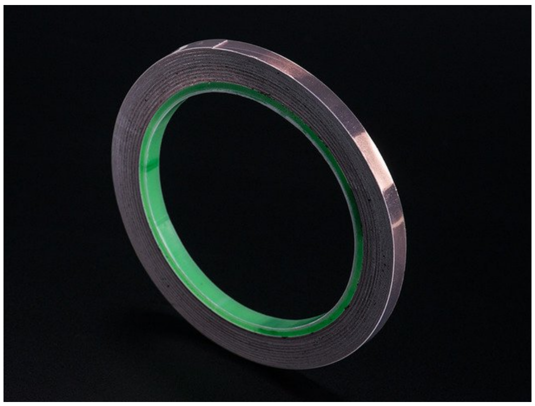
$3.95
IN STOCK
ADD TO CART



Small Alligator Clip to Male Jumper Wire Bundle - 6 Pieces

$3.95
IN STOCK
ADD TO CART

You can also use copper foil tape. Note that if you get foil with conductive adhesive, you can tape the foil right onto the Crickit pads. Otherwise you'll need to use alligator clips to grab onto the copper.



Copper Foil Tape with Conductive Adhesive - 6mm x 15 meter roll

$5.95
IN STOCK
ADD TO CART

Copper Foil Tape with Conductive Adhesive - 25mm x 15 meter roll

$19.95
IN STOCK
ADD TO CART

You can use other conductive materials like paints! Either drip the paint into the pad itself and let it harden, or use alligator clips to connect from one pad to a paper with conductive paint on it.

Bare Conductive Paint Pen - 10mL

$9.95
IN STOCK
ADD TO CART

Bare Conductive Paint - 50mL

$29.95
IN STOCK
ADD TO CART

MakeCode

With MakeCode, you can create robots simply and easily, using a drag-and-drop block interface. It's perfect for first time robot-makers, people who don't have a lot of coding experience, or even programmers who just want to get something going *fast*

MakeCode uses a web browser only, so no IDE is required to install. When you download a binary from MakeCode it is compiled for the Circuit Playground Express and you will overwrite any Arduino code or the CircuitPython runtime. You can always go back to Arduino (just use the Arduino IDE) or CircuitPython (by re-installing CircuitPython as shown here (https://adafru.it/Bfh))

Get Comfy With MakeCode

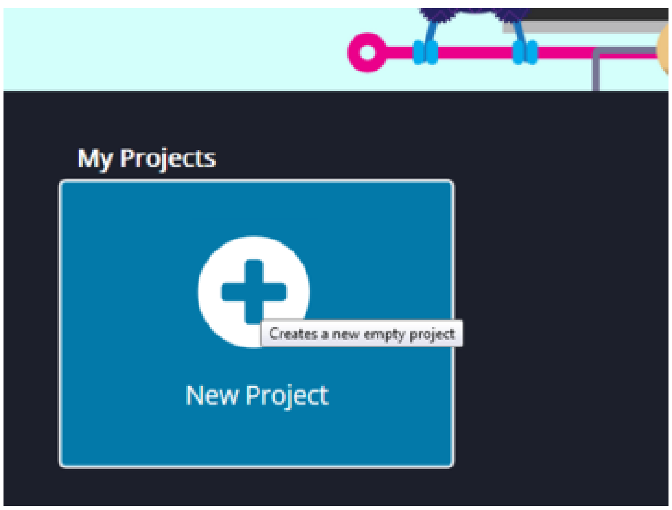
We recommend starting out by trying out the simple blinking NeoPixel example in our MakeCode guide, so you get a hang of how to install MakeCode apps on your Circuit Playground Express (https://adafru.it/wWd)

Once you feel comfortable with MakeCode, come back here and we'll add Crickit support!

Adding Crickit Extension

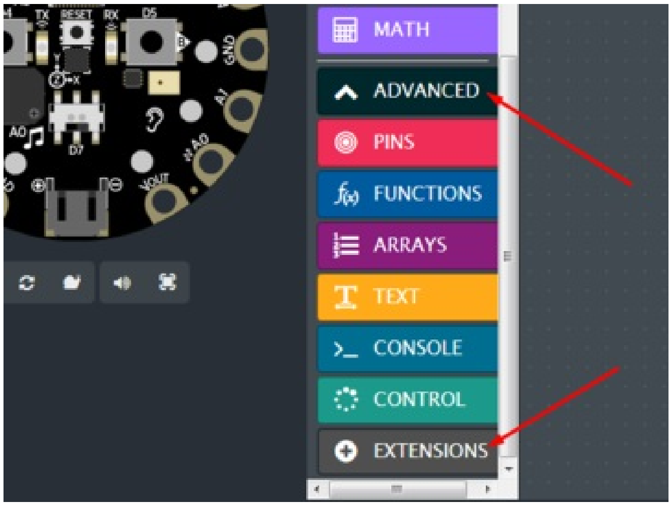
Now you're a MakeCode'r and you are ready to add Crickit support.

At this time, MakeCode support is being worked on and we're improving it every day, but it is Beta

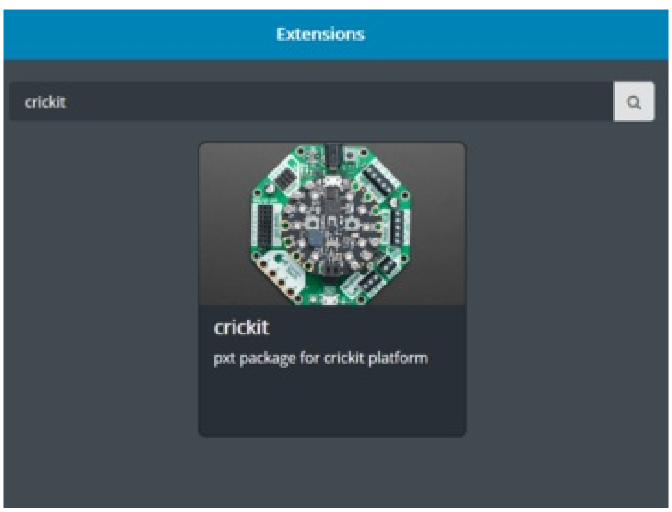


Start by visiting https://makecode.adafruit.com (https://adafru.it/Bly)

Click on **New Project**

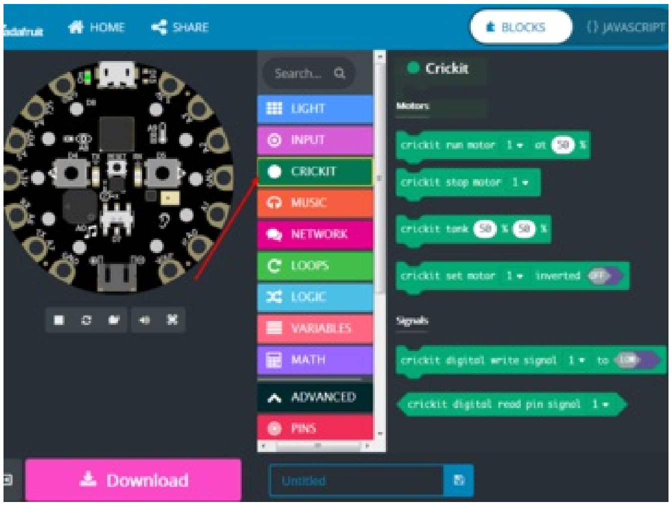


In the list of blocks, select **ADVANCED** and then **EXTENSIONS**



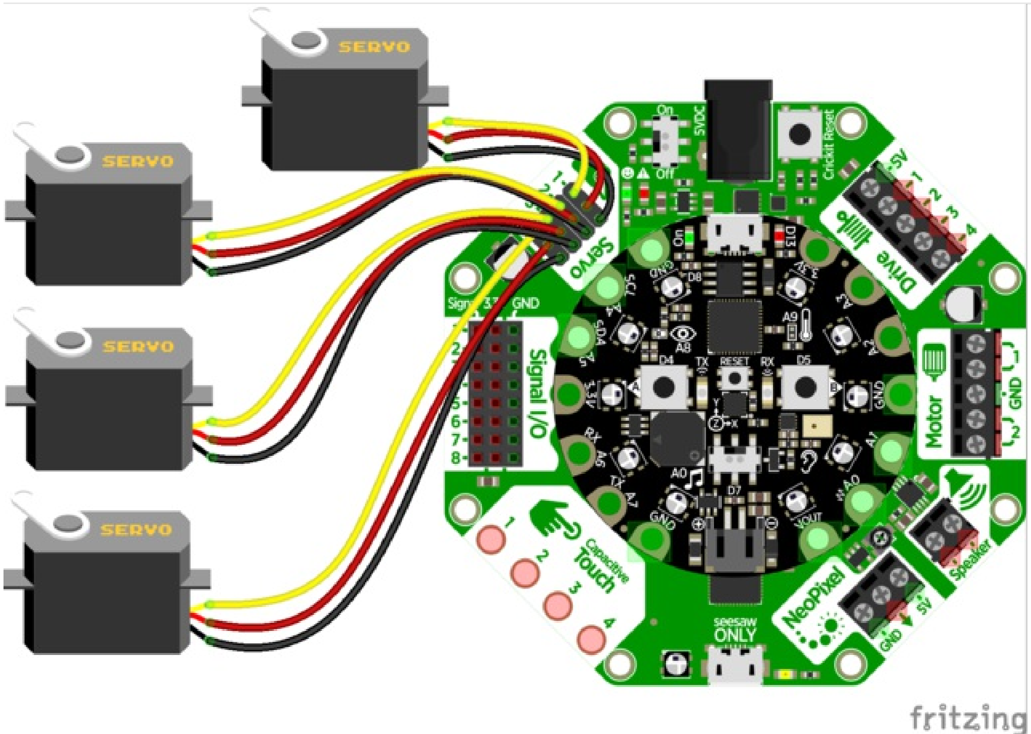
In the **Search Bar** type in **Crickit** and click the magnifying glass.

Click on the Crickit block that shows up to install Crickit support!



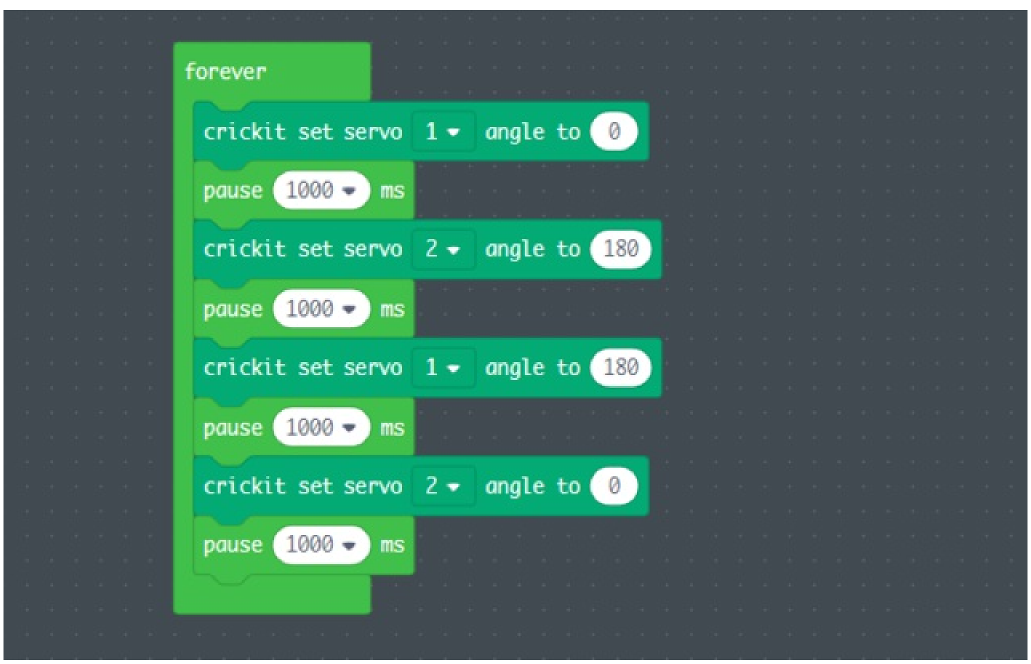
You will now have a new **CRICKIT** bin of blocks you can use! Continue on to learn how to use these blocks

MakeCode Servos



Servos are so easy to use, you can control **four** independent servos - micro, mini, standard, metal gear or continuous rotation. Basically, if it has a 3-pin plug on the end and has 'servo' in the name, it'll work just fine.

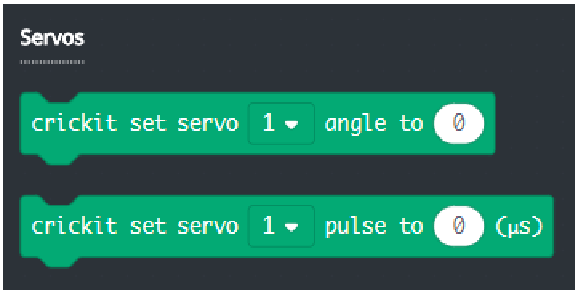
Let's start with a simple demo that moves two servos back and forth:



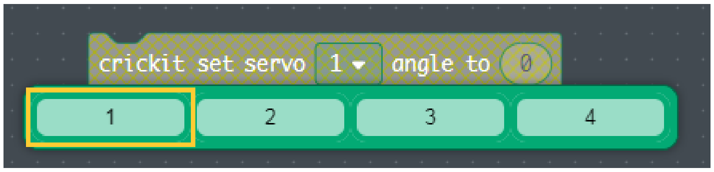
Are your servos not moving a full 180 degrees? Don't fret! This is normal, see below about putting in custom pulse lengths to get the 'max range' your servo!

Controlling servos is basically the same through a Crickit as through MakeCode directly.

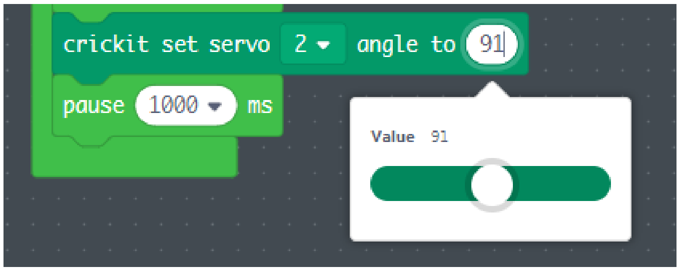
There's two blocks you can use, one for setting the angle and one for setting the pulse width directly



We recommend using the angle block, its easier! Select which servo you want to use, from 1 through 4



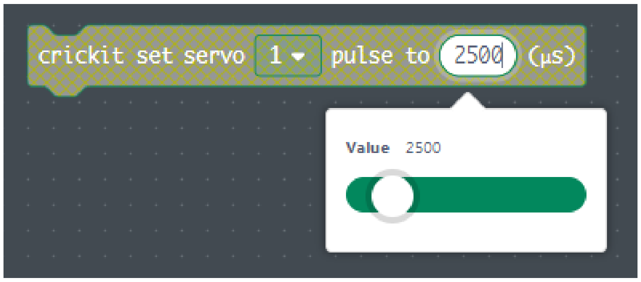
Then adjust the angle. Remember it does take a little time for the servo motor to move, so you can't just set it back and forth instantly, try adding a delay of a second after moving to make sure it got to the angle you want!



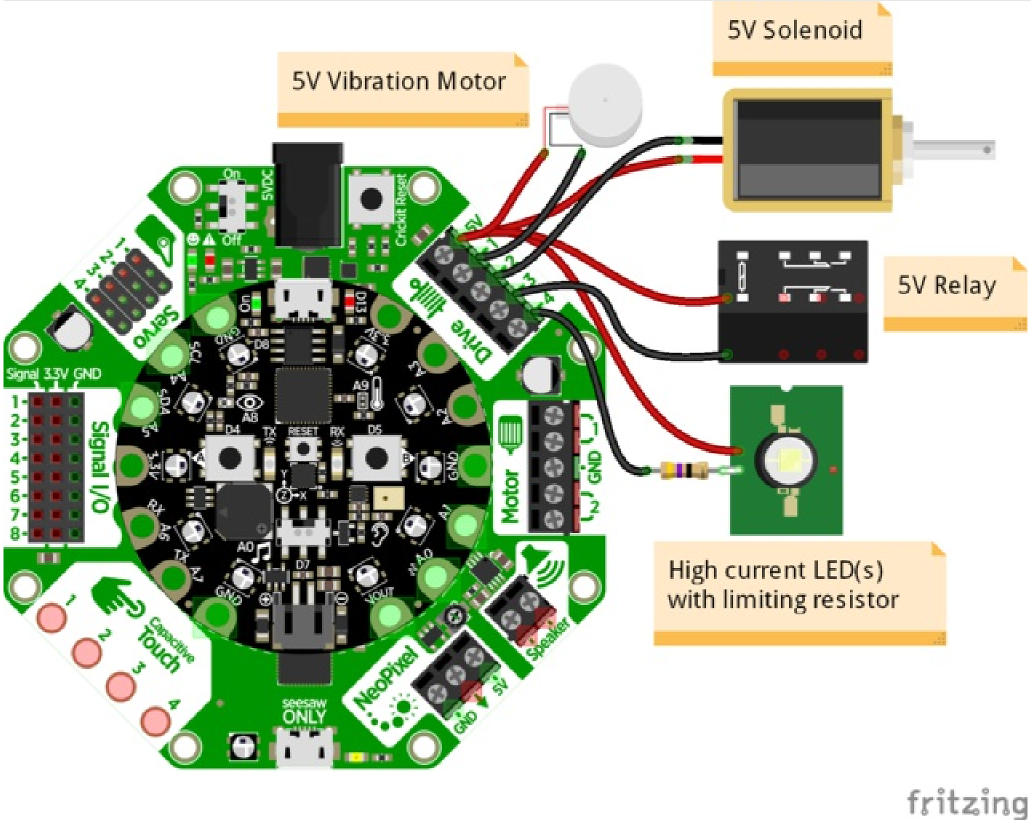
Although the angles range from 0 to 180, servos may have different ranges depending on the make and model. Also, each servo is a little different, so you may not get precisely the same angle even if its the same servo! Tweaking/adjusting the angle may be necessary.

Precise Pulses

For advanced use, you can hand-tune the pulse width. The 'standard' for servos is that 0 degrees is 1000 microseconds (us), 90 degrees is 1500 and 180 degrees is 2000 us. But...like we said, it can vary. You may want to try values as low as 750us and as high as 2500us! Just go slow, changing the values only 100us at a time, so you dont thwack the servo gears too far, they could be damaged if they push too far! For that reason, we recommend using angles only until you're comfy with servo usage

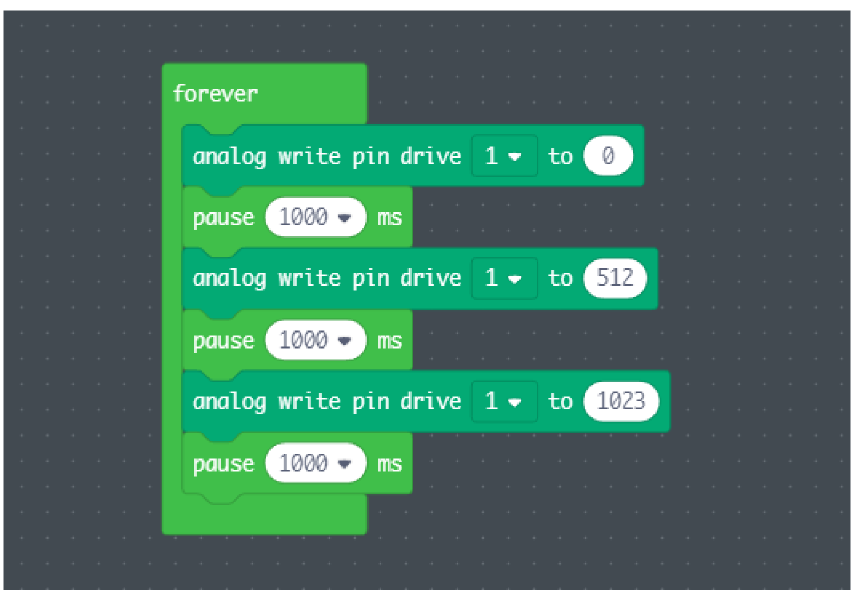


MakeCode Drives

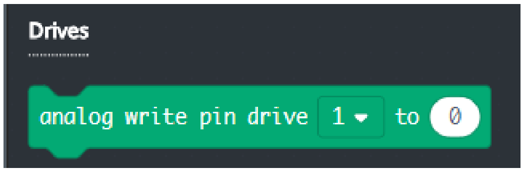


The **Drive** output of your Crickit is perfect for 5V-powered solenoids, relays, vibration motors or high powered LEDs. You can drive up to 500mA per output, and 4 outputs available.

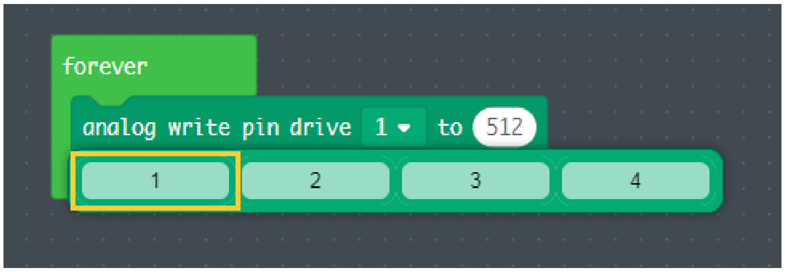
Note that the 'positive' side of the electronic part you're driving has to connect to 5V not Ground. You can just double/triple/quadruple wires into the same 5V terminal block.



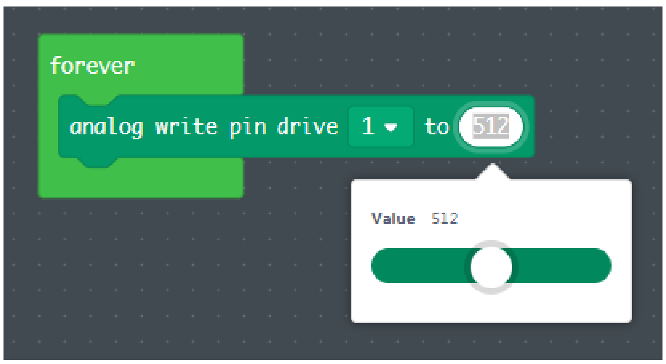
Each Drive output is a PWM output, that means you can change the amount of current or speed of whatever is connected.



Select which Drive pin you want to control with the pull down, Drive 1 through 4 are labeled on the Crickit



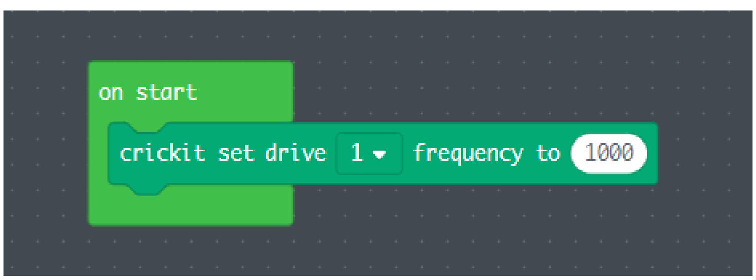
Then you can set the value from 0 (drive off) to 1023 (drive all the way on). If you want to dim an LED or run a vibration motor at half power, use 512. For quarter power, use 256!



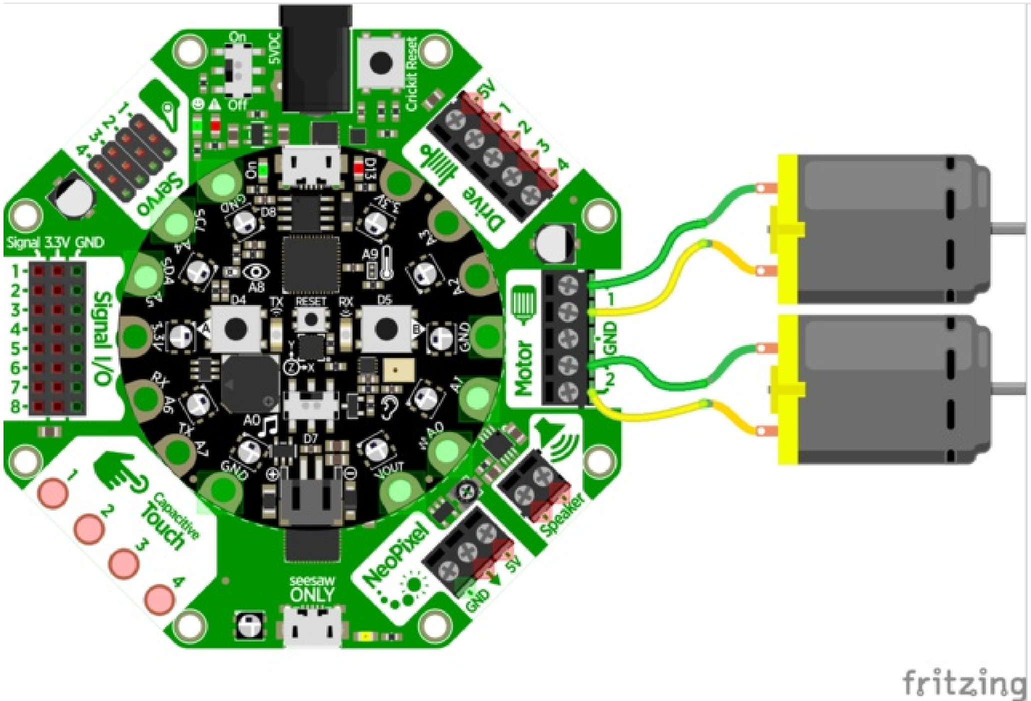
Remember you get 4 drive pins, so you can control them independently

Changing the Drive Analog/PWM Frequency

You can set the analog frequency in an **On Start** block. We recommend 1000 Hz (1 KHz) its a good standard number. Advanced makers can tweak this!



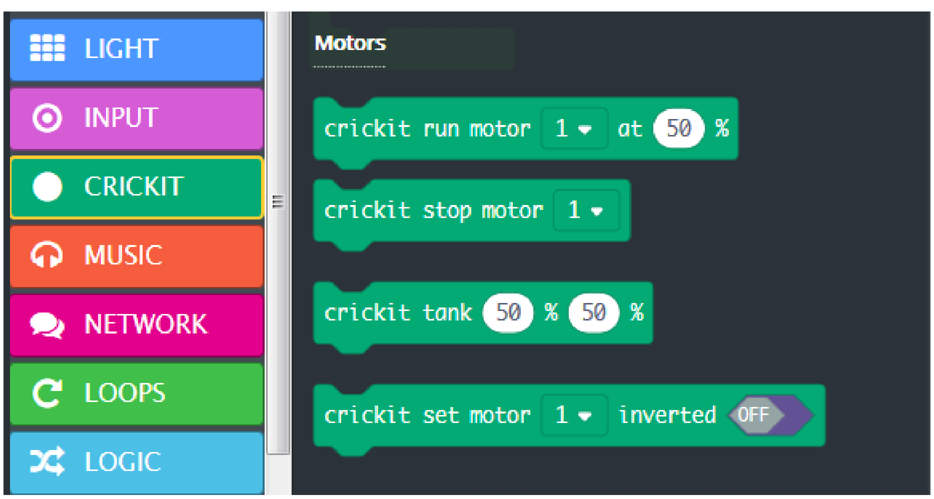
MakeCode DC Motors



You can drive two separate DC motors, so lets go ahead and get right to it!

DC motors are controlled by 4 PWM (adjustable speed) output pins, the 4 pins let you control speed *and* direction. And we'll use our **CRICKIT Motors** block set to help us manage the speed and direction for us, making it very easy to control motors

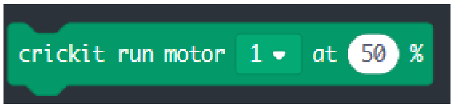
Note that each DC motor is a little different, so just because you have two at the same speed does not mean they'll rotate at the *exact* same speed! Some tweaking may be required



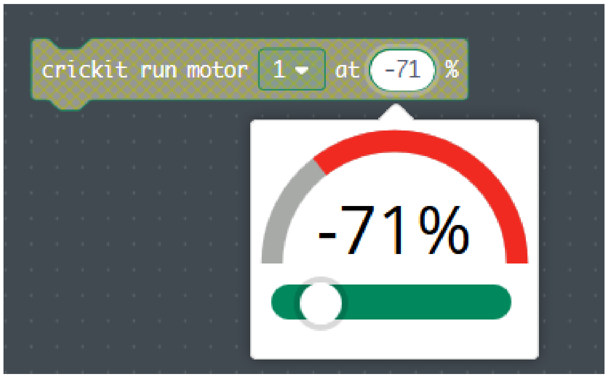
Here's an example program that will move a single motor in different speeds and directions

Setting Motor Speed

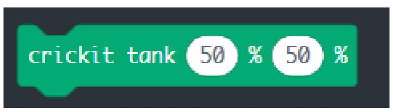
You can set the speed of the motor from 0% to 100% with this block. You can select which motor to use, 1 or 2. Once you set the speed of the motor it will continue at that speed until you change it or ask it to stop.



