Using Interrupt Code - IRQ

We are exploring coding using a Pico and LED strips.

LED strips are a great addition to your home, and great fun to work with.

Today, we explore the problem of dead-end loops in our program.

A program running in a while loop cannot respond to a button, because it is looping.

We can fix this issue using an IRQ = Interrupt Request.

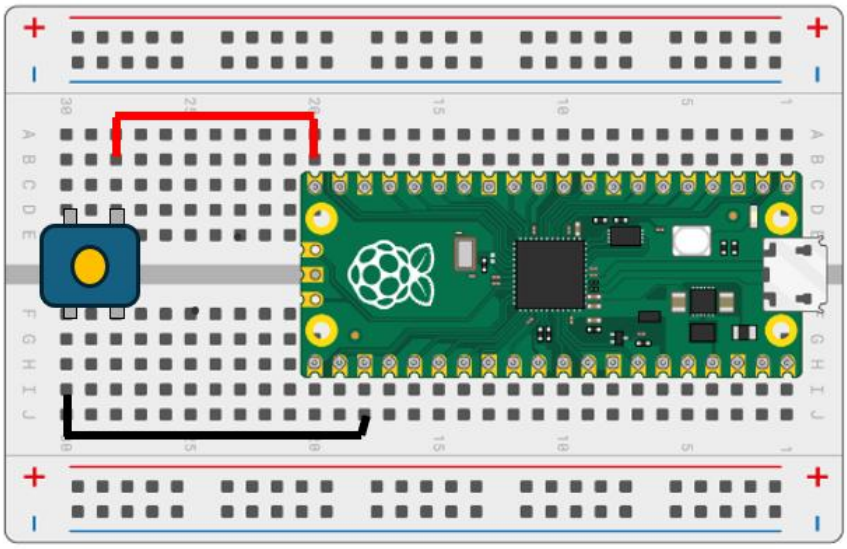
Contents

Hardware Set Up

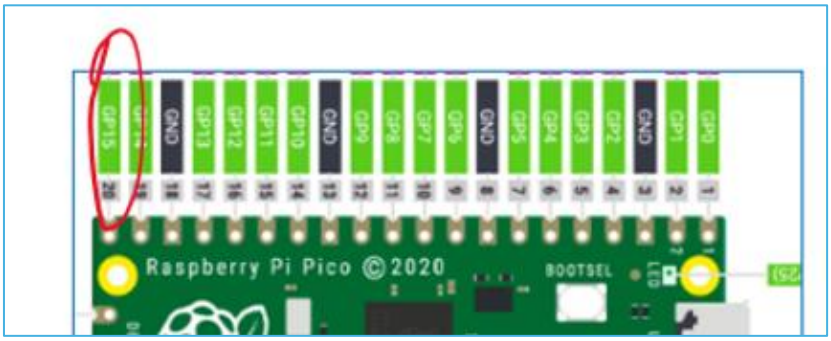
You need a Pico H. A Pico H is a standard Raspberry Pico with headers (legs that stick into the breadboard). The Pico W version includes WiFi capabilities. We are not using this today. You also need a breadboard with a button, and some jumper wires. Have a LED strip ready for the second part of this guide.

You will need a computer with an IDE app installed such as Thonny.

Wire the button as follows:



Note that we are connecting the button to Pico pin 15 (top left)



Software Setup

Our Pico H has micro python installed.

You will need the Neopixel library as well for the second part of this project. See the "LED strip Intro" guide for details on installing the Neopixel library.

Plug the Pico into the computer using a micro usb cable. Be aware that some micro usb cables provide power only, and cannot be used to send data to the Pico. If you are having problems connecting the Pico to the Thonny IDE app, this could be why.

As always, we run the simplest code possible to check our connections are working.

Check the Pico is connected to the PC.

Check the Thonny app is connected to the Pico (see the drop down in the botton right corner of the app).

Copy the test code below into Thonny and run it.

