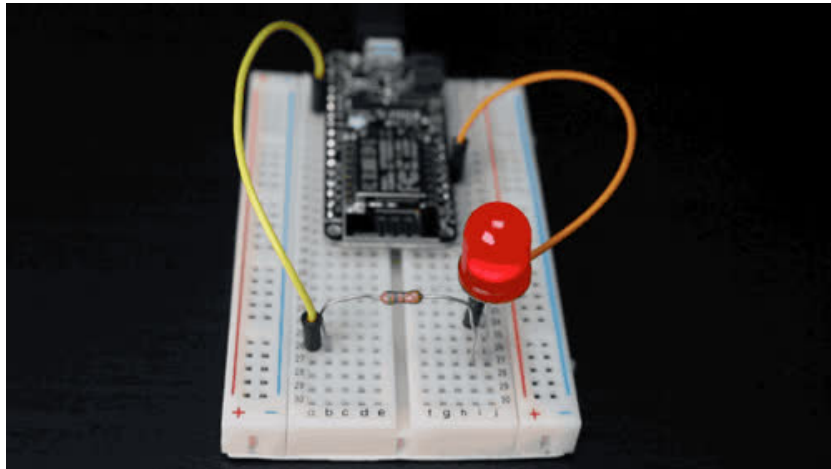


□

MicroPython Hardware: Analog I/O

Created by Tony DiCola



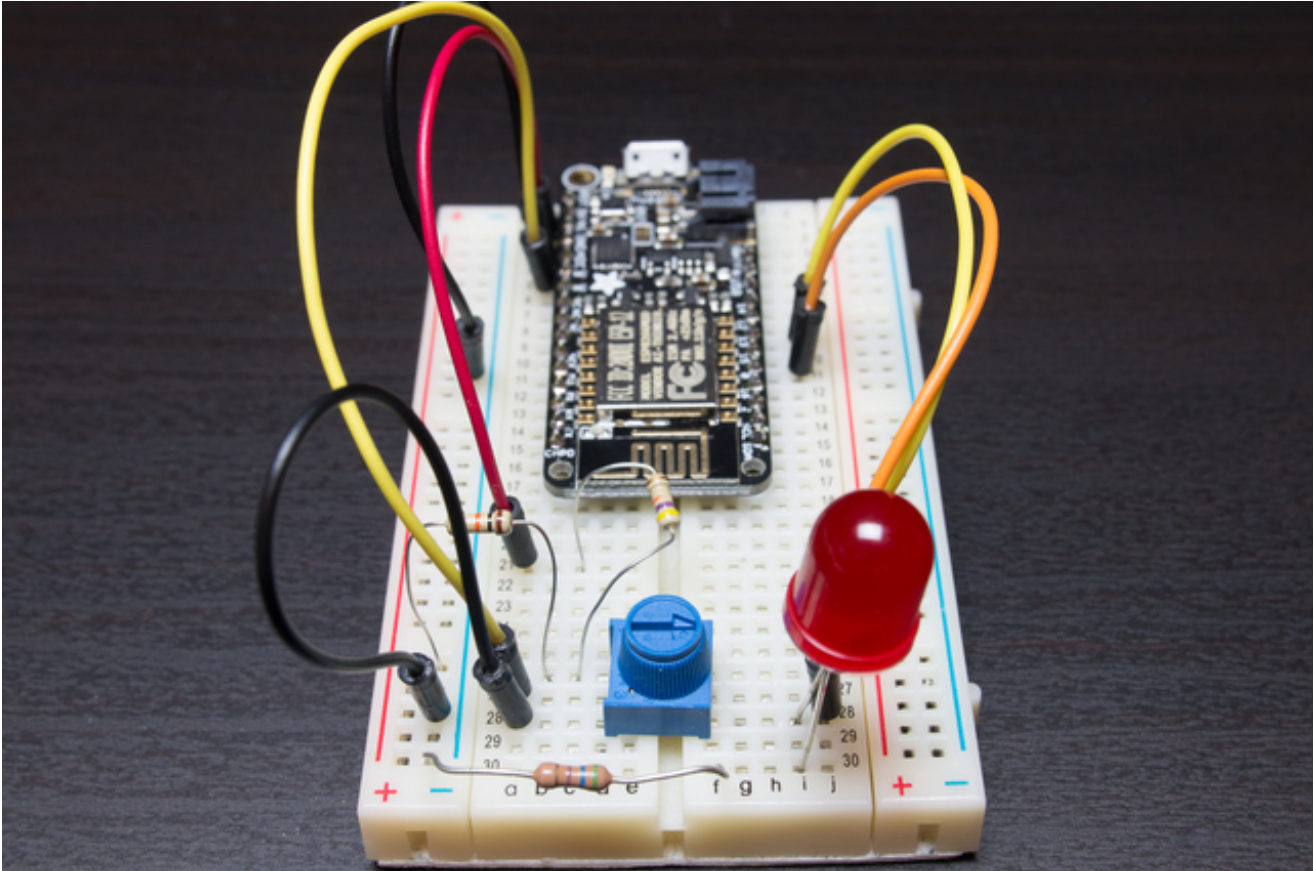
Last updated on 2017-01-27 10:53:43 AM UTC

Guide Contents

Guide Contents	2
Overview	3
Analog to Digital Converter	5
Read Potentiometer Value	5
Digital to Analog Converter	11
Pulse-Width Modulation	12
Control LED Brightness	12

Overview

Note this guide was written for MicroPython.org firmware and not Adafruit CircuitPython firmware.



This guide explores analog input and output (I/O) with MicroPython. [Previously you saw with digital I/O](http://adafru.it/reD) (<http://adafru.it/reD>) how values can be 'on' or 'off' depending on their high or low voltage value. You might wonder though can you deal with values somewhere in-between on and off or high and low? With analog signals you absolutely can go beyond simple digital on/off or high/low. In fact an analog signal can be any one of *infinite* values between on and off!

In this guide you'll see how to use an analog to digital converter (ADC) to read an analog voltage value. You'll also learn about a digital to analog converters (DAC) and pulse-width modulation (PWM) that let you go the other way and effectively output voltages of a certain value. Using analog I/O you can read sensors or potentiometers, and control the intensity of signals like when driving a speaker or just dimming a LED. MicroPython makes it just as quick and easy to perform analog I/O as it does digital I/O.

