How to use interrupts with Python on the Raspberry Pi and RPi.GPIO

3 parties,

Première partie

https://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio

The latest big news in the world of Raspberry Pi Python GPIO programming is that Ben Croston has released an update for RPi.GPIO. Why is that a big deal? Because this version has interrupts. "What's an interrupt?" I hear you say. It's a way of waiting for something to happen without checking constantly whether or not it's happening.

Imagine that you're waiting for a delivery – something you're really excited about – like a Pi camera.You spend far too much time looking down the street in eager anticipation of the postman's arrival. You can't fully focus on what you're supposed to be doing because you know it's imminent. Every time you hear something in the street, you jump up and look out of the window. Woe betide any door-knocking salesman who calls when you're expecting a delivery.

What I've just described in human terms is a bit like polling. Polling is continually checking for something. For example, if you want to make sure your program reacts as quickly as possible to a button press, you can check the button status about ten thousand times per second. This is great if you need a quick reaction, but it uses quite a bit of the computer's processing power. In the same way that you can't fully focus when you're expecting a delivery, a large part of your CPU is used up with this polling.

There has to be a better way, right?

Yes. And there is. It's interrupts. This is the first in a series of articles which aim to show you how to use this new interrupt facility in Python.

Interrupts are a much more efficient way of handling the "wait for something to happen and react immediately when it does" situation. They free up the resources you would have wasted on polling, so that you can use them for something else. Then, when the event happens, the rest of your program is "interrupted" and your chosen outcome occurs.

So, to carry on our human example…
An interrupt is like having an automatic postman detector that will tell you for sure when the postman arrives, so you can get on with something else. You now know you will not miss that knock on the door and end up with one of those "we tried to deliver your item but you were out and the collection office is closed for the next two days, so enjoy the wait" cards.

So interrupts are good, as you can set them up to wait for events to occur without wasting system resources.

So how do you code them?

I'm going to show a simple "wait for a button press" example in this blog article and follow up with other examples in subsequent articles. But before you try this, you will quite likely need to update your RPi.GPIO package. You can check what version of RPi.GPIO you have in the command line with…

```
sudo python
import RPi.GPIO as GPIO
GPIO.VERSION
```

This should show you what RPi.GPIO version you have. You need 0.5.1 or higher for this example.
You can exit the python environment with `CTRL+Z`