Test code to check button is connected (not using IRQ interrupt code)

```
from machine import Pin
import time

#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)

#commands
while True:
  if button.value() == 0:
     print("Button is Pressed")
  else:
     print("Button is not Pressed")
  time.sleep(0.1)
```

The code is in a while loop.

It constantly checks whether the button is pressed.

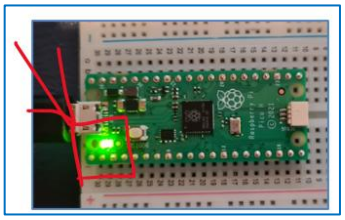
If pin 15 receives a signal, the "button is pressed" text is output.

```
1  from machine import Pin
2  import time
3
4  #button variable
5  button = Pin(15, Pin.IN, Pin.PULL_UP)
6
7  #commands
8  while True:
9     if button.value() == 0:
10        print("Button is Pressed")
11    else:
12        print("Button is not Pressed")
13    time.sleep(0.1)
14
```

Challenge - develop this to make the green on-board LED light up.

Can you make the green light on the Pico light up when the button is pressed?

Solution



A variable for the on-board green LED is assigned on line 6.

Line 12 turns the LED on

Line 15 turns the LED off.

```
1  from machine import Pin
2  import time
3  #button variable
4  button = Pin(15, Pin.IN, Pin.PULL_UP)
5  #On-Board LED
6  led_pin = Pin(25, Pin.OUT)
7
8  #commands
9  while True:
10     if button.value() == 0:
11         print("Button is Pressed")
12         led_pin.value(1)
13     else:
14         print("Button is not Pressed")
15         led_pin.value(0)
16     time.sleep(0.1)
```

Using IRQ Interrupt code.

Let's make the on-board LED flash in a while loop.

```
1  from machine import Pin
2  import time
3  #button variable
4  button = Pin(15, Pin.IN, Pin.PULL_UP)
5  #On-Board LED
6  led_pin = Pin(25, Pin.OUT)
7
8  #commands
9  while True:
10     led_pin.value(1)
11     time.sleep(1)
12     led_pin.value(0)
13     time.sleep(1)
```

This is an endless loop.

It is not listening out for any inputs.

We can use an IRQ interrupt to stop this endless while loop.

Test code for IRQ Interrupt:

This code will flash the green LED slowly.

When the button is pressed the LED will flash fast.

Copy this code into Thonny and run it.

```
from machine import Pin
import time
#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)
#On-Board LED
led_pin = Pin(25, Pin.OUT)

#commands
def pir_interrupt(pin):
    if pin.value() == 1:
        for x in range(10):
            led_pin.value(1)
            time.sleep(0.2)
            led_pin.value(0)
            time.sleep(0.2)
        else:
            led_pin.value(0)

# Configure the interrupt on the PIR pin for both rising and falling edges
button.irq(trigger=(Pin.IRQ_RISING | Pin.IRQ_FALLING), handler=pir_interrupt)

while True:
    led_pin.value(1)
    time.sleep(1)
    led_pin.value(0)
    time.sleep(1)
```

It is likely you are using a Pico H. But, if your Pico has a big silver chip on it, it is a Pico W. The Pico W codes the on-board green LED slightly differently.

Use this code if you have a Pico W:

```
#Button IRQ interrupt, with Pico W
from machine import Pin
import time
#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)
#On-Board LED
led = machine.Pin("LED", machine.Pin.OUT)

#commands
def pir_interrupt(pin):
    if pin.value() == 1:
        for x in range(10):
            led.on()
            time.sleep(0.2)
            led.off()
            time.sleep(0.2)
        else:
            led.off()

# Configure the interrupt on the PIR pin for both rising and falling edges
button.irq(trigger=(Pin.IRQ_RISING | Pin.IRQ_FALLING), handler=pir_interrupt)

while True:
    led.on()
    time.sleep(1)
    led.off()
    time.sleep(1)
```

