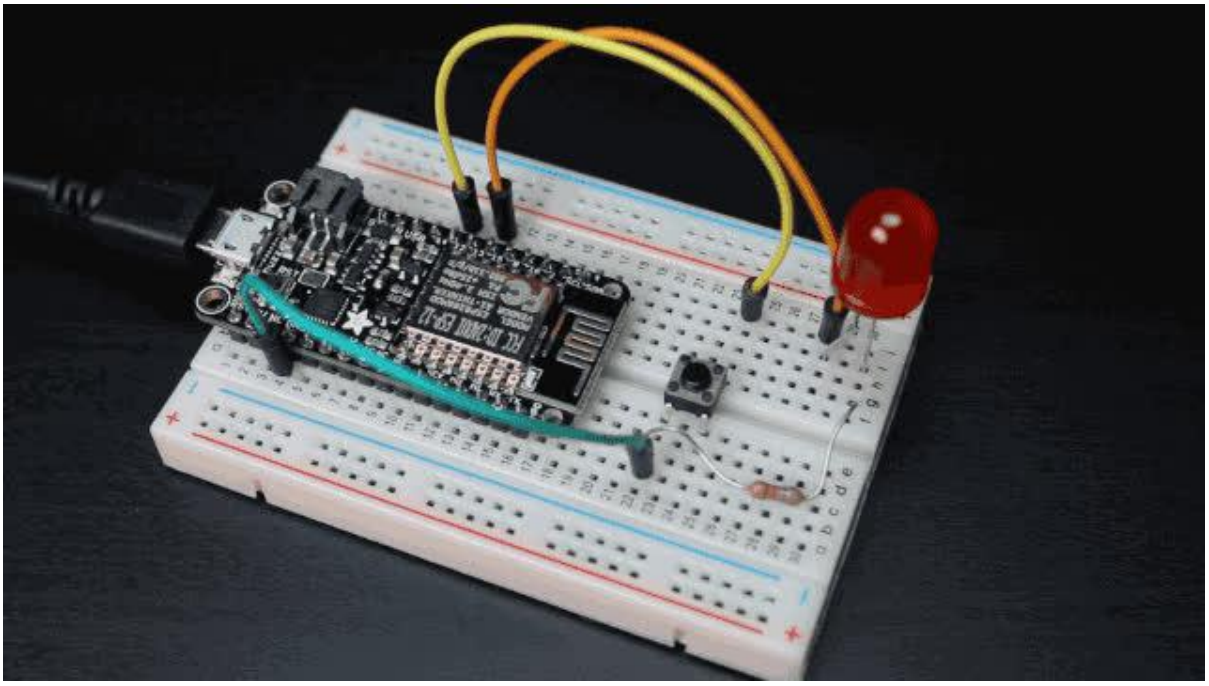
MicroPython Hardware: Digital I/O

Created by Tony DiCola



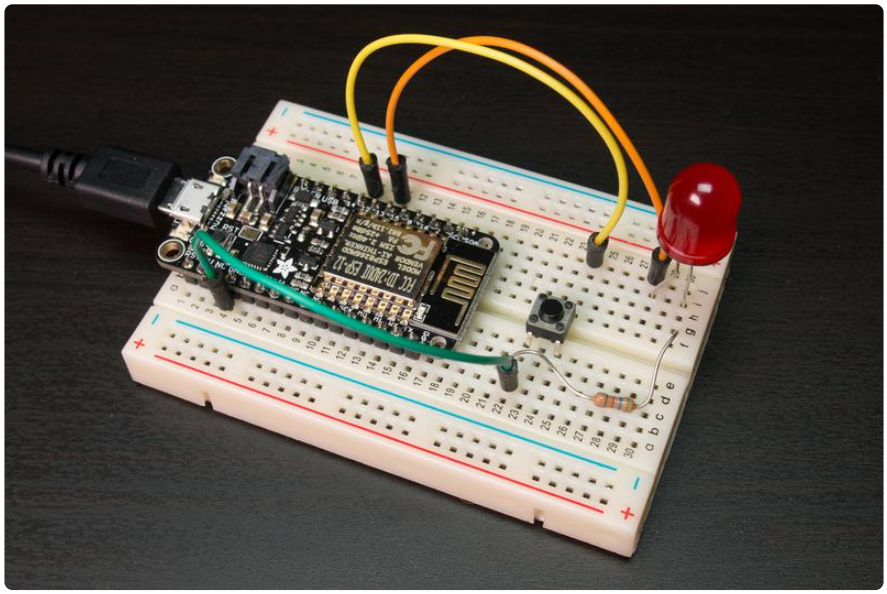
https://learn.adafruit.com/micropython-hardware-digital-i-slash-o

