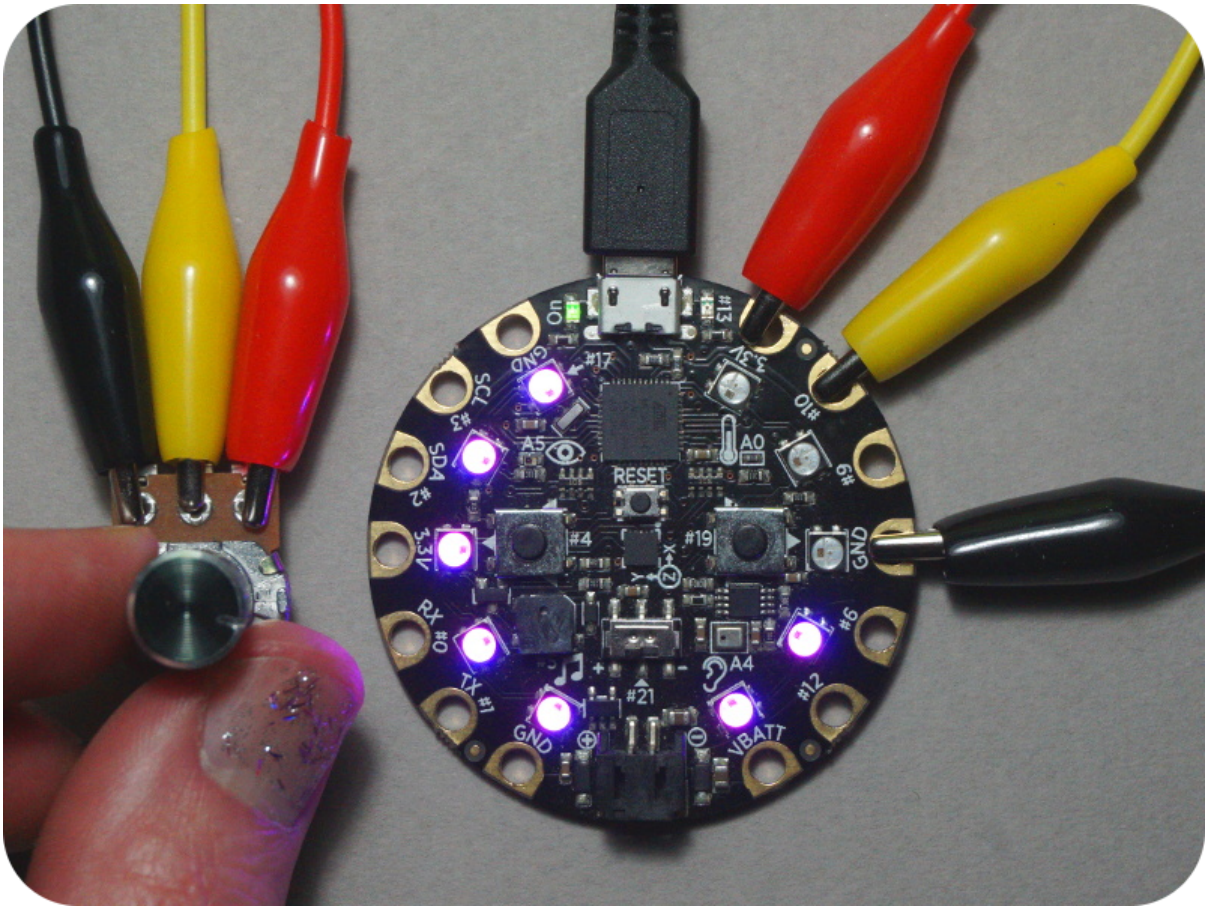
Circuit Playground Analog Input

Created by Carter Nelson

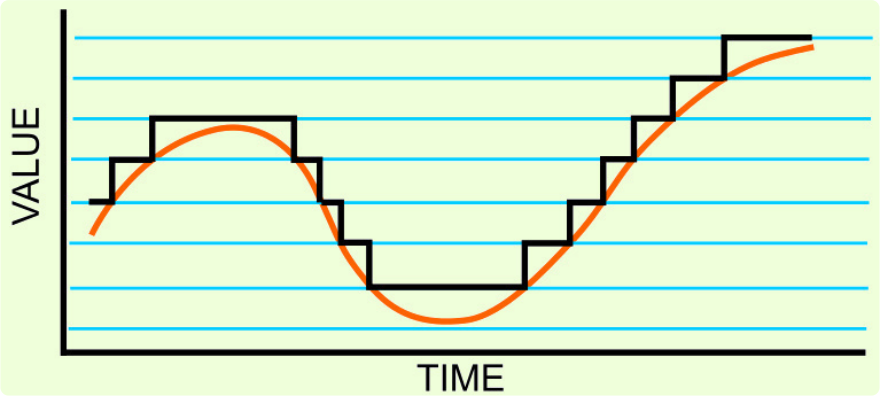


https://learn.adafruit.com/circuit-playground-analog-input

Last updated on 2024-06-03 02:04:42 PM EDT

Table of Contents

Overview

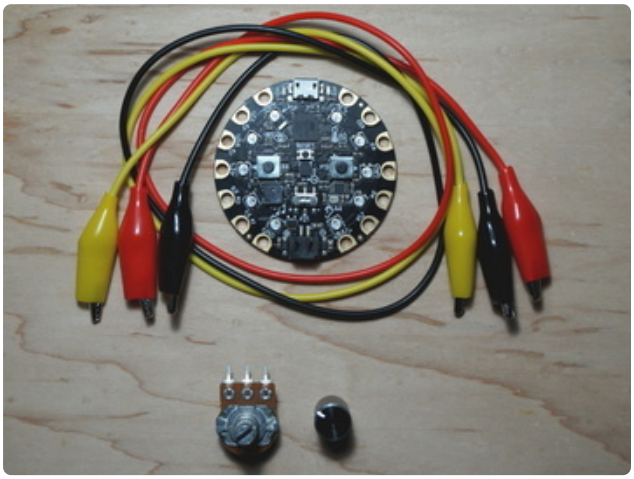


In this guide we will use our Circuit Playground along with a potentiometer and some alligator clips to explore how we can read analog signals into our digital computer.

No soldering required!

Required Parts

In addition to the Circuit Playground, you will need some alligator clips and a potentiometer. While not required, a knob makes turning the potentiometer easier and provides a nice reference mark to see where you are at.



Circuit Playground (http://adafru.it/3000)
Alligator Clip Test Leads (http://adafru.it/1008)
Panel Mount 10K Potentiometer (http://adafru.it/562)
Knob (http://adafru.it/2057)

Optional Parts

