raspberry-gpio-python (/p/raspberry-gpio-python/)

A Python module to control the GPIO on a Raspberry Pi

Brought to you by: croston (/u/croston/)

Summary (/projects/raspberry-gpio-python/) | Files (/projects/raspberry-gpio-python/files/) | Reviews (/projects/raspberry-gpio-python/reviews) | Support (/projects/raspberry-gpio-python/support) | Tickets (/p/raspberry-gpio-python/tickets/) | Wiki (/p/raspberry-gpio-python/wiki/) | Code (/p/raspberry-gpio-python/code/)

Search Wiki

Wiki Home (/p/raspberry-gpio-python/wiki/)

Browse Pages (/p/raspberry-gpio-python/wiki/browse

Browse Labels (/p/raspberry-gpio-python/wiki/browse

Formatting Help (/p/raspberry-gpio-python/wiki/markdo

Inputs (history) (feed) (../search)

Authors: Anonymous (/u/croston/)

Inputs

There are several ways of getting GPIO input into your program. The first and simplest way is to check the input value at a point in time. This is known as 'polling' and can potentially miss an input if your program reads the value at the wrong time. Polling is performed in loops and can potentially be processor intensive. The other way of responding to a GPIO input is using 'interrupts' (edge detection). An edge is the name of a transition from HIGH to LOW (falling edge) or LOW to HIGH (rising edge).

Pull up / Pull down resistors

If you do not have the input pin connected to anything, it will 'float'. In other words, the value that is read in is undefined because it is not connected to anything until you press a button or switch. It will probably change value a lot as a result of receiving mains interference.

To get round this, we use a pull up or a pull down resistor. In this way, the default value of the input can be set. It is possible to have pull up/down resistors in hardware and using software. In hardware, a 10K resistor between the input channel and 3.3V (pull-up) or 0V (pull-down) is commonly used. The RPi.GPIO module allows you to configure the Broadcom SOC to do this in software:

```
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_UP)
  # or
GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

(where channel is the channel number based on the numbering system you have specified - BOARD or BCM).

Testing inputs (polling)

You can take a snapshot of an input at a moment in time:

```
if GPIO.input(channel):
    print('Input was HIGH')
else:
    print('Input was LOW')
```

To wait for a button press by polling in a loop:

```
while GPIO.input(channel) == GPIO.LOW:
    time.sleep(0.01)  # wait 10 ms to give CPU chance to do other things
```

(this assumes that pressing the button changes the input from LOW to HIGH)

Interrupts and Edge detection

An edge is the change in state of an electrical signal from LOW to HIGH (rising edge) or from HIGH to LOW (falling edge). Quite often, we are more concerned by a change in state of an input than it's value. This change in state is an *event*.

To avoid missing a button press while your program is busy doing something else, there are two ways to get round this:

- the wait_for_edge() function
- the event_detected() function
- a threaded callback function that is run when an edge is detected

wait_for_edge() function

The wait_for_edge() function is designed to block execution of your program until an edge is detected. In other words, the example above that waits for a button press could be rewritten as:

```
GPIO.wait_for_edge(channel, GPIO.RISING)
```

Note that you can detect edges of type GPIO.RISING, GPIO.FALLING or GPIO.BOTH. The advantage of doing it this way is that it uses a negligible amount of CPU, so there is plenty left for other tasks.

event_detected() function

The event_detected() function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy working on other things. This could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis.

```
GPIO.add_event_detect(channel, GPIO.RISING)  # add rising edge detection on a channel
do_something()
if GPIO.event_detected(channel):
    print('Button pressed')
```

Note that you can detect events for GPIO.RISING, GPIO.FALLING or GPIO.BOTH.

Threaded callbacks

RPi.GPIO runs a second thread for callback functions. This means that callback functions can be run at the same time as your main program, in immediate response to an edge. For example:

```
def my_callback(channel):
    print('This is a edge event callback function!')
    print('Edge detected on channel %s'%channel)
    print('This is run in a different thread to your main program')

GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback)  # add rising edge detection o
...the rest of your program...
```

If you wanted more than one callback function:

```
def my_callback_one(channel):
    print('Callback one')

def my_callback_two(channel):
    print('Callback two')

GPIO.add_event_detect(channel, GPIO.RISING)
GPIO.add_event_callback(channel, my_callback_one)
GPIO.add_event_callback(channel, my_callback_two)
```

Note that in this case, the callback functions are run sequentially, not concurrently. This is because there is only one thread used for callbacks, in which every callback is run, in the order in which they have been defined.

Switch debounce

You may notice that the callbacks are called more than once for each button press. This is as a result of what is known as 'switch bounce'. There are two ways of dealing with switch bounce:

- add a 0.1uF capacitor across your switch.
- software debouncing
- a combination of both

To debounce using software, add the bouncetime= parameter to a function where you specify a callback function. Bouncetime should be specified in milliseconds. For example:

```
# add rising edge detection on a channel, ignoring further edges for 200ms for switch bounce han
GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)
```

or

```
GPIO.add_event_callback(channel, my_callback, bouncetime=200)
```

Remove event detection

If for some reason, your program no longer wishes to detect edge events, it is possible to stop them:

```
GPIO.remove_event_detect(channel)
```

Related

Wiki: BasicUsage (/p/raspberry-gpio-python/wiki/BasicUsage/)
Wiki: Examples (/p/raspberry-gpio-python/wiki/Examples/)