

GPIO: LEDs and Buttons

Your Raspberry Pi is more than just a small computer; it is a hardware prototyping tool! The RPi has **bi-directional I/O pins**, which you can use to drive LEDs, spin motors, or read button presses. To drive the RPi's I/O lines requires a bit of programming. You can use a [variety of programming languages](#), but we decided to use a really solid, easy tools for driving I/O: **Python**.

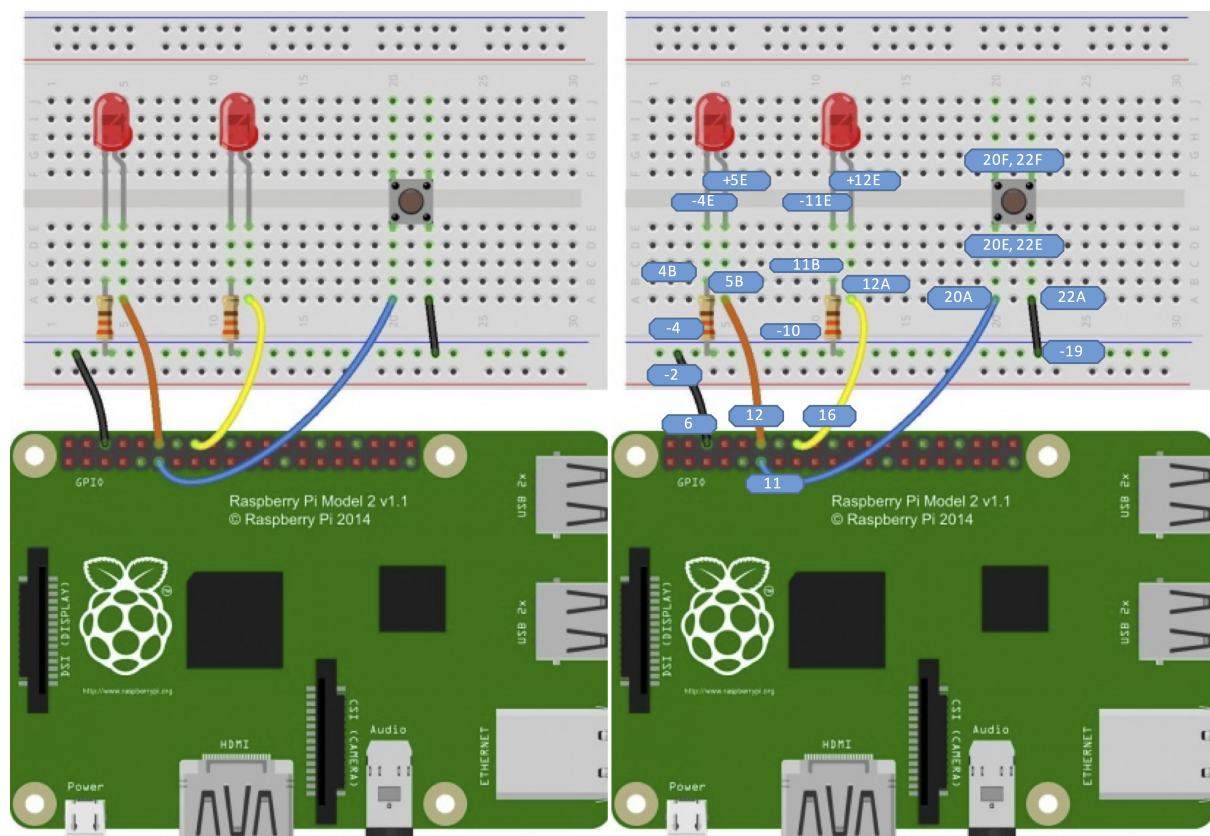
Material needed

- Raspberry Pi 3 B
- Breadboard
- Jumper Wires(M/F)
- Momentary Pushbutton Switch
- 2 Resistors
- 2 LEDs

If you need to remember more about the GPIO Pinout, just got to the [introduction notes for RPi](#).