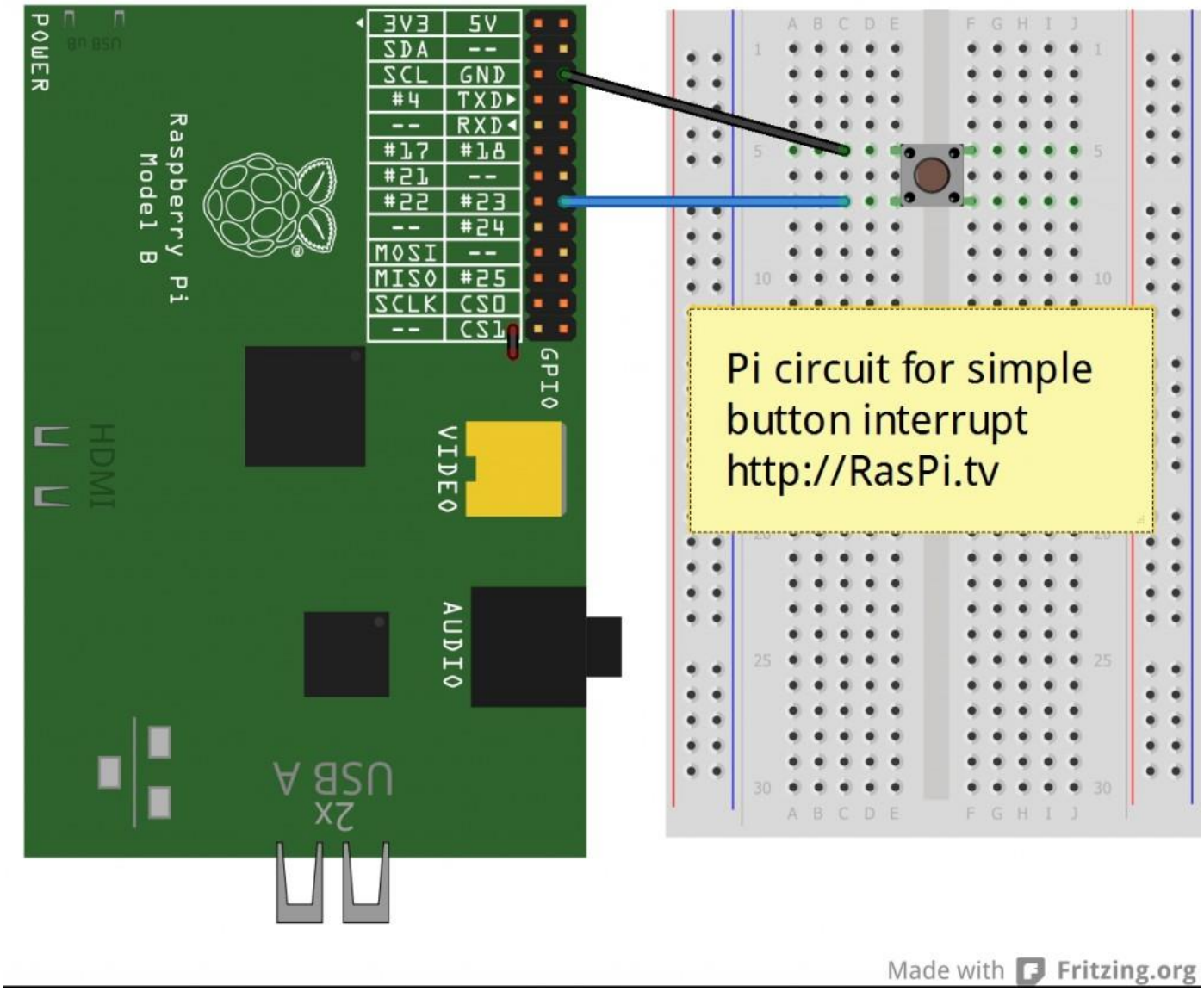
~~**Install RPi.GPIO version 0.5.1 for simple interrupts**~~

~~If you need to, you can install 0.5.1 or later with~~
~~sudo apt-get update~~
~~sudo apt-get dist-upgrade (This will update all your Raspbian packages and may take up to an hour)~~

~~or, from the command line prompt (this will only update RPi.GPIO)…~~

~~wget http://raspberry-gpio-python.googlecode.com/files/python-rpi.gpio_0.5.1a-1_armhf.deb~~
~~wget http://raspberry-gpio-python.googlecode.com/files/python3-rpi.gpio_0.5.1a-1_armhf.deb~~
~~sudo dpkg -i python-rpi.gpio_0.5.1a-1_armhf.deb~~
~~sudo dpkg -i python3-rpi.gpio_0.5.1a-1_armhf.deb~~

And now the circuit



Circuit for simple button press interrupt

It's simply a question of rigging up a button connecting 23 to GND when pressed.

And now onto the code

I've put most of the explanation in the code, so that if you use it, you will still have it.

view plaincopy to clipboardprint?

```
1. #!/usr/bin/env python2.7
2. # script by Alex Eames https://raspi.tv/
3. # https://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-
   gpio
4. import RPi.GPIO as GPIO
5. GPIO.setmode(GPIO.BCM)
6.
```

```
7.  # GPIO 23 set up as input. It is pulled up to stop false signals
8.  GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
9.
10. print "Make sure you have a button connected so that when pressed"
11. print "it will connect GPIO port 23 (pin 16) to GND (pin 6)\n"
12. raw_input("Press Enter when ready\n>")
13.
14. print "Waiting for falling edge on port 23"
15. # now the program will do nothing until the signal on port 23
16. # starts to fall towards zero. This is why we used the pullup
17. # to keep the signal high and prevent a false interrupt
18.
19. print "During this waiting time, your computer is not"
20. print "wasting resources by polling for a button press.\n"
21. print "Press your button when ready to initiate a falling edge interrupt."
22. try:
23.     GPIO.wait_for_edge(23, GPIO.FALLING)
24.     print "\nFalling edge detected. Now your program can continue with"
25.     print "whatever was waiting for a button press."
26. except KeyboardInterrupt:
27.     GPIO.cleanup()      # clean up GPIO on CTRL+C exit
28. GPIO.cleanup()          # clean up GPIO on normal exit
```

