
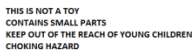


## Neat products, low cost, no frills

Optional

## 24 Experiments for the RASPBERRY PI PICO™.



## Contents

Index of tables.....	4
Index of figures .....	4
1. Introduction .....	5
1.1. Kit component list: .....	6
2.1. Soldering .....	8
2.2. The development Environment .....	8
3. Projects .....	9
3.1. Flashing LEDs.....	9
3.2. A quick aside; lighting up an LED .....	9
3.3. Lighting up an LED using the Pi Pico .....	11
3.4. Adding a button input.....	13
3.5. The Buzzer.....	14
3.6. Traffic light. ....	15
3.7. PIR Sensor Burglar alarm.....	18
3.8. Potentiometer and ADC.....	20
3.9. I2C and LCD display.....	22
3.10. WS2812 Controllable RGB LED strip .....	24
4. More fun. ....	26
4.1. The Mathematics of driving LEDs (You can skip this bit if you like).....	26
4.2. Anatomy of an LED.....	28
4.3. Louder sounder .....	28
4.4. Rising pitch using PIO and sounder.....	29
4.5. Slightly improved pedestrian crossing .....	30
4.6. Piezo sounder as a knock sensor .....	31
4.7. Burglar alarm with PIO driven sounder .....	34
4.8. Potentiometer PWM LED and PIO tone.....	35
4.9. More WS2812 examples .....	36
4.9.1. Rainbow on WS2812 .....	36
4.9.2. Random colours .....	37
4.10. Installing Thonny on Windows.....	38
4.11. LCD_API Commands.....	40
5. Pi-Pico ideas .....	42
6. The small print .....	43
7. Glossary.....	44

## **The experiments**

Experiment 1. Lighting an LED. ....	9
Experiment 2. Lighting up an LED using the Pi Pico. ....	11
Experiment 3. Sequencing 5 LEDs. ....	11
Experiment 4. Button LED and Pi Pico ....	13
Experiment 5. Passive sounder using the Processor ....	14
Experiment 6. Passive sounder using the PIO. ....	15
Experiment 7. Traffic light with passive sounder. ....	15
Experiment 8. Testing the PIR. ....	18
Experiment 9. Using the PIR. ....	19
Experiment 10 PIR Burglar alarm with PIO driven sounder ....	20
Experiment 11. Testing the potentiometer. ....	20
Experiment 12. PWM controlled LED ....	21
Experiment 13. I2C LCD display ....	22
Experiment 14. PIO controlled RGB LED strip. ....	24
Experiment 15. Louder sounder ....	28
Experiment 16. Rising pitch sounder ....	29
Experiment 17. improved pedestrian crossing ....	30
Experiment 18. Energy harvesting with a Piezo sounder ....	31
Experiment 19. Using the Piezo sounder as a simple sensor. ....	32
Experiment 20. Using the ADC to make the Piezo into more sensitive sensor ....	33
Experiment 21. Burglar alarm with PIO driven sounder ....	34
Experiment 22. Using the potentiometer to control PWM and PIO. ....	35
Experiment 23. Rainbow on WS2812 ....	36
Experiment 24. Random colours. ....	37

## Index of tables

Table 1. Component list .....	6
Table 2. Pictures of individual components .....	7
Table 3. Wiring table for Experiment 3 .....	12
Table 4. I2C LCD Display circuit wiring guide. ....	23
Table 5. Forward voltage of LEDs.....	27
Table 6. LCD_API commands .....	40
Table 7. Glossary .....	44

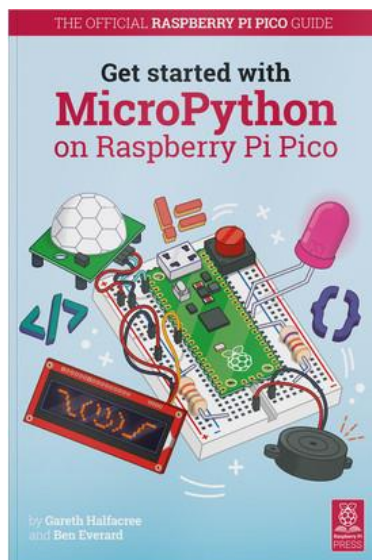
## Index of figures

Figure 1. Raspberry Pi Pico Pinout .....	9
Figure 2. Lighting a white LED.....	10
Figure 3. Bending legs on a resistor. ....	10
Figure 4. Sequencing 5 LEDs .....	12
Figure 5. 2 wire button. ....	13
Figure 6. LED and button wiring.....	13
Figure 7. LED will stay lit for 2 seconds after releasing the button. ....	13
Figure 8. Sounder wiring. ....	14
Figure 9. Traffic light with PIO driven sounder. ....	17
Figure 10 PIR with wires.....	18
Figure 11. PIR controls .....	18
Figure 12. Testing the PIR. ....	19
Figure 13. Burglar alarm circuit.....	19
Figure 14. Testing the potentiometer.....	21
Figure 15. I2C LCD display. ....	22
Figure 16. Level shifter details. ....	22
Figure 17. WS2812 RGB LED strip .....	24
Figure 18. Running WS2812 LED strip.....	25
Figure 19. Resistor and LED circuit.....	26
Figure 20. Wiring for louder sounder. ....	28
Figure 21 Simple Piezo LED circuit/Energy Harvester .....	31
Figure 22. Sounder driving 2 LEDs .....	32
Figure 23. Detecting the sounder on a logic pin. ....	32
Figure 24. Detecting the sounder using the ADC for more sensitivity. ....	33
Figure 25. Potentiometer PWM LED and PIO tone circuit .....	35
Figure 26. Zagdig .....	40

## 1. Introduction

This kit is inspired by The Book “Getting started with MicroPython on Raspberry Pi Pico” (Otherwise known as “The Book” in this document) published by Raspberry Pi™ Press and the amazing Raspberry Pi Pico™ otherwise known in this document as “Pi-Pico”). You can buy The Book from many outlets and the lovely people at Raspberry Pi™ have also made it free to download as a PDF:

<https://hackspace.raspberrypi.org/books/micropython-pico>



The first edition of The Book had a few errors that were ironed out in the second edition. Corrections are published at the above URL.