Hardware Setup

We start assembling the circuit as shown in the diagram bellow. We will use two LEDs to test the output functionality (digital and PWM-Pulse-width Modulation), and a button to test the input.



In the next table you will see which RPi's pins we are suing:

Leds

Broadcom chip-specific numbers	P1 Pin Number
GPIO 18	12
GPIO 23	16

Button

Broadcom chip-specific numbers	P1 Pin Number
GPIO 17	11

Ground

Broadcom chip-specific numbers	P1 Pin Number
Ground	6

Python (RPi.GPIO) API Example

We will use the **RPi.GPIO module** as the driving force behind our Python examples. These Python files and source is included with Raspbian, so assuming you are running the latest Linux distribution, you do not need to download anything to get started. Let's see an example:

1. From your terminal in your laptop connect to your RPi.
2. Create a folder call "code", then a file call "blinker.py":

```
$ mkdir code
$ cd code
$ touch blinker.py
```

3. Then we open it with our text editor:

```
$ nano blinker.py
```

4. Then, copy the next code in your text editor. This code assumes we have set up the circuit as we arranged above.

```
#!/usr/bin/env python

# External module imports GPIO
import RPi.GPIO as GPIO
# Library to slow or give a rest to the script
```

```

import time

# Pin definiton using Broadcom scheme
# PWM (Analog)
pwmPin = 18 # Broadcom pin 18 (P1 pin 12)
# Led
ledPin = 23 # Broadcom pin 23 (P1 pin 16)
# Button
butPin = 17 # Broadcom pin 17 (P1 pin 11)

dc = 95 # duty cycle (0 i.e 0%/LOW and 100 ie.e 100%/HIGH) for PWM pin

# Pin Setup:
GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output
GPIO.setup(pwmPin, GPIO.OUT) # PWM pin set as output
# PWM (Analog) Output
pwm = GPIO.PWM(pwmPin, 50) # Initialize PWM on pwmPin 100Hz frequency
# Button pin set as input w/ pull-up resistors
GPIO.setup(butPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Initial state for LEDs:
GPIO.output(ledPin, GPIO.LOW)
# This function set an initial value of the frequency
pwm.start(dc)

print("Here we go! Press CTRL+C to exit")
try:
    while 1:
        # The input() function will return either a True or False
        # indicating whether the pin is HIGH or LOW.
        if GPIO.input(butPin): # button is released
            pwm.ChangeDutyCycle(dc) # Adjust the value of the PWM output
            GPIO.output(ledPin, GPIO.LOW)
        else: # button is pressed:
            pwm.ChangeDutyCycle(100-dc)
            GPIO.output(ledPin, GPIO.HIGH)
            # Delay of 75 milliseconds
            time.sleep(0.075)
            GPIO.output(ledPin, GPIO.LOW)
            # Delay of 75 milliseconds
            time.sleep(0.075)
except KeyboardInterrupt: # If CTRL+C is pressed, exit cleanly:
    pwm.stop() # stop PWM
    GPIO.cleanup() # cleanup all GPIO

```

5. Running the script needs administrator privileges because of the RPi.GPIO module requires it. So we run the following commands:

To make the script an executable:

```
$ sudo chmod u+x blinker.py
```

```
$ ./blinker.py
```

With the code running, press the button to turn on the digital LED. The PWM-ing LED will invert its brightness when you press the button as well.

Python (RPi.GPIO) API: Overview of the basic function calls used in our example.

Setup Section

- When we use Python to control our GPIO pins, we always need to import the corresponding Python module, which goes at the top of the script:

```
import RPi.GPIO as GPIO
```

In here, we are giving a shorter name to the module "GPIO", to call the module through our script.

- It is important to define which of the two **pin-numbering schemes** you want to use:
 - GPIO.BOARD** – **Board numbering scheme**. The pin numbers follow the **pin numbers on header P1**.
 - GPIO.BCM** – **Broadcom chip-specific pin numbers**. These pin numbers follow the lower-level numbering system defined by the Raspberry Pi's Broadcom-chip brain.