Another option is to use a breadboard (http://adafru.it/64) along with a trim pot (http://adafru.it/356) and alligator to male jumper wires (http://adafru.it/3255). The hardware outlined in the previous section is recommended, but if you already have these items, they will work as well.

Before Starting

If you are new to the Circuit Playground, you may want to first read these overview guides.

- Overview (https://adafru.it/ncG)
- Lesson #0 (https://adafru.it/rb4)

This project will use the Arduino IDE. Make sure you have added the board support for the Circuit Playground as well as installed the Circuit Playground library. **MUST DO BOTH.** This is covered in the guides linked above.

Analog vs. Digital

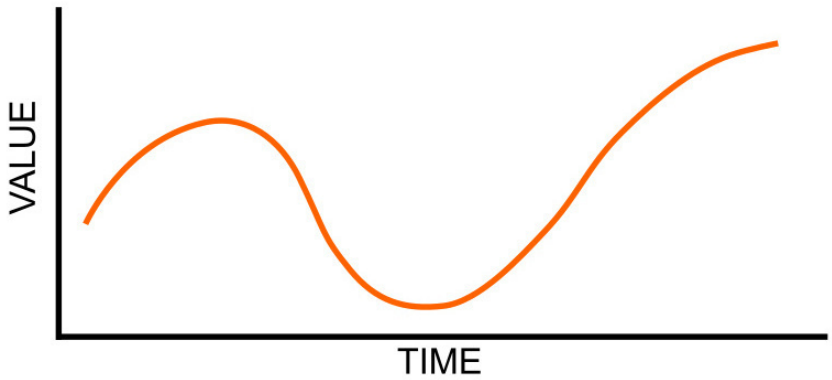
Analog World

Pretty much every phenomenon in the natural world is analog. What does "analog" mean? The key bits boil down to this phrase in the definition for analog (https://adafru.it/uen):

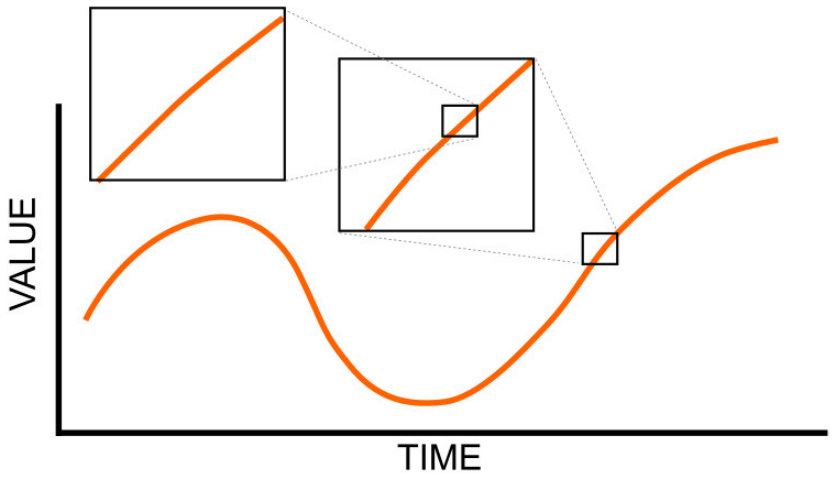
> ...continuously variable physical quantities...