This document describes the components in our kit and how to use them. In many cases the components are identical to those described in The Book and in others they are slightly different. In particular we have added wires with pin connections so that you will not need to use a soldering iron. Where there are differences, they are explained with examples. In all cases we highlight known pitfalls and problems.

The photographs in this document are taken whilst developing the code and this document so are a good representation of the real thing working.

The code examples in this document are all available as a download from our website [www.easyDAQ.co.uk](http://www.easyDAQ.co.uk)

We also document how to use a PC or Mac™ instead of a Raspberry Pi™ as a development platform. See section 3.10 Installing Thonny on Windows™.

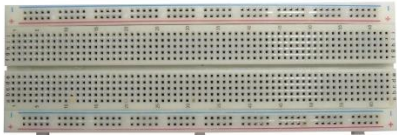


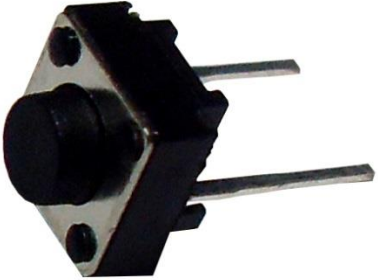



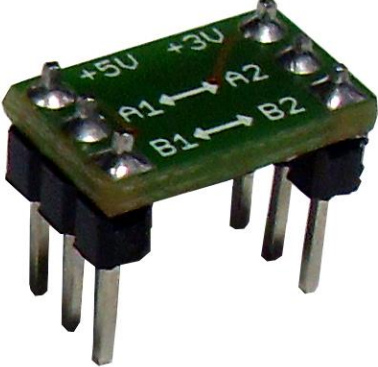




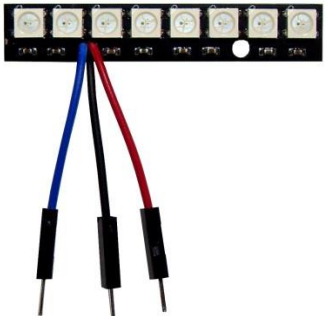

**1.1. Kit component list:**

Table 1. Component list.

Item	Description	Qty.
1	Raspberry Pi Pico with soldered headers ( <u>OPTIONAL</u> )	1
2	MB-102 Breadboard	1
3	LEDs. 1 of each of Red, Yellow, Green, Blue, White	5
4	Resistors 10 x 330R, 2 x 1Meg Ohm	12
5	Switch/button	2
6	Sounder without oscillator	1
7	I2C, 2x16 char LCD display	1
8	3V to 5V level shifter	1
9	HC-SR501 PIR module with wires	1
10	Link wires M-M (Male to Male)	20
11	Link wires M-F (Male to Female)	10
12	10k linear potentiometer	1
13	WS2812 RGB LED strip (8 LEDs) with wires	1
14	EasyDAQ Notes and examples link card	1



Table 2. Pictures of individual components.

 <p>MB-102 Breadboard</p>	 <p>LEDs</p>	 <p>Resistors</p>
 <p>Button</p>	 <p>Sounder</p>	<p>Front view</p>   <p>Rear view</p> <p>I2C, 2x16 char LCD Display</p>
 <p>Level Shifter</p>	 <p>PIR module</p>	 <p>20 Link wires M-M</p>
 <p>10 Link wires M-F</p>	 <p>10k linear potentiometer</p>	 <p>WS2812 RGB LED</p>
<p><b>Optional</b> The EasyDAQ-Kit 1+ is supplied with a pre soldered Raspberry Pi Pico with pre-loaded MicroPython.</p> 		

## Getting started

### 1.2. Soldering

We have devised this kit so that there is no soldering required. Our EasyDAQ-Kit1+ even provides a Pi Pico with pins soldered, loaded with MicroPython and tested. Alternatively, Chapter 1 of The Book gives a good introduction to soldering which will be needed to attach the Pin Headers of your Pi Pico. If you don't have a soldering iron then we suggest that you either buy our EasyDAQ-Kit1+ or a Pi Pico with the pre-soldered Headers, Pimoroni stock them. See <https://shop.pimoroni.com/products/raspberry-pi-pico?variant=32402092326995> . Alternatively, find a friendly electronics engineer to do it for you as soldering is a bit of an art which requires a reasonable amount of practice to master. Most technical colleges and universities are likely to have someone willing to help with this too.

### 1.3. The development Environment

The Book is based around using a standard Raspberry Pi™ connected to your Raspberry Pi Pico™ to develop code using Thonny and the IDE (Integrated Development Environment). Although the Raspberry Pi has sold in the millions there are many more PCs and MACs in the world and it is also possible to run Thonny on a Windows™ or Linux PC or on a MAC™.

For installation on Windows™ see section 3.9 [Installing Thonny on Windows 7](#)

**Remember.** Once the MicroPython has been loaded onto the Pi Pico then you will no longer need to hold down the Boot button when plugging in the Pi Pico.

Chapters 2 and 3 of The Book will get you started using Thonny and MicroPython and so we will skip straight to the electronics projects.