Decoding the code

lines 4-5 import the RPi.GPIO module and set up the BCM port numbering scheme

line 8 sets GPIO 23 as an input with the pullup resistor set to UP.
This means that the signal will be HIGH all the time until the button is pressed connecting the port to GND, which makes it LOW. This avoids false event detection.

lines 10-11 print some instructions
line 12 waits for user to hit enter before starting. This gives an opportunity to check the wiring

lines 14-21 further instructions and documentation

line 22 `try:` & line 26 `except KeyboardInterrupt:`
This allows us to run the program and exit cleanly if someone presses `CTRL-C` to stop the program. If we didn't do this, the ports would still be set when we forcefully exit the program.

line 23 sets up the "wait for the signal on port 23 to start falling towards 0"

lines 24-25 further on-screen instructions

line 27 cleans up the GPIO ports we've used during this program when `CTRL-C` is pressed
line 28 cleans up the GPIO ports we've used during this program when the program exits normally

Two ways to get the above code on your Pi

If you are in the command line on your Pi, type…

```
nano interrupt1.py
```

Then click "copy to clipboard" (above) and paste into the nano window.

```
CTRL+O
Enter
CTRL+X
```

Alternatively, you can download this directly to your Pi using…

```
wget https://raspi.tv/download/interrupt1.py.gz
gunzip interrupt1.py.gz
```

Then you can run it with…

```
sudo python interrupt1.py
```

That's cool, what next?

So that was a simple "wait for a button press" interrupt. There's a lot more you can do with them, as I will show you in the next article, which will cover "threaded callback", which allows us to use the spare capacity we've freed up by not polling continually.

Click here for the next article (part 2)
Or check out the official documentation here and press on by yourself.

Seconde partie

https://raspi.tv/2013/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio-part-2

Interrupts are an efficient way for a program to be able to respond immediately to a specific event. In the previous article I explained the basics of using interrupts in RPi.GPIO and gave an example of a simple "wait for an event" interrupt program.

In this second article I will introduce "threaded callback" which opens up a lot of new possibilities.

Threaded callback – what the heck is that?

I know it sounds complicated. And it probably is complicated in the C code it's written in, but we're Pythonites and we don't have to go there. ;)

If you remember the previous example program was just a simple "wait for port 23 to be connected to GND when we press the button and then print a message and exit the program".

So, while it was waiting, the program wasn't doing anything else. The program only had one thread, which means only one thing was being done at once. Python is capable of running more than one thread at once. It's called multi-threading. It means that you can go through more than one piece of code simultaneously. This is where we can reap the benefit of interrupts because we can do something else while we wait for our "event" to happen. (Just like your "postman detector" allowed you to get on with something else instead of being distracted by waiting for the mail.)