Code explained

```
1  from machine import Pin
2  import time
3  #button variable
4  button = Pin(15, Pin.IN, Pin.PULL_UP)
5  #On-Board LED
6  led_pin = Pin(25, Pin.OUT)
7
8  #commands
9  def pir_interrupt(pin):
10     if pin.value() == 1:
11         for x in range(10):
12             led_pin.value(1)
13             time.sleep(0.2)
14             led_pin.value(0)
15             time.sleep(0.2)
16         else:
17             led_pin.value(0)
18
19 # Configure the interrupt on the PIR pin for both rising and falling e
20 button.irq(trigger=(Pin.IRQ_RISING | Pin.IRQ_FALLING), handler=pir_interrupt)
21
22 while True:
23     led_pin.value(1)
24     time.sleep(1)
25     led_pin.value(0)
26     time.sleep(1)
```

This function is called when the button is pressed.

It makes the green led flash quickly 10 times.

This IRQ code is linked to the **button** variable. When the button is pressed the **pir_interrupt()** function is called.

This is the while loop that iterates when the button is not pressed.

It will go back to this after the function has finished.

Line 20 includes an IRQ Interrupt Request.

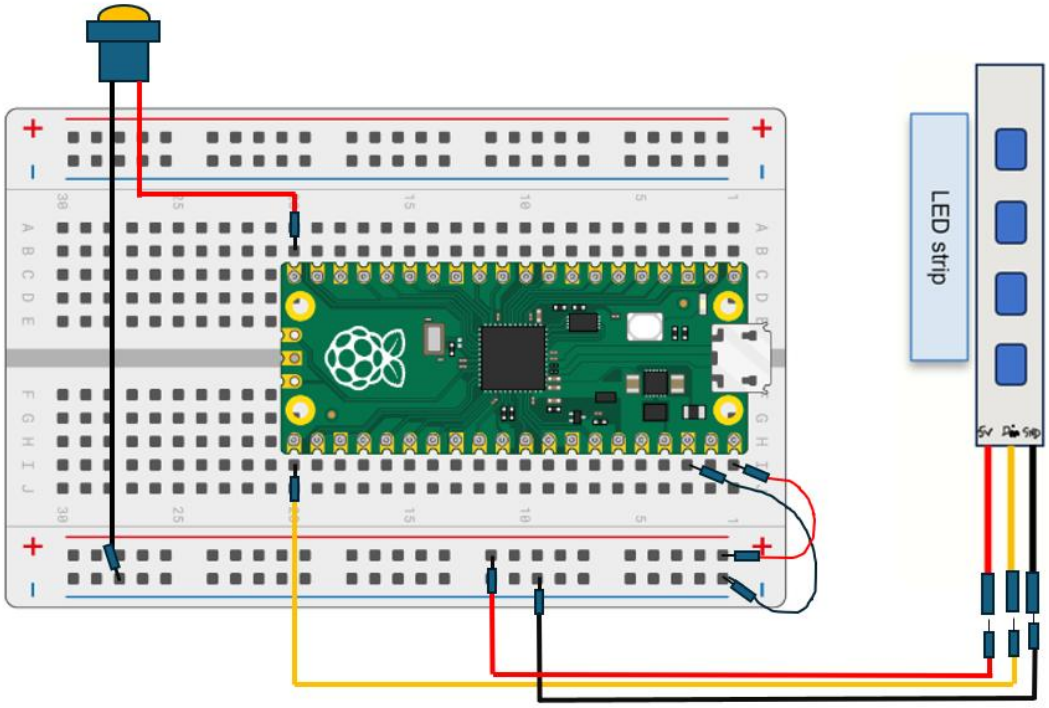
Using an IRQ interrupt allows us to interrupt an endless while loop with a button press.