**Note.** If Thonny loses contact with the Pi Pico it may be that the Pi Pico is busy. Try pushing the “Stop” Icon a couple of times to try to interrupt the Pi Pico. If that fails then try unplugging the USB connection to the Pi Pico, wait a few seconds, plug it back in, wait a few seconds for the PC's operating system to recognise the Pi Pico then press the Thonny “Stop” icon again. This should re-establish a connection. We have found that you may have to repeat this procedure a couple of times. Longer delays seem to help.

Once you get used to the Thonny – Pi Pico IDE the problem of losing communication reduces.



## 2. Projects

### 2.1. Flashing LEDs

The following experiment relates to pages 51 and 52 of The Book.

LEDs are wonderful devices that are increasingly lighting our world. If treated carefully they are more efficient and longer lasting than all the current alternatives<sup>1</sup>. LEDs are very easy to use, particularly if raw efficiency is not required such as their use as an indicator in an electronic circuit. All that is needed is a resistor to limit the current that flows through the LED.

We include 5 LEDs in our kit, Red, Green, Blue and White. These LEDs have a maximum operating current of 20mA but will be plenty bright enough at just 4mA or even 2mA.

If you are familiar with LEDs then skip section 2.2.

### 2.2. A quick aside; lighting up an LED

#### *Experiment 1. Lighting an LED.*

Use the diagram Pico-R3-A4-Pinout.pdf from Raspberry Pi™ to help you with the wiring connections.

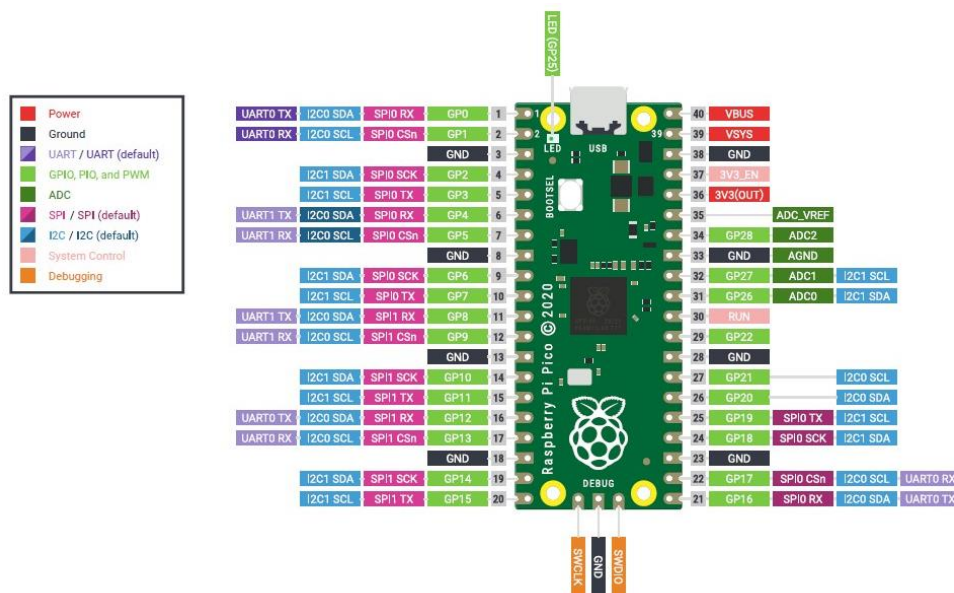


Figure 1. Raspberry Pi Pico Pinout

The Pi Pico board has numerous functions on many of the pins so the same pin is often referenced by different names. For example, Pin 1 of the board is known as GP0, SPI0\_RX, I2C0\_SDA and UART\_TX. To make matters even more complex the software will often reference a Pin in the fashion “led1 = Pin(11, Pin.OUT)” but in this case it is referring to GP11 not Pin 11 of the board.

If you are not familiar with electronics you can set up the following circuit just to light up an LED. We can use the Pi Pico to act as a power supply. This is actually just a connection to the power that is available from a USB connection.

<sup>1</sup> Excepting the sun of course. But controlled thermo-nuclear power is a bit beyond our current ability lighting wise ☺.

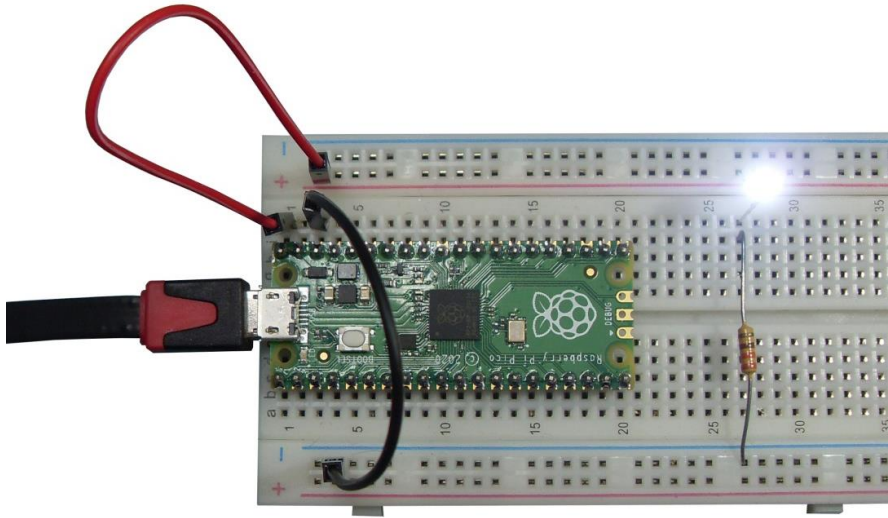


Figure 2. Lighting a white LED.