Last updated on 2023-08-29 03:13:10 PM EDT

Table of Contents

Overview

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



Digital I/O, or input/output, is one of the most basic ways to talk to hardware. We say 'digital' because it only deals with on or off values and nothing in-between. For example with digital I/O a LED can be turned completely on and completely off repeatedly to blink. Or using digital I/O you can read when a switch or button is moved on or off. With just a few lines of MicroPython code you'll be a master of digital I/O in no time.

This guide explores how to use digital I/O pins with MicroPython. You'll see how to control a LED using a digital output pin, and read a pushbutton using a digital input pin. These basic digital I/O examples can serve as the starting point for more complex and interesting projects that use hardware like LEDs, buttons, switches, relays, transistors, and even simple digital sensors!

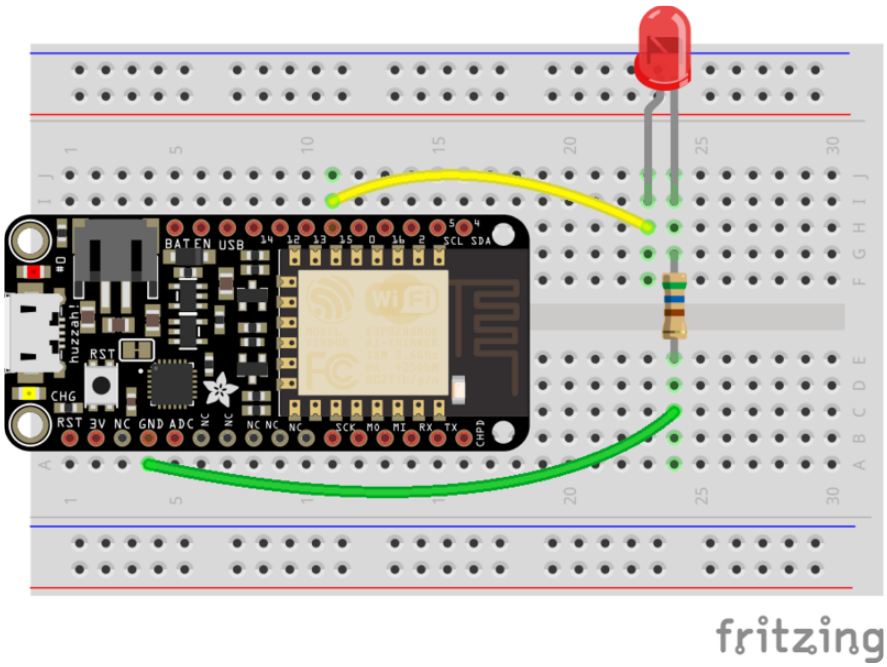
Before you get started you'll want to be familiar with the basics of using MicroPython by reading these guides:

- MicroPython Basics: What is MicroPython? ()
- MicroPython Basics: How to Load MicroPython on a Board ()
- MicroPython Basics: Blink a LED ()
- MicroPython Basics: Load Files & Run Code ()

Digital Outputs

A digital output is a simple high (on) or low (off) signal from a board's pin that you can control with MicroPython. Using a digital output you can turn something on or off, like a LED, a relay, a transistor, or more.

