Raspberry Pi GPIO. Input and PWM

Introduction to IoT and Cloud Architectures

GITHUB.COM/CLAUDIUGROZA/MSA

Contents

1. Basic input and output

1.1 Digital input read

One of the primary tasks while interacting with Raspberry Pi would be to read some data coming from a physical device connected to the board, e.g. a button or a temperature sensor. The section that follows will exemplify how to read the state of a push-button [Swi].

1.1.1 Building the circuit

An example of wiring a push-button to the board is illustrated by Figure 1.1.

The following list of items are required to build the circuit:

- a small sized breadboard
- one tactile push switch
- two jumper wires
- one 330 Ohm resistor (optional)

You may need the resistor connected between the button and physical pin if the second wire is connected to 3.3V. It will protect your input pin from receiving too much current.

1.1.2 Scripting mode

The previous activity was built around Python's interactive interpreter by showing how to toggle the state of an LED. Next step will focus on how to write our programs in a script and then invoke the Python interpreter to execute it.

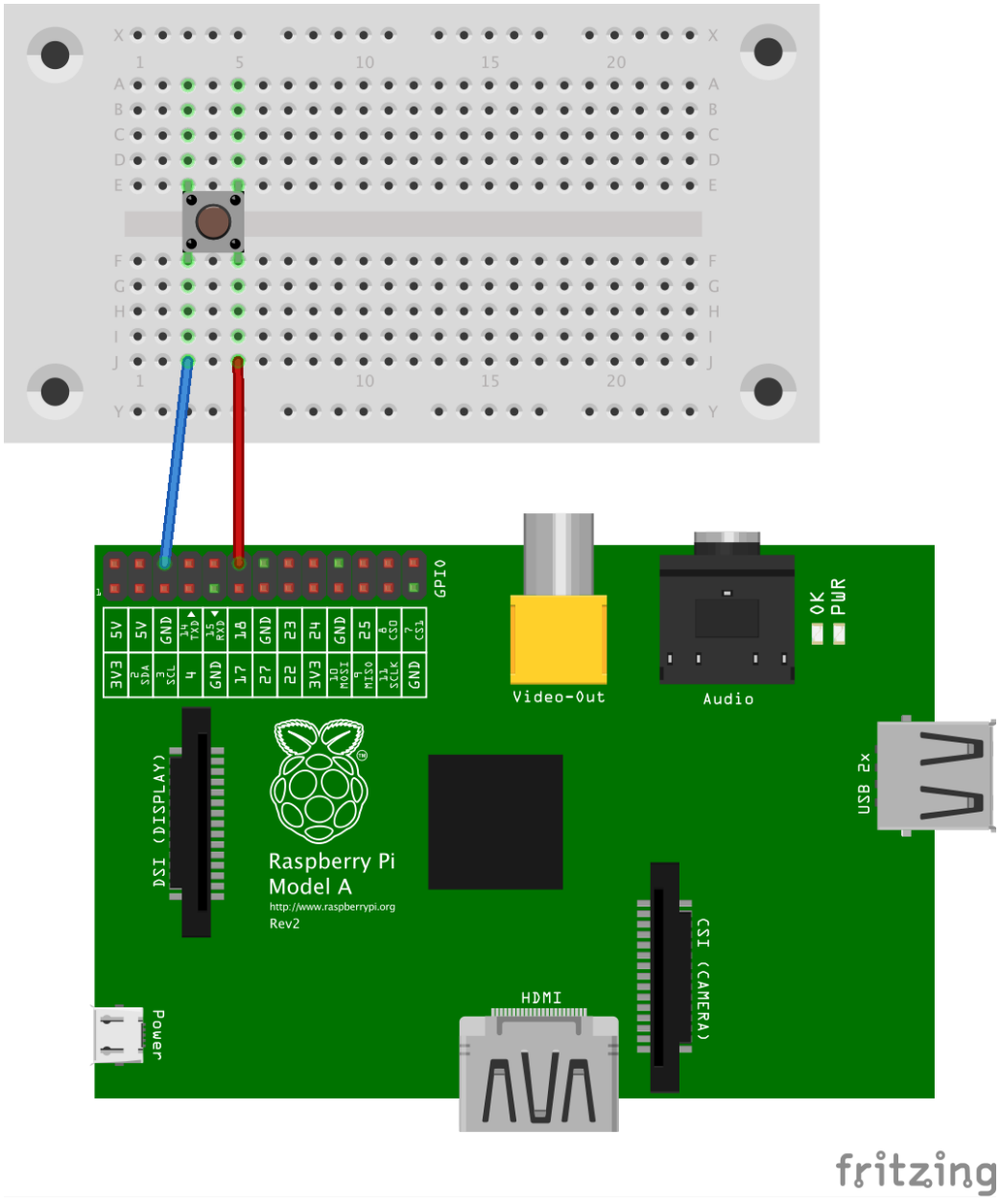

Figure 1.1: Push-button wiring example

Let's start by importing the basic modules and configuring the GPIO:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

You may have noticed that the button is wired so that when it is pressed then it connects *pin 12* to *ground*. We consider that the input pin is "pulled-up" to high state (3.3V) by using the optional argument *GPIO.PUD_UP*. To state that simpler, once the button is pressed then the digital reading of that pin returns *FALSE*.