You can change direction by having a **negative** percentage speed!



You may want to have two motors move at the same time so they act like wheels on a car. In that case, you can use this handy block that will control two motors at once!

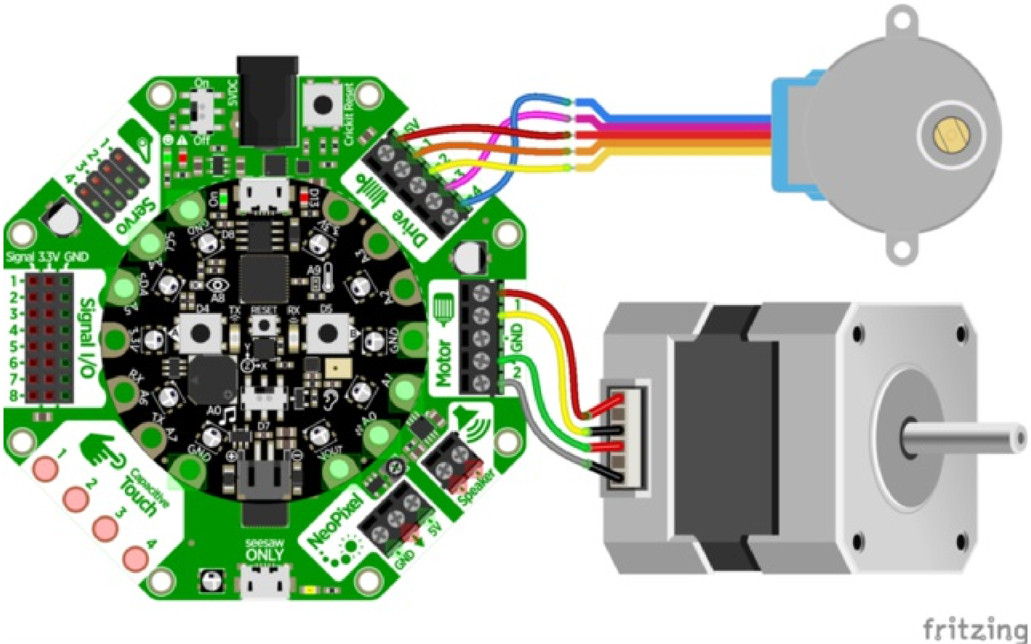


You can set the two speeds at once. If both move at the same positive speed, the tank/car will move forward. Same negative speed it will move backward. If one side moves faster than the other, the car will turn.

If you want to 'invert' the motor, it will flip which direction positive/negative numbers go. That is, if positive was forward, now positive will mean backwards

This is sometimes handy if you want to use only positive numbers or to keep your code looking tidy.

MakeCode Steppers

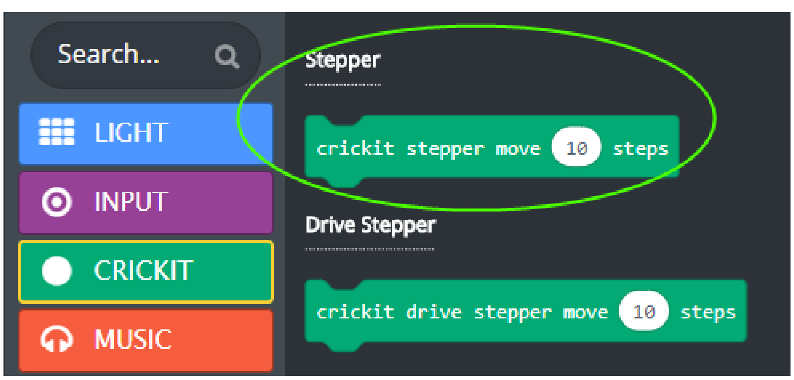


You can control one or two stepper motors on Crickit. The **Motor** block can drive one bipolar stepper (wiring shown above) or one unipolar stepper. In addition, the **Drive** block can control one stepper also but it **must** be unipolar (bipolar will not work on the **Drive** port).

> The MakeCode blocks to control a unipolar/bipolar stepper on the Motor port is DIFFERENT from the block used to control a unipolar stepper on the Drive port. Be sure you use the correct block depending on the block you are wiring the stepper motor to.

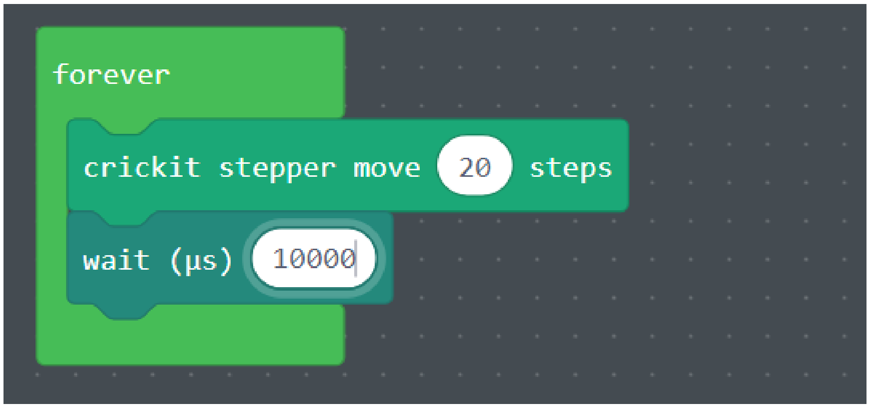
MakeCode for Using a Stepper on the Motor Port

In the **CRICKIT** block group, scroll down until you see the **Stepper** heading and the block `crickit stepper move` block. Be sure **not** to use the `crickit drive stepper move block`, that is for using a unipolar stepper on the **Drive** port, discussed further down the page.



Move the Motor Port Stepper One Direction Forever

Here is a simple program that tells the stepper to move 5 steps, then wait 10 milliseconds (= 10,000 microseconds), and repeats forever:

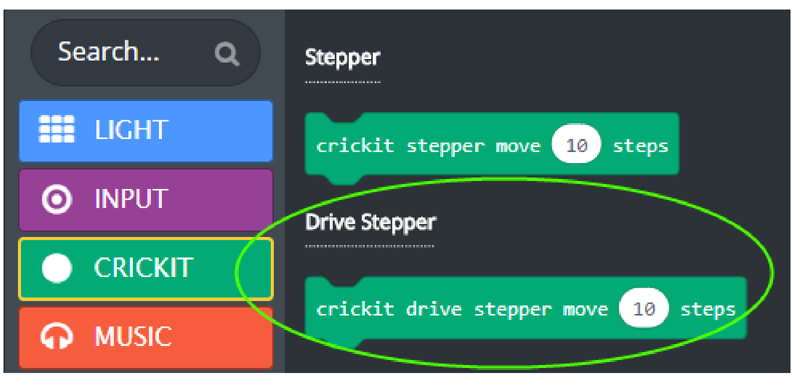


You'll see the motor shaft slowly turning in the "positive" direction. If you use a bit of solid tape on the stepper's shaft as a small flag, you can see the rotation better. If the rotation is in the wrong direction, use a negative value for the number of steps, re. -20 .

At this point, you can vary the parameters: increase or decrease the number of steps moved every loop. If you want the stepper to move faster, increase the steps. This may make the action a bit "jerky". If so, you can decrease the steps. This will be smooth, but slow. To increase the pause between steps, you can use the wait block to get times greater than 10 milliseconds.

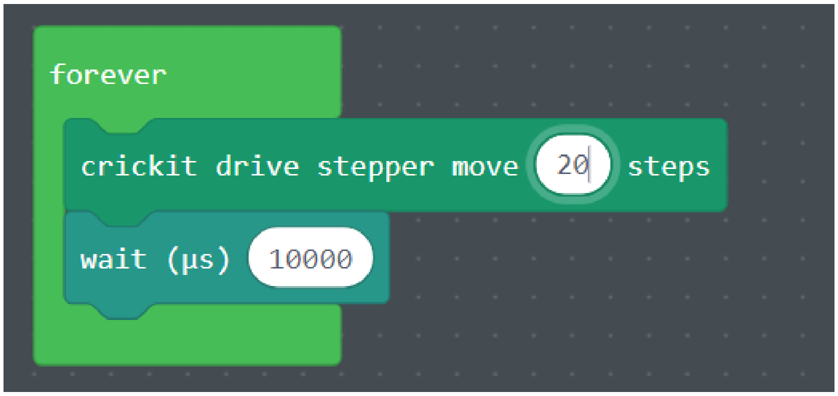
Using a Stepper on the Drive Port in MakeCode

In the **CRICKIT** block group, scroll down until you see the **Stepper** heading and the block crickit drive stepper move block.



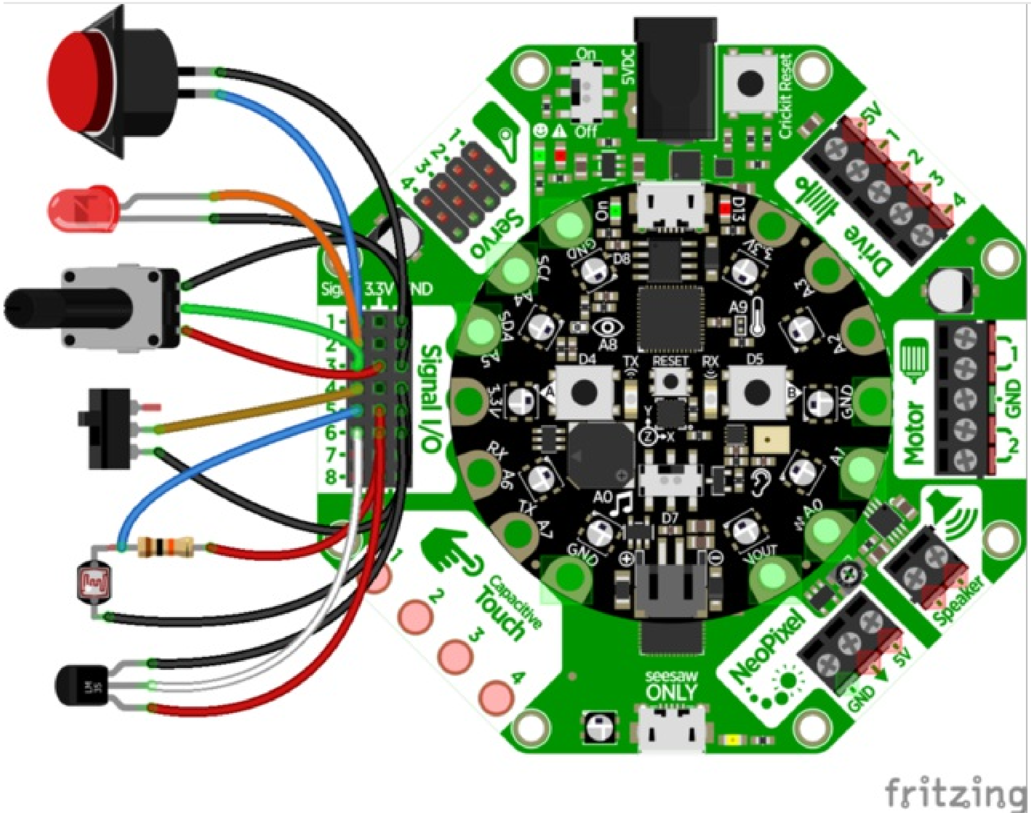
Move the Drive Port Stepper One Direction Forever

Here is a simple program that tells the stepper to move 20 steps, then waits ten milliseconds, and repeats forever. 10 milliseconds minimum delay between step blocks is the minimum to ensure the stepper doesn't miss any steps between blocks.



If you want to move the motor in the opposite direction, make the value negative, re. `-20`. The block takes positive and negative values.

MakeCode Signals



You may want to add buttons, LEDs, switches or simple sensors to your robot project. With Crickit, you get 8 x 'general purpose in/out' (GPIO) pins called **signals**. Each signal can be a digital input (button/switch), digital output (LED, for example), or *analog input.*

This lets you add a ton of external components easily, and its all handled by seesaw. Perfect when you have a Feather without analog inputs (like the ESP8266) or just need a ton of extra pins.

The signal pins are on a 3x8 female header, so you can poke wires directly in!

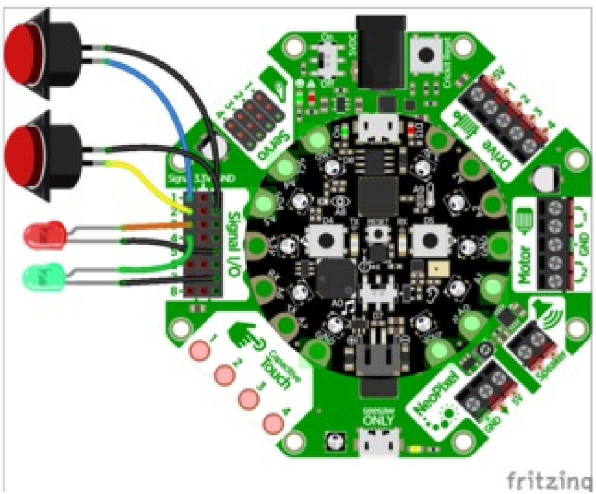
Using Signals in MakeCode

MakeCode has three blocks under the CRICKIT group to help you work with signals:

- `crickit digital read signal` allows you to read digital values in
- `crickit analog read signal` reads a signal and provides an analog value from 0-1023
- `crickit digital write signal` allows you to write out to a signal line

Analog read returns a number so the block is rounded to place where a number may be used. Digital read is angled so it fits where a decision like `if..then..else` blocks use. Write signal is a block of its own and will set a signal (Make it `HIGH` / 3.3 volts or `LOW` / 0 volts).

Digital Reads and Writes

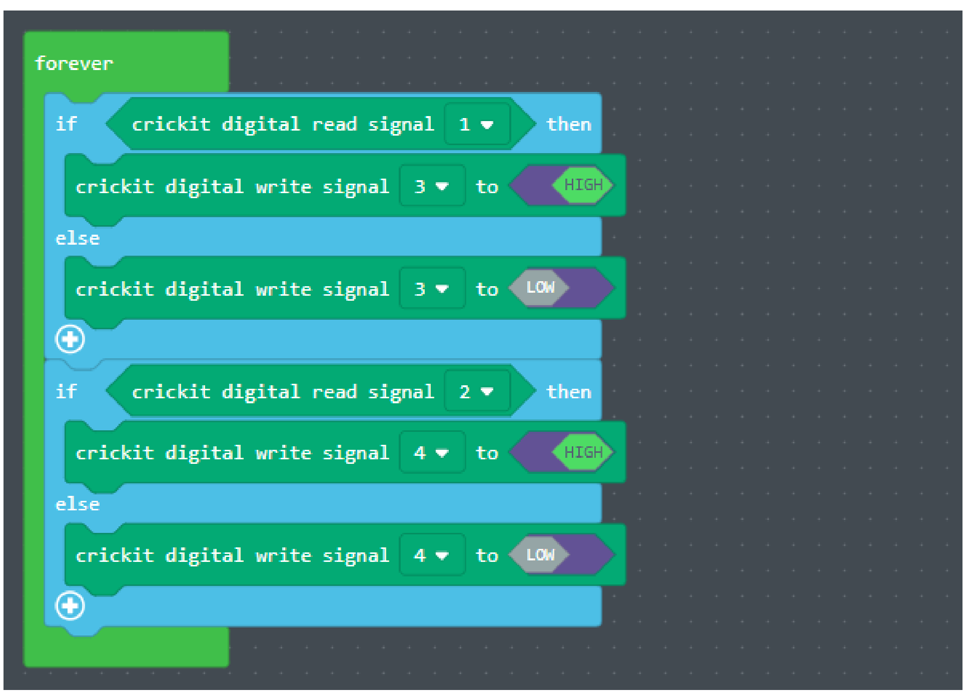


Here's an example wiring that goes with the code below.

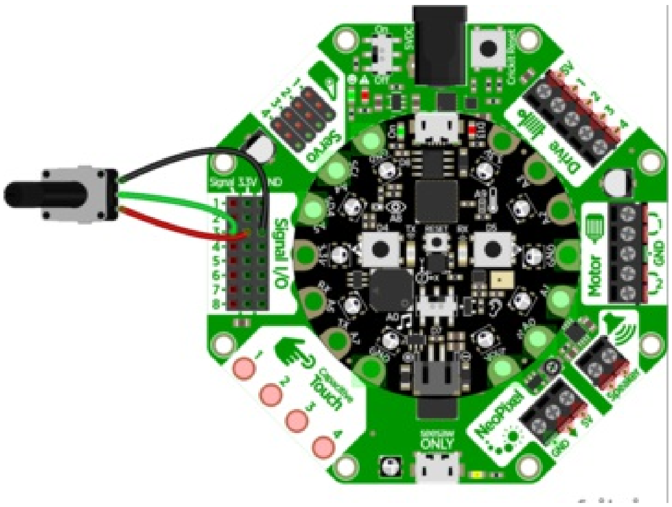
We have two switch buttons, connected to **signals #1** and **#2**, the other side of the buttons connect to ground

There's also two LEDs, connected to the **signals #3** and **#4** and the negative wires connected to ground. (All the 3.3V and Ground pins are connected together so you don't *have* to use the ones right next to the signal pin!)

Here is the MakeCode that reads the buttons on **signal #1** and **#2** and lights **signal #3** and **signal #4** if the corresponding button is pressed:



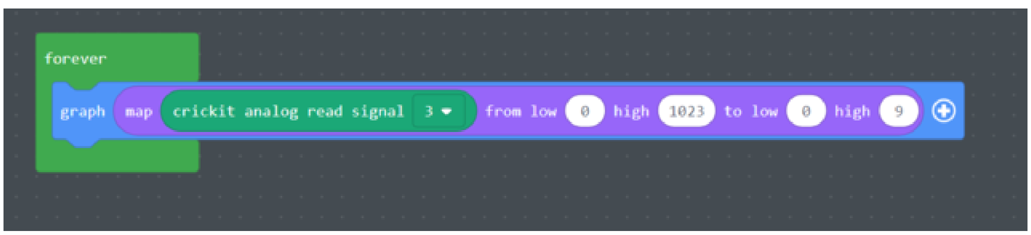
Analog Reads



You can also read analog values like from a potentiometer or sensor.

Lets do a demonstration where the center tap of a potentiometer is hooked up to **Signal #3** - don't forget to also connect one side of the potentiometer to 3.3V and the other side to ground.

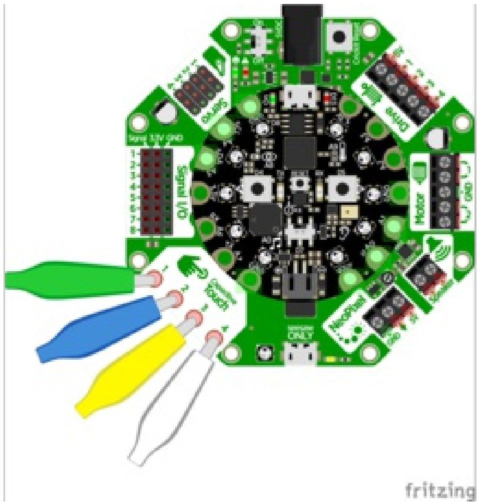
And here is the example code. You can see we read the signal with `crickit analog read signal` which returns a value from 0 to 1023. The `map` **MATH** function changes 0 to 1023 to 0 to 9. The `graph` **NEOPIXEL** block will light the number of NeoPixels `map` returns in rainbow colors.



https://adafru.it/C35

https://adafru.it/C35

MakeCode Touch

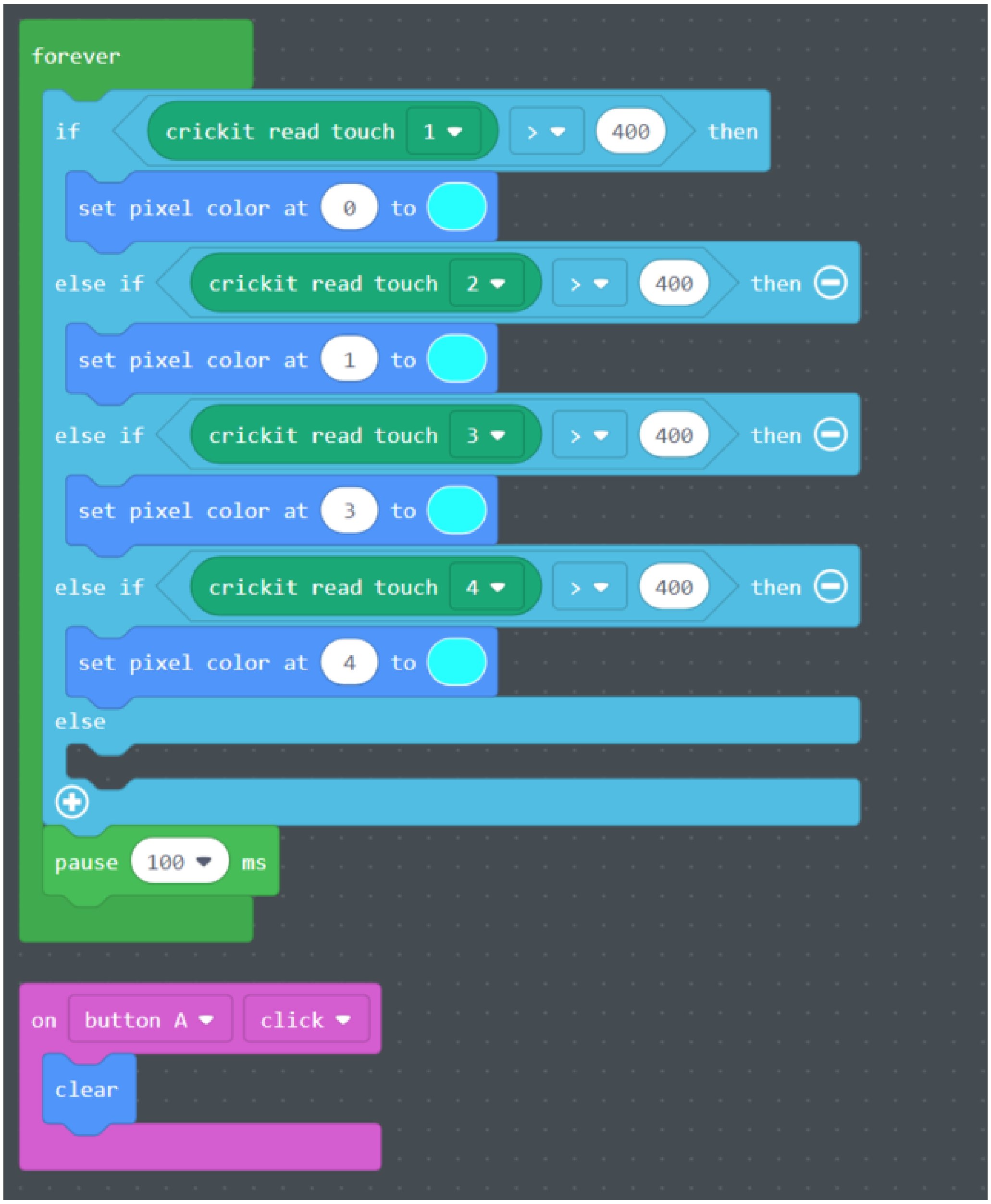


There are four capacitive touch pads you can use to detect human touch. They have big pads you can use to attach alligator clips to extend the pads' reach.

You can connect the other end of the alligator wires to fruit and make your own fruit-touch robot. Or move servo motors based on touch, it's all fun.

You can read the value of the captouch pads from the MakeCode **CRICKIT** block group, block `crickit read touch`. This will return a value that is the change in value, touched vs. not.

The program below sets up Crickit capacitive touch on pads 1, 2, 3 and 4. It then loops forever - if you touch a pad, it lights a NeoPixel. Pressing Button A clears the NeoPixels.

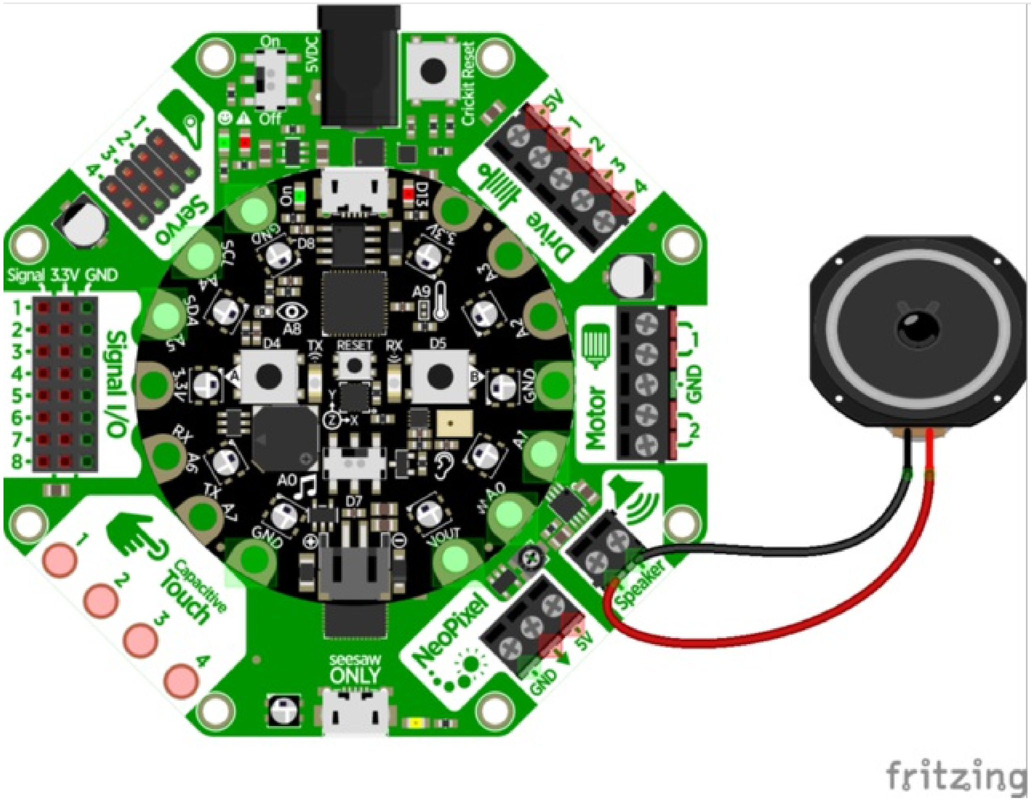


https://adafru.it/C36

https://adafru.it/C36

You can set different actions: if a touch is detected, change the direction of a motor as just one example.

MakeCode Audio



Audio animatronics! By adding a voice or sound effects to your robot you can make a cool interactive project. We take advantage of CircuitPython's ability to play WAV files over the true-analog output pin **A0**.

This is one of the *few* outputs that does not go through the Crickit's seesaw helper chip. Instead, the audio is played directly from the CircuitPython board and the Crickit amplifies it!

Amplifier Details

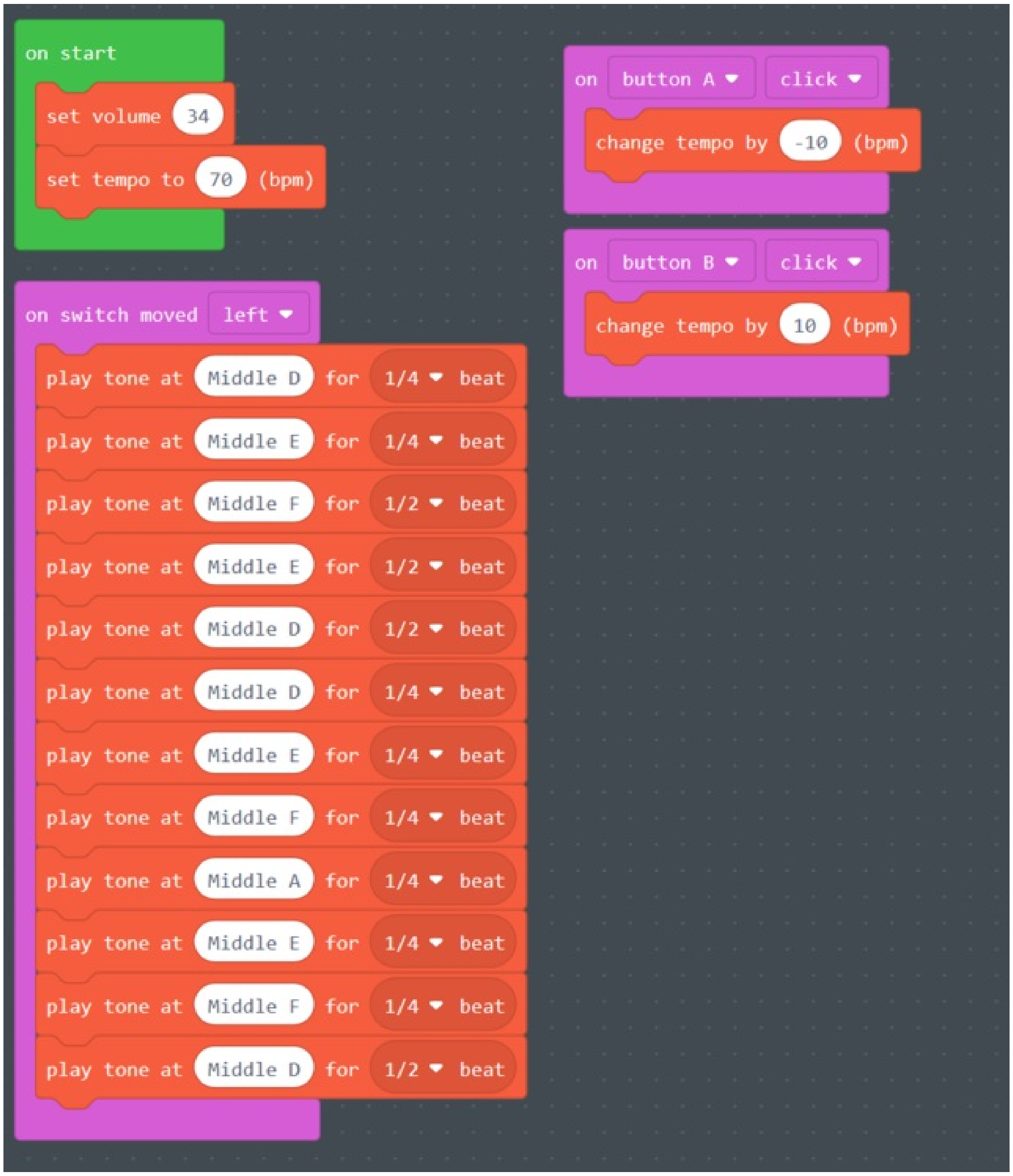
The onboard amplifier is a mono "Class D" audio amp with BTL (Bridge Tied Load) output.