The "physical quantity" can be anything - temperature of an apple, pressure in a balloon, water level of some lake, total internal energy of the third moon of Omicron Persei 8, voltage between two points in an electrical circuit, etc.

To understand the "continuously variable" part, consider the plot of some physical quantity below as it happily marches along in time changing its value.



If we were to zoom in on some part of the curve, we would again see a nice smooth variation. Zoom again, still the same. No matter how far we zoom in, we would see a nice smooth "continuous" variation in the value.



Hurray for nature! But what about digital computers, like our Circuit Playground? Let's look at that next.

Digital World

You've probably heard the phrase before, something like how everything in a computer is either a 1 or 0, either YES or NO, either ON or OFF, either HIGH or LOW, etc. This is entirely true for digital computers (yes, there are analog computers (https://adafru.it/ueo)), and the basic unit of information that stores that 1 or 0 is called a bit (https://adafru.it/uep). All information must somehow be represented by using one or more of these bits.

If we try to use just 1 bit, we don't get very far. It only allows us to represent two values.

1 BIT

PATTERN	NUMBER
0	0
1	1

By simply adding a second bit, we can now represent up to 4 different values.

2 BITS

PATTERN	NUMBER
00	0
01	1
10	2
11	3

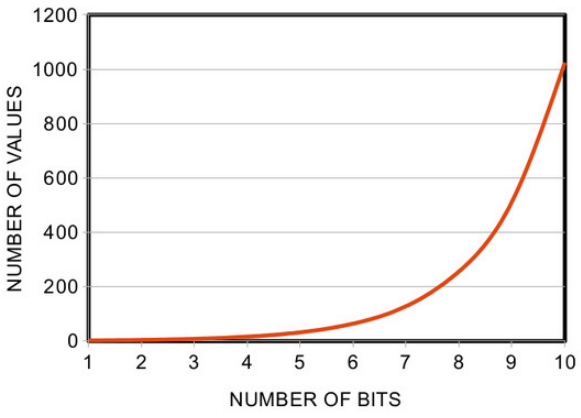
The more bits we add, the more values we can represent. What about 10 bits?

10 BITS

PATTERN	NUMBER
0000000000	0
0000000001	1
0000000010	2
0000000011	3
0000000100	4
0000000101	5
:	:
1111111111	1023

Here's a summary of how many values can be represented by using 1 to 10 bits. Note that the variation is not linear.

NUMBER OF BITS	NUMBER OF VALUES
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024



Analog World to Digital World

So how do we get those nice analog values into our digital computer so we can do cool things with the information? Simple - just use an Analog to Digital Converter (ADC). There's all kinds of technical mumbo jumbo for how an ADC works (https://adafru.it/eYp), but for this guide it's good enough to just think of an ADC as a device where an analog signal goes in and a digital (1s and 0s) signal comes out.

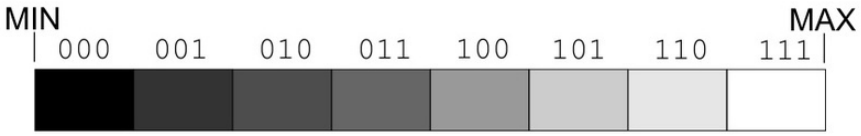
To illustrate this, let's start by considering the gray scale range shown below - this is our analog signal. It is **continuously variable** (remember that from above?) from some MIN value up to some MAX value.

An ADC will take that and chop it up into discrete (digital) segments. ADCs come in different resolutions - the total number of bits they use. That's why you here people say things like "it has a 10bit ADC". The more bits, the more segments, and the better the ADC can represent (resolve) the analog values.

A 1 bit ADC would be pretty crude. All that beautiful grayness would just become black or white.

Let's add one more bit and see what happens. Here's what a 2 bit ADC would do.

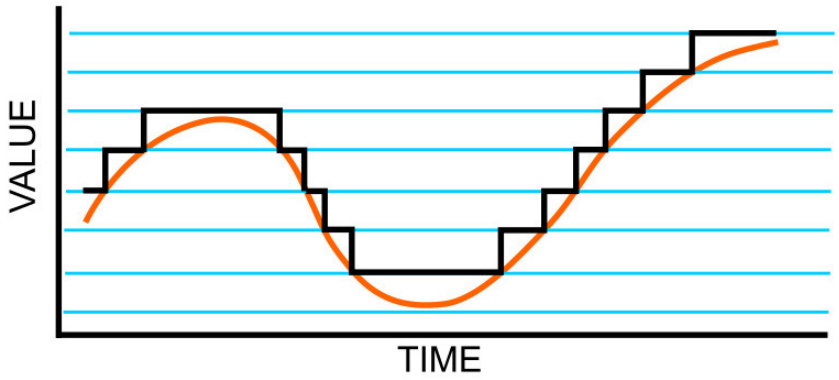
That's much better. We can start to see some grayness. But it's still pretty crude. Let's add one more bit. Here's what a 3 bit ADC would do.