Let's walk through how to turn a LED on and off with a digital output from a MicroPython board. There's an entire guide just on blinking a LED with MicroPython () that you'll want to read and follow first. In that guide you'll see how to wire a LED to a MicroPython board (), for example like with the Feather Huzzah ESP8266 (), a red LED, and a 560 ohm resistor as below:



- 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!
- 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.

Once the LED is wired to the board connect to the serial or other MicroPython REPL and create a digital output pin by running the following code:

```
import machine
pin = machine.Pin(15, machine.Pin.OUT)
```

The import machine line will import the machine module which provides much of the hardware access API for MicroPython. In particular the machine module has a Pin class that allows you to create pin objects for all of the digital I/O pins on a board.

In this case the second line will create an object called pin and set it as an instance of the machine module Pin class. The initializer for the Pin class takes two important parameters:

- The number or name of the board pin. Check your board's documentation () for details on the pin values--some boards like the pyboard use a string identifier (like 'X1') while other boards like the ESP8266 use simple numbers (like 15, 14, etc.).
- The type of digital I/O pin, in this case a digital output. The machine.Pin.OUT value is a special constant value that tells MicroPython we intend to use this pin as an output we can control.

Now control the output level of the pin by calling the value function on the pin object in different ways:

```
pin.value(0)
pin.value(1)
pin.value(True)
pin.value(False)
```

The value function takes a single parameter which indicates if the output should be high or low (on or off). Notice when the pin value is set to 0 or False the LED turns off, and when set to 1 or True the LED turns on! The LED turns on and off because the digital output connected to it changes from high to low voltage levels.

There's also a handy shortcut for setting a pin to a high or low level directly with the hi gh and low functions:

```
pin.high()
pin.low()
```

The high function will set the pin to a high level, and the low function will set the pin to a low level. Notice again the LED turns on and off as the level from the pin changes.

That's all there is to using digital outputs with MicroPython! Although they're simple digital outputs are very handy for talking to devices that expect an on or off signal like a LED, a relay (http://adafru.it/2895) or power tail (), or a transistor (http://adafru.it/756)

controlling high power devices like a solenoid (http://adafru.it/412), laser (http://adafru.it/1054), super bright LED (http://adafru.it/754) and more.

Digital Inputs

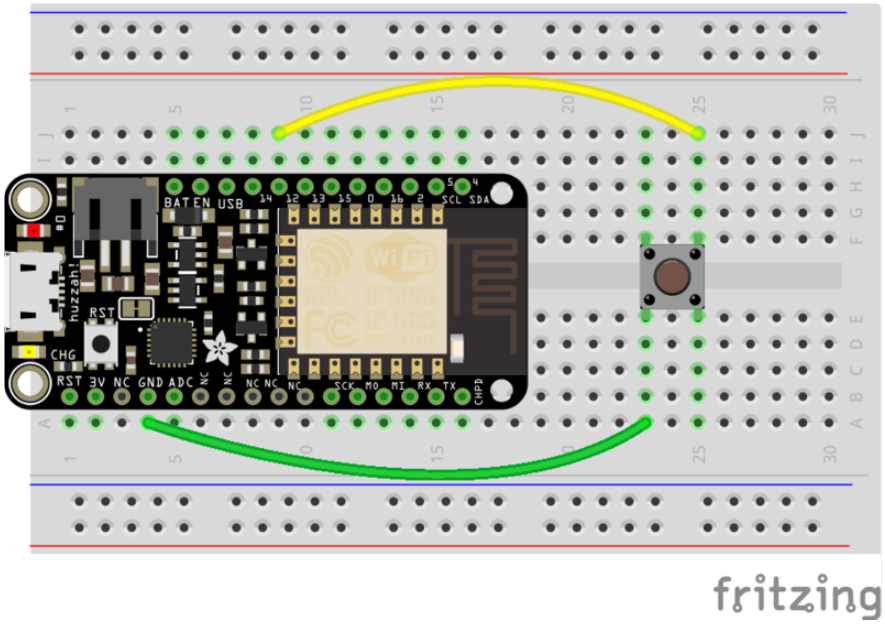
A digital input is a board pin which can read if it sees a high (on) or low (off) logic level connected to it. For example if you connect a button to a digital input you can detect when it's pressed and released. MicroPython makes it just as easy to use a pin digital inputs as it does outputs.