That means **you cannot plug the speaker output into another amplifier, it must connect directly to a speaker!**

You can use just about *any* 4 to 8Ω speaker (6 Ω is OK too, just not as common). The amplifier can drive up to 3 Watts into 4Ω and 1 Watt into 8Ω. That means its ok to drive a 5 Watt speaker, it just wont be as loud as it *could* be with a bigger amp (but you wont damage the amp). You can also drive speakers that are smaller, like an 8Ω 0.5 W but make sure you don't turn the audio volume potentiometer up, as it could damage the speaker by overpowering it.

Playing Sounds on Crickit with MakeCode

If you are using a Crickit with Circuit Playground Express (CPX), the Crickit becomes an amplified extension of the regular audio out. The sound is very clear and the volume can be higher than the CPX on-board speaker. Below I've taken a cute song snippet and reduced the volume from a previous value of `100` to `34` so one's ears don't hurt when it starts. If the slide switch is moved left (towards the on-board speaker on CPX), the song will play, moving the switch right silences it. If you think the speed of the sound (the *tempo*) is too fast, press button A to slow it down. If you think the tempo is too slow, press the B button.



You can download the code by clicking this link to link to the MakeCode website.

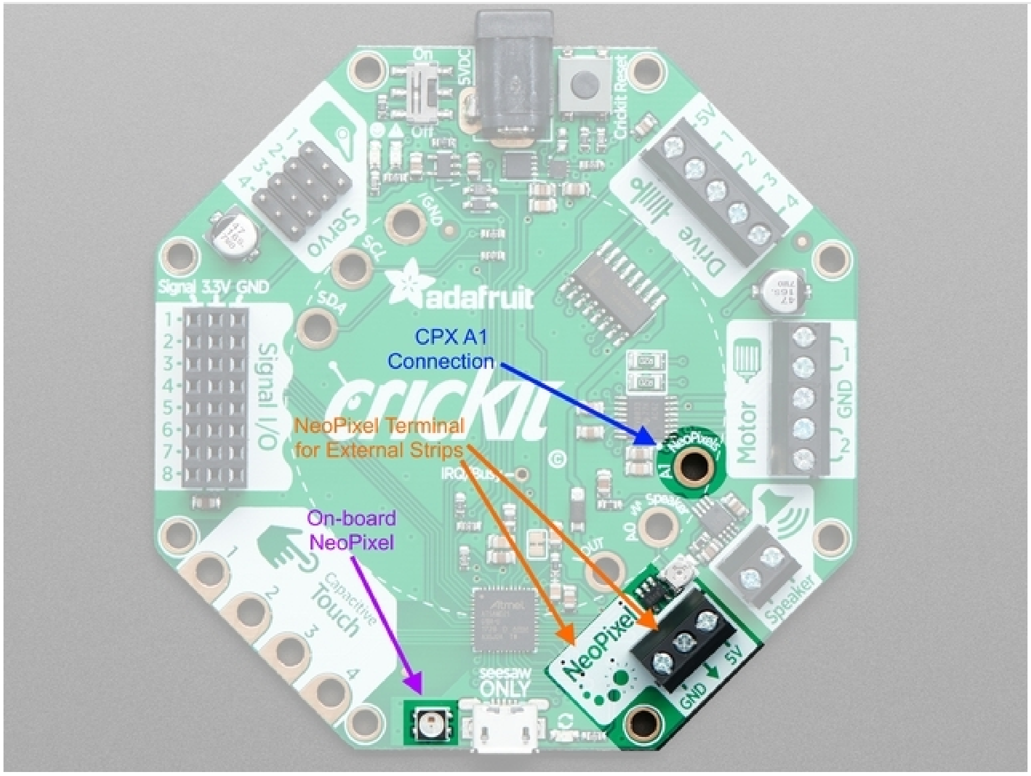
https://adafru.it/BPC

https://adafru.it/BPC

Check out all the music blocks, you can have Crickit using sounds in projects with just a couple of clicks!


Search...
LIGHT
INPUT
CRICKIT
MUSIC
NETWORK
LOOPS
LOGIC
VARIABLES
MATH
ADVANCED
PINS
FUNCTIONS
ARRAYS
TEXT
CONSOLE
Music
play sound power up
play sound power up until done
stop all sounds
play tone at Middle C for 1/2 beat
ring tone at Middle C
rest for 1/2 beat
set volume 128
change tempo by 20 (bpm)
set tempo to 120 (bpm)
tempo (bpm)
0 Hz
1/2 beat

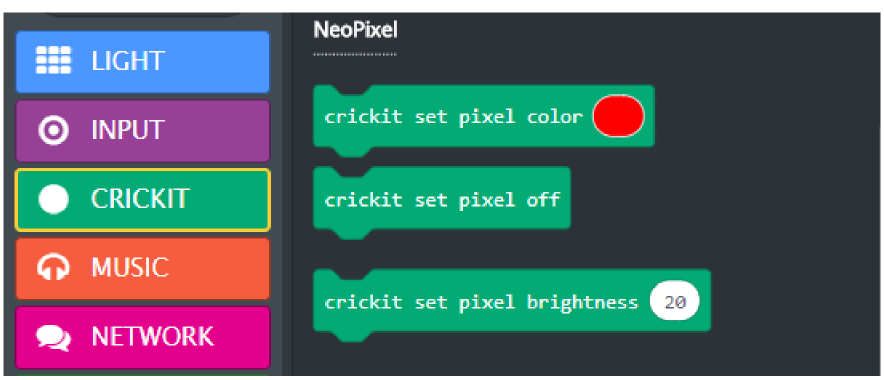
MakeCode NeoPixels



Crickit easily allows you to work with NeoPixels. There is one on-board to the right of the Capacitive Touch pads. There is also a terminal block called NeoPixel next to the Crickit Speaker terminal output. The NeoPixel terminal connections makes it super easy to use a strip or ring of NeoPixels to light up anything. You can see the Crickit connection to Circuit Playground Express (CPX) A1, that is where CPX will send the codes to the Crickit NeoPixel driver to make external NeoPixels light up.

MakeCode for Crickit NeoPixels

Using the Crickit Onboard Single NeoPixel



You can use the three special NeoPixel blocks in the **CRICKIT** block group extension to change the single NeoPixel on-board Crickit. They work just like the NeoPixel blocks under the **LIGHT** block group but just for the one Crickit pixel.

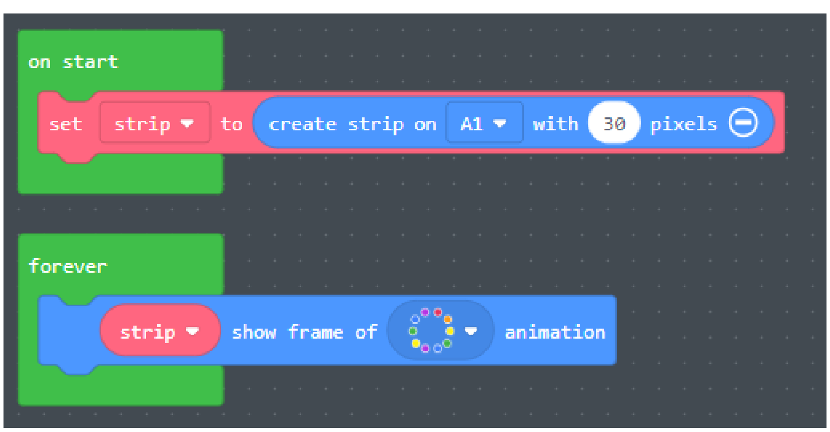
External NeoPixels on the NeoPixel Terminal Block

In the **LIGHT** block group, there is a special subgroup that pops below **LIGHT** when **LIGHT** is pushed called **... NEOPIXEL**. This provides a huge number of blocks to work with NeoPixels that are not on your Crickit or the microcontroller on Crickit like a Circuit Playground Express.

For Circuit Playground Express, the Crickit NeoPixel Terminal is connected to CPX Pin `A1`. When you use the MakeCode NeoPixel blocks to manipulate your externally connected NeoPixels, you need to use the **NEOPIXEL** subgroup block labeled `set strip to create strip`.

For the code below, I have connected a [30 NeoPixel strip (https://adafru.it/BPD)](https://adafru.it/BPD) to the Crickit NeoPixel terminal block. When the program starts, the `on start` code up the variable named `strip` to refer to a NeoPixel strip connected to `A1` (which all Crickit strips are connected to) with `30` NeoPixels on it (You have to click the `+` on the block to specify the pin `A1` and add the number of NeoPixels.

Then the program shows the rainbow animation on the strip forever. You can change the animation type or do lots of other things on your strip. It's that easy!



For more information on NeoPixels and Crickit, see the guide Make It Glow with Crickit (https://adafru.it/BZ0).

CircuitPython Code

To use Crickit, we recommend CircuitPython. Python is an easy programming language to use, programming is fast, and its easy to read.

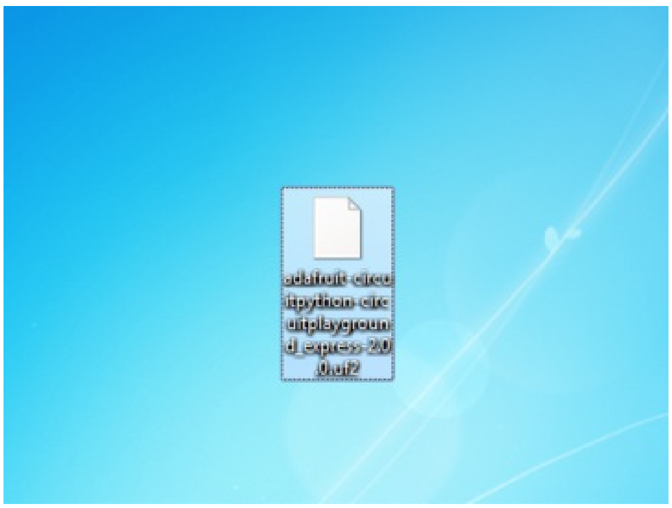
Install CPX Special Build

If you're using Circuit Playground Express (CPX), Please install this special 'seesaw' version of the CPX firmware. Plug the USB cable into the CPX, double click the reset button until you see **CPLAYBOOT** drive, then drag the UF2 file onto the disk drive:

https://adafru.it/BMc

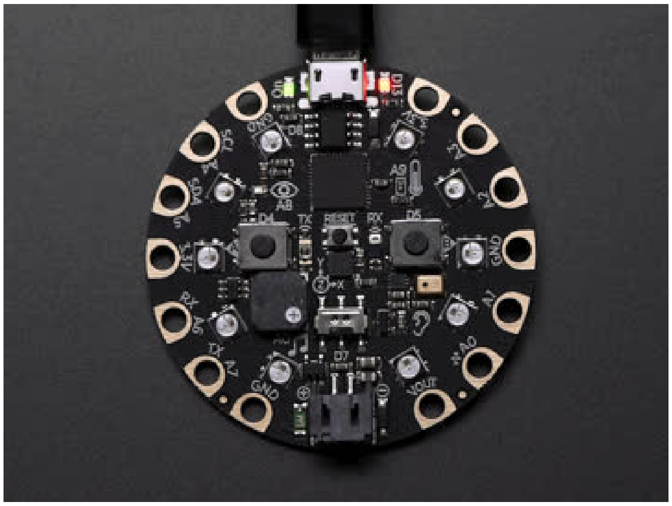
https://adafru.it/BMc

What's nice about this special version is that the **adafruit_crickit**, **adafruit_seesaw** and **adafruit_motor** library is built in, which saves you tons of space and makes it really fast to get started



Click the link above to download the latest UF2 file

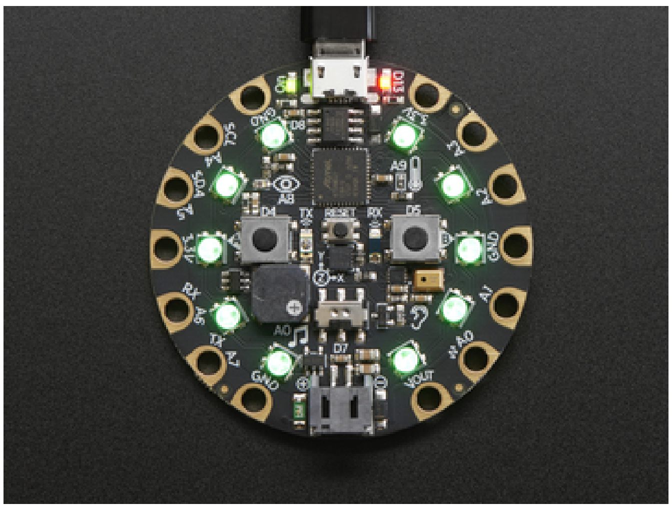
Download and save it to your Desktop (or wherever is handy)



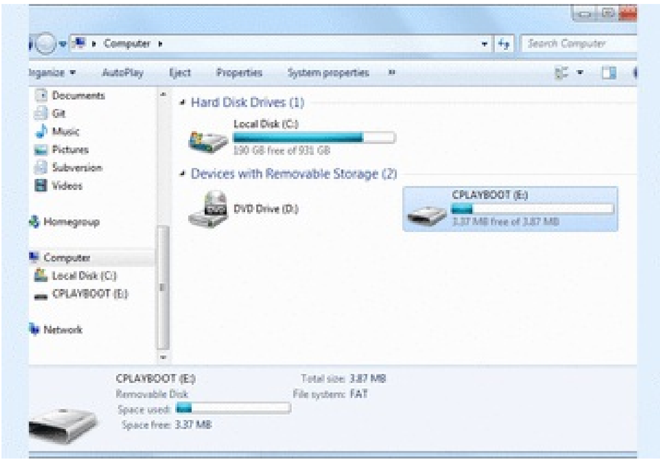
Plug your Circuit Playground Express into your computer using a known-good USB cable

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync

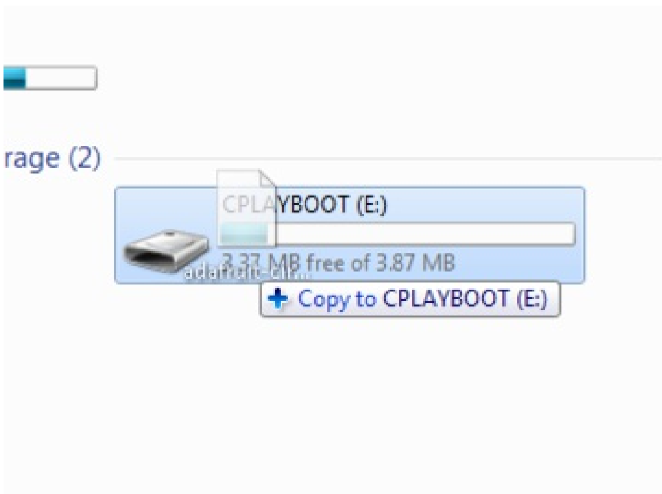
Double-click the small **Reset** button in the middle of the CPX, you will see all of the LEDs turn green. If they turn all red, check the USB cable, try another USB port, etc.



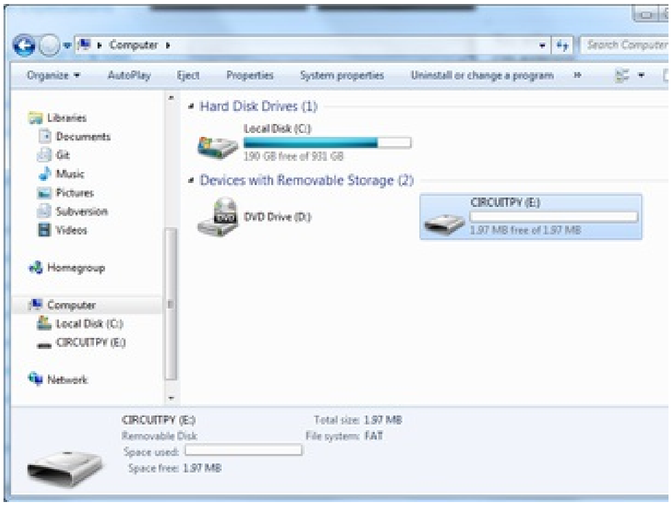
(If double-clicking doesn't do it, try a single-click!)



You will see a new disk drive appear called CPLAYBOOT



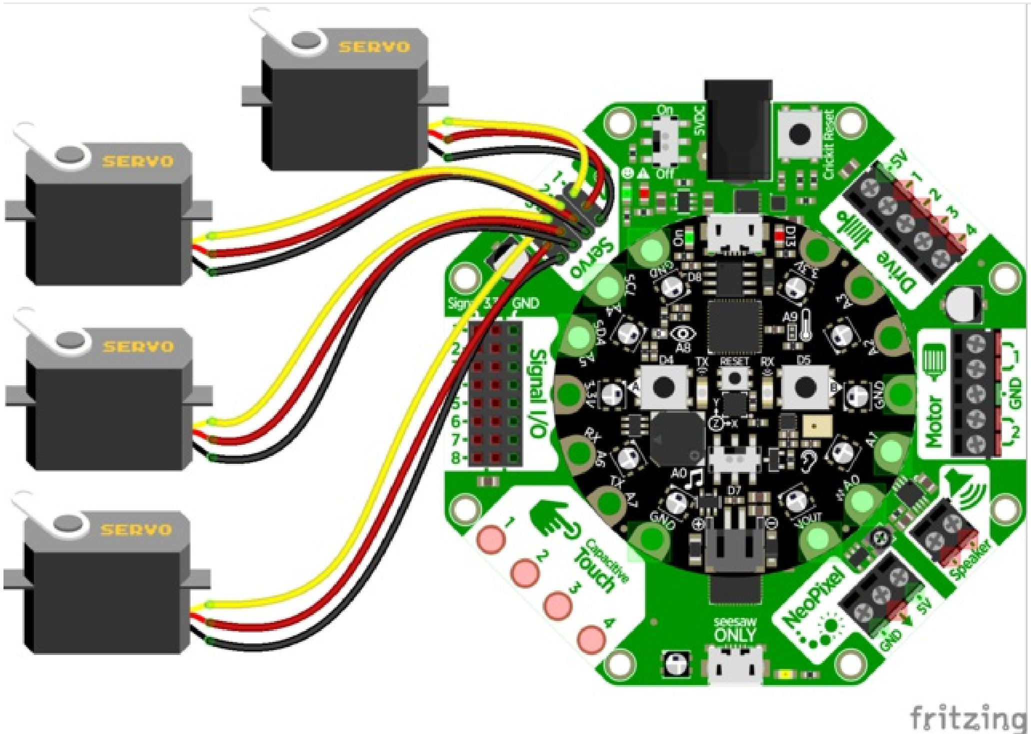
Drag the **.uf2** file onto it.



The **CPLAYBOOT** drive will disappear and a new disk drive will appear called **CIRCUITPY**

That's it! You're done :)

CircuitPython Servos



https://adafru.it/Be1

https://adafru.it/Be1

Test Servos

Lets start by controlling some servos. You'll want at least one servo to plug in and test out the servo code. Visit our recommended servo page to check that you have a servo that works (https://adafru.it/Bfo). Once you do, plug in a servo into **SERVO #1** spot, making sure the yellow or white wire is next to the **1** text label.

This example will show rotating one servo from 0 to 180 degrees with a stop at 90 degrees.

```
import time
from adafruit_crickit import crickit

print("1 Servo demo!")

while True:
    print("Moving servo #1")
    crickit.servo_1.angle = 0      # right
    time.sleep(1)
    crickit.servo_1.angle = 90     # middle
    time.sleep(1)
    crickit.servo_1.angle = 180    # left
    time.sleep(1)
    crickit.servo_1.angle = 90     # middle
    time.sleep(1)
    # and repeat!
```

Are your servos not moving a full 180 degrees? Don't fret! This is normal, see below about min/max pulse lengths to 'tune' your servo!

We start by importing the libraries that we need to have time delays (import time) and then the main crickit python library that will make it super easy to talk to the motors and sensors on crickit (from adafruit_crickit import crickit)

The crickit object represents the motors and servos available for control. The servos are available on the sub-objects named servo_1 , servo_2 , servo_3 , servo_4

Each of these are adafruit_motor.servo (https://adafru.it/BMX) type objects for the curious

Control Servo

Now that we know the servo objects, we can simply assign the angle! crickit.servo_1.angle = 0 is all the way to the right, crickit.servo_1.angle = 90 is in the middle, and crickit.servo_1.angle = 180 is all the way to the right.

More Servos!

OK that was fun but you want MORE servos right? You can control up to four!

```
import time
from adafruit_crickit import crickit

print("4 Servo demo!")

# make a list of all the servos
servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)

while True:
    # Repeat for all 4 servos
    for my_servo in servos:
        # Do the wave!
        print("Moving servo #", servos.index(my_servo)+1)
        my_servo.angle = 0      # right
        time.sleep(0.25)
        my_servo.angle = 90     # middle
        time.sleep(0.25)
        my_servo.angle = 180    # left
        time.sleep(0.25)
        my_servo.angle = 90     # middle
        time.sleep(0.25)
        my_servo.angle = 0      # right
```

This example is similar to the 1 servo example, but instead of accessing the crickit.servo_1 object directly, we'll make a **list** called servos that contains 4 servo objects with

servos = (crickit.servo_1, crickit.servo_2, crickit.servo_3, crickit.servo_4)

Then we can access the individual using servo[0].angle = 90 or iterate through them as we do in the loop. You don't *have* to do it this way, but its very compact and doesn't take a lot of code lines to create all 4 servos at once!

One thing to watch for is that if you use a list like this, servo[0] is the name of the Servo #1 and servo[3] is Servo #4!

Min/Max Pulse control

Originally servos were defined to use 1.0 millisecond to 2.0 millisecond pulses, at 50 Hz to set the 0 and 180 degree locations. However, as more companies started making servos they changed the pulse ranges to 0.5ms to 2.5ms or even bigger ranges. So, not all servos have their full range at thoe 'standard' pulse widths. You can easily tweak your code to change the min and max pulse widths, which will let your servo turn more left and right. **But** don't set the widths too small/large or you can hit the hard stops of the servo which could damage it, so try tweaking the numbers slowly until you get a sense of what the limits are for your motor.

All you need to do is add a line at the top of your code like this

`crickit.servo_1.set_pulse_width_range(min_pulse=500, max_pulse=2500)`

The above is for Crickit Servo #1, you'll need to duplicate and adjust for all other servos, but that way you can customize the range uniquely per servo!

Here we've change the minimum pulse from the default ~750 microseconds to 500, and the default maximum pulse from 2250 microseconds to 2500. Again, **each servo differs**. Some experimentation may be required!

```
import time
from adafruit_crickit import crickit

print("1 Servo demo with custom pulse widths!")

crickit.servo_1.set_pulse_width_range(min_pulse=500, max_pulse=2500)

while True:
    print("Moving servo #1")
    crickit.servo_1.angle = 0      # right
    time.sleep(1)
    crickit.servo_1.angle = 180    # left
    time.sleep(1)
```

Continuous Rotation Servos

If you're using continuous servos, you can use the angle assignments and just remember that 0 is rotating one way, 90 is 'stopped' and 180 and rotating the other way. Or, better yet, you can use the `crickit.continuous_servo_1` object instead of the plain `servo_1`

Again, you get up to 4 servos. You can mix 'plain' and 'continuous' servos

```
import time
from adafruit_crickit import crickit

print("1 Continuous Servo demo!")

while True:
    crickit.continuous_servo_1.throttle = 1.0 # Forwards
    time.sleep(2)
    crickit.continuous_servo_1.throttle = 0.5 # Forwards halfspeed
    time.sleep(2)
    crickit.continuous_servo_1.throttle = 0   # Stop
    time.sleep(2)
    crickit.continuous_servo_1.throttle = -0.5 # Backwards halfspeed
    time.sleep(2)
    crickit.continuous_servo_1.throttle = -1 # Forwards
    time.sleep(2)
    crickit.continuous_servo_1.throttle = 0   # Stop
    time.sleep(2)
```

If your continuous servo doesn't stop once the loop is finished you may need to tune the `min_pulse` and `max_pulse` timings so that the center makes the servo stop. Or check if the servo has a center-adjustment screw you can tweak.

Disconnecting Servos or Custom Pulses

If you want to 'disconnect' the Servo by sending it 0-length pulses, you can do that by 'reaching in' and adjusting the underlying PWM duty cycle with:

`crickit.servo_1._pwm_out.duty_cycle = 0`

or

`crickit.servo_1._pwm_out.fraction = 0`

Likewise you can set the duty cycle to a custom value with

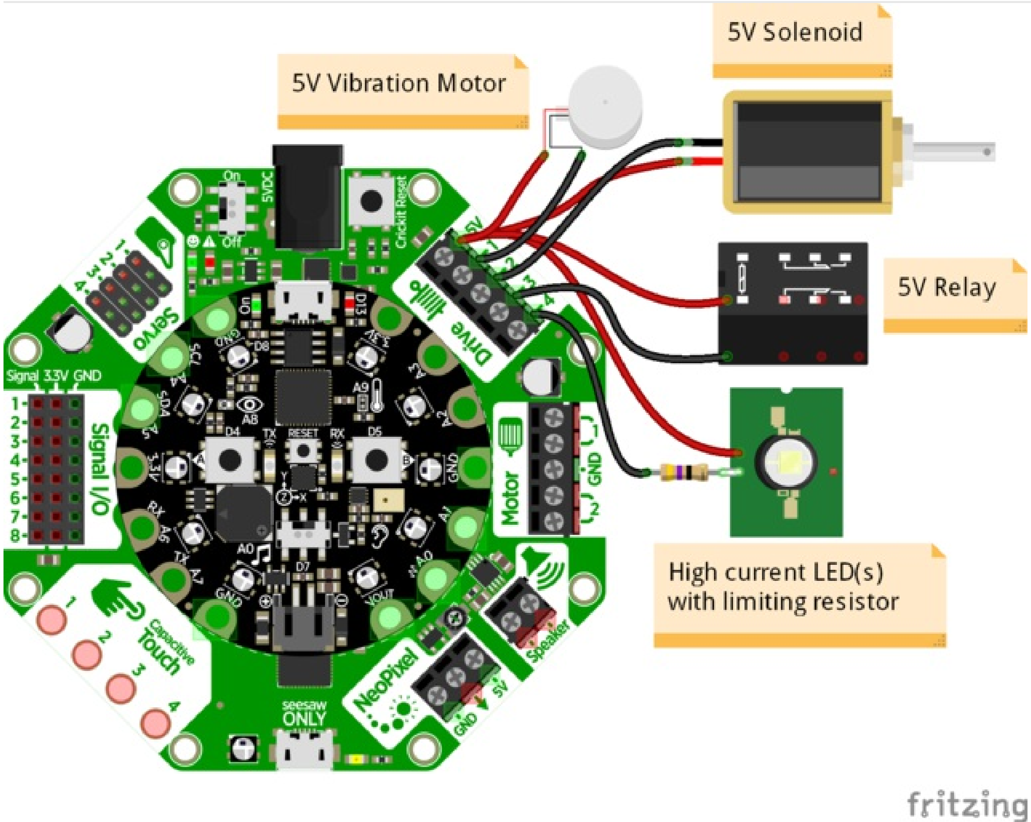
`crickit.servo_1._pwm_out.duty_cycle = number`

where *number* is between 0 (off) and 65535 (fully on). For example, setting it to 32767 will be 50% duty cycle, at the 50 Hz update rate

Or you can use fractions like `crickit.servo_1._pwm_out.fraction = 0.5`

```
import time
from adafruit_crickit import crickit

print("1 Servo release demo!")

while True:
    print("Moving servo #1")
    crickit.servo_1.angle = 0      # right
    time.sleep(10)
    print("Released")
    crickit.servo_1._pwm_out.duty_cycle = 0
    time.sleep(10)
    # and repeat!
```

CircuitPython Drives



https://adafru.it/Be2

https://adafru.it/Be2

Test Drive

Lets start by controlling a drive output. You'll need to plug something into the **5V** and **DRIVE1** terminal blocks. I'm just using a simple LED with resistor but anything that can be powered by 5V will work.