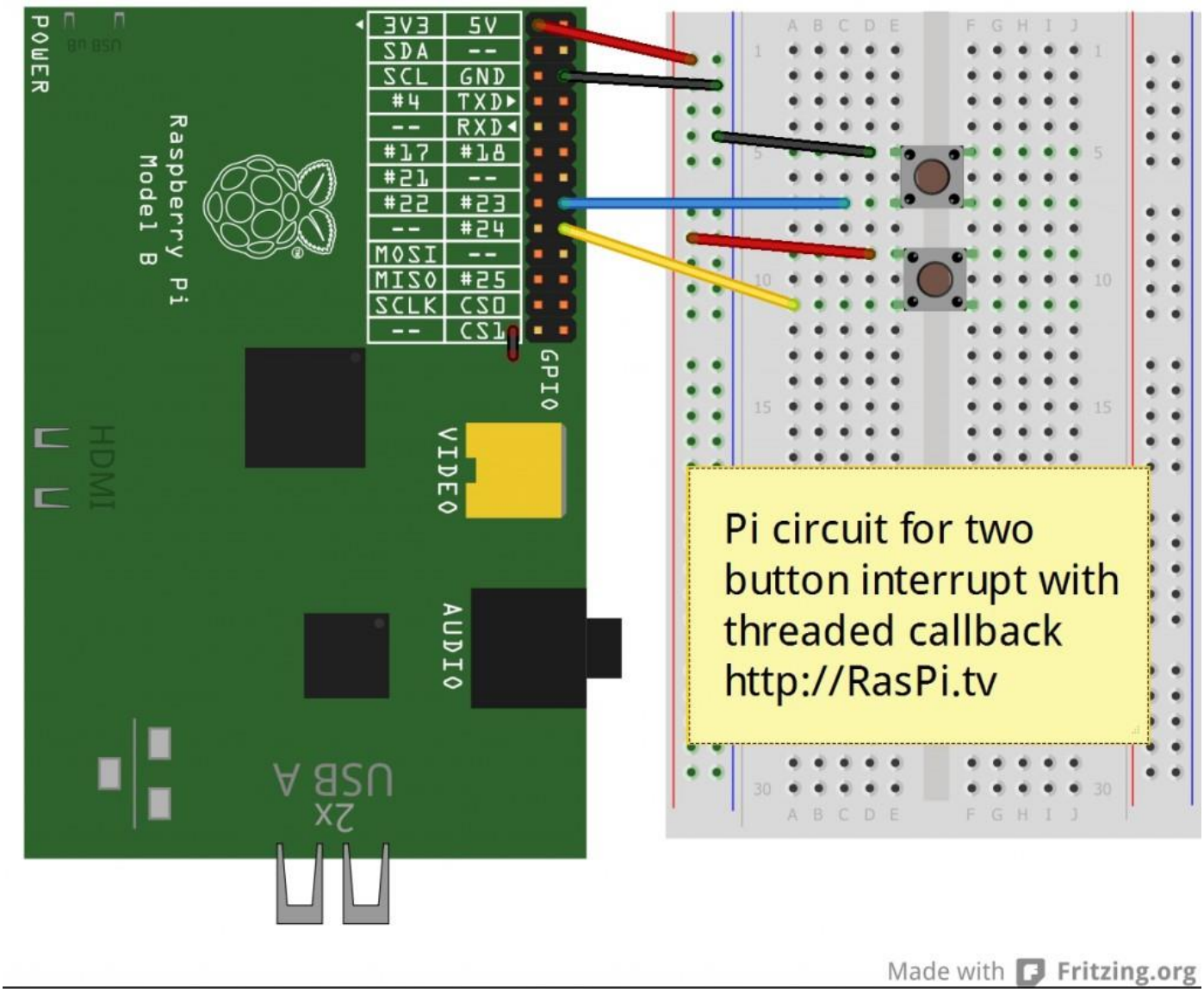
So that covers the threading part of threaded callback. What's a callback?
When an event is detected in the second thread, it communicates this back to the main thread (calls back). What we now have in RPi.GPIO is the ability to start a new thread for an interrupt and specify a set of instructions (function) that will run when the interrupt occurs in the second thread. This is a **threaded callback function.**

This is like your "postman detector" giving you a list of reminders of things you wanted to do when your delivery arrives AND doing them for you, so you can carry on with what you want to be doing.

So What are we going to do now?

We'll keep most of what we did before and add another button and an event detect threaded callback that runs when the new button is pressed, even though we are still waiting for the first button to be pressed.

But this time, the new button will connect GPIO port 24 to 3.3V (3V3) when pressed. This will allow us to demonstrate a rising edge detection. So we'll be setting up port 24 with the built in pulldown resistor enabled.



Circuit for second interrupt experiment

Later on we'll have to modify the code to cope with 'button bounce', but we won't say any more about that just yet.

Do you need to update RPi.GPIO?

If you didn't do it for the first example, you will quite likely need to update your RPi.GPIO package. You can check what version of RPi.GPIO you have in the command line with…

```
sudo python
import RPi.GPIO as GPIO
GPIO.VERSION
```

This should show you what RPi.GPIO version you have. You need 0.5.2a or higher for this example.
You can exit the python environment with `CTRL+Z`

Install RPi.GPIO version 0.5.2 for simple interrupts

If you need to, you can install 0.5.2 or later with
`sudo apt-get update`
`sudo apt-get dist-upgrade` (This will update all your Raspbian packages and may take up to an hour)

Update July 2014
The best way to get the latest RPi.GPIO (currently 0.5.5) is to flash a new SD card with the latest NOOBS or Raspbian. This will give you a clean start with the latest version of RPi.GPIO.