For this experiment:

- 1). Plug your Pi Pico into the breadboard so that the USB connector is in the leftmost position as shown in Figure 2.
- 2). Use a black wire between pin 38 of the Pi Pico and the lower blue rail
- 3). Use a red wire between pin 40 of the Pi Pico and the upper red rail
- 4). Connect a 330 Ohm resistor (330R) between the lower blue rail and the upper bank of vertical connections as shown in Figure 2. You will need to bend 10mm of the wire of the resistor down at the end to poke into the holes in the breadboard.
- 5). Connect an LED between the other end of the resistor and the upper red rail with the longest pin of the LED going to the top rail.
- 6). With the PC powered up; plug in a USB cable between your PC and the Pi Pico™ and your LED should light<sup>2</sup>.



Figure 3. Bending legs on a resistor.

**Note.** When bending the legs of a component like a resistor it is best to make sure that the joint between the wire and the body of the resistor is not strained. It is possible to break the leg away from the component or fracture its connection to the resistive element inside. Hold the wire both sides of the bend while forming it.

Possible problems:

If the LED does not light there are a few possible problems:

- 1). Is the LED on the Pi Pico illuminated or flashing? If not, the PC may not be powering the USB cable.
- 2). Check the wiring is as shown in Figure 2. Page 49 of The Book gives a good explanation of how the breadboard is wired internally.

<sup>2</sup>The Pi Pico requires a USB-A to USB-micro-B cable to connect from the Raspberry Pi, PC or MAC to Pi Pico.

3). The LED may be in the wrong way around. Don't worry. This will not damage the LED. Just take it out and make sure that the long lead is in the top red rail.

Once you have it working, try swapping the LEDs and light up different colours. You could also try lighting up several LEDs.

For the mathematics of an LED circuit see section 3.1 The mathematics of driving LEDs (You can skip this bit if you like)

### 2.3. Lighting up an LED using the Pi Pico

#### *Experiment 2. Lighting up an LED using the Pi Pico.*

Page 52 and Figure 4.4 of The Book show how to link your LED to the Pi Pico. You can choose any colour LED you like. The long pin will be connected to a 330 ohm resistor which in turn is connected to pin 20 (GP15) of the Pi Pico. The short pin of the LED connects via a black wire to the 0v rail created by connecting a black wire between pin 38 (GND) of the Pi Pico and the top blue rail of the breadboard.

Load up the Blink.py software as described in The Book. It would be a good idea to run this program again to confirm that the on-board LED is still controlled. Now modify to control of the LED from the internal to the external LED.

If all goes well you are now controlling an LED from the Pi Pico. It really is the "Hello World" of the Microcontroller environment.

If there are any problems then check the wiring and the orientation of the LED. If that doesn't help check the modification from internal to external LED in the software carefully.

#### *Experiment 3. Sequencing 5 LEDs.*

(Page 53 challenge)

As there are 5 LEDs in the kit, how about trying to control all of them, each on a separate pin, to make a light show.

The following code example defines 5 LEDs on I/O pins 11 to 15 respectively. A counter is declared which will be used to cycle between the LEDs. When the timer triggers, "sequence(timer)" turns the currently selected LED on and the previous LED off then increments the counter.

Create a new file for this program. In Thonny, select File > New and rename it something like "Sequential\_LEDs\_code\_example.py".

Copy or write the following code in the code space: Note. The file name for this program in our download is "LED\_Sequence.py". The names of the files are given in the headers of the programs we have supplied.

```
# Sequential LEDs code example "LED_Sequence.py"
# Nigel .J. Halse 3rd April 2021
```

```
from machine import Pin, Timer
```

```
led1 = Pin(11, Pin.OUT)          # Give the LEDs 5 port names
```

```

led2 = Pin(12, Pin.OUT)
led3 = Pin(13, Pin.OUT)
led4 = Pin(14, Pin.OUT)
led5 = Pin(15, Pin.OUT)
counter = 0                                     # Declare a variable

LEDs = [led1,led2,led3,led4,led5]              # Put the LEDs in a list
timer = Timer()

def sequence(timer):                             # Define a function for dealing the timer interrupt
    global counter                               # Indicate that the global variable is required
    LEDs[counter % 5].on()                     # Turn on an LED
    LEDs[(counter-1) % 5].off()                 # Turn off the previous LED
    counter = counter + 1                       # Increment the counter

# Set up a timer interrupt
timer.init(freq=2.5, mode=Timer.PERIODIC, callback=sequence)

```

Now rewire the breadboard as shown in Figure 4. Sequencing 5 LEDs using the wiring table and Figure 1. Raspberry Pi Pico Pinout to check.