- Note that the drive outputs cannot have 5V output so you must connect the **positive** pin of whatever you're driving to **5V**. Don't try connecting the positive pin to the drive, and the negative pin to **GND**, it wont work!
- Drive outputs are PWM-able!

This example will show turning the drive output fully on and off once a second:

```
import time
from adafruit_crickit import crickit

print("1 Drive demo!")

crickit.drive_1.frequency = 1000

while True:
    crickit.drive_1.fraction = 1.0  # all the way on
    time.sleep(0.5)
    crickit.drive_1.fraction = 0.0  # all the way off
    time.sleep(0.5)
    crickit.drive_1.fraction = 0.5  # half on/off
    time.sleep(0.5)
    # and repeat!
```

We start by importing the libraries that we need to have time delays (`import time`) and then the main crickit python library that will make it super easy to talk to the motors and sensors on crickit (`from adafruit_crickit import crickit`)

The `crickit` object represents the drive outputs available for control. The drives are available on the sub-objects named `drive_1` , `drive_2` , `drive_3` , `drive_4`

Set PWM Frequency

Drive outputs are all PWM outputs too, so not only can they turn fully on and off, but you can also set it half-way on. In general, the default frequency for PWM outputs on seesaw is 1000 Hz, so set the frequency to 1 KHz with `crickit.drive_1.frequency = 1000` . Even if you aren't planning to use the PWM output, please set the frequency!

Note that all the Drive outputs share the same timer so if you set the frequency for one, it will be the same for all of them.

Control Drive Output

Now that we have a drive pwm object, we can simply assign the PWM duty cycle with the fraction property!

- `crickit.drive_1.fraction = 0.0` turns the output completely off (no drive to ground, no current draw).
- `crickit.drive_1.fraction = 1.0` turns the output completely on (fully drive to ground)
- And, not surprisingly `crickit.drive_1.fraction = 0.5` sets it to 1/2 on and 1/2 off at the PWM frequency set above.

More Drivers!

OK that was fun but you want MORE drives right? You can control up to four!

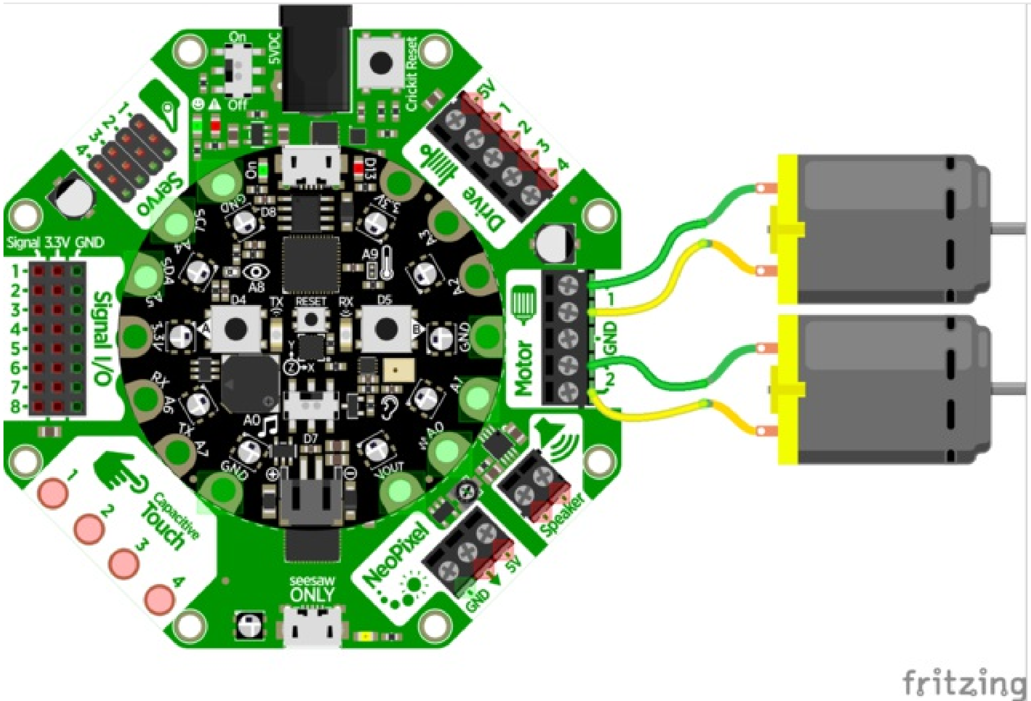
```
import time
from adafruit_crickit import crickit

print("4 Drive demo!")

drives = (crickit.drive_1, crickit.drive_2, crickit.drive_3, crickit.drive_4)

for drive in drives:
    drive.frequency = 1000

while True:
    for drive in drives:
        print("Drive #", drives.index(drive)+1)
        drive.fraction = 1.0  # all the way on
        time.sleep(0.25)
        drive.fraction = 0.0  # all the way off
        time.sleep(0.25)
        # and repeat!
```

This example is similar to the 1 drive example, but instead of accessing the `crickit.drive_1` object directly, we'll make a **list** called `drives` that contains 4 drive objects with

```
drives = (crickit.drive_1, crickit.drive_2, crickit.drive_3, crickit.drive_4)
```

Then we can access the individual using `drives[0].fraction = 0.5` or iterate through them as we do in the loop. You don't *have* to do it this way, but its very compact and doesn't take a lot of code lines to create all 4 drives at once!

CircuitPython DC Motors



https://adafru.it/BdY

https://adafru.it/BdY

You can drive two separate DC motors, so lets go ahead and get right to it!

DC motors are controlled by 4 PWM output pins, the 4 PWM pins let you control speed *and* direction. And we'll use our **adafruit_motor** library to help us manage the throttle (speed) and direction for us, making it very easy to control motors

Note that each DC motor is a little different, so just because you have two at the same throttle does not mean they'll rotate at the *exact* same speed! Some tweaking may be required

The two wires of the DC motor can be plugged in either way into each Crickit Motor port. If the motor spins the opposite way from what you want to call 'forward', just flip the wires!

```
import time
from adafruit_crickit import crickit

print("Dual motor demo!")

# make two variables for the motors to make code shorter to type
motor_1 = crickit.dc_motor_1
motor_2 = crickit.dc_motor_2

while True:
    motor_1.throttle = 1  # full speed forward
    motor_2.throttle = -1 # full speed backward
    time.sleep(1)

    motor_1.throttle = 0.5  # half speed forward
    motor_2.throttle = -0.5 # half speed backward
    time.sleep(1)

    motor_1.throttle = 0  # stopped
    motor_2.throttle = 0  # also stopped
    time.sleep(1)

    motor_1.throttle = -0.5  # half speed backward
    motor_2.throttle = 0.5   # half speed forward
    time.sleep(1)

    motor_1.throttle = -1  # full speed backward
    motor_2.throttle = 1   # full speed forward
    time.sleep(1)

    motor_1.throttle = 0  # stopped
    motor_2.throttle = 0  # also stopped
    time.sleep(0.5)

    # and repeat!
```

Import Libraries

We start by importing the libraries that we need to have time delays (`import time`) and then the main crickit python library that will make it super easy to talk to the motors and sensors on crickit (`from adafruit_crickit import crickit`)

The `crickit` object represents the motors and servos available for control. The motors are available on the sub-objects named `dc_motor_1` and `dc_motor_2`

Each of these are adafruit_motor.motor (https://adafru.it/BNE) type objects for the curious

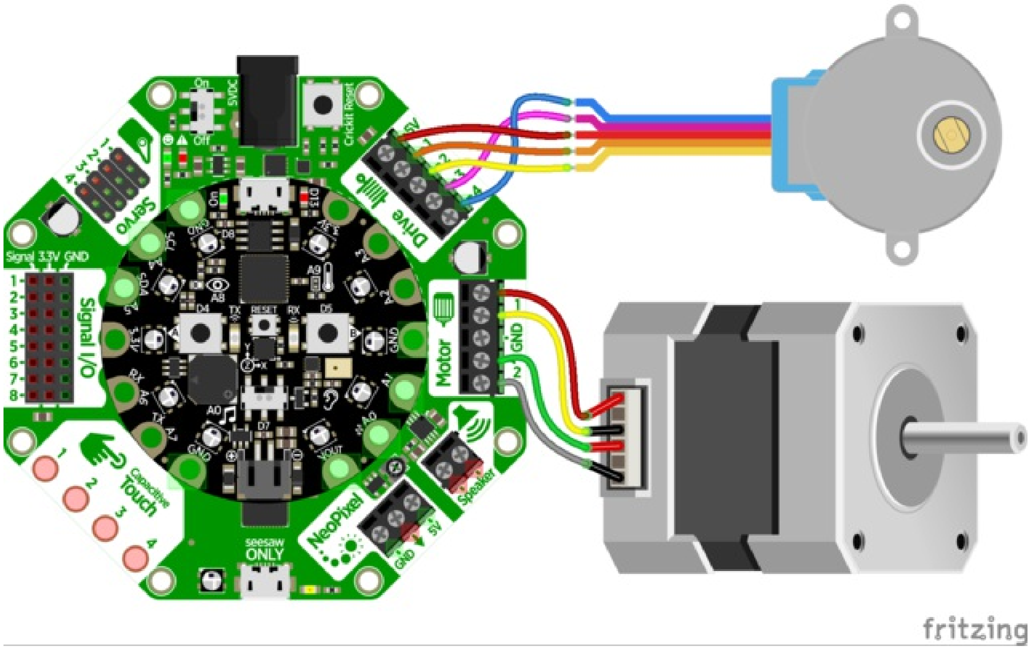
To make our code easier to read, we'll make new names for each motor:

```
# make two variables for the motors to make code shorter to type
motor_1 = crickit.dc_motor_1
motor_2 = crickit.dc_motor_2
```

Control Motor

Now that we have our motor objects, we can simply assign the throttle, this will set the direction and speed. For example, to set the speed to full forward, use `motor_1.throttle = 1` and to set to full speed backward use `motor_1.throttle = -1`. For speeds in between, use a fraction, such as `0.5` (half speed) or `0.25` (quarter speed). Setting the `throttle = 0` will stop the motor.

CircuitPython Steppers



https://adafru.it/BNF

https://adafru.it/BNF

Even though we show the Circuit Playground Crickit, the wiring and code works with the Feather and CPX Crickit

Even though we don't make it really obvious, you *can* drive stepper motors from the Crickit.

Stepper motors rotate all the way around but only one 'step' at a time. Usually there's a few hundred steps per turn, making them great for precision motion. The trade off is they're very slow compared to servos or steppers. Also, unlike servos they don't know 'where' they are in the rotation, they can only step forward and backwards.

There's *two* kinds of stepper motors: bipolar (4-wire) and unipolar (5 or 6-wire). We can control both kinds but with some restrictions!

- The voltage we use to power the motor is 5V only, so 5V power steppers are best, but sometimes you can drive 12V steppers at a slower/weaker rate
- You can drive **one** bi-polar stepper motor via the Motor port
- You can drive **two** uni-polar stepper motors, one via the Motor port and one via the Drive port
- That means you have have two uni-polar steppers or one uni and one bi-polar. But you cannot drive two bi-polar steppers.

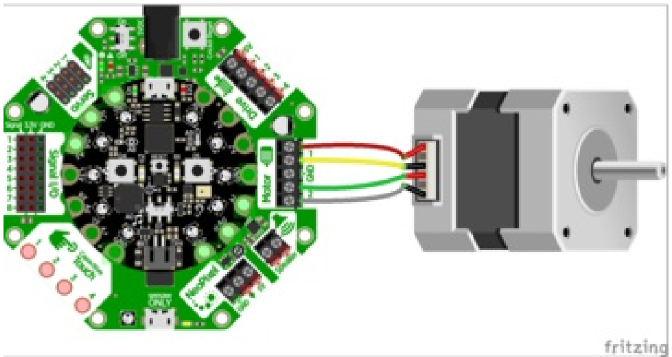
Bi-Polar or Uni-Polar Motor Port

The Crickit **Motor** port can run a unipolar (5-wire and 6-wire) or bipolar (4-wire) stepper. It cannot run steppers with any other # of wires!

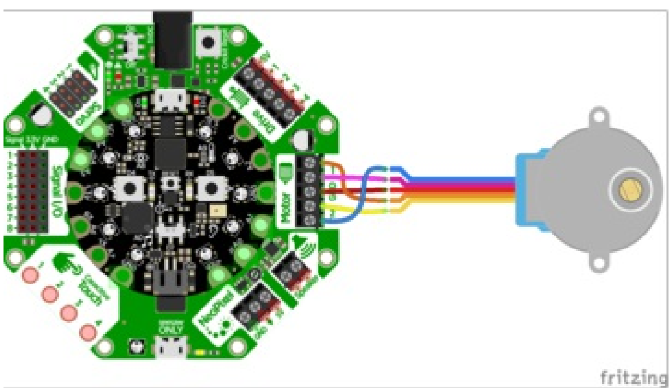
The code is the same for unipolar or bipolar motors, the wiring is just slightly different.

Unlike DC motors, the wire order **does** matter. Connect one coil to the Motor pair #1. Connect the other coil to the Motor pair #2

- If you have a bipolar motor, connect one motor coil to #1 and the other coil to #2 and do not connect to the center GND block.
- If you are using a unipolar motor with 5 wires, connect the common wire to the center GND port.
- If you are using a unipolar motor with 6 wires, you can connect the two 'center coil wires' together to the center GND port



If you are using our "12V" bi-polar stepper, (https://adafru.it/BxE) wire in this order: red, yellow, (skip GND center), green, gray



If you are using our 5V uni-polar stepper (https://adafru.it/BxF), wire in this order: orange, pink, red (ground), yellow, blue.

Here is the CircuitPython code for stepping various ways. You can try tweaking the `INTERSTEP_DELAY` to slow down the motor.

CircuitPython supports 4 different waveform stepping techniques. More on each is detailed at Wikipedia. (https://adafru.it/BxG)

- `SINGLE` stepping (one coil on at a time) - fast, lowest power usage, weak strength
- `DOUBLE` stepping (two coils on at a time) - fast, highest power, high strength
- `INTERLEAVE` stepping (alternates between one and two coils on) - slow (half the speed of single or double!), medium power, medium strength

- MICROSTEPPING - while this is supported its so slow with Crickit we're going to just 'skip' this one!

Unless you have power limiting requirements, DOUBLE is great for most projects. INTERLEAVE gives you smoother motion but is slower. SINGLE is simplest but weakest turning strength.

```
import time
from adafruit_crickit import crickit
from adafruit_motor import stepper

print("Bi-Polar or Uni-Polar Stepper motor demo!")

# make stepper motor a variable to make code shorter to type!
stepper_motor = crickit.stepper_motor
# increase to slow down, decrease to speed up!
INTERSTEP_DELAY = 0.01

while True:
    print("Single step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD)
        time.sleep(INTERSTEP_DELAY)

    print("Double step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD, style=stepper.DOUBLE)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD, style=stepper.DOUBLE)
        time.sleep(INTERSTEP_DELAY)
    print("Interleave step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD, style=stepper.INTERLEAVE)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD, style=stepper.INTERLEAVE)
        time.sleep(INTERSTEP_DELAY)
```

CircuitPython stepper motor control is pretty simple - you can access the motor port for stepper control via the `crickit.stepper_motor object` (it's an adafruit_motor.stepper type object (https://adafru.it/BNE)).

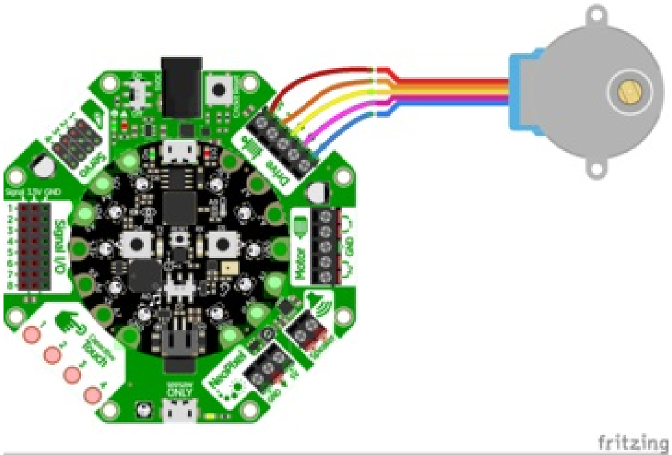
With that object, you can call `onestep()` to step once, with the `direction` and stepping `style` included. The default direction is `FORWARD` and the default style is `SINGLE`.

Note that 'forward' and 'backward' are, like DC motors, dependent on your wiring and coil order so you can flip around the coil wiring if you want to change what direction 'forward' and 'backward' means.

Putting `time.sleep()`'s between steps will let you slow down the stepper motor, however most steppers are geared so you may not want any delays.

Uni-Polar Only Drive Port

The Drive port can also control steppers although it can only do uni-polar! Don't try connecting a 4-wire bi-polar stepper, it won't work at all.

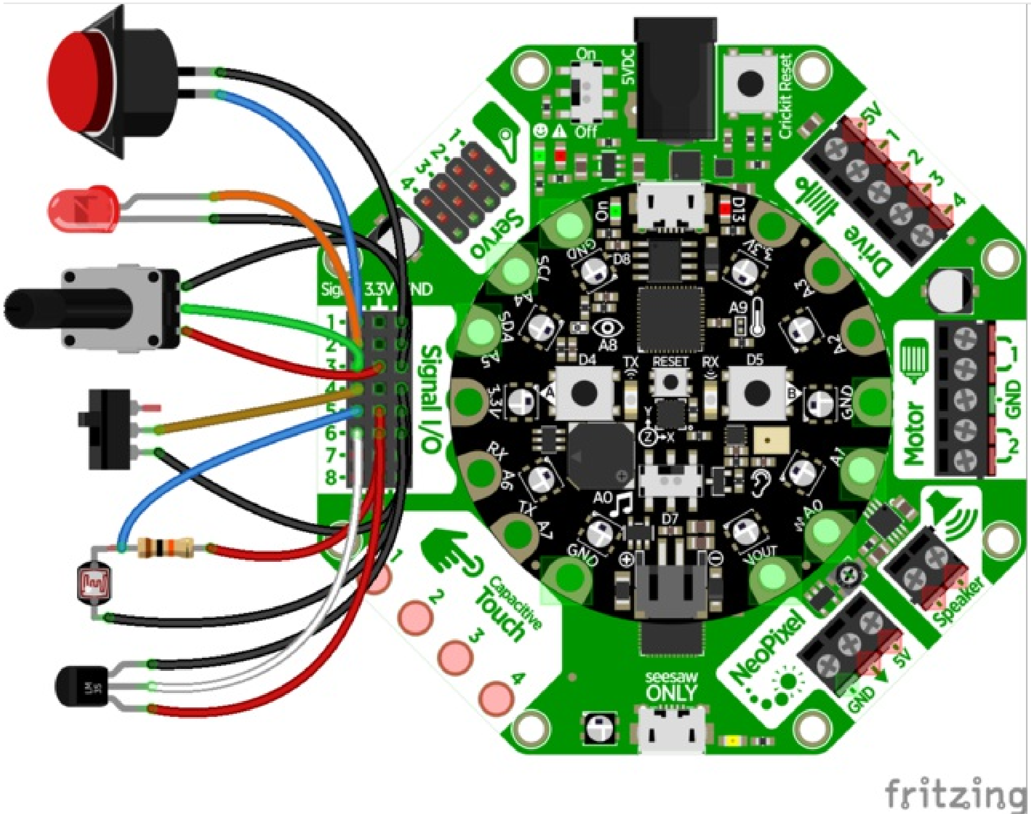


If you are using our 5V uni-polar stepper (https://adafru.it/BxF), wire in this order: red (5V), orange, yellow, pink, blue. That should line up with the wires on the plug

And here's the CircuitPython code. Note that the only difference is we're using the `crickit.drive_stepper_motor` object now!

```
import time
from adafruit_crickit import crickit
from adafruit_motor import stepper

print("Uni-Polar Stepper motor demo!")

# make stepper motor a variable to make code shorter to type!
stepper_motor = crickit.drive_stepper_motor  # Use the drive port

# increase to slow down, decrease to speed up!
INTERSTEP_DELAY = 0.02
while True:
    print("Single step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD)
        time.sleep(INTERSTEP_DELAY)

    print("Double step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD, style=stepper.DOUBLE)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD, style=stepper.DOUBLE)
        time.sleep(INTERSTEP_DELAY)
    print("Interleave step")
    for i in range(200):
        stepper_motor.onestep(direction=stepper.FORWARD, style=stepper.INTERLEAVE)
        time.sleep(INTERSTEP_DELAY)
    for i in range(200):
        stepper_motor.onestep(direction=stepper.BACKWARD, style=stepper.INTERLEAVE)
        time.sleep(INTERSTEP_DELAY)
```

CircuitPython Signals



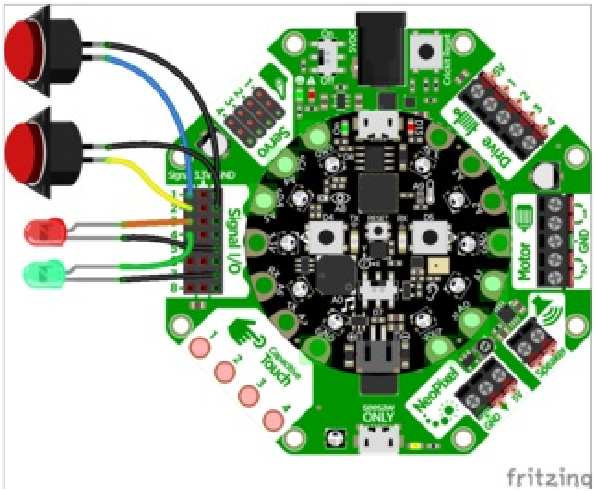
https://adafru.it/BNG

https://adafru.it/BNG

You may want to add buttons, LEDs, switches or simple sensors to your robot project. With Crickit, you get 8 x 'general purpose in/out' (GPIO) pins called **signals**. Each signal can be a digital input (button/switch), digital output (LED, for example), or *analog input.*

This lets you add a ton of external components easily, and its all handled by seesaw. Perfect when you have a Feather without analog inputs (like the ESP8266) or just need a ton of extra pins.

The signal pins are on a 3x8 female header, so you can poke wires directly in!



Here's an example wiring that goes with the code below.

We have two switch buttons, connected to signals #1 and #2, the other side of the buttons connect to ground

There's also two LEDs, connected to the signals #3 and #4 and the negative wires connected to ground. (All the 3.3V and Ground pins are connected together so you don't *have* to use the ones right next to the signal pin!)

```
import time
from adafruit_crickit import crickit

# For signal control, we'll chat directly with seesaw, use 'ss' to shorted typing!
ss = crickit.seesaw

# Two buttons are pullups, connect to ground to activate
BUTTON_1 = crickit.SIGNAL1  # button #1 connected to signal port 1 & ground
BUTTON_2 = crickit.SIGNAL2  # button #2 connected to signal port 2 & ground

ss.pin_mode(BUTTON_1, ss.INPUT_PULLUP)
ss.pin_mode(BUTTON_2, ss.INPUT_PULLUP)

# Two LEDs are outputs, on by default
LED_1 = crickit.SIGNAL3    # LED #1 connected to signal port 3 & ground
LED_2 = crickit.SIGNAL4    # LED #2 connected to signal port 4 & ground

ss.pin_mode(LED_1, ss.OUTPUT)
ss.pin_mode(LED_2, ss.OUTPUT)
ss.digital_write(LED_1, True)
ss.digital_write(LED_2, True)

while True:
    if not ss.digital_read(BUTTON_1):
        print("Button 1 pressed")
        ss.digital_write(LED_1, True)
    else:
        ss.digital_write(LED_1, False)

    if not ss.digital_read(BUTTON_2):
        print("Button 2 pressed")
        ss.digital_write(LED_2, True)
    else:
        ss.digital_write(LED_2, False)
```

Each of the 8 signal pin numbers is available under the `crickit` object as `SIGNAL1` through `SIGNAL8`. **Note these are not DigitalInOut or Pin objects!** We need to use the `crickit.seesaw` object to set the mode, direction, and readings

To simplify our code we shorted the `crickit.seesaw` object to just `ss`

```
# For signal control, we'll chat directly with seesaw, use 'ss' to shorted typing!
ss = crickit.seesaw
```

Digital Pin Modes

You can set the mode of each signal pin with `ss.pin_mode(signal, mode)` where *`signal`* is the `crickit.SIGNAL#` from above and *mode* can be `ss.OUTPUT`, `ss.INPUT` or `ss.INPUT_PULLUP`

```
ss.pin_mode(BUTTON_1, ss.INPUT_PULLUP)
ss.pin_mode(BUTTON_2, ss.INPUT_PULLUP)
...
ss.pin_mode(LED_1, ss.OUTPUT)
ss.pin_mode(LED_2, ss.OUTPUT)
```

Digital Read

Then, you can read the values True or False with `ss.digital_read(signal)`

Don't forget you have to set it to be an INPUT first!

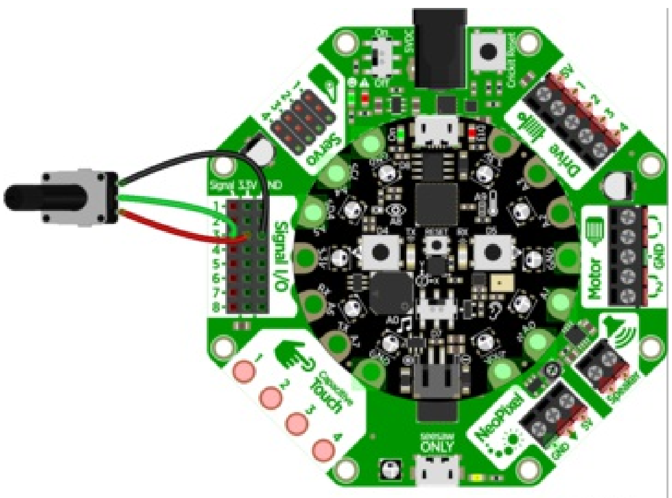
```
ss.digital_read(BUTTON_1)
```

Digital Write

Or, you can set the signal you want to a high value with `ss.digital_write(signal, True)` , or set to low value with `ss.digital_write(signal, False)` . Don't forget you have to set it to be an OUTPUT first!

```
# LED On
ss.digital_write(LED_2, True)
# LED Off
ss.digital_write(LED_2, False)
```

Analog Reads



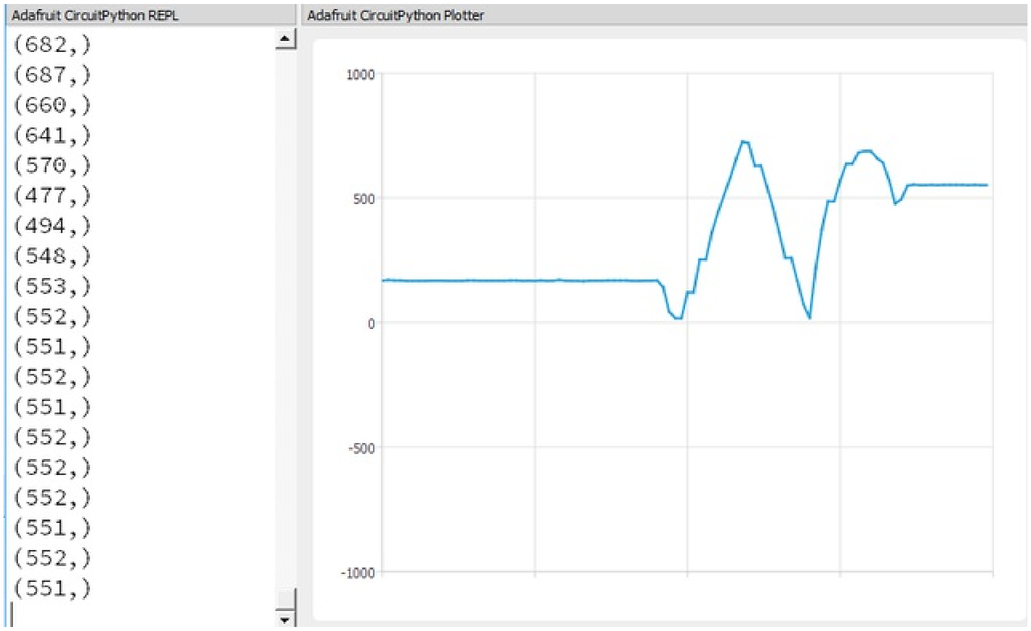
You can also read analog values like from a potentiometer or sensor.