And now onto the code

I've put most of the explanation in the code, so that if you use it, you will still have it.

view plaincopy to clipboardprint?

```
1.  #!/usr/bin/env python2.7
2.  # script by Alex Eames https://raspi.tv
3.
4.  import RPi.GPIO as GPIO
5.  GPIO.setmode(GPIO.BCM)
6.
7.  # GPIO 23 & 24 set up as inputs. One pulled up, the other down.
8.  # 23 will go to GND when button pressed and 24 will go to 3V3 (3.3V)
9.  # this enables us to demonstrate both rising and falling edge detection
10. GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
11. GPIO.setup(24, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
12.
13. # now we'll define the threaded callback function
14. # this will run in another thread when our event is detected
15. def my_callback(channel):
16.     print "Rising edge detected on port 24 - even though, in the main thread,"
17.     print "we are still waiting for a falling edge - how cool?\n"
18.
19. print "Make sure you have a button connected so that when pressed"
20. print "it will connect GPIO port 23 (pin 16) to GND (pin 6)\n"
21. print "You will also need a second button connected so that when pressed"
22. print "it will connect GPIO port 24 (pin 18) to 3V3 (pin 1)"
23. raw_input("Press Enter when ready\n>")
24.
25. # The GPIO.add_event_detect() line below set things up so that
26. # when a rising edge is detected on port 24, regardless of whatever
27. # else is happening in the program, the function "my_callback" will be run
28. # It will happen even while the program is waiting for
29. # a falling edge on the other button.
30. GPIO.add_event_detect(24, GPIO.RISING, callback=my_callback)
31.
32. try:
33.     print "Waiting for falling edge on port 23"
```

```
34.     GPIO.wait_for_edge(23, GPIO.FALLING)
35.     print "Falling edge detected. Here endeth the second lesson."
36.
37. except KeyboardInterrupt:
38.     GPIO.cleanup()       # clean up GPIO on CTRL+C exit
39. GPIO.cleanup()           # clean up GPIO on normal exit
```

Two ways to get the above code on your Pi

If you are in the command line on your Pi, type…

```
nano interrupt2.py
```

Then click "copy to clipboard" (above) and paste into the nano window. Then

```
CTRL+O
Enter
CTRL+X
```

Alternatively, you can download this directly to your Pi using…

```
wget https://raspi.tv/download/interrupt2.py.gz
gunzip interrupt2.py.gz
```

Then you can run it with…

```
sudo python interrupt2.py
```

What's supposed to happen?

When you run the code it gives you a message "Waiting for falling edge on port 23"
If you press button 1, it will terminate the program as before and give you a message "Falling edge detected."

If, instead of button 1, you press button 2, you'll get a message
"Rising edge detected on port 24".

This will occur as many times as you press the button. The program is still waiting for the original falling edge on port 23, and it won't terminate until it gets it. Because your second button press is detected in another thread, it doesn't affect the main thread.

You may also notice, depending on how cleanly you press the second button, that sometimes you get more than one message for just one button press. This is called "switch bounce".

Bouncy, bouncy, bouncy

When you press a button switch, the springy contacts may flex and rapidly make and break contact one or more times. This may cause more than one edge detection to trigger, so you may get more than one message for one button press. There is, of course, a way round it, in software.

Why didn't this happen before?

I hear you ask. The answer is simple. Last time the program was simply waiting for a single button press. As soon as that button press was detected, it stopped waiting. So if the switch bounced, it was ignored. The program had already moved on. In our case, it had closed. But,

when the event detection is running constantly in another thread, this is not the case and we actually need to slow things down a bit in what is called “software debouncing”.