And that's even better.

You can see the trend here. The more bits we use, the closer we get to the original analog signal. If we could use an infinite number of bits, we'd actually get the original signal. But that's just not practical. No one's come up with a way store an infinite number of 1s and 0s on a digital computer.

If we were to feed the signal we introduced at the beginning into a 3 bit ADC, it would get turned into something like shown below. The orange (analog) signal becomes the black (digital) signal.



How Many Bits?

So why not just use the biggest most bit-est ADC there is for everything? The answer is cost. The ADC hardware will generally cost more, but there is also the consideration of storage cost. Even if money didn't matter and you could buy a 900000000bit ADC, do you have a place to store all those 1 and 0s? The speed of the ADC conversion is another consideration. Hey. Trade offs.

So you wave your engineering magic wand and pick the ADC that is best suited for your application. Or, you just work with what you've been given. In the case of the Circuit Playground, that's a 10bit ADC.

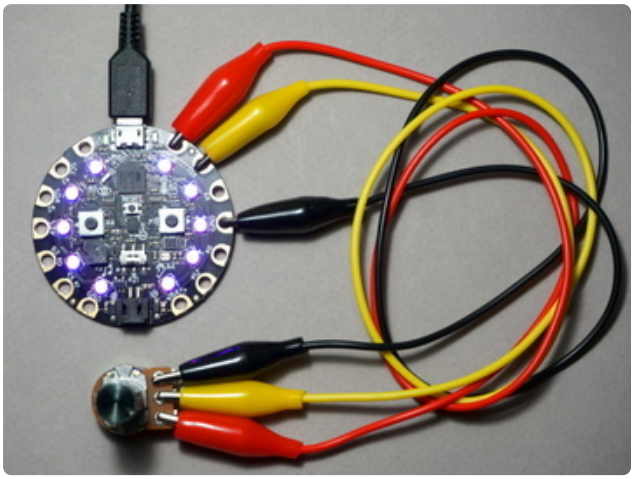
Circuit Hookup

Alright. Fun time.

In order to create an analog signal we can control, we will connect a potentiometer to the Circuit Playground. By connecting the ends of the potentiometer to the 3.3V and GND there will be a variable voltage on the middle (sweep) pin.

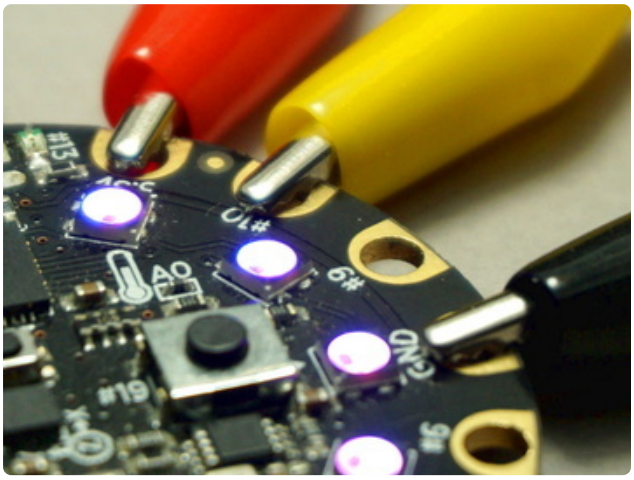
This is called a 'voltage divider'.

Using Panel Mount Potentiometer



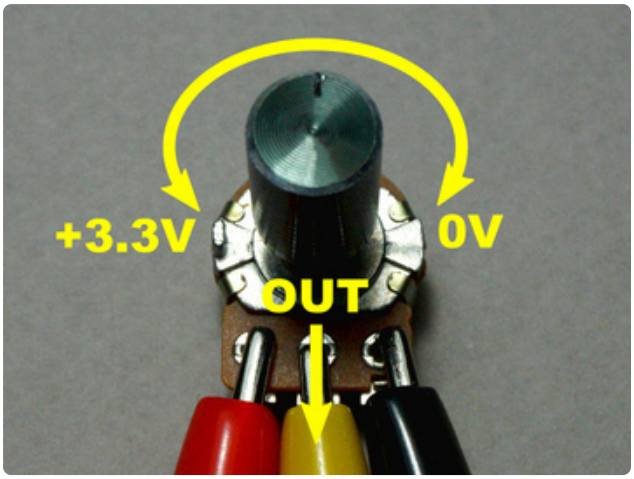
This is the overall view of the hookup.