Let's do a demonstration where the center tap of a potentiometer is hooked up to **Signal #3** - don't forget to also connect one side of the potentiometer to 3.3V and the other side to ground.

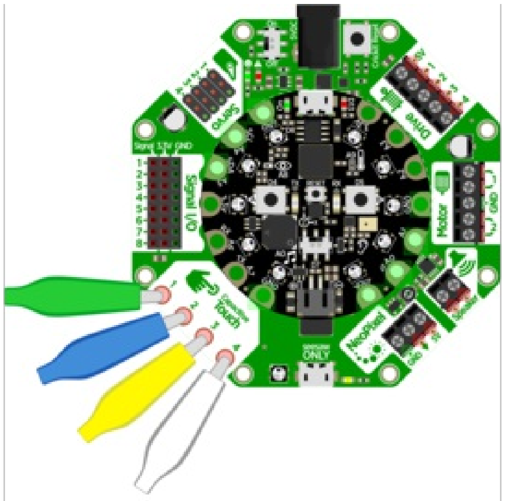
And here is the example code. You can see we read the signal with `ss.analog_read(signal)` which returns a value from 0 to 1023.

```
import time
from adafruit_crickit import crickit

# For signal control, we'll chat directly with seesaw, use 'ss' to shorted typing!
ss = crickit.seesaw
# potentiometer connected to signal #3
pot = crickit.SIGNAL3

while True:
    print((ss.analog_read(pot),))
    time.sleep(0.25)
```

By printing the value in a python tuple `(ss.analog_read(pot),)` we can use the Mu plotter to see the values immediately!



CircuitPython Touch



There's four capacitive touch pads you can use to detect human touch. They have big pads you can use to attach alligator/croc clips

You can read the value of the captouch pads from `crickit.touch_#.value` This will return `True` (if touched) or `False` (if not). This is the simplest/easiest way to detect touch, but it has a catch!

We determine if the touch is active by seeing the difference between the current 'raw' reading value and the first value. That means you do need to read the crickit touch pads *without* touching them first.

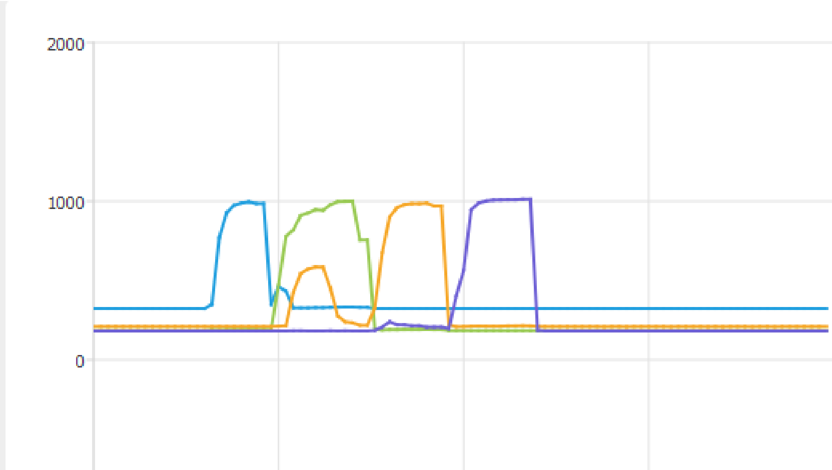
Try loading this code and touching the four pads while looking at the REPL

```
import time
from adafruit_crickit import crickit

# Capacitive touch tests

while True:
    if crickit.touch_1.value:
        print("Touched Cap Touch Pad 1")
    if crickit.touch_2.value:
        print("Touched Cap Touch Pad 2")
    if crickit.touch_3.value:
        print("Touched Cap Touch Pad 3")
    if crickit.touch_4.value:
        print("Touched Cap Touch Pad 4")
```

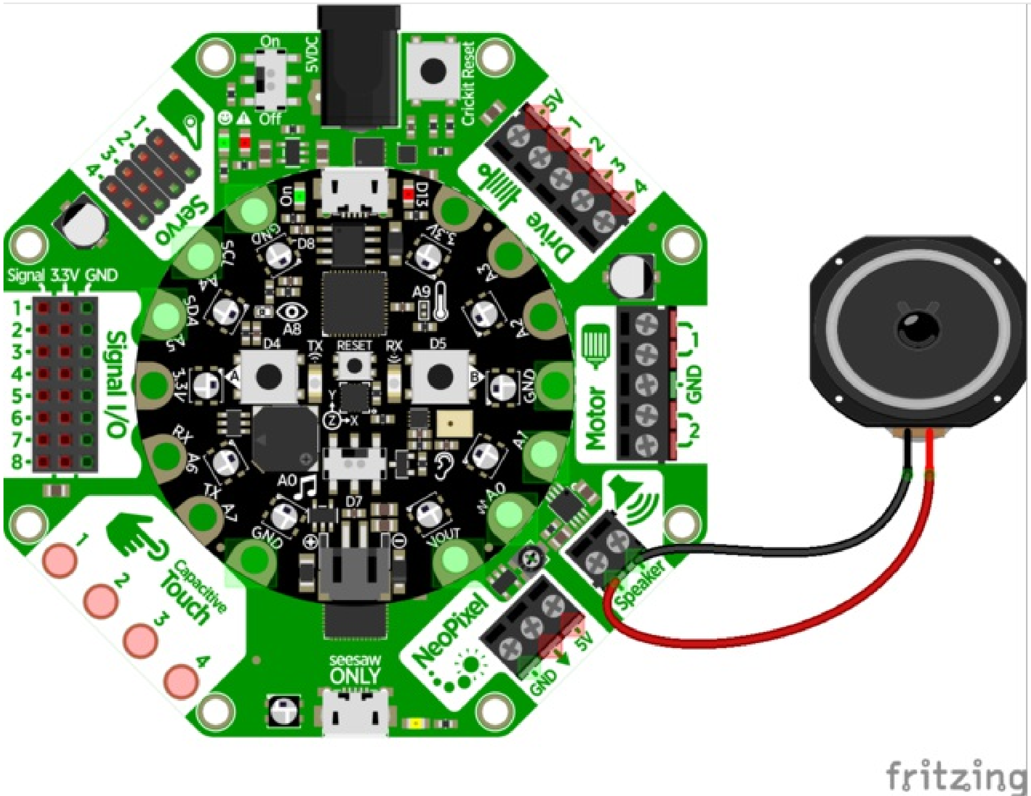
If you want to get more specific, you can read the '`raw_value`' value which is a number between 0 and 1023. Because there's always *some* capacitance its reading you'll see a starting value of about 250.

You can then test against a threshold, or use it to measure how hard someone is pressing against something (a fingertip vs a palm will give different readings)

```
import time
from adafruit_crickit import crickit

# Capacitive touch graphing test
touches = (crickit.touch_1, crickit.touch_2, crickit.touch_3, crickit.touch_4)

# Open up the serial Plotter in Mu to see the values graphed!

while True:
    touch_raw_values = (crickit.touch_1.raw_value, crickit.touch_2.raw_value,
                        crickit.touch_3.raw_value, crickit.touch_4.raw_value)
    print(touch_raw_values)
    time.sleep(0.1)
```



CircuitPython Audio



https://adafru.it/Be3

https://adafru.it/Be3

Audio animatronics! By adding a voice or sound effects to your robot you can make a cool interactive project. We take advantage of CircuitPython's ability to play WAV files over the true-analog output pin **A0**.

This is one of the *few* outputs that does not go through the seesaw chip. Instead, the audio is played directly from the CircuitPython board and the Crickit only amplifies it!

Audio File Formats

CircuitPython supports **Mono** (not stereo) **22 KHz sample rate** (or less) and **16-bit** WAV format. The reason for mono is that there's only one output, 22 KHz or less because the Circuit Playground can't handle more data than that (and also it wont sound much better) and the DAC output is 10-bit so anything over 16-bit will just take up room without better quality

CircuitPython does not support OGG or MP3. Just WAV!

Since the WAV file must fit on the CircuitPython file system, it must be under 2 MB

We have a detailed guide on how to generate WAV files here (https://adafru.it/s8f)

Amplifier Details

The onboard amplifier is a mono "Class D" audio amp with BTL (Bridge Tied Load) output.

That means **you cannot plug the speaker output into another amplifier, it must connect directly to a speaker!**

You can use just about *any* 4 to 8Ω speaker (6 Ω is OK too, just not as common). The amplifier can drive up to 3 Watts into 4Ω and 1 Watt into 8Ω. That means its ok to drive a 5 Watt speaker, it just wont be as loud as it *could* be with a bigger amp (but you wont damage the amp). You can also drive speakers that are smaller, like an 8Ω 0.5 W but make sure you don't turn the audio volume potentiometer up, as it could damage the speaker by overpowering it.

Basic Audio Playback

```
import audioio
import board

wavfile = "howto.wav"
f = open(wavfile, "rb")
wav = audioio.WaveFile(f)
a = audioio.AudioOut(board.A0)
a.play(wav)

# You can now do all sorts of stuff here while the audio plays
# such as move servos, motors, read sensors...

# Or wait for the audio to finish playing:
while a.playing:
    pass

f.close()
```

Here is the audio file we're using for this example

https://adafru.it/Be4

https://adafru.it/Be4

You must drag/copy this onto your **CIRCUITPY** disk drive, it's a big file so it will take a minute to copy over

Import Libraries

We start by importing the libraries that we need to make audio output `import audioio` Then we `import` `board`, our standardhardware library.

Create wave file and audio output

Next we set the name of the file we want to open, which is a wave file `wavfile = "howto.wav"` and then open the file as a **r**eadable **b**inary and store the file object in **f** which is what we use to actually read audio from: `f = open(wavfile, "rb")`

Now we will ask the audio playback system to load the wave data from the file `wav = audioio.WaveFile(f)` and finally request that the audio is played through the A0 analog output pin `a = audioio.AudioOut(board.A0)`

The audio file is now locked-and-loaded, and can be played at any time with `a.play(wav)`

Audio playback occurs in the background, using "DMA" (direct memory access) so you can control servos, motors, read sensors, whatever you like, while the DMA is happening. Since it happens asynchronously, you may want to figure out when its done playing. You can do that by checking the value of `a.playing` if it's `True` then its still processing audio, it will return `False` when complete.

Interactive Audio

OK just playing an audio file is one thing, but maybe you want to have some interactivity, such as waiting for the person to touch something or press a button? Here's an example of using a time-delay and then pausing until something occurs:

```
from busio import I2C
from adafruit_seesaw.seesaw import Seesaw
import audioio
import board
import time

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# what counts as a 'touch'
CAPTOUCH_THRESH = 500

wavfile = "howto.wav"
f = open(wavfile, "rb")
wav = audioio.WaveFile(f)
a = audioio.AudioOut(board.A0)
a.play(wav)

t = time.monotonic()  # this is the time when we started

# wait until we're at timecode 5.5 seconds into the audio
while time.monotonic() - t < 5.5:
    pass

a.pause()   # pause the audio

print("Waiting for Capacitive touch!")
while seesaw.touch_read(0) < CAPTOUCH_THRESH:
    pass

a.resume()   # resume the audio

# You can now do all sorts of stuff here while the audio plays
# such as move servos, motors, read sensors...

# Or wait for the audio to finish playing:
while a.playing:
    pass
print("Done!")
```

You may want to have the audio track match to motion or events in your robot. To do that you can do some tricks with `time.monotonic()`. That's our way of know true time passage, it returns a floating point (fractional) value in seconds. Note its hard to get the exact precise second so use `>` and `<` rather than checking for `=` equality because minute variations will make it hard to get the time delta exactly when it occurs.

CircuitPython Examples

Need some...err...inspiration? Here's some example projects with CircuitPython code and wiring diagrams. They're not full-featured guides but they provide a good basis for seeing how to use Crickit!

Bubble Bot

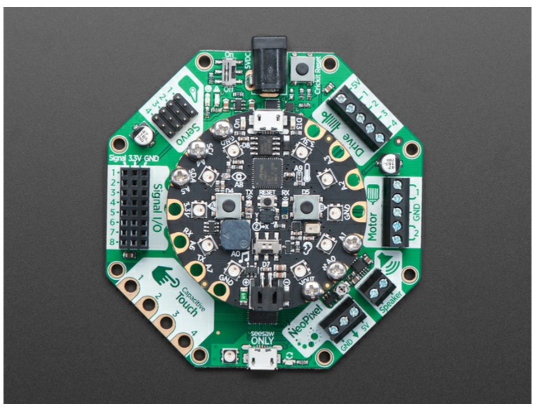
Its summer time and that means tank tops, ice cream and **bubbles!** This robot friend makes a bountiful burst of bubbles all on its own.

Parts List

Your browser does not support the video tag.

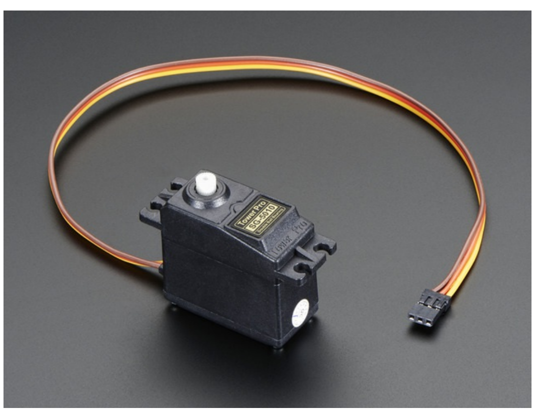
Circuit Playground Express

$24.95
IN STOCK
ADD TO CART



Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK
ADD TO CART



Standard servo - TowerPro SG-5010

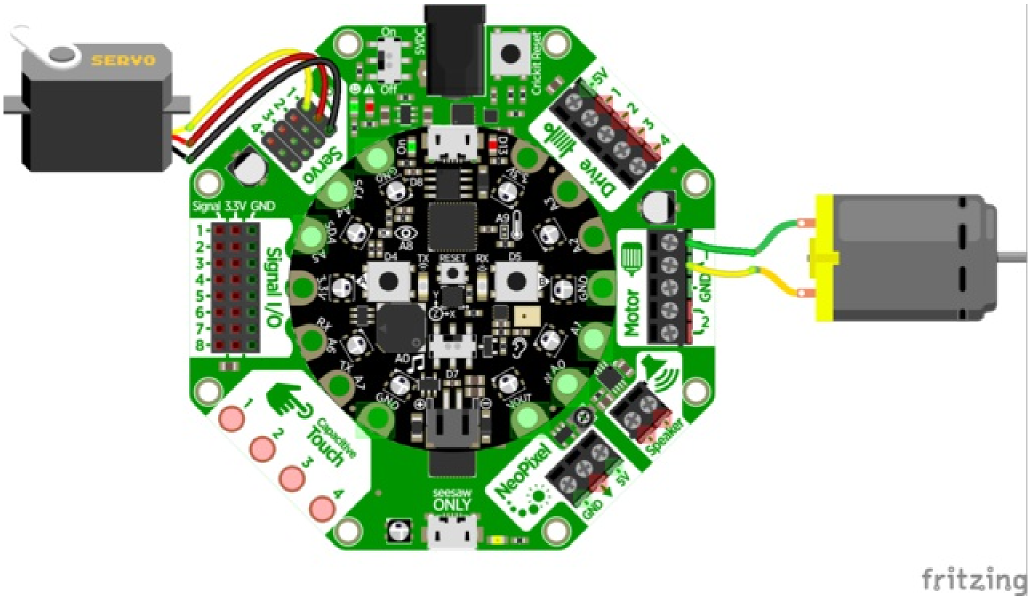
$12.00
IN STOCK
ADD TO CART

[DC Toy / Hobby Motor - 130 Size]

$1.95
IN STOCK

ADD TO CART

Wiring Diagram



Code

This simple robot doesn't do a lot but it does it very well!

We have one DC motor with a fan attachment, and one servo motor where we connect the bubble wand. Every few seconds, the wand goes down into the bubble mix, then back up, the fan turns on for 3 seconds, then turns off and the process repeats!

```python
# CircuitPython 3.0 CRICKIT demo
import time

import board
from adafruit_motor import servo, motor
from adafruit_seesaw.pwmout import PWMOut
from adafruit_seesaw.seesaw import Seesaw
from busio import I2C

i2c = I2C(board.SCL, board.SDA)
ss = Seesaw(i2c)

print("Bubble machine!")

SERVOS = True
DCMOTORS = True

# Create 4 Servos
servos = []
if SERVOS:
    for ss_pin in (17, 16, 15, 14):
        pwm = PWMOut(ss, ss_pin)
        pwm.frequency = 50
        _servo = servo.Servo(pwm)
        _servo.angle = 90  # starting angle, middle
        servos.append(_servo)

# Create 2 DC motors
motors = []
if DCMOTORS:
    for ss_pin in ((18, 19), (22, 23)):
        pwm0 = PWMOut(ss, ss_pin[0])
        pwm1 = PWMOut(ss, ss_pin[1])
        _motor = motor.DCMotor(pwm0, pwm1)
        motors.append(_motor)

while True:
    print("servo down")
    servos[0].angle = 180
    time.sleep(1)
    print("fan on")
    motors[0].throttle = 1
    time.sleep(3)
    print("fan off")
    time.sleep(1)
    motors[0].throttle = 0
    print("servo up")
    servos[0].angle = 0
    time.sleep(1)
```

Feynman Simulator

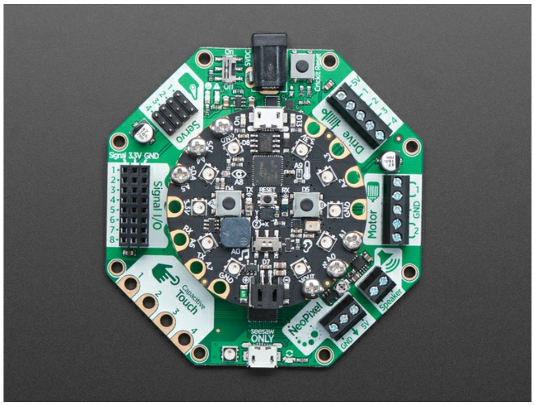
If you are a fan of physics wunderkind Richard Feynman *and* you like bongo drums, this Feynman simulator will satisfy your every desire. Between wise quips, this Feyn-bot will dazzle you with its drumming expertise (https://adafru.it/BxR).

Parts List

Your browser does not support the video tag.

Circuit Playground Express

$24.95
IN STOCK
ADD TO CART



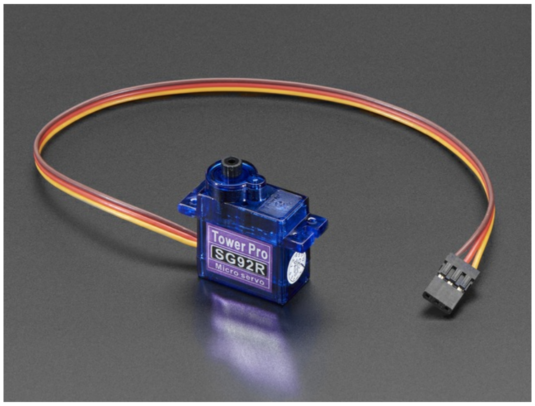
Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK
ADD TO CART

Your browser does not support the video tag.

Mini Push-Pull Solenoid - 5V

$4.95
IN STOCK
ADD TO CART



[Micro servo](https://www.adafruit.com)

$5.95
IN STOCK

ADD TO CART

Mono Enclosed Speaker - 3W 4 Ohm

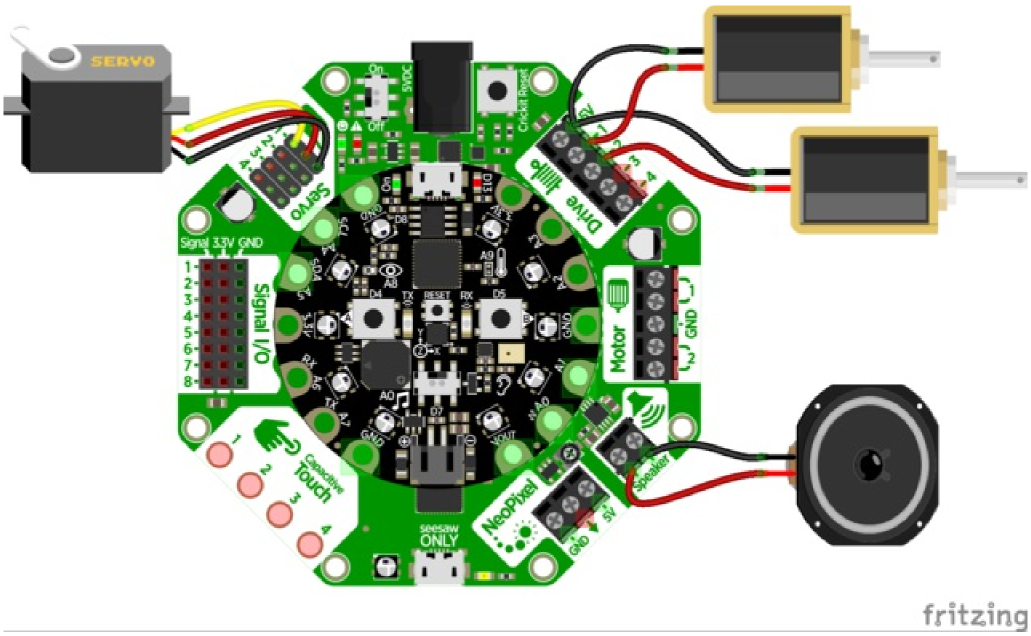
$3.95
IN STOCK

ADD TO CART

Wiring Diagram

Solenoids don't have 'direction' - any current will make them push. So even though we wired the black wire to 5V and the red wires to the #1 and #2 drive ports, they'll work just fine.

The microservo is taped to a wooden stick that moves the paper cut-out mouth up and down, for a Monty-Python-style puppet (https://adafru.it/BxS).



Code

Our code plays through a few wave file quips and quotes we found online, with some interstitial bongo drumming. Once all the audio has been played, it bongos for a long time, then repeats!

```
# CircuitPython 3.0 CRICKIT demo

import gc
import time

import audioio
import board
from adafruit_motor import servo
from adafruit_seesaw.pwmout import PWMOut
from adafruit_seesaw.seesaw import Seesaw
from busio import I2C

i2c = I2C(board.SCL, board.SDA)
ss = Seesaw(i2c)

print("Feynbot demo!")

# 1 Servo
pwm = PWMOut(ss, 17)
pwm.frequency = 50
myservo = servo.Servo(pwm)
myservo.angle = 180  # starting angle, highest

# 2 Drivers
drives = []
for ss_pin in (13, 12):
    _pwm = PWMOut(ss, ss_pin)
    _pwm.frequency = 1000
    drives.append(_pwm)

# Audio files
wavfiles = ["01.wav", "02.wav", "03.wav", "04.wav", "05.wav"]
a = audioio.AudioOut(board.A0)


# Start playing the file (in the background)
def play_file(wavfile):
    f = open(wavfile, "rb")
    wav = audioio.WaveFile(f)
    a.play(wav)


# Tap the solenoids back and forth
def bongo(t):
    for _ in range(t):
        drives[0].duty_cycle = 0xFFFF
        time.sleep(0.1)
        drives[0].duty_cycle = 0
        time.sleep(0.1)
        drives[1].duty_cycle = 0xFFFF
        time.sleep(0.1)
        drives[1].duty_cycle = 0
        time.sleep(0.1)
```

```
        time.sleep(0.1)


# Move mouth back and forth
def talk(t):
    for _ in range(t):
        myservo.angle = 150
        time.sleep(0.1)
        myservo.angle = 180
        time.sleep(0.1)


filenum = 0  # counter to play all files

while True:
    gc.collect()
    print(gc.mem_free())

    # time to play the bongos!
    bongo(5)
    time.sleep(1)

    # OK say something insightful
    play_file(wavfiles[filenum])
    # and move the mouth while it does
    while a.playing:
        talk(1)

    # Done being insightful, take a break
    time.sleep(1)

    # If we went thru all the files, JAM OUT!
    filenum += 1
    if filenum >= len(wavfiles):
        bongo(20)
        filenum = 0
```

Slime Night

Ladyada was unable to get to sleep. Feeling restless she decided to visit her workshop and make some slime to help soothe her soul. Then her companion showed up to lend a hand and have fun together!

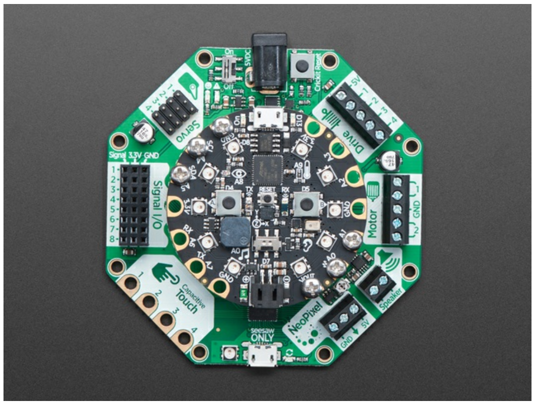
How to Make Slime

- **1 Bottle Elmers Glue** - we like the glitter glue but you can use plain white glue and add food coloring!
- **1/2 Tablespoon Baking Soda** - not baking powder! You probably have some in your freezer, fridge, or baking cabinet
- **1 Tablespoon Contact Lens Solution** - make sure to get the stuff with **Boric Acid!**

Put glue in a glass container, add soda and solution, mix & enjoy!

The quantities are flexible and you don't have to be exact. Add a little more or less to change gooeyness.

Parts Used



Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK

ADD TO CART

Your browser does not support the video tag.

Circuit Playground Express

$24.95
IN STOCK

ADD TO CART

DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC

$2.95

IN STOCK

ADD TO CART

Foot Pedal Potentiometer - Sewing Machine Speed Controller

$26.95

IN STOCK

ADD TO CART

3.5mm (1/8") Stereo Audio Jack Terminal Block

$2.50

IN STOCK

ADD TO CART

Mono Enclosed Speaker - 3W 4 Ohm

$3.95

IN STOCK

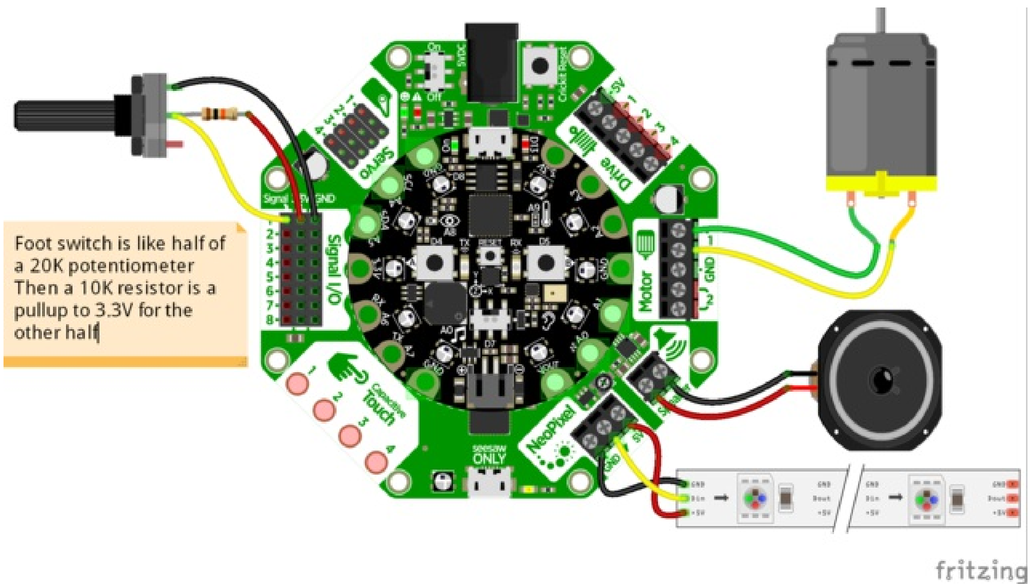
ADD TO CART

[Adafruit NeoPixel LED Dots Strand - 20 LEDs at 2" Pitch]

$27.50
IN STOCK

ADD TO CART

Wiring Diagram



CircuitPython Code

This project has a foot pedal potentiometer that controls the speed of the TT motor that spins the platter around. Since foot pedals are rheostats (variable resistors) you need another resistor to finish the divider. We use a plain 10K, any value from about 4.7K to 47K will do fine.

When not pressed, the analog reading value is about 700. When pressed, the reading goes down to about 50. You may need to calibrate these numbers for your foot pedal!

We map the analog press values to motor speed, our max speed we want is 0.5 throttle (1.0 was waaay to fast) using our simple mapper helper. If its our first time pressing the pedal, we play the audio file 3 seconds later, to give some ambience.