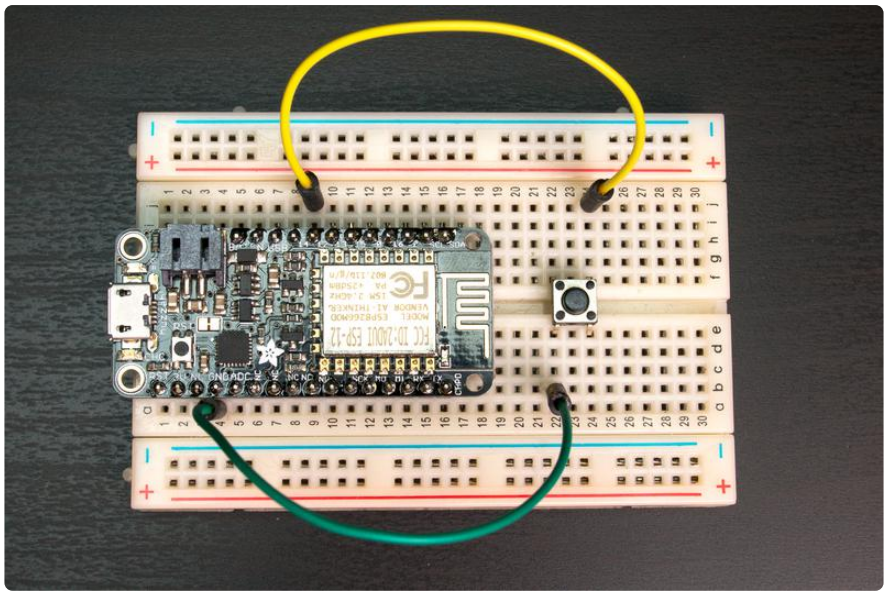
To learn about digital inputs we'll wire a button to a MicroPython board and detect when it's pressed. You'll need the following hardware for this example:

- A board running MicroPython, for this example we'll use the Feather HUZZAH ESP8266 () but any MicroPython board should work.
- A momentary push button, like one of these breadboard friendly buttons (http://adafru.it/367).
- A breadboard (http://adafru.it/64) and hookup wires (http://adafru.it/153).

All of the necessary parts except the MicroPython board are available in the Adafruit Parts Pal pack (http://adafru.it/2975).

Connect the button to the board ground and a GPIO pin as follows:





- Board pin 12 to one corner of the button. Note you don't want to use pin 15 on the ESP8266 because it's connected to a pull-down resistor internally that will conflict with reading other signals.
- Board ground to the opposite corner of the button.

With this setup when the button is pressed it will connect pin 12 to ground and read a low signal on the pin. When the button is released it won't be connected to ground anymore and will actually 'float' or fluctuate between random high & low values. Floating at random values isn't a problem though because you can enable an internal pull-up resistor on the pin. The pull-up resistor will pull the input to read a high level instead of floating when the button is released.

Be aware that on some boards not all the pins support an internal pull-up resistor. Check your board's documentation () for more details.

If the board doesn't support an internal pull-up resistor on the pin you can add an external one easily. Just connect a 10 kilo ohm or higher resistor from the input pin to a high logic level like the 3.3V output of the board.