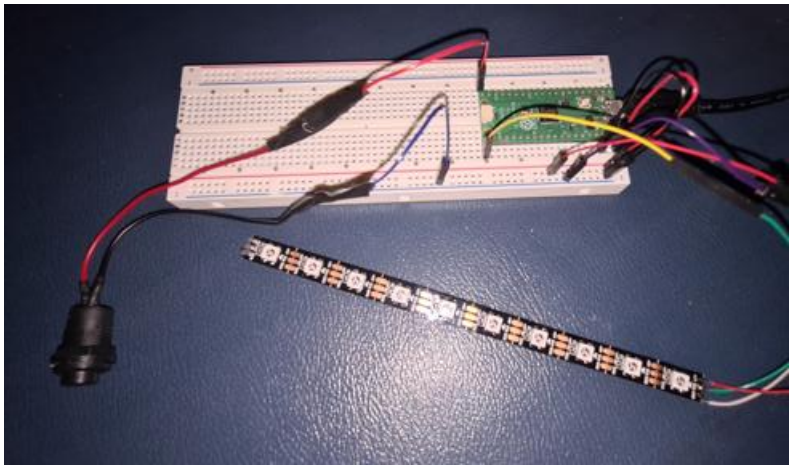
Using IRQ Interrupt code with LED strip.

Instead of using the on-board green LED let's control a LED strip.

The Pico listens out for the **button** press on Pin **GP15**(top left).

The **LED** yellow strip signal wire coming from **Pin GP16** (bottom left).





Test Code to Check LED is wired to Pico H (or Pico W) correctly

This code will work with a Pico H or Pico W. It is only the on-board green LED that is coded differently. The Pico H and W handle LED strips the same way.

```
#Button IRQ interrupt, with Pico W or Pico H
from machine import Pin
import time
from neopixel import Neopixel
```

```
import random
#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)
#LED variables
numpix = 10 # change this value to match the number of pixels in your LED strip
strip = Neopixel(numpix, 0, 16, "GRB")
#colour variables
red = (0, 255, 0)
green = (255, 0, 0)
blue = (0, 0, 255)
white = (204, 255, 255)
blank = (0,0,0)
#---------------commands---------------#

strip.fill(blue)
strip.show()
time.sleep(1)
strip.fill(blank)
strip.show()
```

Hopefully, when you ran this code, the LED strip lit up blue for one second. This confirms our wiring is correct so far.

Create an endless loop.

We are going to make the LED strip flash on and off in an endless loop.

This can be quite bright, so I have reduced the brightness with the command on line 18. Strip brightness can be set to anything between 0 and 255.

Put the **strip.fill** commands into a while loop

```
15 white = (204, 255, 255)
16 blank = (0,0,0)
17
18 strip.brightness(25)
19 #---------------commands---------------#
20
21 while True:
22     strip.fill(blue)
23     strip.show()
24     time.sleep(1)
25     strip.fill(blank)
26     strip.show()
27     time.sleep(1)
```

Run the program.

It should flash blue, then blank, endlessly.

Now we are ready to code a button IRQ interupt.

IRQ interrupt code listens out for a button press.

It can respond to a button press ***even though*** the program is running a while loop.

Add the IRQ Interrupt Request

Line 21 listens for the button press.

When the button is pressed it calls a function named **pir_interrupt().**

```
19 #---------------commands---------------#
20 #IRQ Interupt code. This listens for the button press.
21 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
22
23 while True:
24     strip.fill(blue)
25     strip.show()
26     time.sleep(1)
27     strip.fill(blank)
28     strip.show()
29     time.sleep(1)
```

If we try to run the code now, we get an error message.

```
NameError: name 'pir_interrupt' isn't defined
```

This is because it is trying to call a function that does not exist yet.

Define the pir_interrupt() function

We now need to add the function that gets called when the button is pressed.

When line 21 runs, it is looking for this function

```
21 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
```

We need to write the code for this function before the command on line 21.

Test Code (entire code so far):

```
#Button IRQ interrupt, with Pico W or Pico H
from machine import Pin
import time
from neopixel import Neopixel
import random
#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)
#LED variables
numpix = 10 # change this value to match the number of pixels in your LED strip
strip = Neopixel(numpix, 0, 16, "GRB")
#colour variables
red = (255, 0, 0)
green = (0, 255, 0)
blue = (0, 0, 255)
white = (204, 255, 255)
blank = (0,0,0)

strip.brightness(25)
#---------------commands---------------#
def pir_interrupt(pin):
    strip.fill(red)
    strip.show()
    time.sleep(3)

#IRQ Interupt code. This listens for the button press.
button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)

while True:
    strip.fill(blue)
    strip.show()
    time.sleep(1)
    strip.fill(blank)
    strip.show()
    time.sleep(1)
```

Understanding the code

Near the start of our program we created a variable named **button**.