You can also press the **A** button to turn on/off some pretty NeoPixels.

```
import time
from digitalio import DigitalInOut, Direction, Pull
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.analoginput import AnalogInput
from adafruit_seesaw.pwmout import PWMOut
```

```
from adafruit_motor import motor
from busio import I2C
import neopixel
import audioio
import board

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# built in CPX button A
button = DigitalInOut(board.BUTTON_A)
button.direction = Direction.INPUT
button.pull = Pull.DOWN

# NeoPixels
pixels = neopixel.NeoPixel(board.A1, 10, brightness=0)
pixels.fill((0,0,250))

# Analog reading from Signal #1 (ss. #2)
foot_pedal = AnalogInput(seesaw, 2)

# Create one motor on seesaw PWM pins 22 & 23
motor_a = motor.DCMotor(PWMOut(seesaw, 22), PWMOut(seesaw, 23))
motor_a.throttle = 0

def map_range(x, in_min, in_max, out_min, out_max):
    # Maps a number from one range to another.
    mapped = (x-in_min) * (out_max - out_min) / (in_max-in_min) + out_min
    if out_min <= out_max:
        return max(min(mapped, out_max), out_min)
    return min(max(mapped, out_max), out_min)

# Get the audio file ready
wavfile = "unchained.wav"
f = open(wavfile, "rb")
wav = audioio.WaveFile(f)
a = audioio.AudioOut(board.A0)

time_to_play = 0  # when to start playing
played = False  # have we played audio already? only play once!
while True:
    # Foot pedal ranges from about 700 (unpressed) to 50 (pressed)
    # make that change the speed of the motor from 0 (stopped) to 0.5 (half)
    press = foot_pedal.value
    speed = map_range(press, 700, 50, 0, 0.5)
    print("%d -> %0.3f" % (press, speed))
    motor_a.throttle = speed

    if not time_to_play and speed > 0.1:
        print("Start audio in 3 seconds")
        time_to_play = time.monotonic() + 3
    elif time_to_play and time.monotonic() > time_to_play and not played:
        print("Playing audio")
        a.play(wav)
        played = True

    # turn on/off blue LEDs
    if button.value:
```

```
        if pixels.brightness < 0.1:
            pixels.brightness = 1
        else:
            pixels.brightness = 0
        time.sleep(0.5)

    # loop delay
    time.sleep(0.1)
```

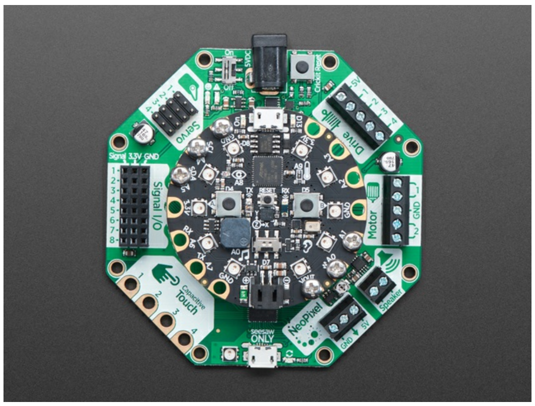
Flying Trapeze

Feel the excitement, the thrill, the rushing air beneath your wings - without having to leave home or run any risk of injury or sweating!

This Flying Trapeze bot uses a servo claw to grip onto a willing gymnast, and release it into the air when the detected acceleration has reached a sufficient peak!

Parts List

The servo claw we used had a built in metal gear servo that could draw significant current when actuated! We found a 4xAA battery pack with good NiMH batteries would last a while but 3xNiMH couldn't power it sufficiently



Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK

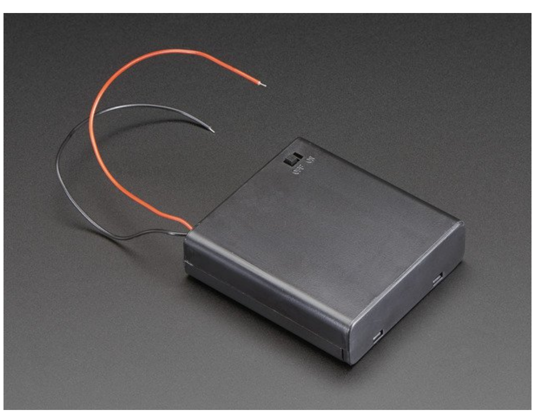
ADD TO CART

Your browser does not support the video tag.

Circuit Playground Express

$24.95
IN STOCK

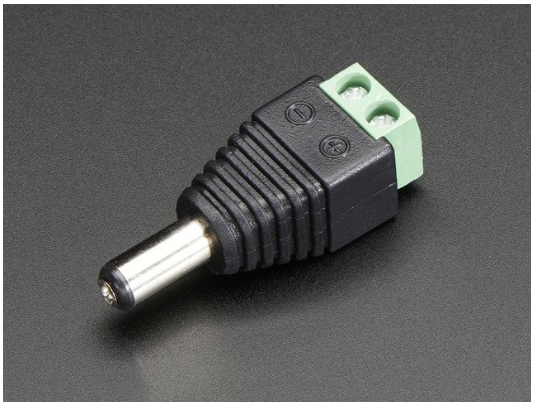
ADD TO CART



4 x AA Battery Holder with On/Off Switch

$2.95
OUT OF STOCK

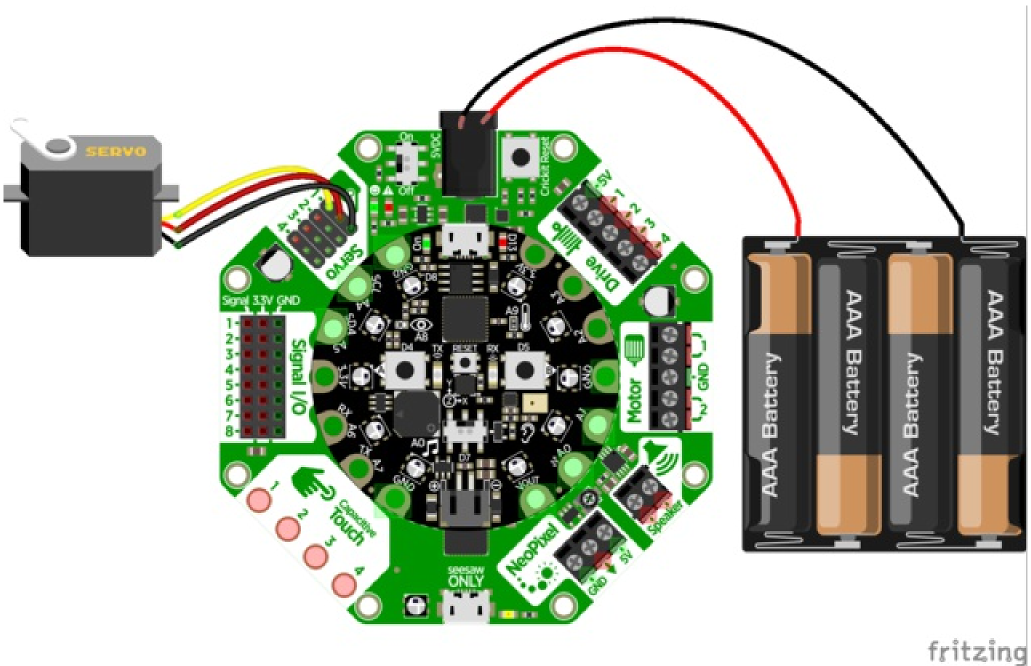
OUT OF STOCK



Male DC Power adapter - 2.1mm plug to screw terminal block

$2.00
IN STOCK
ADD TO CART

Wiring



Boot.py

Since we want to have the ability to data log the accelerometer, we need to put the CPX into 'filesystem write mode' - this boot.py will let you use the switch on the CPX to select whether you want to log data or go into trapeze-release mode

```
# Save as boot.py to turn on/off datalogging capability

import digitalio
import board
import storage

switch = digitalio.DigitalInOut(board.D7)  # For Circuit Playground Express
switch.direction = digitalio.Direction.INPUT
switch.pull = digitalio.Pull.UP

# If the switch pin is connected to ground CircuitPython can write to the drive
storage.remount("/", switch.value)
```

CircuitPython Code

Our Python code is dual use. You can use the slide switch to select whether you want to log the accelerometer data to the onboard storage. If you do, its easy to plot it and see the magnitude of the forces on your trapeze artist!

We mostly used data log mode to calibrate how 'hard' we required the person to push the trapeze to make the servo release the gymnast-stand-in.

We also have two buttons on the CPX we use for different tasks. In logging mode, you use button **A** to turn on/off logging. The red LED blinks to let you know logging is occuring. In trapeze mode, **A** and **B** let you manually open/close the servo gripper so you can have it grab the gymnasts head. Hey life's tough all around!

Finally, if we're in trapeze mode, we look for when we're at the beginning of a swing, that's when the Z axis acceleration drops below 3 m/s^2 and the Y axis has positive acceleration (we used the data log info to figure this out!) If so, the next time we reach max-acceleration, at the lowest point of the swing, we start opening the gripper, which takes a little time so that when we are at the end of the swing, it's opened enough for the gymnast to be released!

We change the NeoPixel colors to help debug, by flashing when we reach the different sensor states, since we don't have wireless data transfer on the CPX.

```
import time
from digitalio import DigitalInOut, Direction, Pull
import adafruit_lis3dh
from busio import I2C
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import servo
import neopixel
import board

# create accelerometer
i2c1 = I2C(board.ACCELEROMETER_SCL, board.ACCELEROMETER_SDA)
lis3dh = adafruit_lis3dh.LIS3DH_I2C(i2c1, address=0x19)
lis3dh.range = adafruit_lis3dh.RANGE_8_G

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# Create servo object
pwm = PWMOut(seesaw, 17)    # Servo 1 is on s.s. pin 17
pwm.frequency = 50          # Servos like 50 Hz signals
```

```
my_servo = servo.Servo(pwm) # Create my_servo with pwm signal

# LED for debugging
led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

# two buttons!
button_a = DigitalInOut(board.BUTTON_A)
button_a.direction = Direction.INPUT
button_a.pull = Pull.DOWN
button_b = DigitalInOut(board.BUTTON_B)
button_b.direction = Direction.INPUT
button_b.pull = Pull.DOWN

# NeoPixels!
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=1)
pixels.fill((0,0,0))

#################### log file for logging mode!
logfile = "/log.csv"
# check that we could append if wanted to
try:
    fp = None
    fp = open(logfile, "a")
    print("File system writable!")
# pylint: disable=bare-except
except:
    print("Not logging, trapeeze mode!")

# If we log, have some helper variables
logging = False
logpoints = 0
outstr = ""

# When its time to release the trapeze
release = False

while True:
    if button_a.value: # A pressed
        while button_a.value: # wait for release
            pass
        if fp:  # start or stop logging
            logging = not logging
            print("Logging: ", logging)
            time.sleep(0.25)
        else:
            my_servo.angle = 180      # open

    if button_b.value: # B pressed
        while button_b.value: # wait for release
            pass
        my_servo.angle = 0      # close

    x, y, z = lis3dh.acceleration

    # To keep from corrupting the filesys, take 25 readings at once
    if logging and fp:
        outstr += "%0.2F, %0.2F, %0.2F\n" % (x, y, z)
        logpoints += 1
```

```
        if logpoints > 25:
            led.value = True
            #print("Writing: "+outstr)
            fp.write(outstr+"\n")
            fp.flush()
            led.value = False
            logpoints = 0
    else:
        # display some neopixel output!
        if z > 20:
            # MAXIMUM EFFORT!
            pixels.fill((0, 255, 0))
            if release:
                my_servo.angle = 180

        elif z < 3 and y > 0: # means at the outer edge
            release = True
            # flash red when we peak
            pixels.fill((255, 0, 0))

        else:
            pixels.fill((0,0,int(abs(z)*2)))

    time.sleep(0.05)
```

For the curious, our data log file is here! (https://adafru.it/BkU)

R.O.B. GyroBot

We picked up a Nintendo R.O.B. robot from our local online auction site and when it appeared we decided to figure out how to get it working. There's 3 motors inside, and the R.O.B. already comes with motor drivers and end-stops, so instead of driving the robot directly, we decided to control the R.O.B. using Circuit Playground Express (CPX) and Crickit!

The code is all in CircuitPython.

We use the Crickit for the amplified audio effects (we snagged some audio from gameplay to give it that authentic chiptune sound), driving an IR LED for signalling at 500mA burst current so we could have it a few feet away, and the capacitive touch inputs for our desk controller.

With the addition of a D battery for the gyro turner, we had a fun live-action game without the need of a CRT!

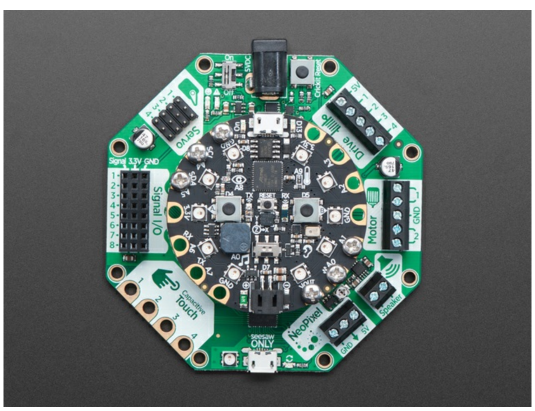
Parts List

Your browser does not support the video tag.

Circuit Playground Express

$24.95
IN STOCK

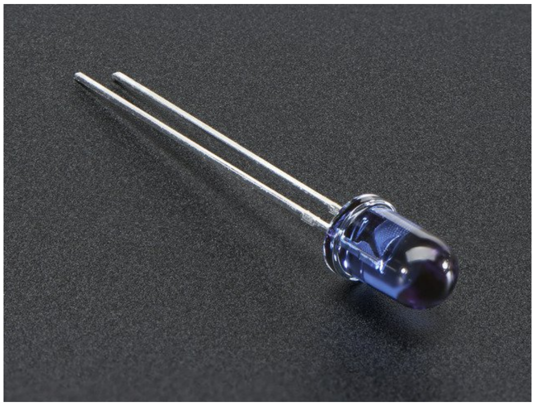
ADD TO CART



Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK

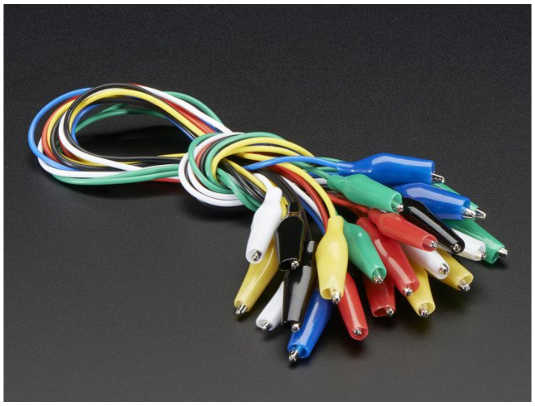
ADD TO CART



[Super-bright 5mm IR LED]

$0.75
IN STOCK

ADD TO CART



[Small Alligator Clip Test Lead (set of 12)]

$3.95
IN STOCK

ADD TO CART

[Mono Enclosed Speaker - 3W 4 Ohm]

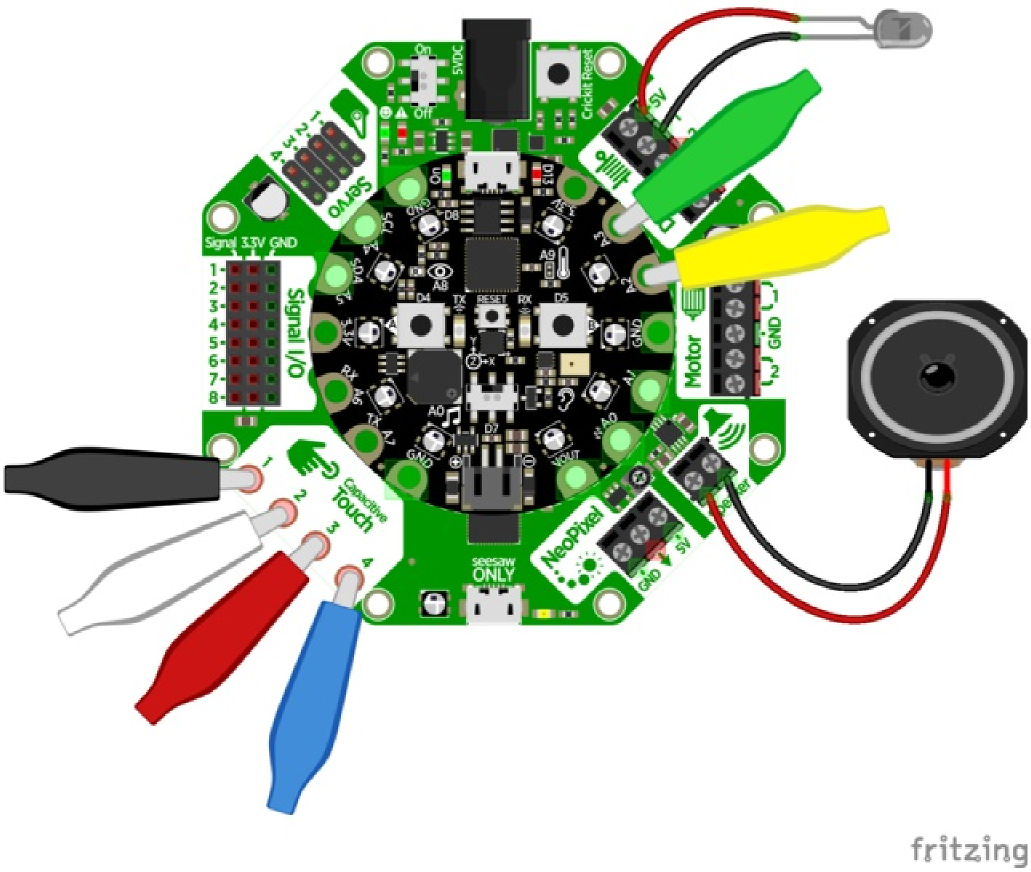
$3.95
IN STOCK

ADD TO CART

Wiring Diagram

The IR LED can handle up to 1 Amp peak current, so don't use a resistor, just wire it up to Drive 1 directly!

We use 4 capacitive touch sensors from the Crickit and 2 from CPX for 6 total (there's more capacitive touch inputs available on Crickit Signal pins but we wanted to use plain alligator pads!)



Code!

Save to your CPX as **code.py** and touch the alligator clips to control your R.O.B.

The IR LED should be 1-2 feet away and pointed at the R.O.B's left eye (or, the right-most eye when you are looking at R.O.B)

It will calibrate when first starting up, and play some tunes.

Flip the switch on/off on the CPX to turn on/off the capacitive touch detection/command sending (if you need to adjust your cables without having the robot turn around on you!

To help you know what's going on, the NeoPixels on the CPX will glow to match the colors of the alligator clips shown above, so use those same colors! Only exception is black shows up as purple LEDs.

You may need to tweak the capacitive touch threshholds. Try uncommenting

`#touch_vals = (touch2.raw_value, touch3.raw_value, seesaw.touch_read(0), seesaw.touch_read(1), seesaw.touch_read(2), seesaw.touch_read(3))`
`#print(touch_vals)`

And watching the REPL to see what the values read are.

```
import time
import gc
from digitalio import DigitalInOut, Direction, Pull
from busio import I2C
```

```
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
import touchio
import audioio
import neopixel
import board

pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=1)
pixels.fill((0,0,0))

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# switch
switch = DigitalInOut(board.SLIDE_SWITCH)
switch.direction = Direction.INPUT
switch.pull = Pull.UP

# We need some extra captouches
touch2 = touchio.TouchIn(board.A2)
touch3 = touchio.TouchIn(board.A3)

# LED for debugging
led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

# Create drive (PWM) object
INFRARED_LED_SS = 13
my_drive = PWMOut(seesaw, INFRARED_LED_SS)    # Drive 1 is on s.s. pin 13
my_drive.frequency = 1000        # Our default frequency is 1KHz

CAPTOUCH_THRESH = 850

# Commands, each 8 bit command is preceded by the 5 bit Init sequence
Init = [0, 0, 0, 1, 0]            # This must precede any command
Calibrate = [1, 0, 1, 0, 1, 0, 1, 1]  # the initial calibration
Up = [1, 0, 1, 1, 1, 0, 1, 1]     # Move arms/body down
Down = [1, 1, 1, 1, 1, 0, 1, 1]   # Move arms/body up
Left = [1, 0, 1, 1, 1, 0, 1, 0]   # Twist body left
Right = [1, 1, 1, 0, 1, 0, 1, 0]  # Twist body right
Close = [1, 0, 1, 1, 1, 1, 1, 0]  # Close arms
Open = [1, 1, 1, 0, 1, 1, 1, 0]   # Open arms
Test = [1, 1, 1, 0, 1, 0, 1, 1]   # Turns R.O.B. head LED on

print("R.O.B. Start")


def IR_Command(cmd):
    print("Sending ", cmd)
    gc.collect()                     # collect memory now, timing specific!
    # Output initialization and then command cmd
    for val in Init+cmd:             # For each value in initial+command
        if val:                      # if it's a one, flash the IR LED
            seesaw.analog_write(INFRARED_LED_SS, 65535)  # on
            seesaw.analog_write(INFRARED_LED_SS, 0)      # off 2ms later
        time.sleep(0.013)       # 17 ms total
    # pylint: disable=useless-else-on-loop
    else:
        time.sleep(0.015)       # 17 ms total
```

```
        time.sleep(0.015)       # 17 ms total

a = audioio.AudioOut(board.A0)
startfile = "startup.wav"
loopfile = "loop.wav"
with open(startfile, "rb") as f:
    wav = audioio.WaveFile(f)
    a.play(wav)
    for _ in range(3):
        IR_Command(Calibrate)
        time.sleep(0.5)
    while a.playing:
        IR_Command(Open)
        time.sleep(1)
        IR_Command(Close)
        time.sleep(1)
f = open(loopfile, "rb")
wav = audioio.WaveFile(f)
a.play(wav, loop=True)

while True:                          # Main Loop poll switches, do commands
    led.value = switch.value         # easily tell if we're running
    if not switch.value:
        continue

    #touch_vals = (touch2.raw_value, touch3.raw_value, seesaw.touch_read(0), seesaw.touch_read(1),
    #              seesaw.touch_read(2), seesaw.touch_read(3))
    #print(touch_vals)

    if touch2.raw_value > 3000:
        print("Open jaws")
        pixels.fill((50,50,0))
        IR_Command(Open)             # Button A opens arms

    elif touch3.raw_value > 3000:
        print("Close jaws")
        pixels.fill((0,50,0))
        IR_Command(Close)            # Button B closes arms

    elif seesaw.touch_read(0) > CAPTOUCH_THRESH:
        print("Up")
        pixels.fill((50,0,50))
        IR_Command(Up)

    elif seesaw.touch_read(1) > CAPTOUCH_THRESH:
        print("Down")
        pixels.fill((50,50,50))
        IR_Command(Down)

    elif seesaw.touch_read(2) > CAPTOUCH_THRESH:
        print("Left")
        pixels.fill((50,0,0))
        IR_Command(Left)

    elif seesaw.touch_read(3) > CAPTOUCH_THRESH:
        print("Right")
        pixels.fill((0,0,50))
        IR_Command(Right)

    time.sleep(0.1)
```

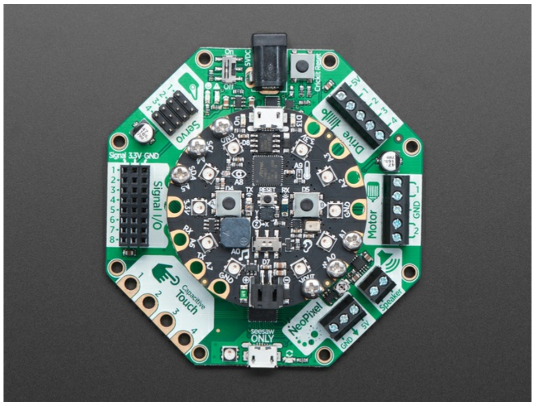
```
    pixels.fill((0,0,0))
```

Gear Tower

Plastic toys are great for hacking, modding and improving using Crickit! This box o' gears is fun for about 10 minutes...but when you add motors, sensors and robotics you can make cool interactive art

This example shows how to use the light sensor on the Circuit Playground Express to trigger a motor to rotate. With some audio effects it becomes a Force trainer, or a moving Theremin

Parts List



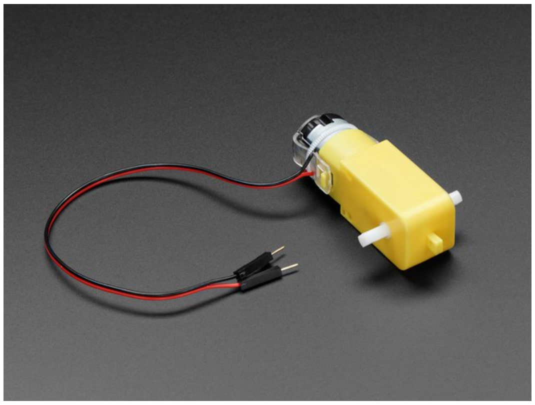
Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK
ADD TO CART

Your browser does not support the video tag.

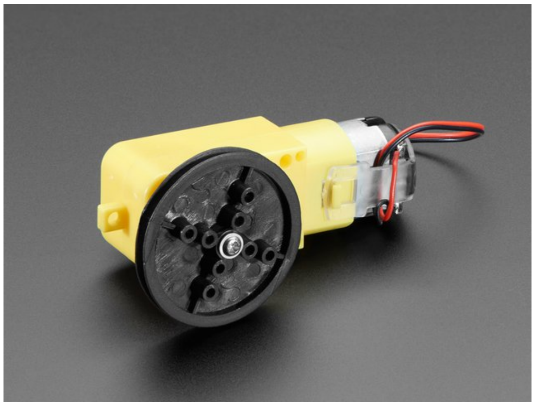
Circuit Playground Express

$24.95
IN STOCK
ADD TO CART



DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC

$2.95
IN STOCK
ADD TO CART



[TT Motor Pulley - 36mm Diameter](https://www.adafruit.com/product/3766)

$0.75

IN STOCK

ADD TO CART

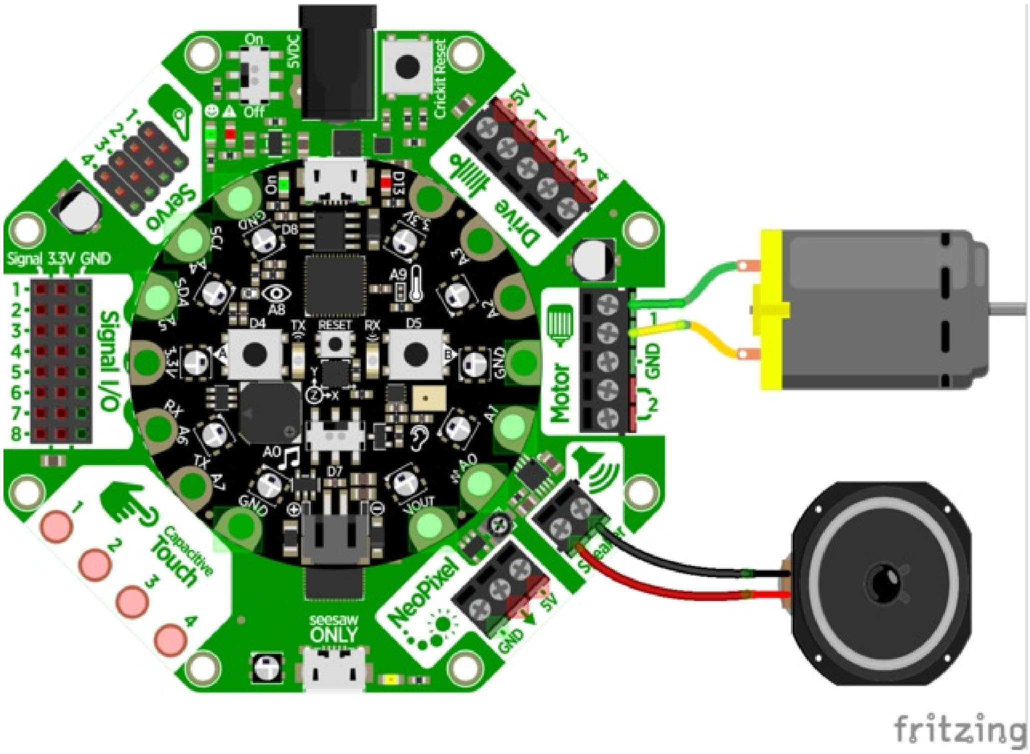
[Mono Enclosed Speaker - 3W 4 Ohm](https://www.adafruit.com/product/3351)

$3.95

IN STOCK

ADD TO CART

Wiring



CircuitPython Code For "Force Wave" demo

This project is pretty simple, it looks to see when the light sensor is shaded by your hand and changes the motor from running to off or vice versa.

```
import time
from busio import I2C
import analogio
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import motor
import board

light = analogio.AnalogIn(board.LIGHT)

print("Wave on/off to turn")

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# Create one motor on seesaw PWM pins 22 & 23
motor_a = motor.DCMotor(PWMOut(seesaw, 22), PWMOut(seesaw, 23))
motor_a.throttle = 0  # motor is stopped

while True:
    print((light.value,))
    # light value drops when a hand passes over
    if light.value < 4000:
        if motor_a.throttle:
            motor_a.throttle = 0
        else:
            motor_a.throttle = 1  # full speed forward

    while light.value < 5000:
        # wait till hand passes over completely
        pass
    time.sleep(0.1)
```

CircuitPython Code For "Theremin" demo

We can adapt the code above to speed up or slow down the motor based on how far our hand is. The darker the sensor, the faster the motor spins!

```
import time
from busio import I2C
import analogio
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import motor
import board

light = analogio.AnalogIn(board.LIGHT)


print("Theramin-like turning")

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# Create one motor on seesaw PWM pins 22 & 23
motor_a = motor.DCMotor(PWMOut(seesaw, 22), PWMOut(seesaw, 23))
motor_a.throttle = 0 # motor is stopped

def map_range(x, in_min, in_max, out_min, out_max):
    # Maps a number from one range to another.
    mapped = (x-in_min) * (out_max - out_min) / (in_max-in_min) + out_min
    if out_min <= out_max:
        return max(min(mapped, out_max), out_min)
    return min(max(mapped, out_max), out_min)

while True:
    print((light.value,))
    motor_a.throttle = map_range(light.value, 500, 8000, 1.0, 0)
    time.sleep(0.1)
```

CPX-1701

> This is the adventure of the United Space Ship *CircuitPlayground*. Assigned a five year galaxy patrol, the bold crew of the giant starship explores the excitement of strange new worlds, uncharted civilizations, and exotic code. These are its voyages and its adventures.