Before you get started you'll want to be familiar with the basics of using MicroPython by reading these guides:

- [MicroPython Basics: What is MicroPython?](http://adafru.it/pMb) (<http://adafru.it/pMb>)
- [MicroPython Basics: How to Load MicroPython on a Board](http://adafru.it/pNB) (<http://adafru.it/pNB>)
- [MicroPython Basics: Blink a LED](http://adafru.it/q2d) (<http://adafru.it/q2d>)
- [MicroPython Basics: Load Files & Run Code](http://adafru.it/r2B) (<http://adafru.it/r2B>)

Analog to Digital Converter

An analog to digital converter (ADC) is a peripheral that can read an analog voltage value and convert it into a digital or numeric value. This is perfect for reading some types of sensors like thermistors, photocells, force sensitive resistors, load cells, and more that change resistance based on what they're sensing. When a sensor changes its resistance it can cause a change in voltage that an analog to digital converter reads and converts into a number that a microcontroller understands.

Most MicroPython boards have built-in analog to digital converters. This means you can easily connect sensors and other analog components to read their values. However be aware each board is different and its analog to digital converter will have different specs like accuracy and voltage limits. Be sure to check out your board's documentation for more details:

- [MicroPython ESP8266 ADC Documentation \(http://adafru.it/r9b\)](http://adafru.it/r9b)
 - The ESP8266 has one ADC with 10-bit accuracy and a maximum voltage of **1.0 volt**. **Be sure not to exceed 1 volt on the ADC input or you could damage the board!**
- [MicroPython pyboard ADC Documentation \(http://adafru.it/pXb\)](http://adafru.it/pXb)
 - The pyboard has three ADC's with 12-bit accuracy and a maximum voltage of **3.3 volts**.
- [MicroPython WiPy ADC Documentation \(http://adafru.it/reG\)](http://adafru.it/reG)
 - The WiPy has four ADC's with 12-bit accuracy and a maximum voltage of **1.4 volts**.

Read Potentiometer Value

For this example we'll wire up a potentiometer, which is a variable resistor you can change by turning a knob, so that MicroPython on an ESP8266 board can read its value. To follow this example you'll need the following parts (which are all available in the [Adafruit Parts Pal \(http://adafru.it/reH\)](http://adafru.it/reH)):

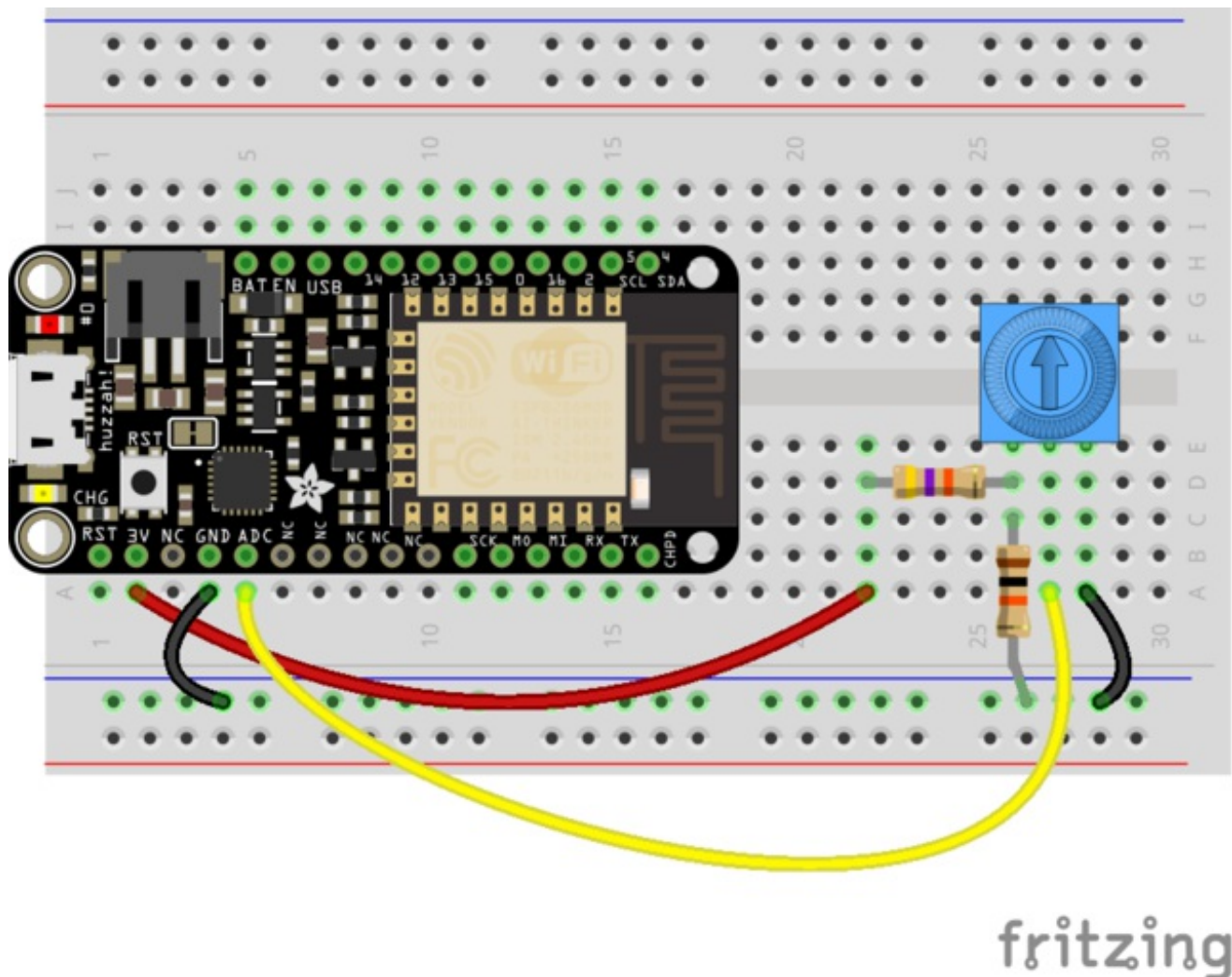
- **MicroPython ESP8266 board**, like the [Adafruit Feather Huzzah ESP8266 \(http://adafru.it/n6A\)](http://adafru.it/n6A). Other MicroPython boards can work too, but you might need to change the syntax slightly as mentioned further below in the code.
- **1x 10 kilo-ohm or 1 kilo-ohm potentiometer** (<http://adafru.it/356>)
 - The value of the potentiometer doesn't actually matter for how it will be used in