RED to 3.3V
YELLOW to #10
BLACK to GND



Simply connect the alligator clips to the gold pads on the edge of the Circuit Playground as shown.

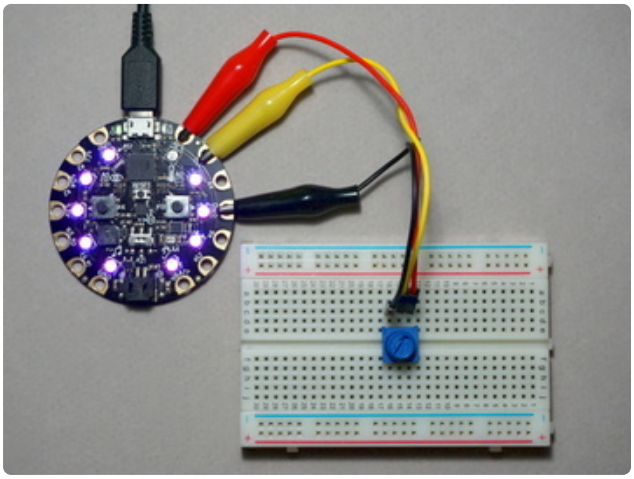
The alligator clips should clamp nicely onto the leads of the potentiometer as shown.



With this hookup, the voltage on the yellow (OUT) will vary with knob position as shown.

Using Breadboard Trim Pot

The test circuit can also be setup on a breadboard as shown below. You can go this route if you want, like if you already have some of these parts, but the knob on the panel mount potentiometer is bigger and easier to work with.



This is the overall view of the hookup.

RED to 3.3V
YELLOW to #10
BLACK to GND

Hello Analog

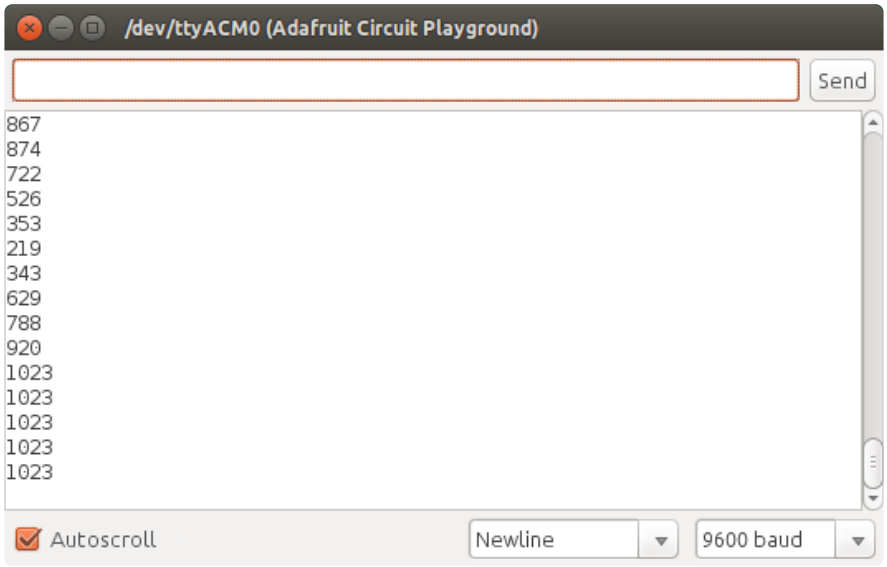
To read the value from the analog input, we will use the good 'ole Arduino library function analogRead() (https://adafru.it/ueq). You can use the sketch below to read the value from the potentiometer (attached to #10) and print it on the serial output.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Analog In - Serial Print
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

uint16_t value;