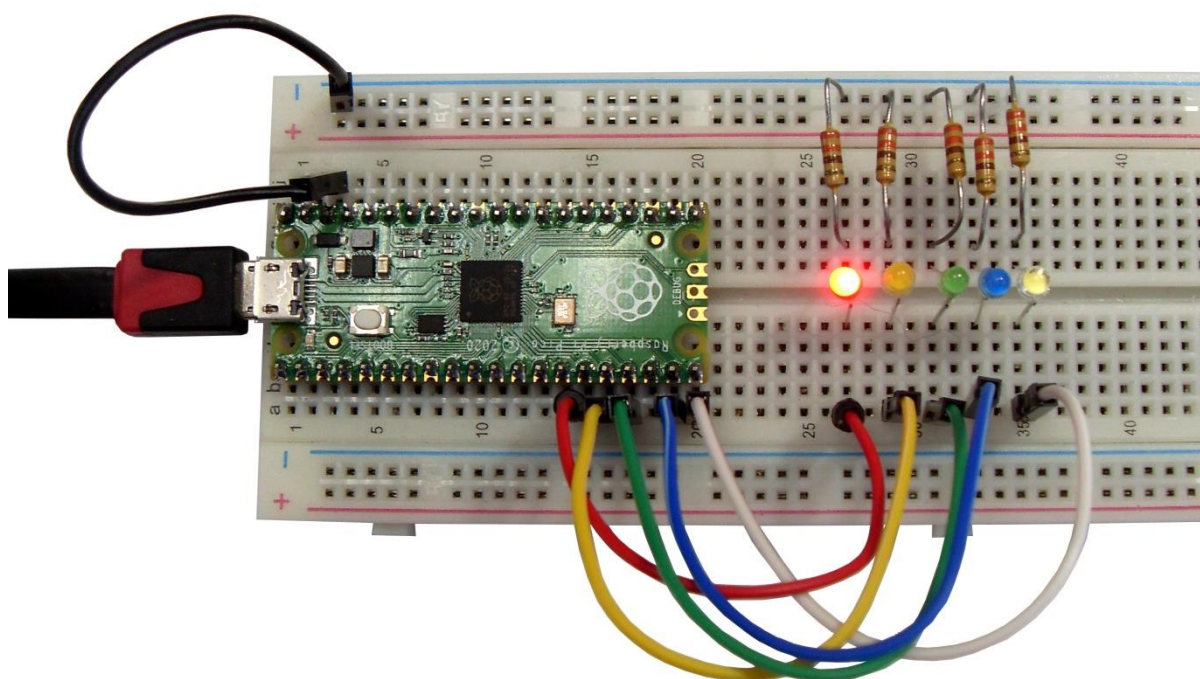


Figure 4. Sequencing 5 LEDs

Table 3. Wiring table for Experiment 3

Wire number	Function	Wire Colour	From	To	M-M or M-F
1	0V	Black	Pi-Pico GND [pin 38]	Breadboard -Bus	M-M
2	Red LED signal	Red	Pi-Pico GPIO(11) [pin 15]	Red LED Anode (+ive)	M-M
3	Yellow LED signal	Yellow	Pi-Pico GPIO(12) [pin 16]	Yellow LED Anode (+ive)	M-M
4	Green LED signal	Green	Pi-Pico GPIO(13) [pin 17]	Green LED Anode (+ive)	M-M
5	Blue LED signal	Blue	Pi-Pico GPIO(14) [pin 19]	Blue LED Anode (+ive)	M-M
6	White LED signal	White	Pi-Pico GPIO(15) [pin 20]	White LED Anode (+ive)	M-M
7	330R resistor	-	Red LED Cathode (-ive)	Breadboard -Bus	-
8	330R resistor	-	Yellow LED Cathode (-ive)	Breadboard -Bus	-
9	330R resistor	-	Green LED Cathode (-ive)	Breadboard -Bus	-
10	330R resistor	-	Blue LED Cathode (-ive)	Breadboard -Bus	-
11	330R resistor	-	White LED Cathode (-ive)	Breadboard -Bus	-

**Note.** The orientation of the LEDs has changed. The negative ends of the LEDs point to the top negative bus. Using the coloured wires helps. Remember that pin18 on the Pi Pico is a GND pin, not an I/O pin, so there is a space between the green and blue wire in the connections to the Pi Pico.



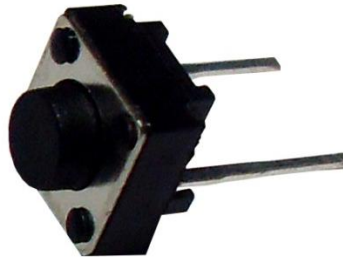
Also... Try adjusting the “freq=”. This value is in Hz (cycles per second). At 10Hz it is obviously faster. At 100Hz it is manic. At 1000Hz it is just a blur. All the LEDs appear to be on at the same time because the human eye can’t respond fast enough. Also, don’t include “Hz”. MicroPython will not understand.

#### **2.4. Adding a button input.**

Now we are on Page 53 of The Book.

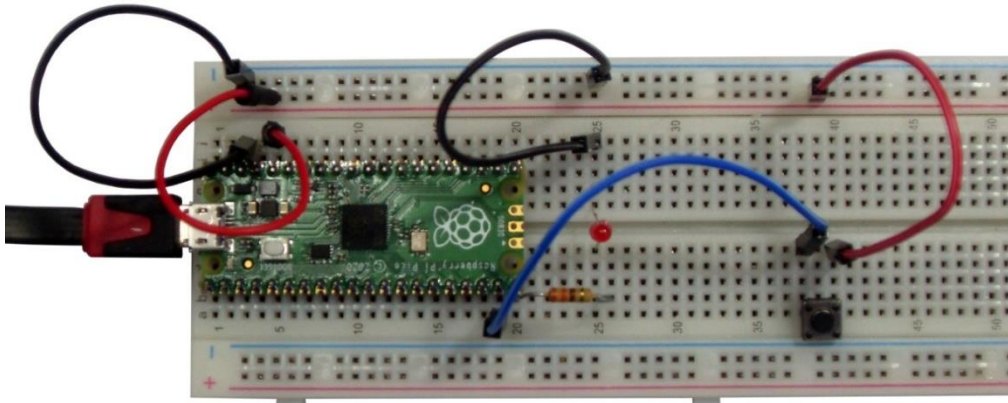
#### **Experiment 4. Button LED and Pi Pico**

The button we have provided in the kit has 2 pins rather than 4. The pins are longer and more likely to make a good contact in the breadboard than 4 pin versions. It is also easier to get the connections right as there are only 2 of them... :-)