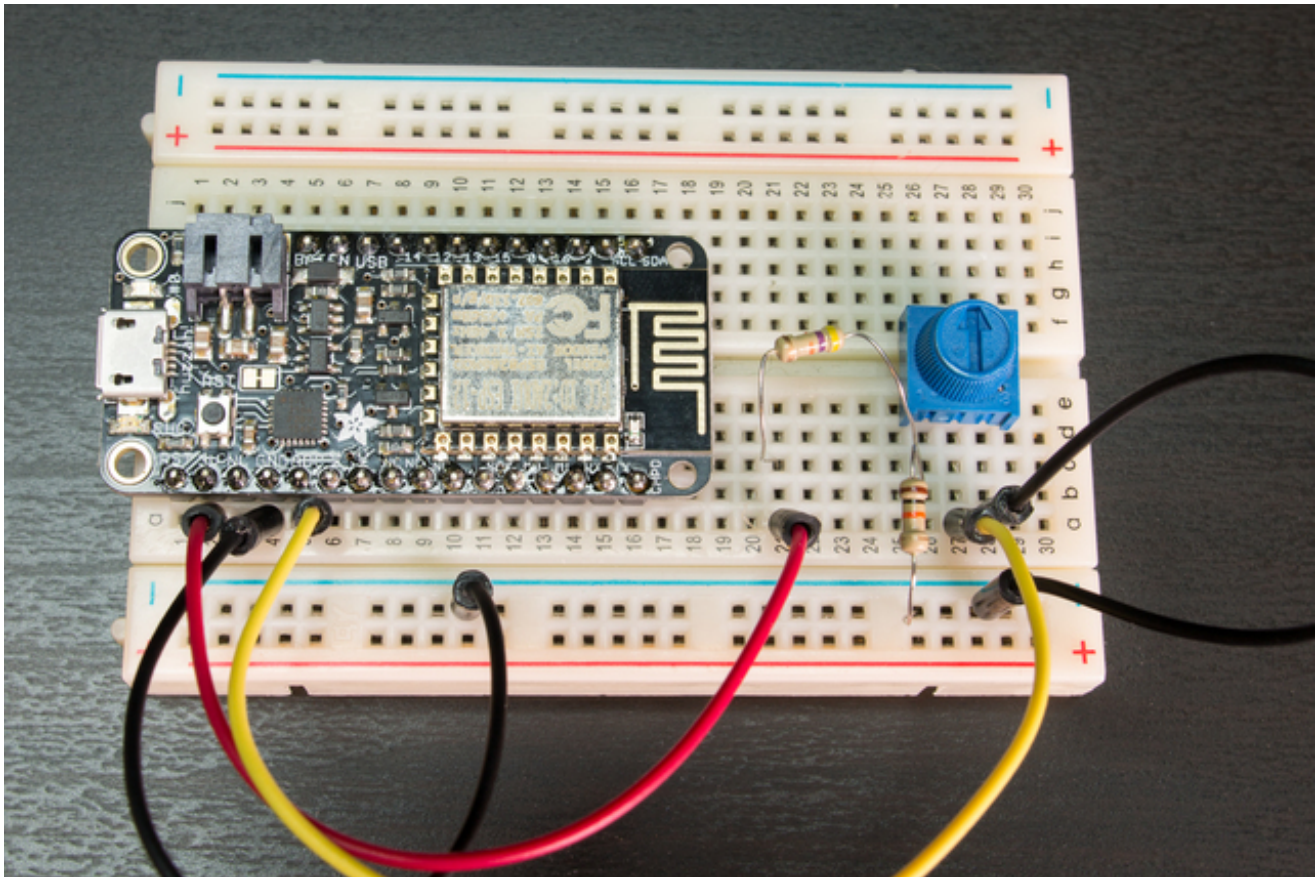
this circuit.

- 1x **10 kilo-ohm resistor** (<http://adafru.it/2784>)
- 1x **47 kilo-ohm resistor** (<http://adafru.it/2786>) OR **22 kilo-ohm resistor** (<http://adafru.it/2785>)
 - If you have a 22 kilo-ohm resistor you can use it to get the full range of 10-bit values. If you only have a 47 kilo-ohm resistor then you'll only get about half the range of values.
- **Breadboard** (<http://adafru.it/64>) and **jumper wires** (<http://adafru.it/153>).

Note the value of the two resistors used in this circuit are very important. These resistors will form a [voltage divider](http://adafru.it/osc) (<http://adafru.it/osc>) that drops 3.3V down to 0.5-1V which is below the voltage limit of the ESP8266 ADC. **It's very important to use a voltage divider or other means to make sure the maximum voltage to the ADC is 1V or less!**

Wire up the components as follows:





- Board **3.3V power** to one side of the **47 kilo-ohm (or 22 kilo-ohm) resistor**.
- The opposite side of the **47 kilo-ohm (or 22 kilo-ohm) resistor** to **one of the outside pins of the potentiometer** (i.e. not the middle pin, pick the left or right outer pin as it doesn't really matter which one you use).
- The **opposite outer pin of the potentiometer** connects to **board ground**.
- One side of the **10 kilo-ohm resistor** connects to the **where the 47/22 kilo-ohm resistor connects to the potentiometer**.
- The opposite side of the **10 kilo-ohm resistor** connects to **board ground**.
- The **middle pin of the potentiometer** connects to the **board ADC input**.

Double check you have the two resistors in the circuit exactly as shown above. If you accidentally swap them you might damage the ADC on the board!