To specify in your code which number-system is being used, use the `GPIO.setmode()` function as:

```
GPIO.setmode(GPIO.BCM)
```

This will activate the Broadcom-chip specific pin numbers.

Setting a Pin Mode

You have to declare a "pin mode" before you can use it as either an input or output. To set a pin mode, use the `setup([pin], [GPIO.IN, GPIO.OUT])` function. So, if you want to set pin 18 (in the BCM) or 12 (in the BOARD) as an output, for example:

```
GPIO.setup(18, GPIO.OUT)
```

Output

Digital Output

To write a pin high or low, use the `GPIO.output([pin], [GPIO.LOW, GPIO.HIGH])` function. For example, if you want to set pin 18 (in the BCM) high:

```
GPIO.output(18, GPIO.HIGH)
```

Writing a pin to `GPIO.HIGH` will drive it to 3.3V, and `GPIO.LOW` will set it to 0V. For the lazy, alternative to `GPIO.HIGH` and `GPIO.LOW`, you can use either `1`, `True`, `0` or `False` to set a pin value.

Pulse-width Modulation (PWM-“Analog”) Output

To initialize PWM, use `GPIO.PWM([pin], [frequency])` function. To make the rest of your script-writing easier, you can assign that instance to a variable. Then use `pwm.start([duty cycle])` function to set an initial value. For example, we can set PWM to pin up with a frequency of 1kHz, and set that output to a 50% duty cycle:

```
pwm = GPIO.PWM(18, 1000)
pwm.start(50)
```

To adjust the value of the PWM output, use the `pwm.ChangeDutyCycle([duty cycle])` function. `[duty cycle]` can be any value between 0 (i.e 0%/LOW) and 100 (i.e 100%/HIGH). So to set a pin to 75% on, for example, you could write:

```
pwm.ChangeDutyCycle(75)
```

To turn PWM on that pin off, use the `pwm.stop()` command. Just don't forget to set the pin as an output before you use it for PWM.

Inputs

If a pin is configured as an input, you can use the `GPIO.input([pin])` function to read its value. The `input()` function will return either a `True` or `False` indicating whether the pin is HIGH or LOW. You can use an if statement to test this. For example, in the next lines of code the GPIO library will read pin 17 (in the BCM) and print whether it is being read as HIGH or LOW:

```
if GPIO.input(17):
    print("Pin 11 is HIGH")
else:
    print("Pin 11 is LOW")
```

Pull-Up/Down Resistors

In the `GPIO.setup()` function, we saw above, where we declared whether a pin was an input or output, we can use a third parameter to set pull-up or pull-down resistors: `pull_up_down=GPIO.PUD_UP` or `pull_up_down=GPIO.PUD_DOWN`. For example, to use a pull-up resistor on GPIO 17 (in the BCM), write this into your setup:

```
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

If nothing is declared in that third value, both pull-resistors will be disabled.

Others

Setting up delays

If you need to slow down your Python script, you can add delays by incorporating the [time module](#) at the top of your script as:

```
import time
```

Then, you can use `time.sleep([seconds])` to give your script a rest. You can use decimals to precisely set your delay. For example, to delay 250 milliseconds, write:

```
time.sleep(0.25)
```

Garbage Collecting

Once your script has run its course, be kind to the next process that might use your GPIOs by cleaning up after yourself. Use the `GPIO.cleanup()` command at the end of your script to release any resources your script may be using. Your RPi will survive if you forget to add this command, but it is good practice to include wherever you can.

Suggested readings

- [Pulse-Width Modulation](#) – You can use PWM to dim LEDs or send signals to servo motors. The RPi has a single PWM-capable pin.
- [Light-Emitting Diodes \(LEDs\)](#) – To test the output capabilities of the Pi we will use some Leds.
- [Switch Basics](#) – To test inputs to the Pi, we will use buttons and switches.
- [Pull-Up Resistors](#) – The Pi has internal pull-up (and pull-down) resistors. These are very handy when you are interfacing buttons with the little computer.

References[1, 2]