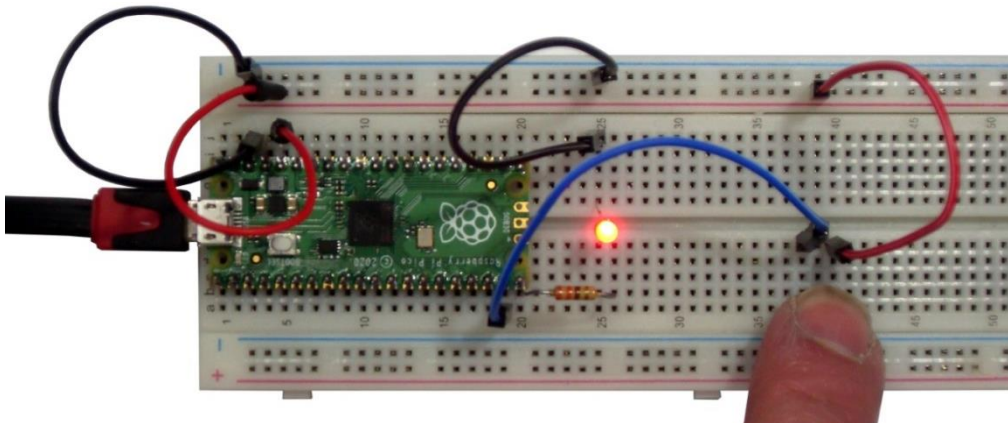
**Figure 5. 2 wire button.**

Using the button:



**Figure 6. LED and button wiring.**

Note position of the button.



**Figure 7. LED will stay lit for 2 seconds after releasing the button.**

You are now all set up for the rest of Chapter 4.

## 2.5. The Buzzer.

### *Experiment 5. Passive sounder using the Processor*

We are now at Chapter 5 page 58 of The Book. The traffic lights project is a good example of multiple inputs and outputs doing different functions. The Piezoelectric buzzer (Piezo) provided in this kit is not “active”. It is a standard passive device which makes it a lot more useful as we shall see in section 3.6 Piezo sounder as a knock sensor.

To start with let's just make a noise.

This sounder will have a peak resonance at about 4000Hz (4kHz). We can make lower and higher pitches but it will be loudest around this frequency.

All we need to do is connect 1 pin of the sounder to a pin we are controlling and the other to 0v. In this case we will use I/O pin 12 and oscillate the pin high and low at 4000 cycles per second.

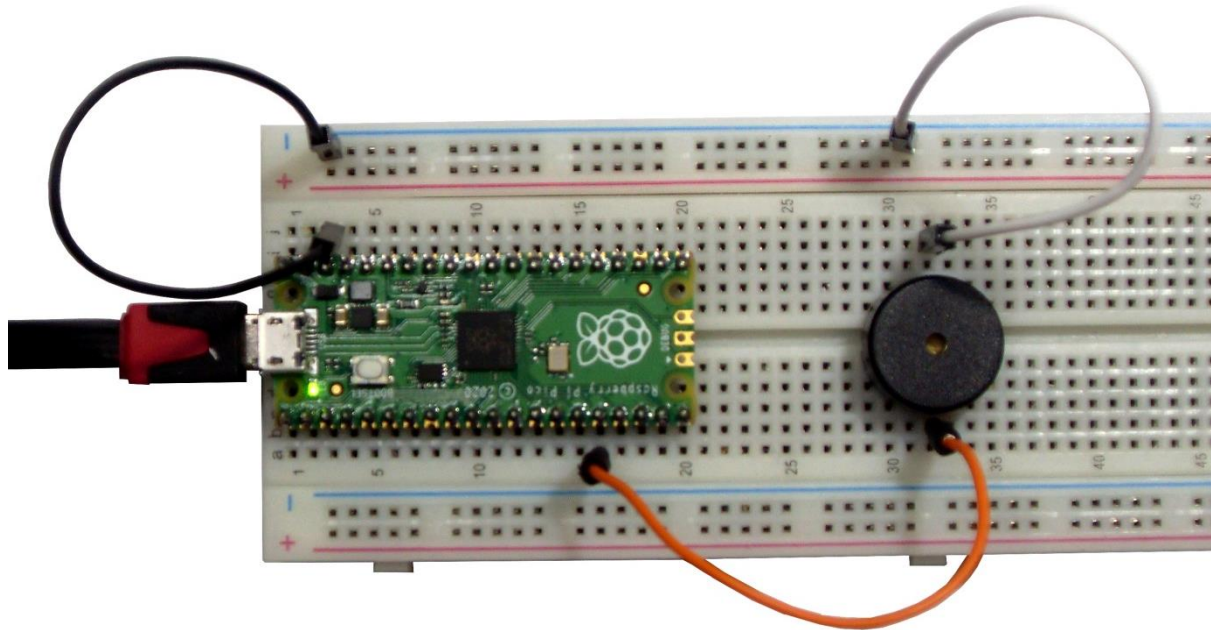


Figure 8. Sounder wiring.

The following code example will produce a 4000Hz tone from the sounder.

```
# Basic Tone generator code example "Tone 1.py"
# Using 1 pin to control the sounder.
# N.J.Halse 6rd April 2021

from machine import Pin, Timer

Sounder1 = Pin(12, Pin.OUT)    # Give the ports names

timer = Timer()

def Sounder(timer):            # Define a function for dealing with the timer interrupt
    Sounder1.toggle()

# Set up a timer interrupt to double the frequency required on the sounder
timer.init(freq=8000, mode=Timer.PERIODIC, callback=Sounder)
```

Try changing the “freq” value in the last line in the code. A piezo sounder is not a brilliant speaker. It really is best at its resonant frequency so there are a lot of over tones at higher frequencies. Also, the processor will not be able to keep up eventually. Lower tones will be quieter but less rough. For a louder sounder see 3.1 [Louder sounder](#) in the [More fun](#) section.