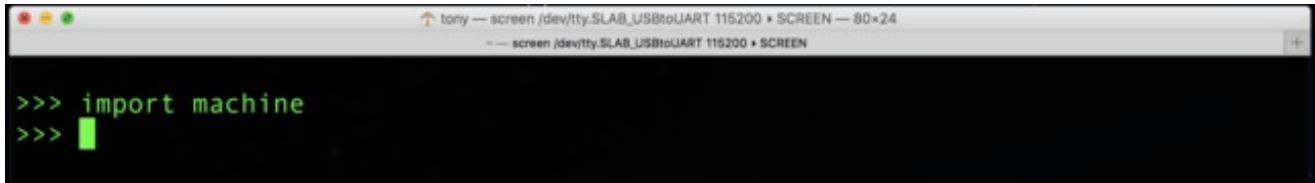
This circuit works by converting the 3.3 volt output of the board to a smaller voltage that's safe for the ADC to read. The two resistors form a voltage divider that cuts the voltage down to about 0.5 volts when using a 47 kilo-ohm resistor and about 1 volt when using a 22 kilo-ohm resistor. This voltage goes into one side of the potentiometer and will travel down to ground on the opposite side.

As the knob of the potentiometer rotates it changes the resistance and voltage seen by the middle/output pin. On one extreme of the knob the middle pin will see the full input voltage

(0.5 or 1 volt) and on the other extreme it will see 0 volts/ground. At the half-way point the output voltage will be half of the input. Try using a multimeter to read and confirm the voltage output by the potentiometer!

Now to read the ADC value open a serial or other REPL on the MicroPython board. First enter the following code to import the machine module:

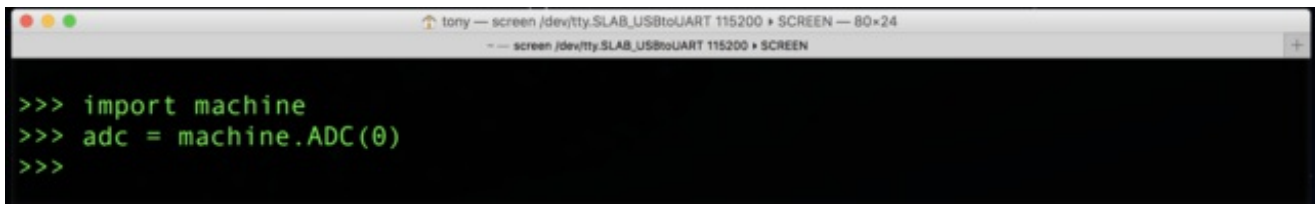
```
import machine
```

A screenshot of a terminal window with a dark background and green text. The window title bar shows 'tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24'. The terminal content shows two lines of code: '>>> import machine' followed by '>>>' with a green cursor on the next line.

pyboard note: Currently the pyboard uses the older pyb module instead of the machine module for ADC access. [Check out the pyboard ADC examples \(http://adafru.it/pXb\)](http://adafru.it/pXb) for the slightly different syntax to use.

Now you can create an instance of the ADC class by running:

```
adc = machine.ADC(0)
```

A screenshot of a terminal window with a dark background and green text. The window title bar shows 'tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24'. The terminal content shows three lines of code: '>>> import machine', '>>> adc = machine.ADC(0)', and '>>>' with a green cursor on the next line.

pyboard note: Since the pyboard has multiple ADC channels you should instead pass a Pin instance to the ADC initializer with the pin name for the desired ADC channel (like X19, X20, etc). [See the pyboard ADC examples \(http://adafru.it/pXb\)](http://adafru.it/pXb) for more details.

WiPy note: The WiPy also has multiple ADC channels but has a slightly different syntax to choose the right one to read. [Check the WiPy ADC example \(http://adafru.it/reG\)](http://adafru.it/reG) to see how you call the **channel** function on the ADC object to pick the channel you want to read.

Once you have an instance of the ADC class (or ADC channel) you can read its value with the **read** function:

```
adc.read()
```


A screenshot of a terminal window with a dark background. The window title bar shows 'tomy — screen /dev/tty.SLAB_USBtoUART 115200 * SCREEN — 80x24'. The terminal content shows the following: a prompt '>>>' followed by 'import machine', another prompt '>>>' followed by 'adc = machine.ADC(0)', a third prompt '>>>' followed by 'adc.read()', and the output '177'. A final prompt '>>>' is followed by a green cursor bar.

```
>>> import machine
>>> adc = machine.ADC(0)
>>> adc.read()
177
>>> █
```

WiPy note: The WiPy has a slightly different syntax where you call **value** or invoke the channel object as a function instead of calling **read**. Check the [WiPy ADC example](http://adafru.it/reG) (<http://adafru.it/reG>) for more details.

This function should return a number that will change depending on where the potentiometer knob is pointing. Try moving the potentiometer knob and calling the read function again. Keep moving the knob to different points and calling read to see how the number changes.

If you turn the knob to one extreme you should get a value that's less than 10 or even close to zero, and on the opposite extreme you should get a large value that's 300 or higher (800 or more if using the 22 kilo-ohm resistor). As you turn the knob between the two extremes you should see the reading grow and shrink evenly between those extreme values.

Note that the value might change slightly each time you call read even if you don't move the knob. It's normal for ADC readings to be 'noisy' and fluctuate up or down a tiny bit. You could read the ADC a few times and average the readings to get a more stable value if necessary.

The value you're seeing is the raw output of the analog to digital converter. The ADC is taking the voltage that it reads from the potentiometer and converting it to a number that ranges from 0 to the maximum bit accuracy of the ADC. For the ESP8266 it has a 10-bit ADC which means it can give values that range from 0 to 1023. For boards with a 12-bit ADC they can range from 0 to 4095.

The ADC determines the value to return based on where the voltage falls within the range it supports. For the ESP8266 it only supports a range of 0 to 1 volts, so an input of 0.5 volts would return a value exactly in between 0 and 1023, or around 512. An input of 1 volt would return 1023 (the maximum), and an input of 0 volts would return 0.

That's all there is to reading a potentiometer with an analog to digital converter in MicroPython! Just create an instance of the ADC class or channel and call its read function. With access to an analog to digital converter you can explore many different types of analog sensing using MicroPython. For example [use a thermistor](http://adafru.it/reI) (<http://adafru.it/reI>) and an ADC to measure temperature!

Digital to Analog Converter

A digital to analog converter (DAC) can take a digital or numeric value and convert it into a voltage. Think of a DAC as the opposite or compliment to an analog to digital converter.

Instead of reading analog voltage values the DAC will generate voltage values based on a digital or numeric input.