Once the hardware is wired up connect to a serial or other MicroPython REPL on the board. Run the following commands to create a digital input pin with an internal pull-up resistor enabled:

```
import machine
button = machine.Pin(12, machine.Pin.IN, machine.Pin.PULL_UP)
```

Just like with creating a digital output on the previous page, you first need to import the machine module. Remember if you've already imported the machine module earlier in a script you don't need to import it again.

Next a button object is created as an instance of the machine module Pin class. However notice the parameters are a little bit different when creating this input as compared to creating an output. The parameters are:

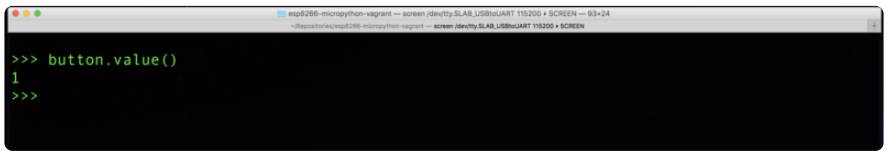
- The board pin name or number, just like with creating a digital output pin.
- The type of pin, in this case a digital input with the value machine.Pin.IN. This means the pin will be set as an input that can read high or low levels.
- The third parameter is the constant value machine.Pin.PULL_UP to enable an internal pull-up resistor on the pin. This parameter isn't actually required and if it's not specified then the pin will still be an input but it won't have any internal pull-up resistor set.

Note: If you're using an external pull-up resistor or reading something other than a button that might not require the internal pull-up be sure to omit the third machine.Pin.PULL_UP parameter!

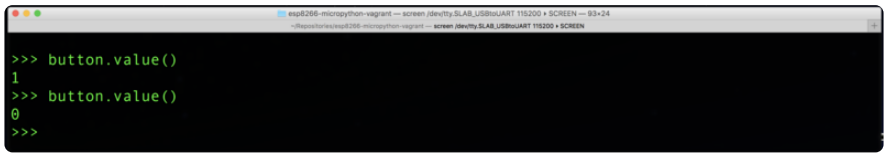
Now read the value of the pin by calling the value function without any parameters:

```
button.value()
```

Without the button being pressed you should see the value 1 returned:



Now press the button and run the same command again while the button is being pressed, you should see a value 0 returned:



If you see different values double check your wiring and check that the board pin doesn't have something else connected to it.

The value function returns the current level of the pin, either 1 for a high logic level or 0 for a low logic level. Notice how the button is at a high level (value returns 1) when it's not pressed. This is because the pull-up resistor keeps the pin at a high level when it's not connected to ground through the button. When the button is pressed then the input pin connects to ground and reads a low level (value returns 0).

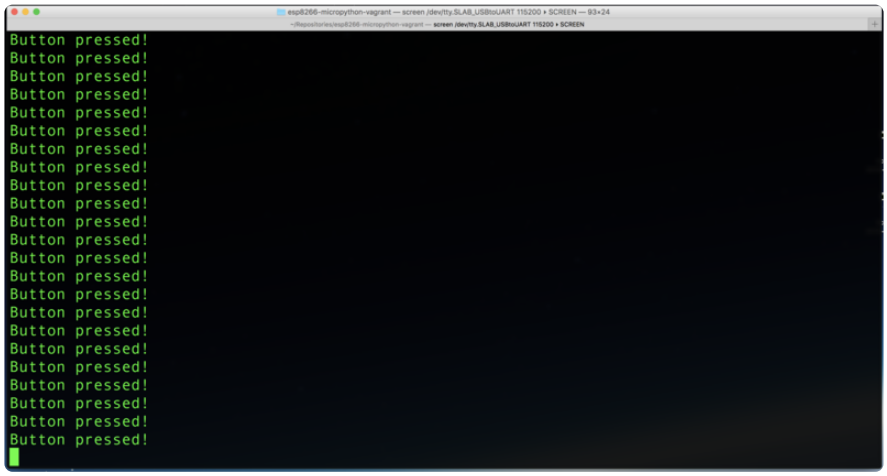
Reading a digital input is as easy as calling the value function!