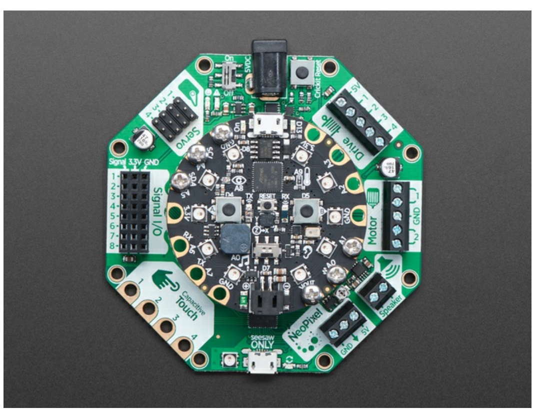
Explore exciting new modes of propulsion by creating a *really big* vibrating motor. This Crickit project attaches a bunch of CD's to an up-cycled CD-ROM motor for a cool looking warp drive. Some popsicle sticks, NeoPixels and sound effects complete the space craft and it's now ready for your command, captain!

Parts List

Your browser does not support the video tag.

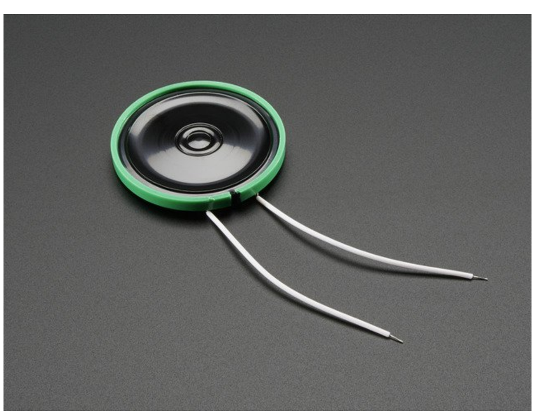
Circuit Playground Express

$24.95
IN STOCK
ADD TO CART



Adafruit CRICKIT for Circuit Playground Express

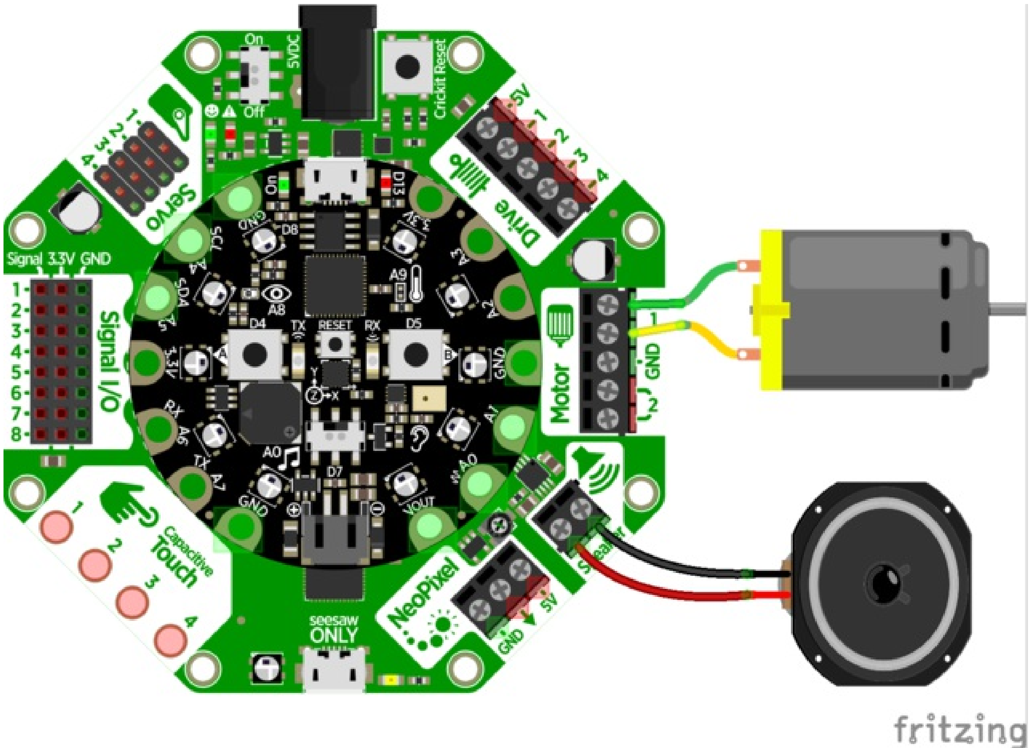
$29.95
IN STOCK
ADD TO CART



Thin Plastic Speaker w/Wires - 8 ohm 0.25W

$1.75
IN STOCK
ADD TO CART

Wiring Diagram



CircuitPython Code

This project is pretty simple, it plays some audio clips, and then lights up the built in NeoPixels and powers up the motor in time with the effects.

```
import time
from busio import I2C
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import motor
import neopixel
import audioio
import board

print("The voyages of the CPX-1701!")

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# Create one motor on seesaw PWM pins 22 & 23
motor_a = motor.DCMotor(PWMOut(seesaw, 22), PWMOut(seesaw, 23))

# audio output
cpx_audio = audioio.AudioOut(board.A0)

# neopixels!
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=1)
pixels.fill((0, 0, 0))
```

```
pixels.fill((0, 0, 0))

# give me a second before starting
time.sleep(1)

motor_a.throttle = 0  # warp drive off

f = open("01space.wav", "rb")
wav = audioio.WaveFile(f)
cpx_audio.play(wav)

t = time.monotonic()  # take a timestamp

# slowly power up the dilithium crystals
for i in range(50):
    pixels.fill((0, 0, i))
    time.sleep(0.05)

# 6 seconds after audio started...
while time.monotonic() - t < 6:
    pass

motor_a.throttle = 1  # full warp drive on!

# wait for music to end
while cpx_audio.playing:
    pass
f.close()

# play the warp drive and theme music!
f = open("02warp.wav", "rb")
wav = audioio.WaveFile(f)
cpx_audio.play(wav)

time.sleep(1)

# blast off!
pixels.fill((255, 0, 0))

# pulse the warp core
while True:
    for i in range(255, 0, -5):
        pixels.fill((i, 0, 0))
    for i in range(0, 255, 5):
        pixels.fill((i, 0, 0))

# wait for music to end
while cpx_audio.playing:
    pass
f.close()
```

Mag Neat-o

We picked up a magnetic foam shape kit, to make a fridge-mounted marble run. But picking up the marble after each run is such a *drag* - wouldn't it be fun if you could use your Crickit to help lift the ball back up and re-start the marble run?

With an electromagnet, we can pick up the stainless steel balls. A DC motor acts as a pulley, and a servo helps align the electromagnet so it can navigate around the foam.

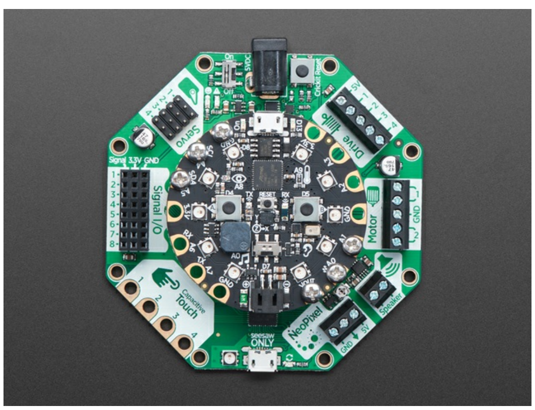
You can DIY, as we did, using the two Circuit Playground Express buttons and switch to control the motors - or you could even automate the whole thing!

Parts List

Your browser does not support the video tag.

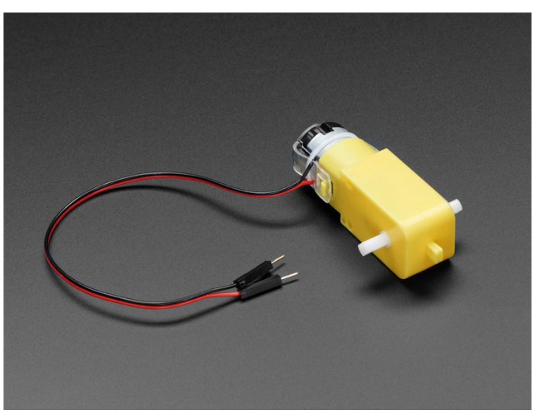
Circuit Playground Express

$24.95
IN STOCK
ADD TO CART



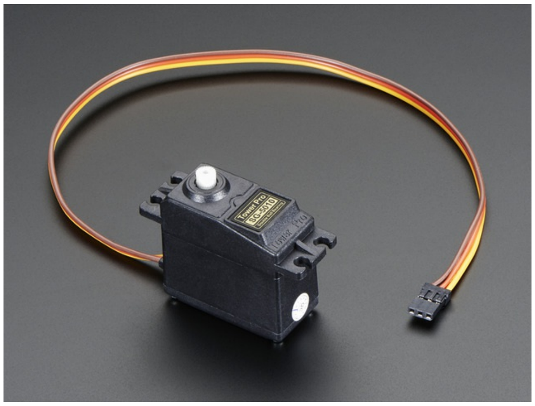
Adafruit CRICKIT for Circuit Playground Express

$29.95
IN STOCK
ADD TO CART



DC Gearbox Motor - "TT Motor" - 200RPM - 3 to 6VDC

$2.95
IN STOCK
ADD TO CART



[Standard servo - TowerPro SG-5010]

$12.00

IN STOCK

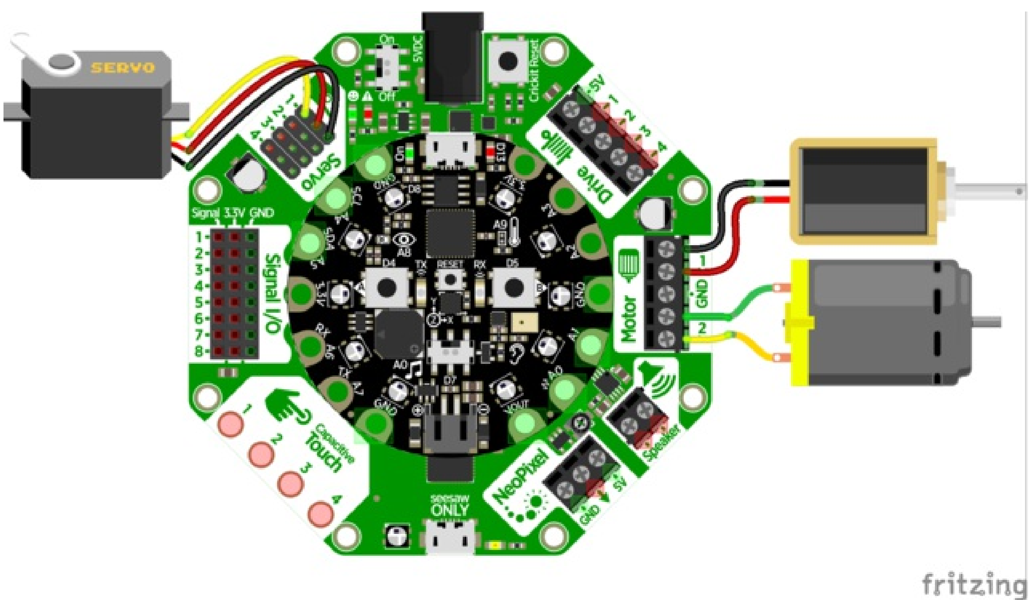
ADD TO CART

1 x 5V Electromagnet

Use a "50N" one for good pick up ability!

Wiring Diagram

Even though an electromagnet doesn't have 'direction' and thus could be controlled by a Drive pin, we opted for a Motor port because these electromagnets can draw up to 700mA and that's more than the Drive pin. But, you could almost certainly get away with using a Drive pin if you like!



Code!

Save to your CPX as **code.py** and press the CPX buttons to move the pulley up and down. Capacitive touch pads #1 and #4 twist the servo and then the switch enables and disables the electromagnet.

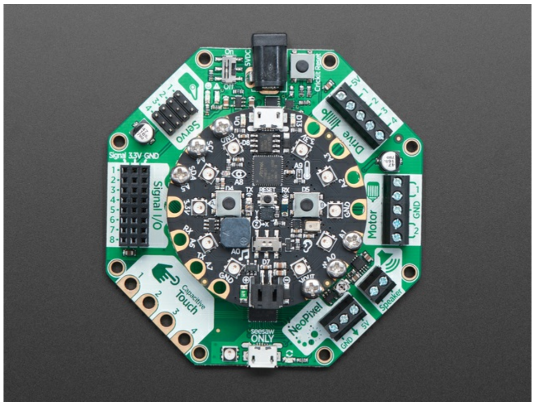
The most interesting part is `smooth_move` which is our gentle servo movement helper, it will carefully move the servo rather than jostling it and the magnet which would possibly cause the balls to fall.

```
import time
from busio import I2C
from adafruit_seesaw.seesaw import Seesaw
```

```
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import motor, servo
from digitalio import DigitalInOut, Direction, Pull
import board

print("Mag Neat-o!")

# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)

# Create one motor on seesaw PWM pins 22 & 23
motor_a = motor.DCMotor(PWMOut(seesaw, 22), PWMOut(seesaw, 23))
# Create another motor on seesaw PWM pins 19 & 18
motor_b = motor.DCMotor(PWMOut(seesaw, 19), PWMOut(seesaw, 18))

# Create servo object
pwm = PWMOut(seesaw, 17)     # Servo 1 is on s.s. pin 17
pwm.frequency = 50           # Servos like 50 Hz signals
my_servo = servo.Servo(pwm)  # Create my_servo with pwm signa
my_servo.angle = 90

def smooth_move(start, stop, num_steps):
    return [(start + (stop-start)*i/num_steps) for i in range(num_steps)]

buttona = DigitalInOut(board.BUTTON_A)
buttona.direction = Direction.INPUT
buttona.pull = Pull.DOWN

buttonb = DigitalInOut(board.BUTTON_B)
buttonb.direction = Direction.INPUT
buttonb.pull = Pull.DOWN

switch = DigitalInOut(board.SLIDE_SWITCH)
switch.direction = Direction.INPUT
switch.pull = Pull.UP

last_buttona = buttona.value
last_buttonb = buttonb.value
last_switch = switch.value

last_touch1 = False
last_touch4 = False


while True:
    touch_vals = (seesaw.touch_read(0), seesaw.touch_read(3))
    # print(touch_vals)

    touch1 = False
    if seesaw.touch_read(0) > 500:
        touch1 = True

    if touch1 != last_touch1:
        if touch1:
            start_angle = my_servo.angle
            end_angle = start_angle - 20
            end_angle = max(0, min(end_angle, 180))
            print("left from", start_angle, "to", end_angle)
            for a in smooth_move(start_angle, end_angle, 25):
```

```
            my_servo.angle = a
            time.sleep(0.03)
        last_touch1 = touch1

    touch4 = False
    if seesaw.touch_read(3) > 500:
        touch4 = True

    if touch4 != last_touch4:
        if touch4:
            start_angle = my_servo.angle
            end_angle = start_angle + 20
            end_angle = max(0, min(end_angle, 180))
            print("right from", start_angle, "to", end_angle)
            for a in smooth_move(start_angle, end_angle, 25):
                my_servo.angle = a
                time.sleep(0.03)
        last_touch4 = touch4

    if buttona.value != last_buttona:
        if buttona.value:
            print("down")
            if motor_a.throttle:
                print("haulin!")
                motor_b.throttle = -0.1
            else:
                motor_b.throttle = -0.1
        else:
            motor_b.throttle = 0
        last_buttona = buttona.value

    if buttonb.value != last_buttonb:
        if buttonb.value:
            print("up")
            if motor_a.throttle:
                print("haulin!")
                motor_b.throttle = 0.4
            else:
                motor_b.throttle = 0.3
        else:
            motor_b.throttle = 0
        last_buttonb = buttonb.value

    if switch.value != last_switch:
        motor_a.throttle = switch.value
    if motor_a.throttle:
        print("GRAB")
    else:
        print("RELEASE")
        last_switch = switch.value

    time.sleep(0.01)
```

Parts List



Adafruit CRICKIT for Circuit Playground Express

$29.95

IN STOCK

ADD TO CART

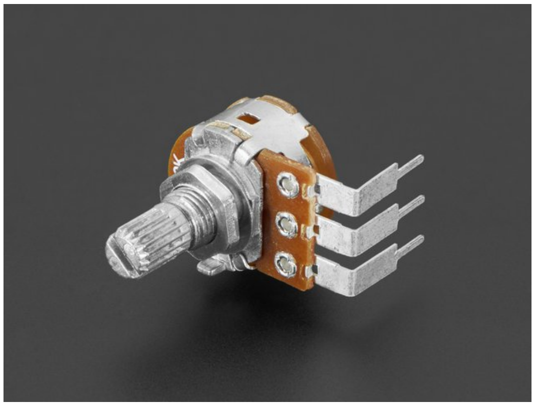
Your browser does not support the video tag.

Circuit Playground Express

$24.95

IN STOCK

ADD TO CART



Panel Mount Right Angle 10K Linear Potentiometer w/On-Off Switch

$1.50

IN STOCK

ADD TO CART

[Solid Machined Metal Knob - 1" Diameter](https://www.adafruit.com)

$3.95
IN STOCK
ADD TO CART

Mono Enclosed Speaker - 3W 4 Ohm

$3.95
IN STOCK
ADD TO CART

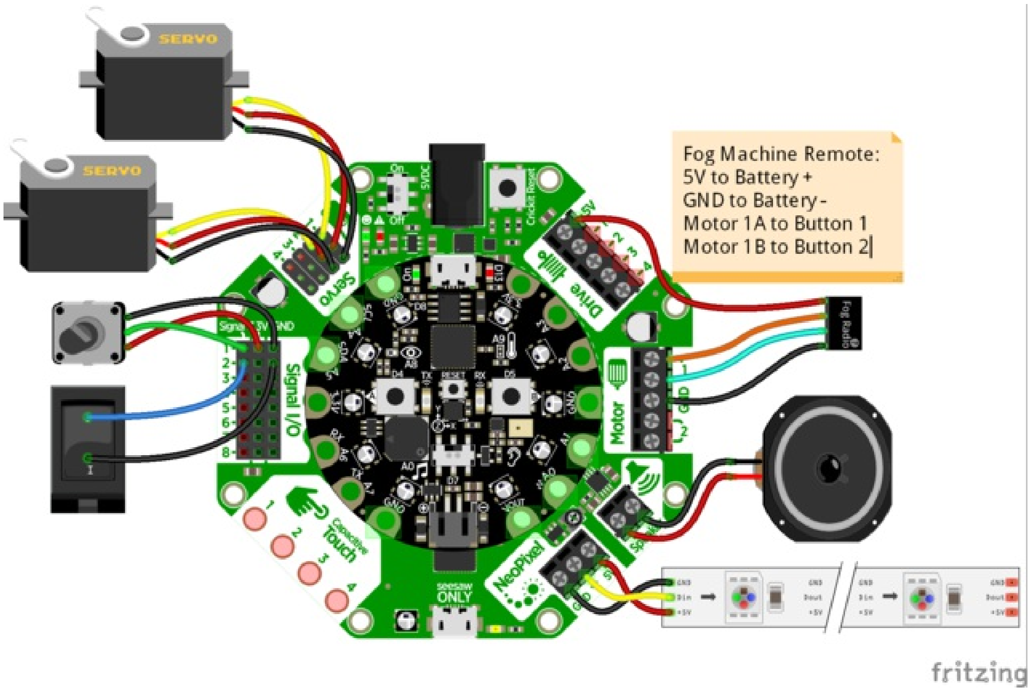
Adafruit NeoPixel LED Strand 20 LED 4" Pitch

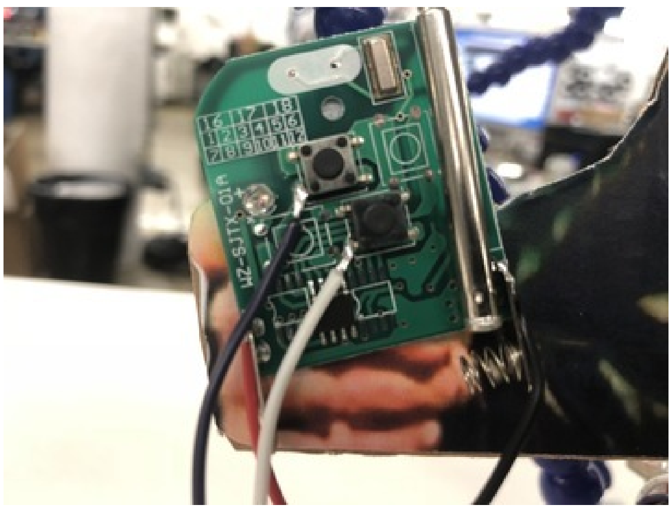
$27.50
IN STOCK
ADD TO CART

1 x Fog Machine with Remote
https://www.amazon.com/Virhuck-400-Watt-Portable-Halloween-Christmas/dp/B074WMWWS5

1 x Cow Bell
Craft store cowbell

Wiring Diagram





For the remote, we soldered four wires

- Black ground wire to the battery spring (negative) terminal
- Red +5V wire to the battery flat (positive) terminal
- White and Purple go to the 'switched' part of each switch, which, when connected to 5V activates that switch

CircuitPython Code

```
import time
import audioio
from digitalio import DigitalInOut, Pull, Direction
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.pwmout import PWMOut
from adafruit_motor import servo
from busio import I2C
import neopixel
import board


# Create seesaw object
i2c = I2C(board.SCL, board.SDA)
seesaw = Seesaw(i2c)
```

```
led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT

# Two onboard CPX buttons for FOG
buttona = DigitalInOut(board.BUTTON_A)
buttona.direction = Direction.INPUT
buttona.pull = Pull.DOWN

buttonb = DigitalInOut(board.BUTTON_B)
buttonb.direction = Direction.INPUT
buttonb.pull = Pull.DOWN


# Use the signal port for potentiometer w/switch
MORECOW = 2    # A switch on Signal #1
SWITCH = 3     # A potentiometer on Signal #2
# Add a pullup on the switch
seesaw.pin_mode(SWITCH, seesaw.INPUT_PULLUP)

# Servo angles
BELL_START = 60
BELL_END = 75
MOUTH_START = 95
MOUTH_END = 105

# Create servos list
servos = []
for ss_pin in (17, 16): #17 is labeled 1 on CRICKIT, 16 is labeled 2
    pwm = PWMOut(seesaw, ss_pin)
    pwm.frequency = 50 #must be 50 cannot change
    _servo = servo.Servo(pwm, min_pulse=400, max_pulse=2500)
    servos.append(_servo)
# Starting servo locations
servos[0].angle = BELL_START
servos[1].angle = MOUTH_START

# For the fog machine we actually use the PWM on the motor port cause it really needs 5V!
fog_off = PWMOut(seesaw, 22)
fog_off.duty_cycle = 0
fog_on = PWMOut(seesaw, 23)
fog_on.duty_cycle = 0


# Audio playback object and helper to play a full file
a = audioio.AudioOut(board.A0)
def play_file(wavfile):
    with open(wavfile, "rb") as file:
        wavf = audioio.WaveFile(file)
        a.play(wavf)
        while a.playing:
            servos[1].angle = MOUTH_START
            time.sleep(.2)
            servos[1].angle = MOUTH_END
            time.sleep(.2)

# NeoPixels for EYES
pixels = neopixel.NeoPixel(board.A1, 9, brightness=0.5)
pixels[8] = (255, 255, 0)
pixels[7] = (255, 255, 0)
```

```
# Maps a number from one range to another.
def map_range(x, in_min, in_max, out_min, out_max):
    mapped = (x-in_min) * (out_max - out_min) / (in_max-in_min) + out_min
    if out_min <= out_max:
        return max(min(mapped, out_max), out_min)
    return min(max(mapped, out_max), out_min)


# Wait before starting up
time.sleep(3)
play_file("i-gotta-have-more-cowbell.wav")
# a pause between audio clips
time.sleep(1)
play_file("only-prescription-more-cowbell.wav")

while seesaw.digital_read(SWITCH):
    pass

print("Ready for playing audio")
time.sleep(1)

f = open("fear11.wav", "rb")
wav = audioio.WaveFile(f)
a.play(wav)

while True:
    if seesaw.digital_read(SWITCH):
        break    # time to bail!
    pot = seesaw.analog_read(MORECOW)
    print(pot)
    eyecolor = (int(map_range(pot, 0, 1023, 255, 0)), int(map_range(pot, 0, 1023, 0, 255)), 0)
    pixels[8] = eyecolor
    pixels[7] = eyecolor

    if buttonb.value:
        fog_on.duty_cycle = 65535
    else:
        fog_on.duty_cycle = 0

    if buttona.value:
        fog_off.duty_cycle = 65535
    else:
        fog_off.duty_cycle = 0

    if pot < 200:  # wait for a bit before we start
        continue
    delay = map_range(pot, 200, 1023, 1.0, 0.1)
    servos[0].angle = BELL_END
    time.sleep(0.1)
    servos[0].angle = BELL_START
    time.sleep(delay)

a.stop()
f.close()

# Fog machine test
fog_off.duty_cycle = 65535
```

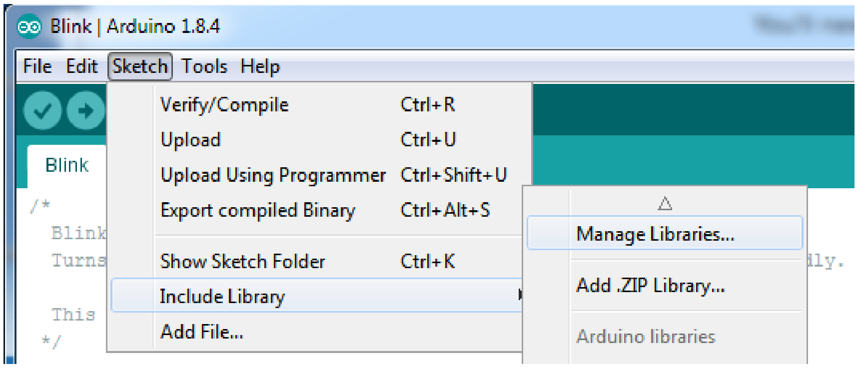
```
fog_on.duty_cycle = 0
time.sleep(0.1)
fog_off.duty_cycle = 0

pixels[8] = (255, 255, 0)
pixels[7] = (255, 255, 0)
time.sleep(1.5)
play_file("i-coulda-used-more-cow-bell.wav")
```

Arduino Code

Download Adafruit_Seesaw library

To begin using your Crickit with the Arduino IDE, you will need to install the Adafruit_seesaw library (https://adafru.it/BkT).