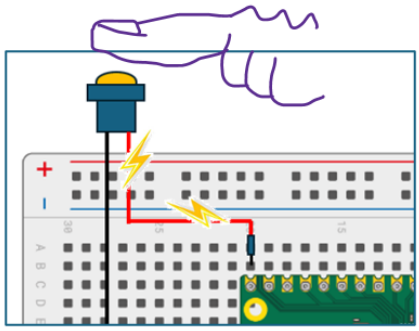
```
7 button = Pin(15, Pin.IN, Pin.PULL_UP)
```

This **button** variable uses the **Pin** class to link the variable with GPIO pin 15.

We use the **button** variable in the command for the IRQ interrupt.

```
26 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
```

When the physical button is pressed by a human finger, it sends an electrical signal to pin 15.



This activity is recorded by the **button** variable, which triggers the IRQ interrupt.

When line 26 runs, it calls the pir_interrupt() function.

You can see the function defined here:

```
20 def pir_interrupt(pin):
```

> Why do we have a lower case **pin** parameter in the brackets, but use **Pin** earlier?
>
> '**pin**' is a variable that holds the instance of the Pin class that caused the interrupt
>
> Do not worry if this is a bit confusing.

What you need to understand is when this command is triggered…

```
26 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
```

…it runs (calls) this function:

```
20 def pir_interrupt(pin):
21     strip.fill(red)
22     strip.show()
23     time.sleep(3)
```

This function makes the LED strip turn red for 3 seconds.

But we can write any code we want in this function.

Multiple Clicks

I am sure you have used xmas tree lights where you tap a button to make it flash different colours and sequences.

We are going to do the same thing with our LED strip and button.

Let's add a new variable to count the number of times the button has been clicked.

```
6  #button variable
7  button = Pin(15, Pin.IN, Pin.PULL_UP)
8  clicks = 0
9  #LED variables
```

Now we can use this variable in our pir_interrupt() function.

```
21 def pir_interrupt(pin):
22     global clicks
23     clicks = clicks + 1
24     print(clicks)
25     if clicks == 1:
26         strip.fill(red)
27         strip.show()
28         time.sleep(3)
29     elif clicks == 2:
30         strip.fill(white)
31         strip.show()
32         time.sleep(3)
33     else:
34         clicks = 0
```

This allows the function to 'see' the **clicks** variable inside the function.

The value for **clicks** is updated by 1.

If **clicks** is equivalent to 1 it makes the LED strip red.

If **clicks** is equivalent to 2 it makes the LED strip white.

If the user clicks again, it resets the value of **clicks** to 0.

Line 24 prints the current value for **clicks** so we can see it in the shell:

```
>>> %Run -c $EDITOR_CONT

  MPY: soft reboot
  1
  2
  3
  1
  2
```

Why does it keep going back to flashing blue?

Currently, we have a while loop running.

When the IRQ interrupt is finished, it goes back to this:

```
36 #IRQ Interupt code. This listens for the button press.
37 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
38
39 while True:
40     strip.fill(blue)
41     strip.show()
42     time.sleep(1)
43     strip.fill(blank)
44     strip.show()
45     time.sleep(1)
46
```

I made this flashing so we could see how an IRQ interrupt can over-ride a while loop.

But, if we are making a lamp, or shelf lighting, we probably do not want this flashing.

So, let's take out the while loop and make it start with a steady blue light.

```
36 #IRQ Interupt code. This listens for the button press.
37 button.irq(trigger = (Pin.IRQ_FALLING), handler = pir_interrupt)
38
39 #colour on start up
40 strip.fill(blue)
41 strip.show()
42
```

Run your program.