Digital to analog converters are useful when interfacing with some analog components, for example a speaker which plays sound and music. Sound files on your computer are stored as numbers in a digital format but speakers can't 'play' these numbers and instead need a special analog signal of varying voltages to make sound. A digital to analog converter can take digital values like music samples and converts them into voltages and analog signals that can drive a speaker.

MicroPython boards with a built-in DAC are somewhat uncommon. Currently the pyboard is the only board with a built-in DAC, other boards like the ESP8266 do not support a DAC. However you will see on the next page for pulse-width modulation that for boards without a DAC you can sometimes get similar 'digital to analog' capabilities.

Since most MicroPython boards don't have a built-in DAC its usage won't be covered in detail in this guide. For the pyboard [consult its DAC documentation](http://adafru.it/reJ) (<http://adafru.it/reJ>) for examples of its use.

Pulse-Width Modulation

[Pulse-width modulation \(http://adafru.it/reK\)](http://adafru.it/reK) (PWM) is a way to use a digital output (i.e. one that is only on or off) to simulate an analog or varying voltage output. PWM works by turning a digital output on and off very quickly. So quickly that the component connected to it can't tell the output signal is changing, instead it just sees the 'average' of how long the signal is on vs. off. If the output is turned on for slightly longer than it's turned off it will appear to be a higher value. Controlling how long the signal is on vs. off, or the duty cycle, allows you to smoothly control the output values.

One great example of using PWM is to dim a LED between full and no brightness. If a PWM signal is fast enough (changing more than about 60 times a second) the human eye won't be able to see the LED turning on and off quickly. Instead the eye will see a brighter light the longer the PWM signal is turned on vs. off. Using PWM you can control the brightness of a LED so that it pulses and glows in interesting ways instead of just blinking fully on and off.

Note that PWM control in MicroPython is somewhat in flux and varies greatly depending on the board. This example will only look at MicroPython on the ESP8266 as it has a simple PWM interface, however be sure to consult the documentation for other boards to learn how to use PWM:

- [MicroPython ESP8266 PWM Documentation \(http://adafru.it/r9b\)](http://adafru.it/r9b)
- [MicroPython pyboard PWM Documentation \(http://adafru.it/pXb\)](http://adafru.it/pXb)
 - PWM on the pyboard currently uses a lower-level timer interface.
- [MicroPython WiPy PWM Documentation \(http://adafru.it/reG\)](http://adafru.it/reG)
 - PWM on the WiPy also uses a lower-level timer interface.

Control LED Brightness

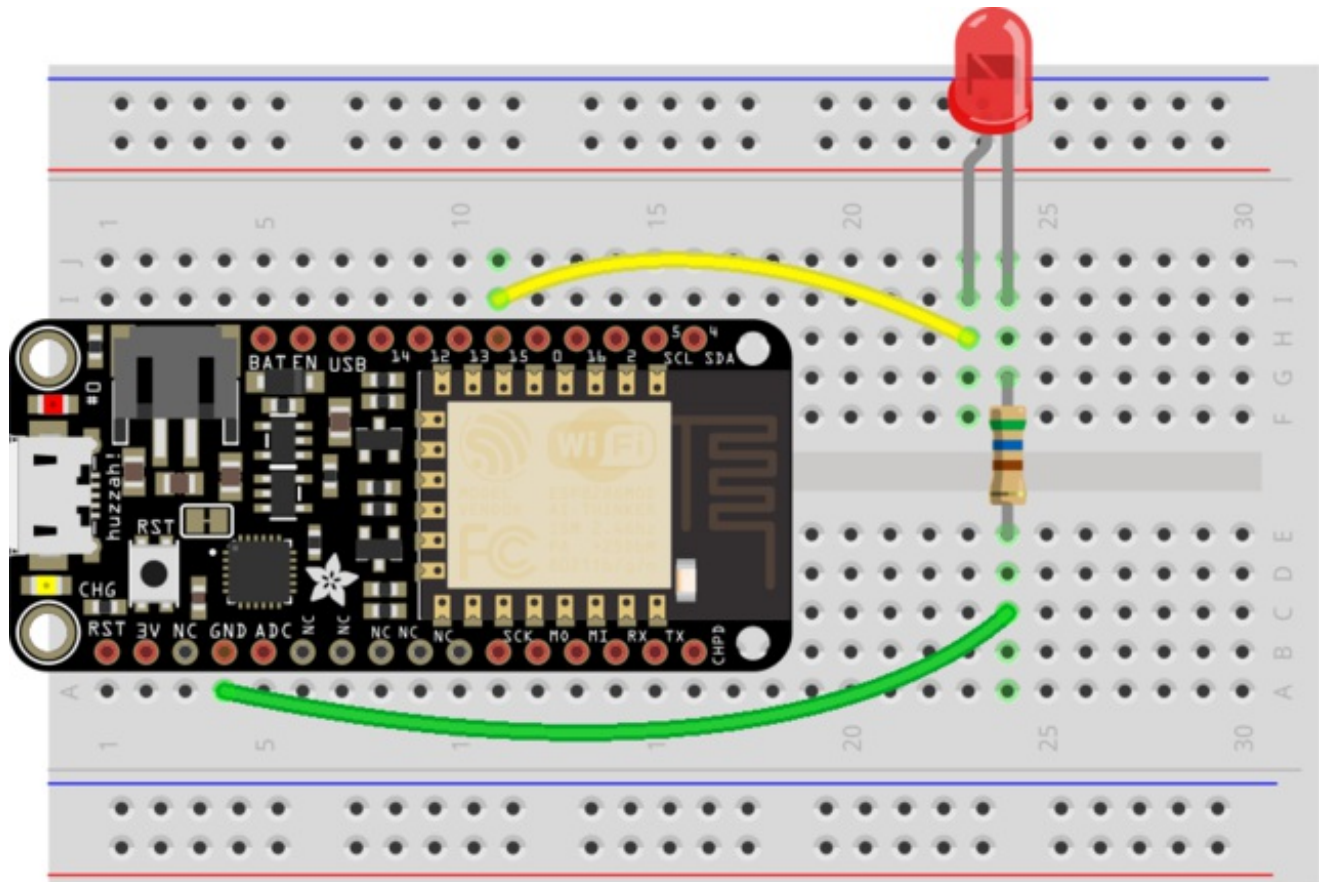
For this example you'll learn how to dim a LED using MicroPython on the ESP8266. To follow this example you'll need the following parts (all available in the [Adafruit Parts Pal \(http://adafru.it/reH\)](http://adafru.it/reH)):