How to do software debouncing

In order to debounce, we need to be able to measure time intervals. To do that, we use the time module. Near the top of the program we need to add a line…

`import time`. I add it immediately after the RPi.GPIO import

view plaincopy to clipboardprint?

```
4. import RPi.GPIO as GPIO
5. import time
6. GPIO.setmode(GPIO.BCM)
```

Then, also quite near the top of the program we need to set an intial value for the variable time_stamp that we will use to measure time intervals

`time_stamp = time.time()` I put this after the GPIO.setup commands. This sets time_stamp equal to the time in seconds right now.

view plaincopy to clipboardprint?

```
7. GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
8. GPIO.setup(24, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
9. time_stamp = time.time()
```

Now we have to change our threaded callback function from

view plaincopy to clipboardprint?

```
15. def my_callback(channel):
16.     print "Rising edge detected on port 24 - even though, in the main thread,"
17.     print "we are still waiting for a falling edge - how cool?\n"
```

to..

view plaincopy to clipboardprint?

```
15. def my_callback(channel):
16.     global time_stamp      # put in to debounce
17.     time_now = time.time()
18.     if (time_now - time_stamp) >= 0.3:
19.         print "Rising edge detected on port 24 - even though, in the main thread,"
20.         print "we are still waiting for a falling edge - how cool?\n"
21.     time_stamp = time_now
```

And now, even if you deliberately press the button twice, as long as you do it within 0.3 seconds it will only register one button press. You have debounced the switch, by forcing the program to ignore a button press if it occurs less than 0.3 seconds after the previous one.

If you are IDLE (Python joke) you can get the amended code here…

```
wget https://raspi.tv/download/interrupt2a.py.gz
gunzip interrupt2a.py.gz
```

How does this work?

`time_now = time.time()` stores the current time in seconds in the variable time_now. Now look at the last line of the function.
`time_stamp = time_now` This stores the time in seconds when the function was started in a global variable called `time_stamp` So next time this function is called, it will be able to check how much time has elapsed since the last time.

That's what is happening in the if statement.
`if (time_now - time_stamp) >= 0.3:` In English this means "If more than 0.3 seconds has elapsed since the last button press, execute the code in the indented block".

So if more than 0.3 seconds have elapsed it would print the message…
"Rising edge detected on port 24 – even though, in the main thread we are still waiting for a falling edge – how cool?"

If less than 0.3 seconds has elapsed, it will do nothing. Job done. Button switch is debounced.

Update – RPi.GPIO 0.5.2 onwards includes this debounce algorithm

Ben has included the above debounce algorithm in 0.5.2 onwards. This article was originally written for 0.5.1. So the above debounce code has been superseded by adding `bouncetime=xxx`, where xxx is a time in milliseconds. e.g.

```
GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback,
bouncetime=200)
```

I will update the next example to reflect this.

So, what's next?

1. We've learnt about simple "wait for" interrupts in the previous article
2. We've covered threaded callback in this article.
3. In the next article, we'll go over the situation where you want to do more than one threaded callback interrupt at once.

If you can't wait for the next article (coming soon to a blog near you) check out the documentation here and press on by yourself.

I hope you are enjoying this series. Click here for Part 3.