### Experiment 6. Passive sounder using the PIO

The above method of controlling the sounder uses up a significant amount of processor resources that could be usefully employed elsewhere in your code. The Pi-Pico has a very powerful feature as described in Appendix C of The Book and chapter 3 of the “rp2040-datasheet”. That is “Programmable I/O”. Basically 8 spare tiny processors that can be dedicated to controlling I/O pins. Here we use 1 of the PIO state machines to generate the tone.

```
# PIO based tone generator "PIO_Tone.py"
# Nigel J. Halse 7th April 2021

import time
from rp2 import PIO, asm_pio
from machine import Pin

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW)      # Let the MicroPython compiler know that the
# following is a PIO machine code definition
def tone():
    wrap_target()
    set(pins, 1)          # 1 instruction
    set(pins, 0)          # 1 instruction plus jump to start
    wrap()

# Instantiate a state machine with the tone program, at 8000Hz, with set bound to Pin(12).
# This will generate a 4kHz square wave on pin 12
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(12))

def Buzzer(value): # A change of name to make the program easier to read
    sm1.active(value)

# 1 second tone bursts
while 1:
    Buzzer(1)          # Turn on the tone
    time.sleep(1)      # Wait for a second
    Buzzer(0)          # Turn off the tone
    time.sleep(1)      # Wait another second
```

For a bit more fun with the sounder see [Rising pitch using PIO and sounder](#) in section 3.4.

## 2.6. Traffic light.

We are now at page 58 of The Book.

### Experiment 7. Traffic light with passive sounder

The traffic lights project is where we start to get the Pi Pico to do some interesting things in The Book.

Work your way through chapter 5 of The Book. The button in your kit has 2 pins, not 4. See Figure 9 on page 17 for the layout that you will use with your kit. You will not be building the page 61 circuit. The sounder in the kit is passive, so it will only click instead of buzzing. So, use the instructions below to convert it to buzz.

To use your kit sounder, you will need to make some changes to the program from the book. You can copy and paste the whole program from below\*, or you can do it manually ...

- Remove “buzzer = machine.Pin(12, machine.Pin.OUT)”



- Make sure that “from rp2 import PIO, asm\_pio” And “from machine import Pin” are included as an import
- Add the following definitions:

```
def tone():
    wrap_target()
    set(pins, 1)          # 1 instruction
    set(pins, 0)          # 1 instruction plus jump to start
    wrap()
```

and

```
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(12))
```

```
def Buzzer(value): # A change of name to make the program easier to read
    sm1.active(value)
```

1. In the body of the program substitute “Buzzer(1)” for `buzzer.value(1)` and `Buzzer(0)` for `buzzer.value(0)`

\*The code now looks like this:

```
# Traffic lights with pedestrian crossing sounder "PIO_Traffic_lights 1.py"
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO" Modified to use a PIO driven sounder
# by Nigel J. Halse 7th April 2021

import machine
import utime
import _thread
from rp2 import PIO, asm_pio
from machine import Pin

led_red = machine.Pin(15, machine.Pin.OUT)
led_amber = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN, machine.Pin.PULL_DOWN)

global button_pressed
button_pressed = False

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW) # Let the MicroPython compiler know that the following is a PIO machine code
definition
def tone():
    wrap_target()
    set(pins, 1) # 1 instruction
    set(pins, 0) # 1 instruction plus jump to start
    wrap()

# Instantiate a state machine with the tone program, at 8000Hz, with set bound to Pin(12).
# This will generate a 4kHz square wave on pin 12
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(12))

def Buzzer(value): # A change of name to make the program easier to read
    sm1.active(value)

def button_reader_thread():
    global button_pressed
    while True:
        if button.value() == 1:
            button_pressed = True
            utime.sleep(0.01)
_thread.start_new_thread(button_reader_thread, ())

while True:
```

```

if button_pressed == True:
    led_red.value(1)
    for i in range(10):
        Buzzer(1)
        utime.sleep(0.2)
        Buzzer(0)
        utime.sleep(0.2)
    global button_pressed
    button_pressed = False
led_red.value(1)
utime.sleep(5)
led_amber.value(1)
utime.sleep(2)
led_red.value(0)
led_amber.value(0)
led_green.value(1)
utime.sleep(5)
led_green.value(0)
led_amber.value(1)
utime.sleep(5)
led_amber.value(0)

```

The program cycles through the standard red, yellow green traffic light sequence until the button is pressed. When the button is pressed, the next time the LEDs return to red it will sound the tone 10 times before continuing the sequence.

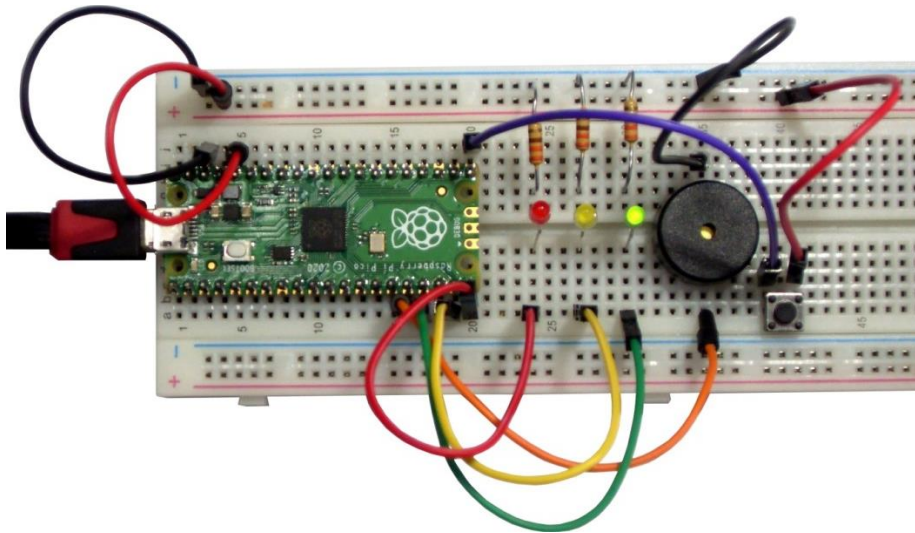


Figure 9. Traffic light with PIO driven sounder.