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  value = analogRead(10);
  Serial.println(value);
  delay(100);
}
```

With this sketch running on the Circuit Playground, open the Serial Monitor:

Tools -> Serial Monitor

You should see the value being printed out. Now try spinning the knob and you should see the value change.

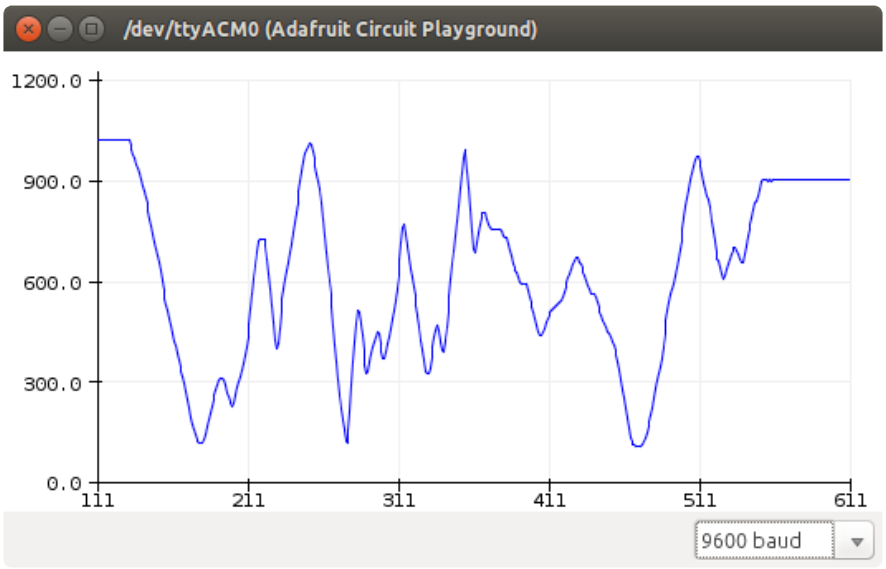


If you turn the knob all the way in one direction, you should get a value of 0. If you turn the knob all the way in the other direction, you should get a value of 1023. This is the range of values provided by the 10 bit analog to digital converter.

You can also use the Serial Plotter to watch the value change.

Tools -> Serial Plotter

Now try turning the knob and watch the line go up and down. Wheeeee!!!!



NeoPixel Fun

Now let's take the analog value and do something fun with it. How about turning on the NeoPixels to represent knob position? Like all NeoPixels off is one direction, and all NeoPixels on is the other direction.

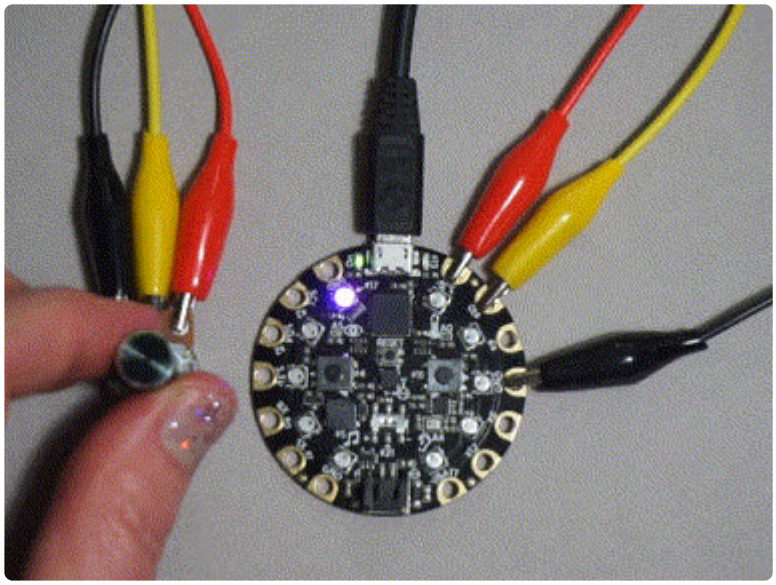
We can do this pretty easy using the map (https://adafru.it/tbc) function. We know the range of values we expect from the analog input, 0-1023, and we just want to map that to how many NeoPixels to light up, 0-10. So, it's just a matter of coding it up.

And here it is.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Analog In - NeoPixel Fun
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;

uint16_t value;
uint8_t pixels;
```

```
////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  value = analogRead(10);
  pixels = map(value, 0, 1023, 0, 10);

  CircuitPlayground.clearPixels();
  for (int p=0; p&lt;pixels; p++) {
    CircuitPlayground.setPixelColor(p, 0xFF00FF);
  }
  delay(100);
}
```

With this code loaded and running on the Circuit Playground, try turning the knob. You should see the NeoPixels light up in accordance with the knob position.



Speaker Fun

Oooooo. What about having the knob change the tone of the speaker? No prob. Same idea as the previous sketch. Just map the knob values to tone frequencies.

The Circuit Playground speaker is really only usable over a range of about 100Hz to 10000Hz, which leads us to the following code.

```
////////////////////////////////////////////////////////////////////////////
// Circuit Playground Analog In - Speaker Fun ?
//
// Author: Carter Nelson
// MIT License (https://opensource.org/licenses/MIT)

#include &lt;Adafruit_CircuitPlayground.h&gt;
```

```
uint16_t value;
uint16_t freq;

////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(9600);
  CircuitPlayground.begin();
}

////////////////////////////////////////////////////////////////////////////
void loop() {
  value = analogRead(10);
  freq = map(value, 0, 1023, 100, 10000);

  CircuitPlayground.playTone(freq, 100, false);
}
```

Load that code up and run it. Spin the knob. Make some weird outer space theremin (https://adafru.it/pFU) like music sounds. Hopefully the dogs don't start howling.

Under The Hood

So you might have noticed we didn't use one of those `CircuitPlayground.something()` functions to get our analog value. Why not, and how did it work without doing so?

The answer to the second question is pretty easy - we just use the Arduino library. The Circuit Playground library is built (largely) on top of the Arduino library and 'hides' the details of using its various bells and whistles. However, you can still use the Arduino library if you want (and know what you're doing).