In case that *pull_up_down* argument is omitted then no internal resistor will be enabled. It is known as "floating" state and it means the voltage can be unpredictable.

The following line will execute a digital read on *pin 12*:

```
state = GPIO.input(12)
```

In order to be efficient in logging different values, we can simply use the print function:

```
print state
```

The last two lines of code presented a one time reading. Our applications will mostly need continuous status monitoring. A primitive solution would be to embed the reading code into a loop block:

```
while True:
      print GPIO.input(12)
```

Our example is complete so the next step is to summon Python interpreter giving the script name as parameter:

```
$ sudo python push_01.py
```

Remember to always run your programs as root whenever you make use of the GPIO layer.

1.1.3 Event subscription

The former approach requires to periodically check the state of the button. We now intend to react only when the state of the button has changed. For achieving this, the edge detection capability comes to our help.

The following block defines a function which serves as a callback when an event is detected:

```
def button_callback(pin):
      print 'Button event'
```

Let's subscribe our previously defined callback for edge detection events on *pin 12*:

```
GPIO.add_event_detect(12, GPIO.BOTH, button_callback)
```

The given *GPIO.BOTH* argument will set a subscription which listens for both *GPIO.RISING* and *GPIO.FALLING* events.

The code section bellow is an extension of the code snippets we defined previously. In addition, it assures us of always releasing the resources used by GPIO library.

```
def button_callback(pin):
    print 'Event occurred on pin ' + str(pin)

# apply 100ms debounce
GPIO.add_event_detect(12, GPIO.BOTH, callback=button_callback, bouncetime=100)

try:
    print 'Start listening'
    # listening 20 seconds for events to be fired
    time.sleep(20)
finally:
    print 'Stop listening'
    GPIO.cleanup()
```

So what is really different from our first approach? Besides the event subscription fact, the moment we register for an event, the GPIO module will create a fresh thread where the **listening** will actually take place. Once an event is fired, the registered callback method will be **notified**.

1.2 Pulse width modulation (PWM)

Doing a quick recap, PWM is related to the concept of a signal being switched relatively quick between on and off. Two important parameters define an instance of PWM: frequency and duty cycle. The next section does not concentrate on the **what** description, but rather on **how** to control these parameters in applications. Our next example captures the variation of an LED brightness.

As usually, we need to set the direction of *pin 12* and then instantiate a PWM object [Pwm] that uses the same pin number. The instance will be timed to a 50 Hz frequency.

```
GPIO.setup(12, GPIO.OUT)
pwm = GPIO.PWM(12, 50)
```

The *start* function instructs the PWM instance to output a signal with 50% duty cycle:

```
pwm.start(50)
```

In order to alter signal's parameters the following methods can be used:

```
pwm.ChangeDutyCycle(duty_cycle)
pwm.ChangeFrequency(frequency)
```

2. Assignments

1. Write a script that controls the state of an LED using a push-button. The LED should change its state only when the button is pressed *(warning: it should not react to button release)*. The detection of an event is confined to the **main thread**.
2. Write a new script which should accomplish the same result as **1)**, but now the detection will be triggered from a **separate thread** (using callback events).
3. Write a new script that will increase or decrease the brightness of an LED when a button event is triggered *(note: it should not react to button release)*.
4. Write a new script that makes an LED to pulse: increase the brightness to a maximum then decrease the brightness to a minimum.
5. Write a new script that takes three arguments as input [Arg]. The program should execute sequences of LED pulses using the previous given configuration: the first argument is the number of pulses contained by one sequence, the second argument is the interval (in seconds) between two pulses in a sequence and the third argument is the pause period (in seconds) between two consecutive sequences.

3. Bibliography

3.1 References

[Arg] *Command line arguments documentation*. https://docs.python.org/2.7/library/sys.html. [Online; accessed March-2018]. 2018 (cited on page 11).

[Swi] *GPIO tutorial*. https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/robot/buttons_and_switches. [Online; accessed March-2018]. 2018 (cited on page 5).

[Pwm] *PWM documentation*. http://pythonhosted.org/RPIO/pwm_py.html. [Online; accessed March-2018]. 2018 (cited on page 8).

3.2 Image credits

- First page illustration.
  http://www.northeastern.edu/levelblog/2018/01/25/guide-iot-careers
- Chapter header background.
  https://blogs.microsoft.com/iot/2015/03/17/simplifying-iot/
- Circuit schematics generated with Fritzing.
  http://fritzing.org