- **MicroPython ESP8266 board**, like the [Adafruit Feather Huzzah ESP8266 \(http://adafru.it/n6A\)](http://adafru.it/n6A).
- **1x LED**, like one of these [5mm diffused red LEDs \(http://adafru.it/299\)](http://adafru.it/299).
- **1x ~560-3k ohm resistor**. You need to use a resistor to limit the current that can flow through the LED (otherwise you might send too much current through the LED).

and damage it or the MicroPython board!). The exact value of the resistor usually doesn't matter here, anything in the 560-3,000 ohm range should work great (higher resistance will be lower current and usually dimmer LED). If unsure grab a [2200 ohm resistor pack](http://adafru.it/2782) (<http://adafru.it/2782>).

- **Breadboard** (<http://adafru.it/64>) & **jumper wires** (<http://adafru.it/153>).

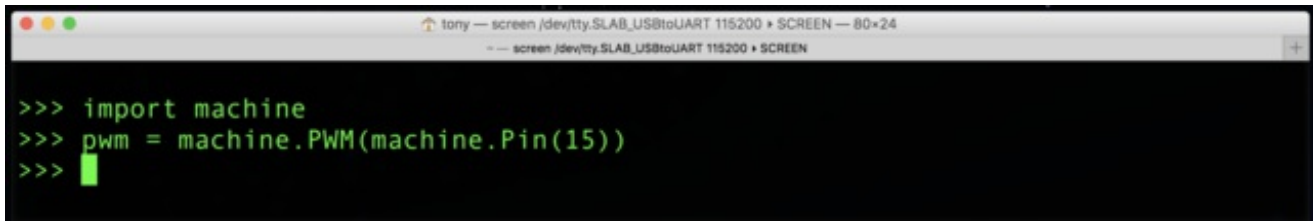
Wire up the components exactly the same as from the [blink a LED guide](http://adafru.it/rax) (<http://adafru.it/rax>):



- **Digital GPIO 15** is connected to the **anode, or longer leg, of the LED**. It's very important to use the correct leg of the LED otherwise it won't light up as expected! Note don't use GPIO 16 as it doesn't support PWM output on the ESP8266.
- **The cathode, or shorter leg, of the LED** is connected to **one side of the resistor** (unlike the LED it doesn't matter which way you orient the resistor).
- **The other side of the resistor** is connected to the board's **ground or GND pin**.

Now connect to a serial or other REPL on your board and run the following code to import the machine module and create a PWM controlled pin:


```
import machine
pwm = machine.PWM(machine.Pin(15))
```



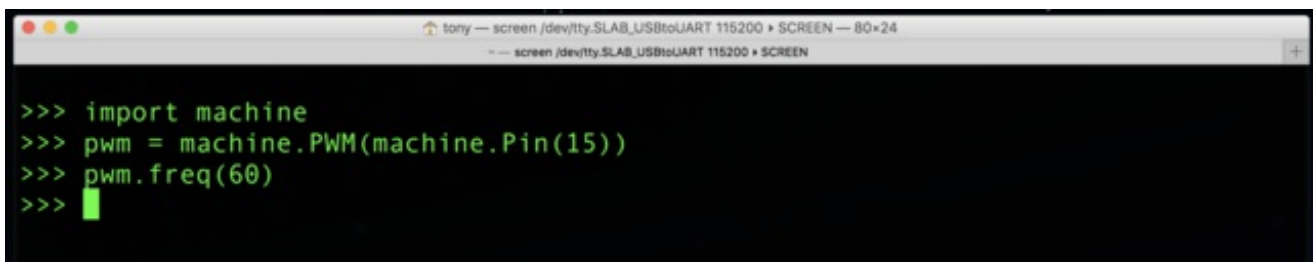
```
tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24
— screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN

>>> import machine
>>> pwm = machine.PWM(machine.Pin(15))
>>> █
```

The second line will create an instance of the PWM class from the machine module and assign it to a variable called pwm. Notice that the initializer takes in a machine Pin class instance, in this case one created for pin 15.

Next you can set the frequency of the PWM signal to 60hz by calling the **freq** function:

```
pwm.freq(60)
```



```
tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24
— screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN

>>> import machine
>>> pwm = machine.PWM(machine.Pin(15))
>>> pwm.freq(60)
>>> █
```

The frequency controls how fast the PWM signal is turned on and off, in this case a frequency of 60 will make the LED turn on and off 60 times a second. You might need to adjust the frequency depending on what you're trying to control with a PWM output. For LEDs you don't need a super fast frequency since humans can't really see something changing hundreds or thousands of times a second.

You can also see the current frequency of the PWM output by calling the **freq** function with no parameter:

```
pwm.freq()
```



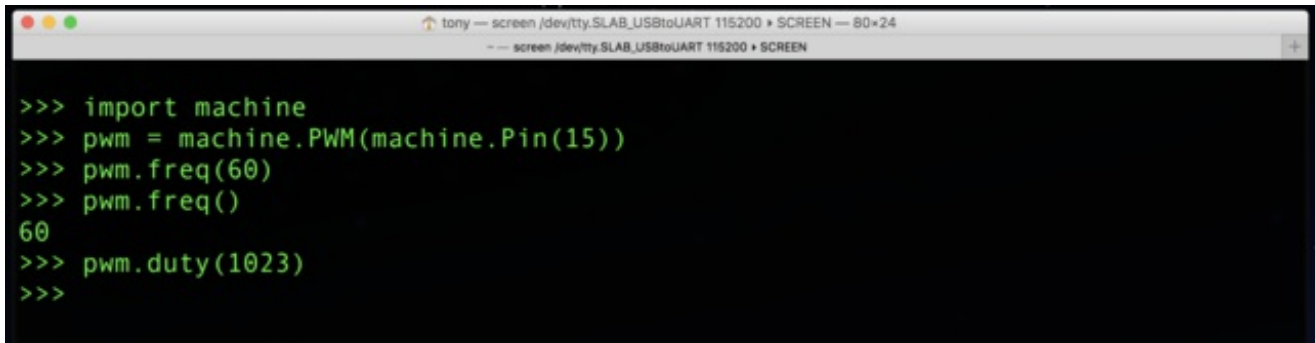
```
tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24
— screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN

>>> import machine
>>> pwm = machine.PWM(machine.Pin(15))
>>> pwm.freq(60)
>>> pwm.freq()
60
>>> █
```

Also note the frequency is global to all PWM instances. If you create multiple PWM objects and change the frequency on one it will change the frequency for all of them.