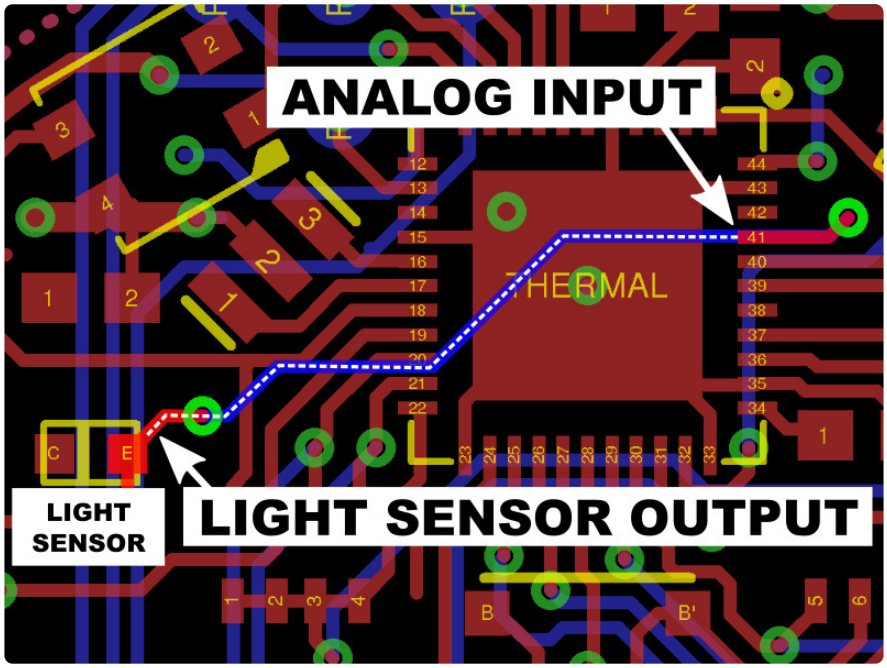
No real good answer to the other question. This could've been done, say with something like `CircuitPlayground.analogRead()` or `CircuitPlayground.giveMeTheAnalogPlease()` or whatever. But this would have likely done nothing more than just call the same Arduino function. So, guess it just never made the cut.

However, there are a few of the Circuit Playground sensors that are essentially the same as the potentiometer we used in this guide, namely the light sensor (https://adafru.it/qHe) , sound sensor (https://adafru.it/uer) , and temperature sensor (https://adafru.it/scy). They all output a voltage that varies with the physical quantity they sense. In order for this to be used in the Circuit Playground, those signals are fed into ADCs, just like we did with the potentiometer. And that's pretty much all there is to it.

Don't believe me? Well, take a look...

Light Sensor

Here's the circuit path for the output of the light sensor. The big red square that says 'THERMAL' is the brain of the Circuit Playground (https://adafru.it/ues).

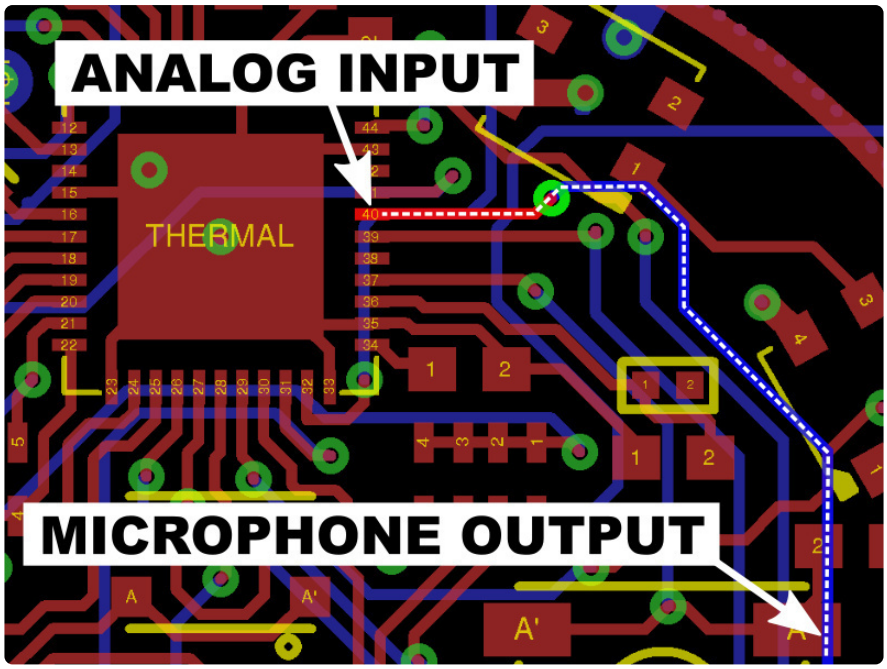


And here's the code for the `CircuitPlayground.lightSensor()` function. (direct link to repo (https://adafru.it/uet))

```
uint16_t Adafruit_CircuitPlayground::lightSensor(void) {
  return analogRead(CPLAY_LIGHTSENSOR);
}
```

Sound Sensor

Here's the circuit path for the output of the soud sensor (microphone off screen).