One common task when reading a digital input like a button is detecting when the button is pressed or released. For example try running the following code which prints a message when the button is pressed:

```
import machine
button = machine.Pin(12, machine.Pin.IN, machine.Pin.PULL_UP)
while True:
    if not button.value():
        print('Button pressed!')
```

This code uses an infinite loop (the while True loop) to continually check if the button is pressed. Remember the value function returns 0 (or false) when the button is pressed so the if statement uses the not operator to invert the button value and detect when it's a 0 or false value.

Press the button quickly and notice the screen is immediately filled with button pressed messages:



Press Ctrl-c to stop the loop and get back to the REPL.

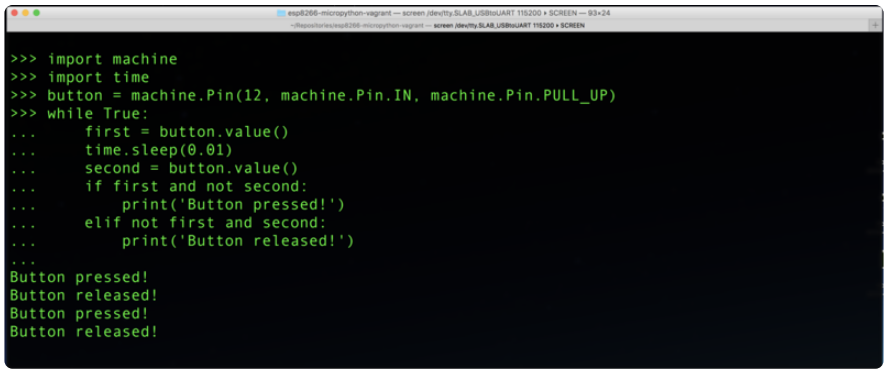
You might wonder if something went wrong to cause all button pressed messages. Nothing is wrong, however the code needs to be a little smarter if it wants to only print the message once when the button is pressed.

Now try the following code which prints a message only once when the button is pressed or released:

```
import machine
import time
button = machine.Pin(12, machine.Pin.IN, machine.Pin.PULL_UP)
while True:
    first = button.value()
    time.sleep(0.01)
```

```
    second = button.value()
    if first and not second:
        print('Button pressed!')
    elif not first and second:
        print('Button released!')
```

Try pressing and releasing the button a few times. You should see a message printed only once!



This code is a little more complex but still easy to explain. The way it works is the code looks for only when the input pin changes from a high to low level (the button was just pressed) or a low to high level (the button was just released). By looking for this change instead of a constant pressed or not pressed value the code can print a message once instead of repeatedly.

To check for this change in level the code reads the button twice. Notice the first call inside the loop is to the button value function. This gets an initial button value and saves it in the variable called first.

Next the time.sleep function is called to pause the program for a small time (10 milliseconds in this case). This very short pause is there to 'debounce' the button press. When a button is physically pressed it can randomly jump from low to high values for a small time as metal contacts inside the button start to touch. By delaying a bit between the two readings it gives time for the button to stabilize and not bounce around randomly. You don't want to delay for too long though or you might miss the change in button state!

Finally the button is read again and its value stored in the variable called second. The first and second variables hold the two button readings.

Once the code has two button readings it can check if there was a change between those readings. For example if the button was pressed it would have a first reading that was true/1 (button not pressed) and a second reading that was false/0 (button pressed). Likewise if the button was released it would first read false/0 (button

pressed) and then read true/1 (button released). The if statement checks for both these cases and prints out a message!

Try combining both of the examples in this guide to light up a LED when a button is pressed. As a hint look at the button code above and think about how you might add extra code to turn a LED on or off along with printing messages.

That's all there is to reading a pushbutton with a digital input using MicroPython! Digital inputs are great for reading buttons, switches (), and simple digital devices like a PIR motion sensor (http://adafru.it/189) or magnetic door open sensor (http://adafru.it/375).