Troisième partie

Multiple threaded callback interrupts in Python

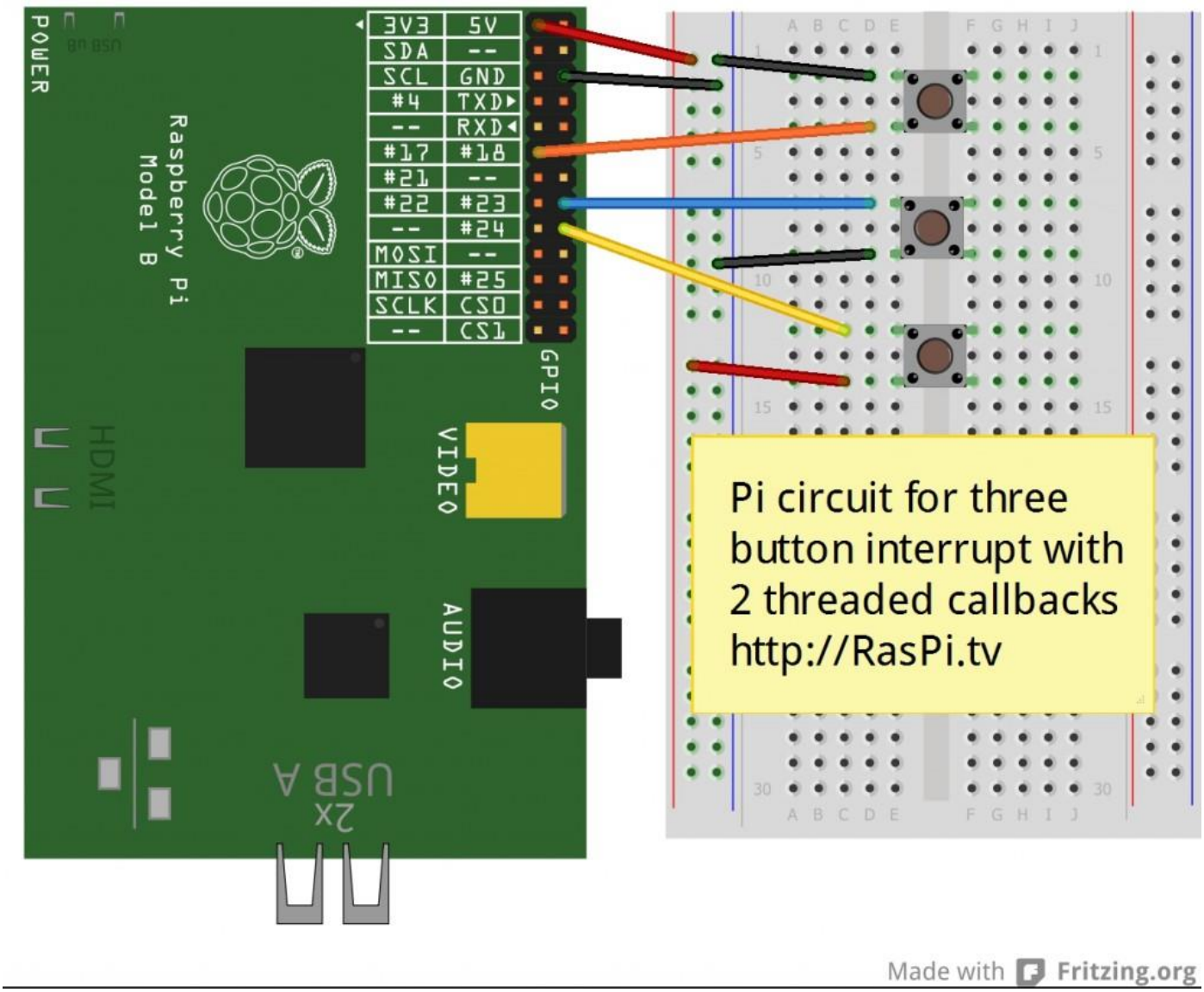
We've been learning about interrupts this week because of the brand new interrupt capabilities of RPi.GPIO. We covered a simple "wait for" interrupt in part 1, threaded callback interrupt and button debouncing in part 2 and today we're getting sophisticated with multiple threaded callbacks. "WooooooooOOOOOOOOOOOOOOooooooooooo", I hear you say. ;)

Well actually, we're not doing much that's very different from last time, except, now there's more of it. We'll add another button and another threaded callback function the same as the first one (but on a different GPIO port). This is just to show that you can do multiple threaded callbacks in one program. After that, your imagination is the limit. (Well actually the number of GPIO ports is probably the limit.)

We're just building on what we did before and this is exactly how programs are made. You do a bit at a time, test it, fix it, make sure it does what it ought to do, then go on to the next bit.

Here's the Circuit

This circuit is a bit different from the previous one. The top two buttons connect port 17 and port 23 to GND when pressed. These are the two which trigger callbacks. The bottom button, connecting port 24 to 3V3 on button press is the "wait for" interrupt this time. So when you press button 3 it's "game over", but buttons 1 and 2 just report that they've been pressed until button 3 is eventually pressed.



Circuit for 2 threaded callbacks and one wait interrupt

We've used all the same building blocks we developed in parts 1 and 2, including button debouncing.

Do you need to update RPi.GPIO?