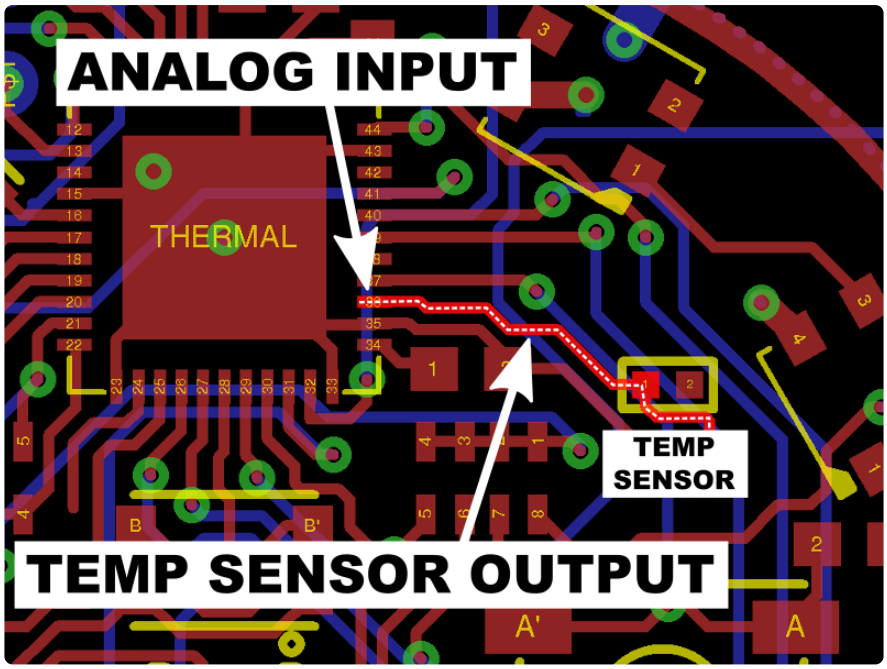
And here's the code for the `CircuitPlayground.soundSensor()` function. (direct link to repo (https://adafru.it/uet))

```
uint16_t Adafruit_CircuitPlayground::soundSensor(void) {
  return analogRead(CPLAY_SOUNDSENSOR);
}
```

Pretty boring. Nothing fancy or magical. Both the light sensor and sound sensor just call `analogRead()` for the pins they are attached to and return the value. And now you know why those functions return values from 0 to 1023. It's just the raw 10 bit analog to digital conversion.

Temperature Sensor

Here's the circuit path for the output of the temperature sensor.



And here's the code for the `CircuitPlayground.temperature()` function. (direct link to repo (https://adafru.it/uet))

```
float Adafruit_CircuitPlayground::temperature(void) {
   // Thermistor test
  float reading;

  reading = analogRead(CPLAY_THERMISTORPIN);

  //Serial.print("Thermistor reading: "); Serial.println(reading);

  // convert the value to resistance
  reading = ((1023.0 * SERIESRESISTOR) / reading);
  reading -= SERIESRESISTOR;

  //Serial.print("Thermistor resistance: "); Serial.println(reading);

  float steinhart;
  steinhart = reading / THERMISTORNOMINAL;     // (R/Ro)
  steinhart = log(steinhart);                  // ln(R/Ro)
  steinhart /= BCOEFFICIENT;                   // 1/B * ln(R/Ro)
  steinhart += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
  steinhart = 1.0 / steinhart;                 // Invert
  steinhart -= 273.15;                         // convert to C

  return steinhart;
}
```

OK, a bit more going on. But not really. Note that pretty much the first thing that happens is to read the analog value. The rest is just math to turn that into a temperature. In this case, that's more useful than the raw analog value. For the light and sound sensors, there's no really useful thing to convert to, so those just return the raw values.

Questions and Code Challenges

The following are some questions related to this project along with some suggested code challenges. The idea is to provoke thought, test your understanding, and get you coding!

While the sketches provided in this guide work, there is room for improvement and additional features. Have fun playing with the provided code to see what you can do with it. Or do something new and exciting!

Questions

- A 10 bit ADC has 1024 values, but why is the highest one 1023?
- Can you think of any natural phenomenon that behaves digitally?

Code Challenges

- Make the knob change the color of the NeoPixels.
- Make the NeoPixels blink at different rates set by the knob.
- Change the range of the tones generated in Speaker Fun.
- Try using something other than `map()`. Something non-linear maybe?
- Combine the NeoPixel and Speaker Fun code into one - lights and sounds!