Now you can set the 'duty cycle', or percent of time that the signal is on vs. off, with the **duty** function:

```
pwm.duty(1023)
```

A screenshot of a terminal window with a dark background and green text. The window title bar shows 'tony — screen /dev/tty.SLAB_USBtoUART 115200 • SCREEN — 80x24'. The terminal contains the following Python code:

```
>>> import machine
>>> pwm = machine.PWM(machine.Pin(15))
>>> pwm.freq(60)
>>> pwm.freq()
60
>>> pwm.duty(1023)
>>>
```

The value you pass to the duty function should be in the range of 0 to 1023. A value of 1023 like the above will set the duty cycle to 100% or completely on. This means the signal never actually turns off--it's always on and the LED is full brightness.

Now try setting the duty cycle to 0:

```
pwm.duty(0)
```

Notice the LED turns off. This is because the duty cycle is 0% or completely off--the LED never actually turns on.

Set the duty cycle somewhere inbetween 0 and 1023, like about halfway at 512:

```
pwm.duty(512)
```

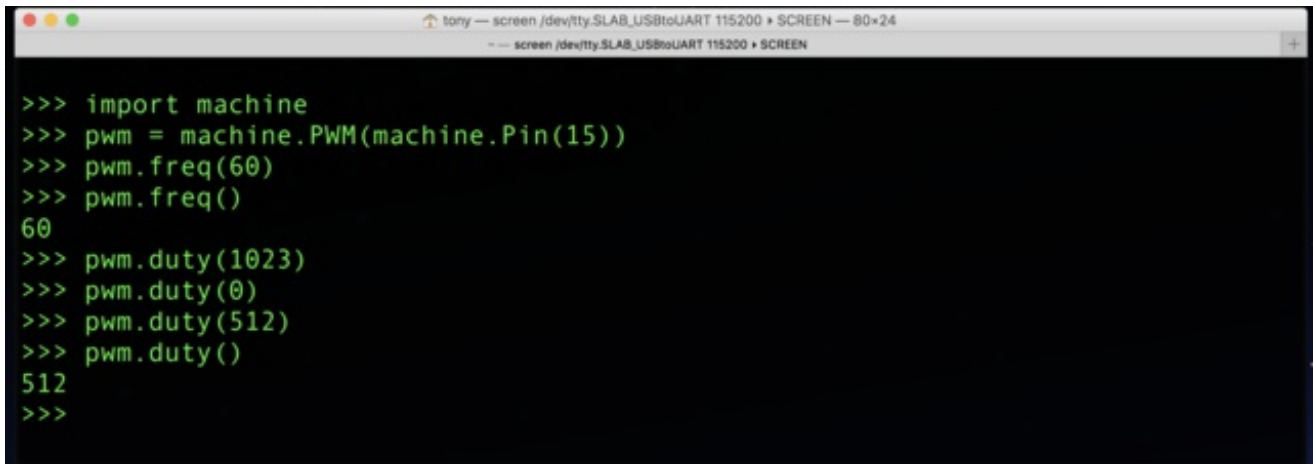
Now you should see the LED light up less bright than before!

Try setting the duty cycle to different values between 0 and 1023 to see how the brightness changes. As you increase the duty cycle to 100% the LED will get brighter! Remember this is because the PWM signal is turned on for a higher percent of time than turned off.

Your eye doesn't see each flash of the LED, instead it just see the 'average' amount of time its on as different brightness levels.

You can also call the duty function without a parameter to see the current duty cycle value:

```
pwm.duty()
```

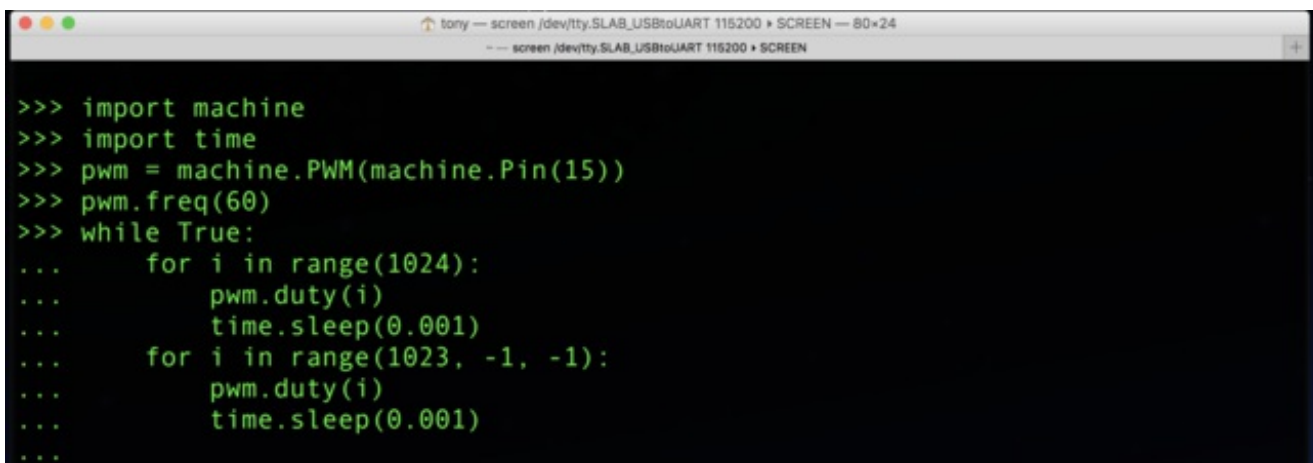
A screenshot of a terminal window with a dark background and green text. The window title bar shows 'tony — screen /dev/tty.SLAB_USBtoUART 115200 * SCREEN — 80x24'. The terminal contains the following code:

```
>>> import machine
>>> pwm = machine.PWM(machine.Pin(15))
>>> pwm.freq(60)
>>> pwm.freq()
60
>>> pwm.duty(1023)
>>> pwm.duty(0)
>>> pwm.duty(512)
>>> pwm.duty()
512
>>>
```