It should cycle through diffferent colours?

Challenge: add more colours

Can you add more clicks so more colours display if you click 4 or 5 times?

You will need to add more colour variables.

```
12 #colour variables
13 red = (255, 0, 0)
14 green = (0, 255, 0)
15 blue = (0, 0, 255)
16 white = (204, 255, 255)
17 blank = (0,0,0)
```

The W3Schools colour page is useful to find the RGB values.
https://www.w3schools.com/colors/colors_picker.asp

You will need to update this IF statement so it knows how to handle 4 or 5 clicks.

```
25     if clicks == 1:
26         strip.fill(red)
27         strip.show()
28         time.sleep(3)
29     elif clicks == 2:
30         strip.fill(white)
31         strip.show()
32         time.sleep(3)
33     else:
34         clicks = 0
```

Using the pir_interrupt() function to call other functions.

I am making a lamp.

It has a button and a LED strip.

I want to be able to cycle through various lamp settings.

I can use functions to create nice LED strip effects.

A new variable has been added that counts the number of button clicks.

The button IRQ interrupt request can be used to call a function, depending on the number of clicks.

Test Code:

```
#Button IRQ interrupt, with Pico W or Pico H
from machine import Pin
import time
from neopixel import Neopixel
import random
#button variable
button = Pin(15, Pin.IN, Pin.PULL_UP)
#LED variables
numpix = 256 # change this value to match the number of pixels in your LED strip
strip = Neopixel(numpix, 0, 16, "RGB")
#colour variables
red = (0, 255, 0)
green = (255, 0, 0)
blue = (0, 0, 255)
white = (204, 255, 255)
blank = (0,0,0)

strip.brightness(25)
clicks = 0
#---------------commands---------------#

def funct1():
    strip.fill(green)
    strip.show()

def funct2():
    strip.fill(red)
    strip.show()

#Function is called when the button is pressed
def pir_interrupt(pin):
    global clicks
    clicks = clicks + 1
    time.sleep(0.5)
    if clicks == 1:
        funct1()
    elif clicks == 2:
        funct2()
    else:
        clicks = 0

#Button press triggers an Interrupt Request, which calls the function
button.irq(trigger=(Pin.IRQ_FALLING), handler=pir_interrupt)

strip.fill(blue)
strip.show()
```

Trouble shooting

Having issues?

Check your micro usb cable does actually allow the sending of data.

Check the Pins you are using are correct. My button is connecting to GPIO15. My LED strip is connecting to GPIO16. But we have often used GPIO28 for LED strips in

previous projects. Check the PinOut diagram and how you are wired up on the breadboard, and check the button variable and strip variable to make sure they are referencing the correct GPIO numbers.

Is the NeoPixel library installed on the Pico. You can check this in Thonny by clicking **View / Files** and then selecting the Raspberry Pi Pico window on the left.

Try a different LED strip. The connecting wires can get a little dodgy after repeated use.

Are you getting the number of clicks increasing by more than one? This is called 'debounce'. It is not unusual for for buttons to give false readings. See below for more on this.

IRQ_RISING or IRQ_FALLING?

The IRQ interrupt request includes options for reading the rising or falling edge.

What is meant by this and which do we want?

```
button.irq(trigger=(Pin.IRQ_RISING | Pin.IRQ_FALLING),
```

We generally use this command:

```
button.irq(trigger=Pin.IRQ_FALLING, handler=pir_interrupt)
```

- **Falling edge**: Detects when the button is pressed (transition from high to low).
- **Rising edge**: Detects when the button is released (transition from low to high).

If you only need to perform actions when the button is pressed,
using Pin.IRQ_FALLING will work perfectly.

Adding Pin.IRQ_RISING would trigger the interrupt twice for each button press (once when pressed and once when released), which might not be necessary for your use case.

Challenge: Build your own

Buy the following equipment and make some accent lighting for your room at home.

- A Pico H micro controller (with headers): link
- Small breadboard: link
- Jumper wires: link
- Button(s): link
- WS2812b LED strip: link