If you didn't do it for the first or second examples, you will quite likely need to update your RPi.GPIO package. You can check what version of RPi.GPIO you have in the command line with…

```
sudo python
import RPi.GPIO as GPIO
GPIO.VERSION
```

This should show you what RPi.GPIO version you have. You need 0.5.1a or higher for this example.
You can exit the python environment with `CTRL+D`

Install RPi.GPIO version 0.5.2a for multiple threaded callback interrupts

If you need to, you can install 0.5.2a or later with
`sudo apt-get update`
`sudo apt-get upgrade` (This will update all your Raspbian packages and may take up to an hour)

Update July 2014
The best way to get the latest RPi.GPIO (currently 0.5.5) is to flash a new SD card with the latest NOOBS or Raspbian. This will give you a clean start with the latest version of RPi.GPIO.

And now onto the code

I've put most of the explanations in the code, so that if you use it, you will still have them.

view plaincopy to clipboardprint?

```
1.  #!/usr/bin/env python2.7
2.  # script by Alex Eames https://raspi.tv
3.  # https://raspi.tv/how-to-use-interrupts-with-python-on-the-raspberry-pi-and-rpi-gpio-part-
    3
4.  import RPi.GPIO as GPIO
5.  GPIO.setmode(GPIO.BCM)
6.
7.  # GPIO 23 & 17 set up as inputs, pulled up to avoid false detection.
8.  # Both ports are wired to connect to GND on button press.
9.  # So we'll be setting up falling edge detection for both
10. GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
11. GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
12.
13. # GPIO 24 set up as an input, pulled down, connected to 3V3 on button press
14. GPIO.setup(24, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
15.
16. # now we'll define two threaded callback functions
17. # these will run in another thread when our events are detected
18. def my_callback(channel):
19.     print "falling edge detected on 17"
20.
21. def my_callback2(channel):
22.     print "falling edge detected on 23"
23.
24. print "Make sure you have a button connected so that when pressed"
25. print "it will connect GPIO port 23 (pin 16) to GND (pin 6)\n"
26. print "You will also need a second button connected so that when pressed"
27. print "it will connect GPIO port 24 (pin 18) to 3V3 (pin 1)\n"
28. print "You will also need a third button connected so that when pressed"
29. print "it will connect GPIO port 17 (pin 11) to GND (pin 14)"
30. raw_input("Press Enter when ready\n>")
31.
32. # when a falling edge is detected on port 17, regardless of whatever
```

```
33. # else is happening in the program, the function my_callback will be run
34. GPIO.add_event_detect(17, GPIO.FALLING, callback=my_callback, bouncetime=300)
35.
36. # when a falling edge is detected on port 23, regardless of whatever
37. # else is happening in the program, the function my_callback2 will be run
38. # 'bouncetime=300' includes the bounce control written into interrupts2a.py
39. GPIO.add_event_detect(23, GPIO.FALLING, callback=my_callback2, bouncetime=300)
40.
41. try:
42.     print "Waiting for rising edge on port 24"
43.     GPIO.wait_for_edge(24, GPIO.RISING)
44.     print "Rising edge detected on port 24. Here endeth the third lesson."
45.
46. except KeyboardInterrupt:
47.     GPIO.cleanup()      # clean up GPIO on CTRL+C exit
48. GPIO.cleanup()          # clean up GPIO on normal exit
```

Bounce

`bouncetime=300` in lines 34 & 39 sets a time of 300 milliseconds during which time a second button press will be ignored. The code from example 2a has been incorporated into RPi.GPIO.

You can download this directly to your Pi using…

```
wget https://raspi.tv/download/interrupt3.py.gz
gunzip interrupt3.py.gz
```

Then you can run it with…

```
sudo python interrupt3.py
```

Can I switch off an event detection?

There's just one more thing. If you want to stop an event detection on a particular port, you can use the following command…

```
GPIO.remove_event_detect(port_number)
```

You now know all you need to know about how to create and manage interrupts in RPi.GPIO in Python on the Raspberry Pi. The official documentation is here if you want to check it out.

Have fun with interrupts. I hope this mini-series has been useful. Let us know, in the comments below, what you're planning to use it for and then come back and tell us how it went. :)

For extra credit, can you tell me why I chose those specific GPIO ports for these experiments?