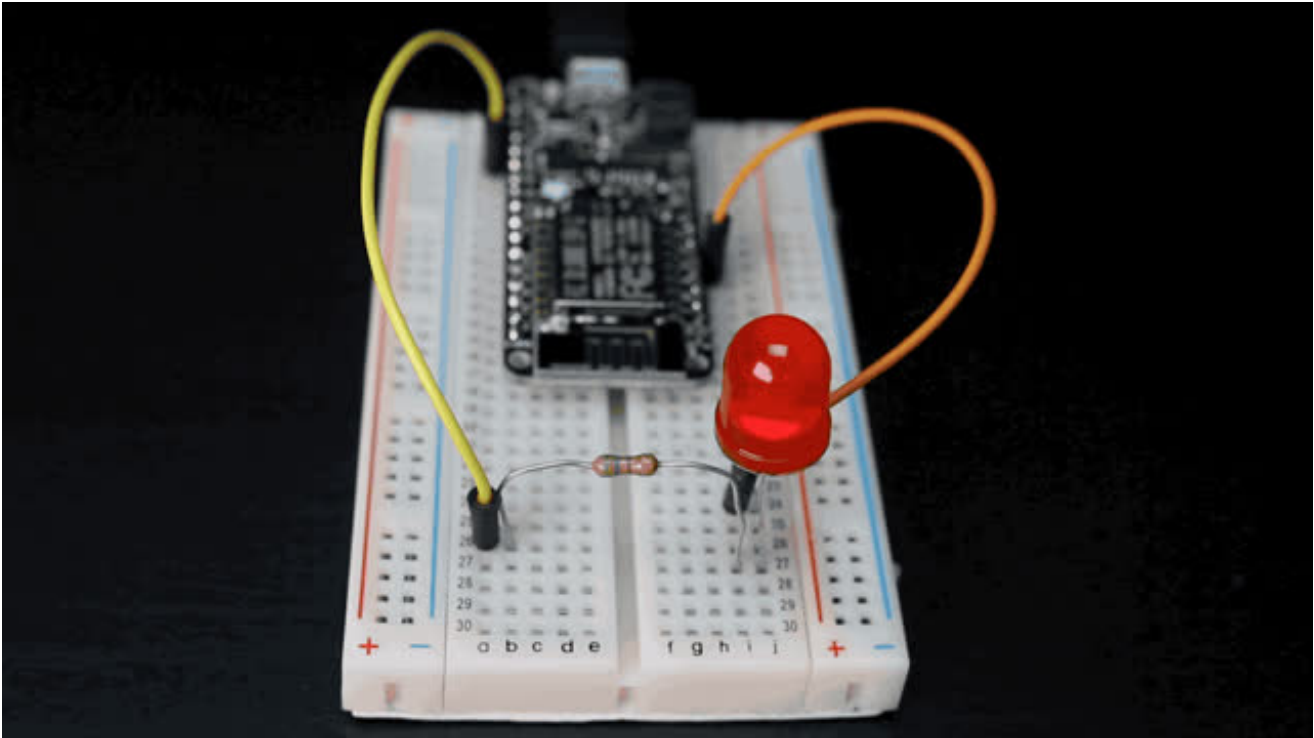
You might wonder why the duty cycle is set with a number between 0 and 1023. Like using the ADC there's a limited 'resolution' or bit accuracy for the PWM output and with the ESP8266 you only have 10 bits available (0 to 1023).

Try making the LED fade or pulse on and off by running the following code:

```
import time
import machine
pwm = machine.PWM(machine.Pin(15))
pwm.freq(60)
while True:
    for i in range(1024):
        pwm.duty(i)
        time.sleep(0.001)
    for i in range(1023, -1, -1):
        pwm.duty(i)
        time.sleep(0.001)
```

A screenshot of a terminal window with a dark background and green text. The window title bar shows 'tony — screen /dev/tty.SLAB_USBtoUART 115200 * SCREEN — 80x24'. The terminal contains the following code:

```
>>> import machine
>>> import time
>>> pwm = machine.PWM(machine.Pin(15))
>>> pwm.freq(60)
>>> while True:
...     for i in range(1024):
...         pwm.duty(i)
...         time.sleep(0.001)
...     for i in range(1023, -1, -1):
...         pwm.duty(i)
...         time.sleep(0.001)
... 
```



You should see the LED quickly fade or pulse on and off about once a second. Press **Ctrl-C** to interrupt and stop the code when finished.

This code will import modules and setup the PWM output as shown previously. Then it jumps into an infinite loop (the **while True** loop) that ramps the LED up from 0 to 100% duty cycle (0 to 1023) and then back down from 100% to 0% duty cycle.

Notice the ramp up is done in a for loop over the range of values 0 to 1023. Inside the loop the duty cycle is set to the appropriate value, then the **time.sleep** function delays for a short time (1 millisecond). Another for loop does the ramp down but this time it uses the range of 1023 down to 0 (notice the different [range function](http://adafru.it/reL) (<http://adafru.it/reL>) syntax to count down to 0 instead of up).

To go further on your own try combining reading a potentiometer value with dimming a LED using PWM. As you saw on the previous page you can use the ESP8266 ADC to read the position of a potentiometer knob. Then using that position you can use a PWM output to control the intensity of a LED as shown on this page. Try making the code to read the potentiometer and convert its value into a duty cycle that adjusts the LED intensity!

That's all there is to controlling the brightness of a LED with pulse-width modulation! PWM is handy for when you need to control the intensity of an analog value and don't have a DAC or other special hardware available. However not all components can be directly controlled with PWM, only components like LEDs which can tolerate the fast signal change and 'average' it out to be a different intensity of light.

As mentioned too other MicroPython boards have a slightly different syntax than the PWM class in the MicroPython ESP8266 port shown above. [Consult your board's documentation](http://adafru.it/pXc) (<http://adafru.it/pXc>) for more details. The basic ideas of frequency and duty cycle are the same but other boards differ in how they expose control over those parameters using timers and other lower-level details.