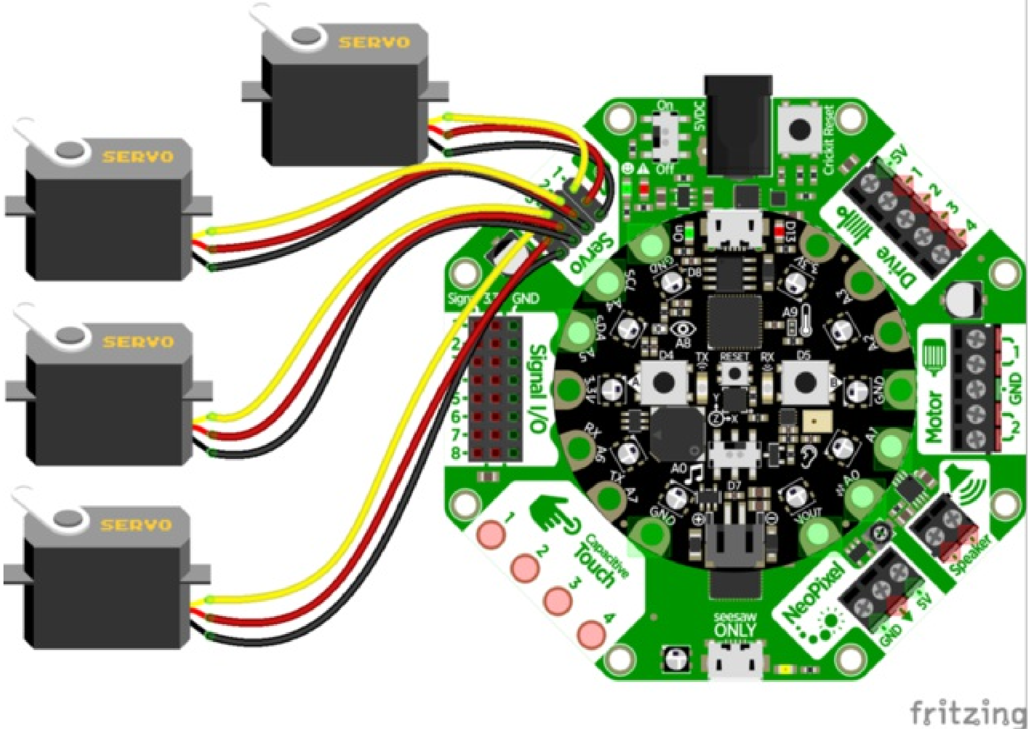
Start up the IDE and open the Library Manager:



Type in **seesaw** until you see the Adafruit Library pop up. 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)

Arduino Servos



Test Servos

Lets start by controlling some servos. You'll want at least one servo to plug in and test out the servo code. Visit our recommended servo page to check that you have a servo that works (https://adafru.it/Bfo). Once you do, plug in a servo into **SERVO #1** spot, making sure the yellow or white wire is next to the **1** text label.

This example will show rotating one servo from 0 to 180 degrees with a stop at 90 degrees.

```
#include "Adafruit_Crickit.h"
#include "seesaw_servo.h"

Adafruit_Crickit crickit;
seesaw_Servo myservo(&crickit);  // create servo object to control a servo

void setup() {
  Serial.begin(115200);

  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  myservo.attach(CRICKIT_SERVO1);  // attaches the servo to CRICKIT_SERVO1 pin
}

void loop() {
  myservo.write(0);
  delay(1000);
  myservo.write(90);
  delay(1000);
  myservo.write(180);
  delay(1000);
  myservo.write(90);
  delay(1000);
}
```

Are your servos not moving a full 180 degrees? Don't fret! This is normal, see below about min/max pulse lengths to 'tune' your servo!

More Servos!

OK that was fun but you want MORE servos right? You can control up to four! The servos are on the seesaw pins **17 (CIRCKIT_SERVO1)**, **16 (CIRCKIT_SERVO2)**, **15 (CIRCKIT_SERVO3)**, **14 (CIRCKIT_SERVO4)**

This example is similar to the 1 servo example, but instead of creating one `myservo` object, we'll make an array called `servos` that contains 4 servo objects. Then we can assign them using `servo[0].write(90);` or iterate through them as we do in the loop. You don't *have* to do it this way, but its very compact and doesn't take a lot of code lines to create all 4 servos at once!

```
#include "Adafruit_Crickit.h"
#include "seesaw_servo.h"

Adafruit_Crickit crickit;

#define NUM_SERVOS 4

//create an array of 4 servos with our crickit object
seesaw_Servo servos[] = { seesaw_Servo(&crickit),
                          seesaw_Servo(&crickit),
                          seesaw_Servo(&crickit),
                          seesaw_Servo(&crickit) };

//these are the pins they will be attached to
int servoPins[] = { CRICKIT_SERVO1, CRICKIT_SERVO2, CRICKIT_SERVO3, CRICKIT_SERVO4 };

void setup() {
  Serial.begin(115200);

  //begin the crickit
  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  //attach the servos to their pins
  for(int i=0; i<NUM_SERVOS; i++)
    servos[i].attach(servoPins[i]);  // attaches the servo to the pin
}

void loop() {
  //repeat for all 4 servos
  for(int i=0; i<NUM_SERVOS; i++){
    servos[i].write(0);
    delay(1000);
    servos[i].write(90);
    delay(1000);
    servos[i].write(180);
    delay(1000);
    servos[i].write(90);
    delay(1000);
  }
}
```

Min/Max Pulse control

In theory, servos should all use 1ms to 2ms long pulses, at 50 Hz to set the 0 and 180 degree locations. However, not all servos have their full range at those pulse widths. You can easily tweak your code to change the min and max pulse widths, which will let your servo turn more left and right. **But** don't set the widths too small/large or you can hit the hard stops of the servo which could damage it, so try tweaking the numbers slowly until you get a sense of what the limits are for your motor.

All you need to do is change the

`myservo.attach(CRICKIT_SERVO1);`

to, say,

`myservo.attach(CRICKIT_SERVO1, 750, 2250);`

Here we've change the minimum pulse from the default 1000 microseconds to 750, and the default maximum pulse from 2000 microseconds to 2250. Again, each servo differs. Some experimentation may be required!

Continuous Rotation Servos

If you're using continuous servos, you can use the angle assignments and just remember that 0 is rotating one way, 90 is 'stopped' and 180 and rotating the other way.

If your continuous servo doesn't stop once the script is finished you may need to tune the `min` and `max` pulse timings so that the center makes the servo stop. Or check if the servo has a center-adjustment screw you can tweak.

Disconnecting Servos or Custom Pulses

If you want to 'disconnect' the Servo by sending it 0-length pulses, you can do that by 'reaching in' and adjusting the underlying PWM duty cycle with:

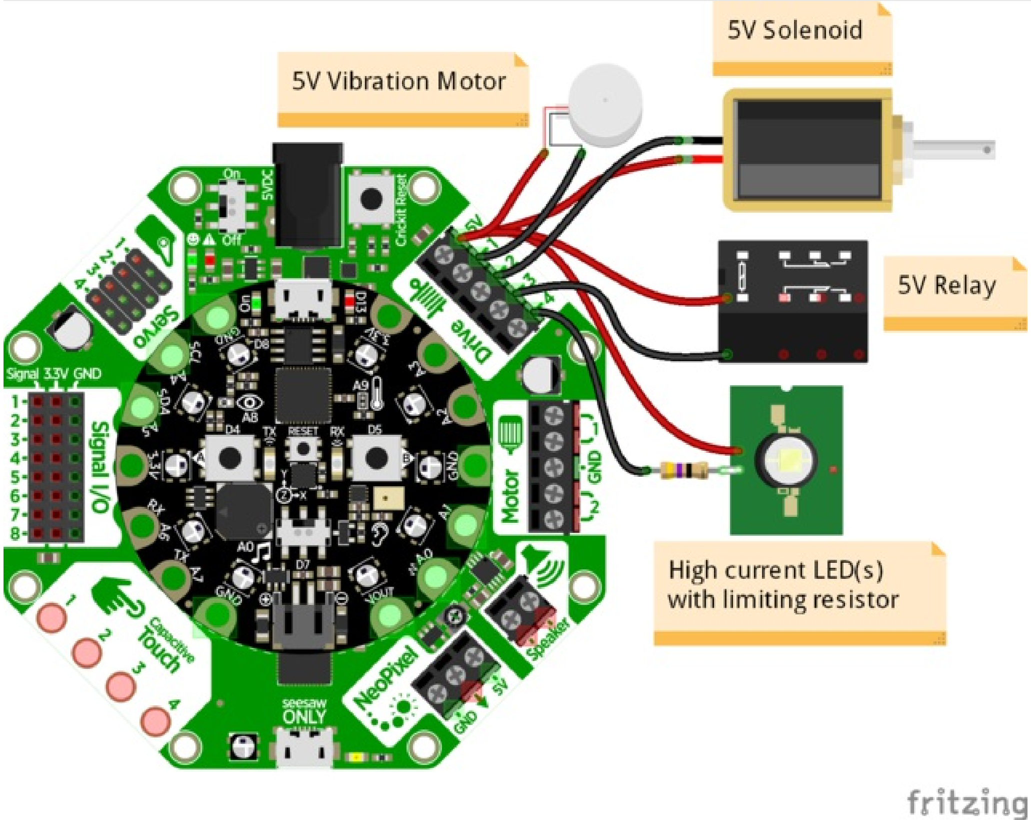
`myservo.writeMicroseconds(0);`

Likewise you can set the duty cycle to a custom value with

`myservo.writeMicroseconds(number);`

where *`number`* is the pulse length is microseconds between 0 (off) and 20000 (fully on). For example, setting it to 10000 will be 50% duty cycle, at the 50 Hz update rate

Arduino Drives



Test Drive

Lets start by controlling a drive output. You'll need to plug something into the **5V** and **DRIVE1** terminal blocks. I'm just using a simple LED with resistor but anything that can be powered by 5V will work.

- Note that the drive outputs cannot have 5V output so you must connect the **positive** pin of whatever you're driving to **5V**. Don't try connecting the positive pin to the drive, and the negative pin to **GND**, it wont work!
- Drive outputs are PWM-able!
- PWM values can be anywhere between **0** (0% duty cycle or always off) and **65535** (100% duty cycle or always on). A value of **32768** would be 50% duty cycle, or on for half of the period and then off for half the period.

This example will show turning the drive output fully on and off once a second. The macros CRICKIT_DUTY_CYCLE_OFF and CRICKIT_DUTY_CYCLE_MAX correspond to 0 and 65535 respectively and are used for readability:

```
#include "Adafruit_Crickit.h"

Adafruit_Crickit crickit;

void setup() {
  Serial.begin(115200);
  Serial.println("1 Drive demo!");

  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  //our default frequency is 1khz
  crickit.setPWMFreq(CRICKIT_DRIVE1, 1000);
}

void loop() {
  //turn all the way on
  crickit.analogWrite(CRICKIT_DRIVE1, CRICKIT_DUTY_CYCLE_OFF);
  delay(500);

  //turn all the way off
  crickit.analogWrite(CRICKIT_DRIVE1, CRICKIT_DUTY_CYCLE_MAX);
  delay(500);
}
```

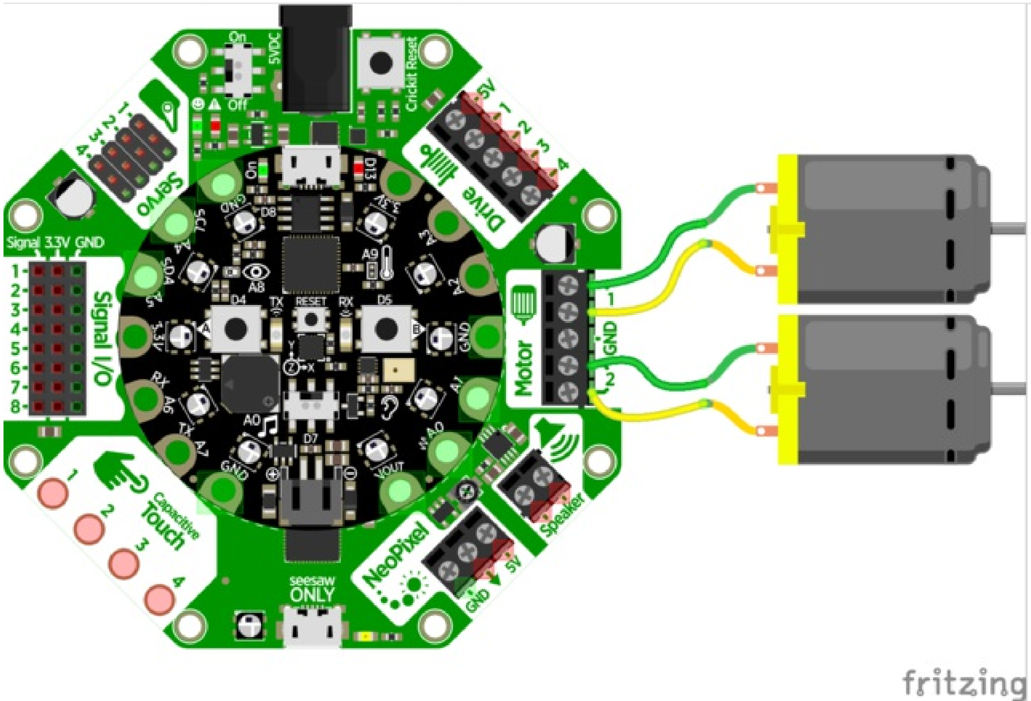
More Drivers!

OK that was fun but you want MORE drives right? You can control up to four! The four drive outputs are on the seesaw pins **13 (CRICKIT_DRIVE1)**, **12 (CRICKIT_DRIVE2)**, **43 (CRICKIT_DRIVE3)**, **42 (CRICKIT_DRIVE4)**

```
#include "Adafruit_Crickit.h"

Adafruit_Crickit crickit;

#define NUM_DRIVES 4
int drives[] = {CRICKIT_DRIVE1, CRICKIT_DRIVE2, CRICKIT_DRIVE3, CRICKIT_DRIVE4};

void setup() {
  Serial.begin(115200);
  Serial.println("4 Drive demo!");

  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  //our default frequency is 1khz
  for(int i=0; i<NUM_DRIVES; i++)
    crickit.setPWMFreq(drives[i], 1000);
}

void loop() {
  for(int i=0; i<NUM_DRIVES; i++){
    //turn all the way on
    crickit.analogWrite(drives[i], CRICKIT_DUTY_CYCLE_OFF);
    delay(100);

    //turn all the way off
    crickit.analogWrite(drives[i], CRICKIT_DUTY_CYCLE_MAX);
    delay(100);
  }
}
```

This example is similar to the 1 drive example, but instead of using just 1 PWM driver, we'll make an array called `drives` that contains the pin numbers of 4 PWM drivers. Then we can assign them using `crickit.analogWrite(drives[0], CRICKIT_DUTY_CYCLE_MAX);` or iterate through them as we do in the loop. You don't *have* to do it this way, but its very compact and doesn't take a lot of code lines to create all 4 drivers at once!

Arduino DC Motors



You can drive two separate DC motors, so lets go ahead and get right to it!

DC motors are controlled by 4 PWM output pins, the 4 PWM pins let you control speed *and* direction. And we'll use our **seesaw_Motor** library to help us manage the throttle (speed) and direction for us, making it very easy to control motors

Note that each DC motor is a little different, so just because you have two at the same throttle does not mean they'll rotate at the *exact* same speed! Some tweaking may be required

The two wires of the DC motor can be plugged in either way into each Crickit Motor port. If the motor spins the opposite way from what you want to call 'forward', just flip the wires!

```
#include "Adafruit_Crickit.h"
#include "seesaw_motor.h"

Adafruit_Crickit crickit;

seesaw_Motor motor_a(&crickit);
seesaw_Motor motor_b(&crickit);

void setup() {
  Serial.begin(115200);
  Serial.println("Dual motor demo!");

  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  //attach motor a
  motor_a.attach(CRICKIT_MOTOR_A1, CRICKIT_MOTOR_A2);

  //attach motor b
  motor_b.attach(CRICKIT_MOTOR_B1, CRICKIT_MOTOR_B2);
}

void loop() {
  motor_a.throttle(1);
  motor_b.throttle(-1);
  delay(1000);

  motor_a.throttle(.5);
  motor_b.throttle(-.5);
  delay(1000);

  motor_a.throttle(0);
  motor_b.throttle(0);
  delay(1000);

  motor_a.throttle(-.5);
  motor_b.throttle(.5);
  delay(1000);

  motor_a.throttle(-1);
  motor_b.throttle(1);
  delay(1000);

  motor_a.throttle(0);
  motor_b.throttle(0);
  delay(500);
}
```

Arduino Signals

Signals on the Crickit are available on the following pins:

Silkscreen Label	Arduino Macro	Seesaw Pin
1	CRICKIT_SIGNAL1	2
2	CRICKIT_SIGNAL2	3
3	CRICKIT_SIGNAL3	40
4	CRICKIT_SIGNAL4	41
5	CRICKIT_SIGNAL5	11
6	CRICKIT_SIGNAL6	10
7	CRICKIT_SIGNAL7	9
8	CRICKIT_SIGNAL8	8

You can use these as analog or digital I/O pins, setting the mode, value and reading with the seesaw library directly:

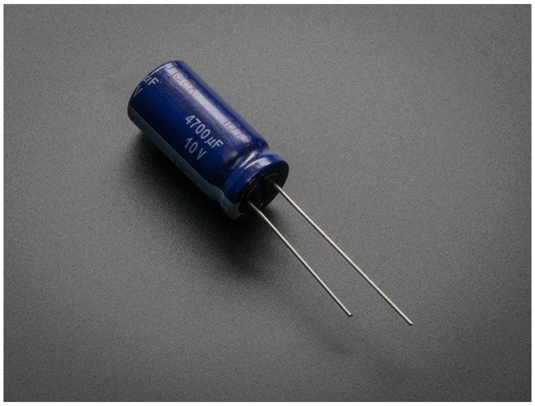
```
#include "Adafruit_Crickit.h"

Adafruit_Crickit crickit;

#define BUTTON_1 CRICKIT_SIGNAL1
#define BUTTON_2 CRICKIT_SIGNAL2
#define LED_1 CRICKIT_SIGNAL3
#define LED_2 CRICKIT_SIGNAL4

void setup() {
  Serial.begin(9600);

  if(!crickit.begin()){
    Serial.println("ERROR!");
    while(1);
  }
  else Serial.println("Crickit started");

  //Two buttons are pullups, connect to ground to activate
  crickit.pinMode(BUTTON_1, INPUT_PULLUP);
  crickit.pinMode(BUTTON_2, INPUT_PULLUP);

  // Two LEDs are outputs, on by default
  crickit.pinMode(LED_1, OUTPUT);
  crickit.pinMode(LED_2, OUTPUT);
  crickit.digitalWrite(LED_1, HIGH);
  crickit.digitalWrite(LED_2, HIGH);
}

void loop() {
  if(!crickit.digitalRead(BUTTON_1))
    crickit.digitalWrite(LED_1, HIGH);
  else
    crickit.digitalWrite(LED_1, LOW);

  if(!crickit.digitalRead(BUTTON_2))
    crickit.digitalWrite(LED_2, HIGH);
  else
    crickit.digitalWrite(LED_2, LOW);
}
```

Hacks & Upgrades

Brown Outs?

The power supply on the Crickit will let you draw 4 Amps at once, which is *a lot*. But perhaps you are turning on all the motors at once, causing the power supply to flicker? An extra large capacitor on the 5V and GND pads may help smooth out that power draw!

Use a large electrolytic capacitor, rated for 10V or higher. Even though the power supply is 5V, you may think you can use a 6.3V capacitor, but you want at least 2x the voltage rating if possible so stick to 10V!



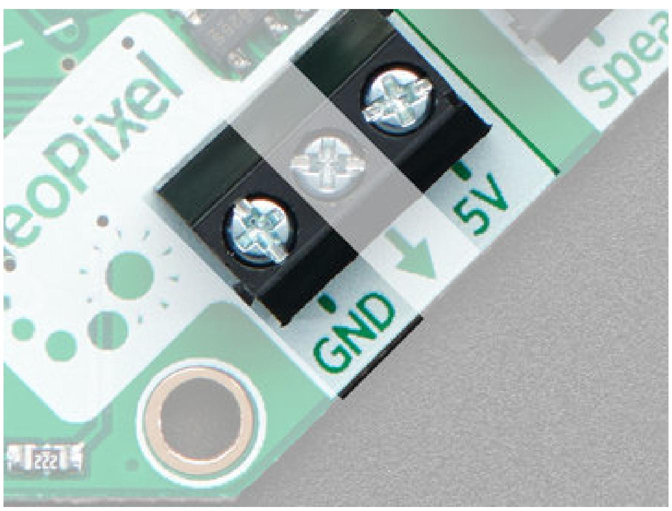
4700uF 10v Electrolytic Capacitor

$1.95

IN STOCK

ADD TO CART

Connect the capacitor using the **NeoPixel** terminal blocks. The **5V** and **GND** lines are shared across the board so even if its a DC motor or servo causing the issues, this will help! It's just the most convenient place to attach a large capacitor because the two terminal blocks are nicely spaced.



Connect the capacitor using the **NeoPixel** terminal blocks. The **5V** and **GND** lines are shared across the board so even if its a DC motor or servo causing the issues, this will help!

Connect the **Positive** (longer leg) to **5V** and the **Negative** (shorter leg) to **GND**

F.A.Q.

Why did you misspell "Cricket"?

We wanted a unique name, inspired by the original Cricket robotics platform from MIT (which then became the PicoCricket), but not with the exact same name!

Downloads

Files

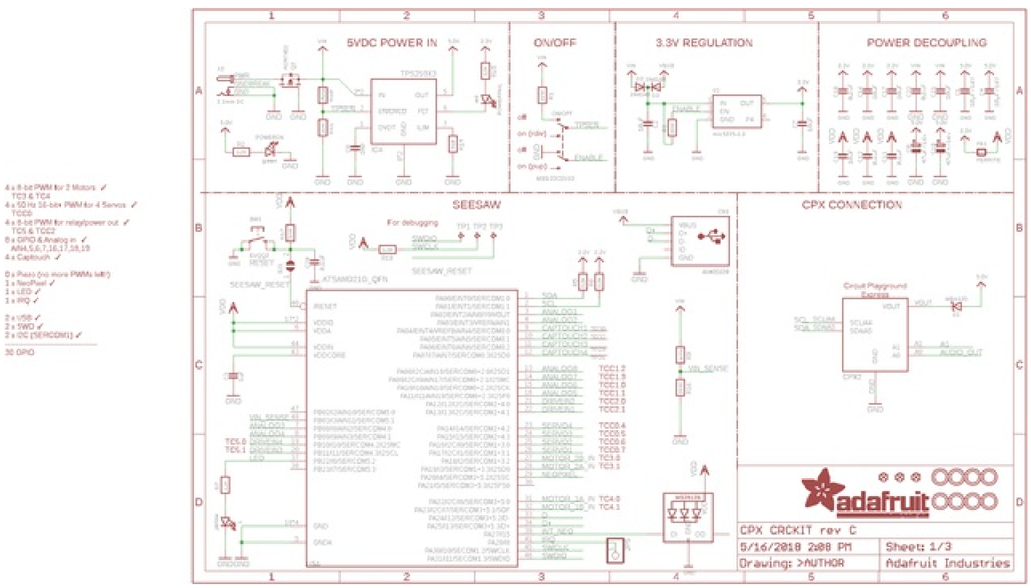
- PCB Files on GitHub (https://adafru.it/BEj)
- Fritzing objects in Adafruit Fritzing Library (https://adafru.it/aP3)

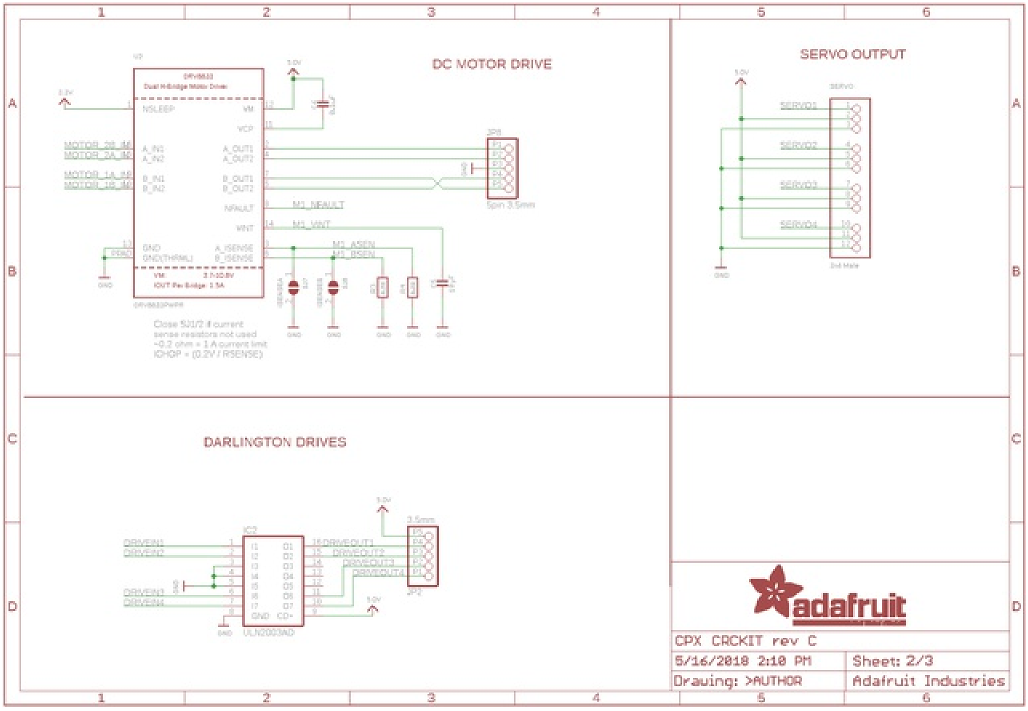
Datasheets

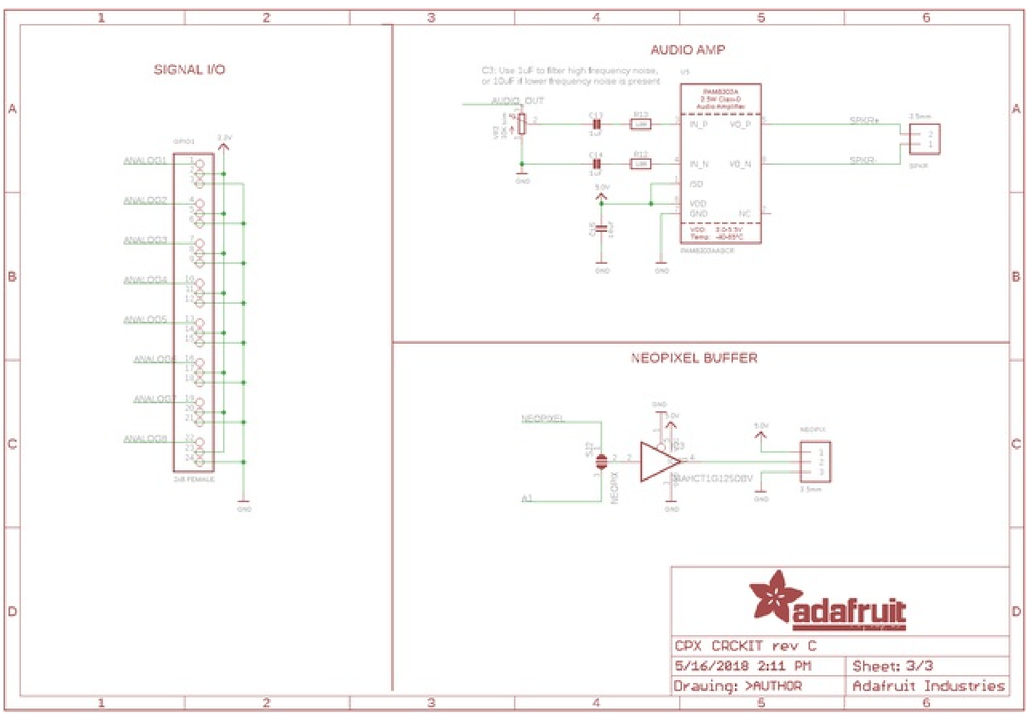
- TPS259573 eFuse power supply protection chip (https://adafru.it/Bfj)
- DRV8833 DC motor driver chip (https://adafru.it/Bfk)
- ULN2003A Darlington driver chip (https://adafru.it/Bfl)

Schematics

Click to embiggen




DC MOTOR DRIVE
SERVO OUTPUT
SERVO1
SERVO2
SERVO3
SERVO4
DARLINGTON DRIVES
ULN2003AD
CPX CRCKIT rev C
5/16/2018 2:10 PM
Sheet: 2/3
Drawing: >AUTHOR
Adafruit Industries


SIGNAL I/O
ANALOG1
ANALOG2
ANALOG3
ANALOG4
ANALOG5
ANALOG6
ANALOG7
ANALOG8
AUDIO AMP
AUDIO_OUT
SPKR+
SPKR-
NEOPIXEL BUFFER
NEOPIXEL
A1
CPX CRCKIT rev C
5/16/2018 2:11 PM
Sheet: 3/3
Drawing: >AUTHOR
Adafruit Industries