For a slightly improved version of the code see section 3.5 [Slightly improved pedestrian crossing](#)

Also, the sounder can be used as an input device. With the right pre-amplification they can function as a microphone but this is beyond this current kit. However, it is entirely possible to make a knock sensor. See section 3.6 [Piezo sounder as a knock sensor](#)

You are now good to go all the way through to Chapter 7 “Burglar alarm” of The Book.

## 2.7. PIR Sensor Burglar alarm.

### Experiment 8. Testing the PIR.

The PIR sensor is a very interesting device. The Book gives a good overview. See <https://lastminuteengineers.com/pir-sensor-arduino-tutorial/><sup>3</sup> for an overview of how these sensors work .

The PIR we have provided is fitted with a 3-wire connector to make it easier for you to use.

Wire colour	Function
Red	+V. Usually +5v. Minimum +4.5V. Maximum 12V
Blue	Signal. 0V = no detection, +3V = detection.
Black	0V (GND)

Testing the functionality for the PIR unit is very easy. If it detects a moving warm object the output will go from 0V to +3V. To see this happen we only need to apply +5V and 0V from the USB on the Pi-Pico and drive an LED via a 330R resistor from the blue, signal, wire to 0V. See Figure 12. Testing the PIR.



Figure 10 PIR with wires.

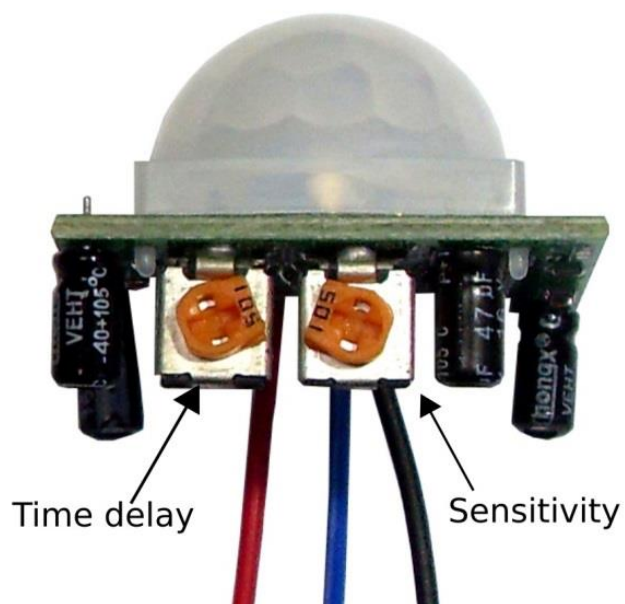


Figure 11. PIR controls

For testing purposes, it is best to set the on-board pre-sets to maximum sensitivity and shortest time delay as shown in Figure 11. PIR controls.

These sensors appear to take a minute or two to settle after powering up. They appear to become more sensitive to movement after they have calibrated to the environment.

<sup>3</sup> Try pasting the URL into a browser if the link doesn't open directly from the page.

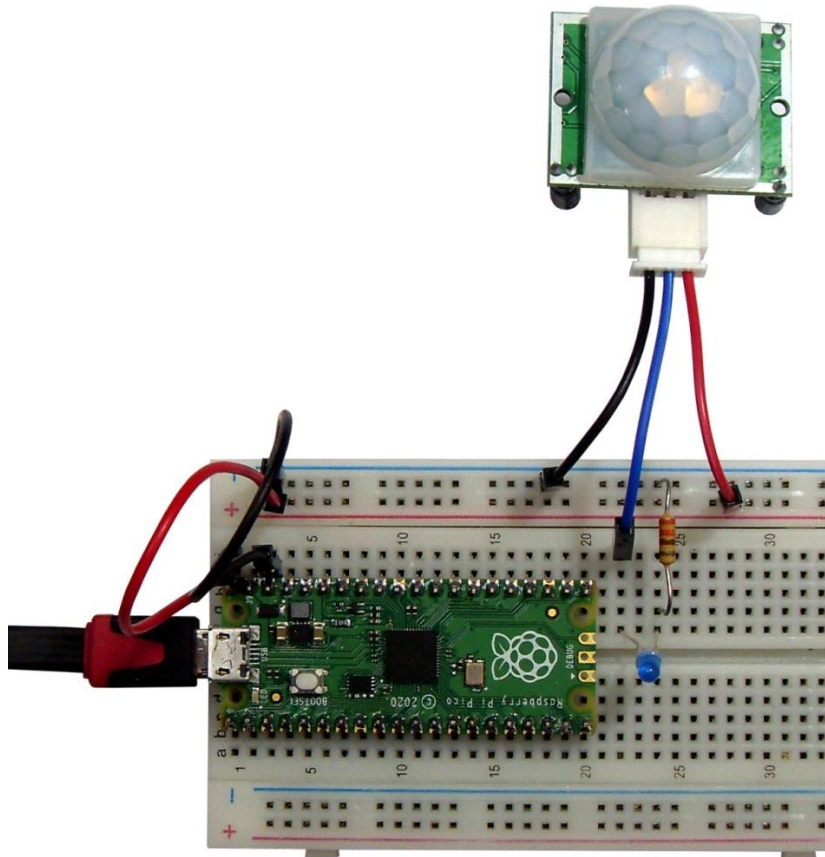


Figure 12. Testing the PIR.

See Figure 13. Burglar alarm circuit. below, for our version using the wired PIR.

*Experiment 9. Using the PIR*

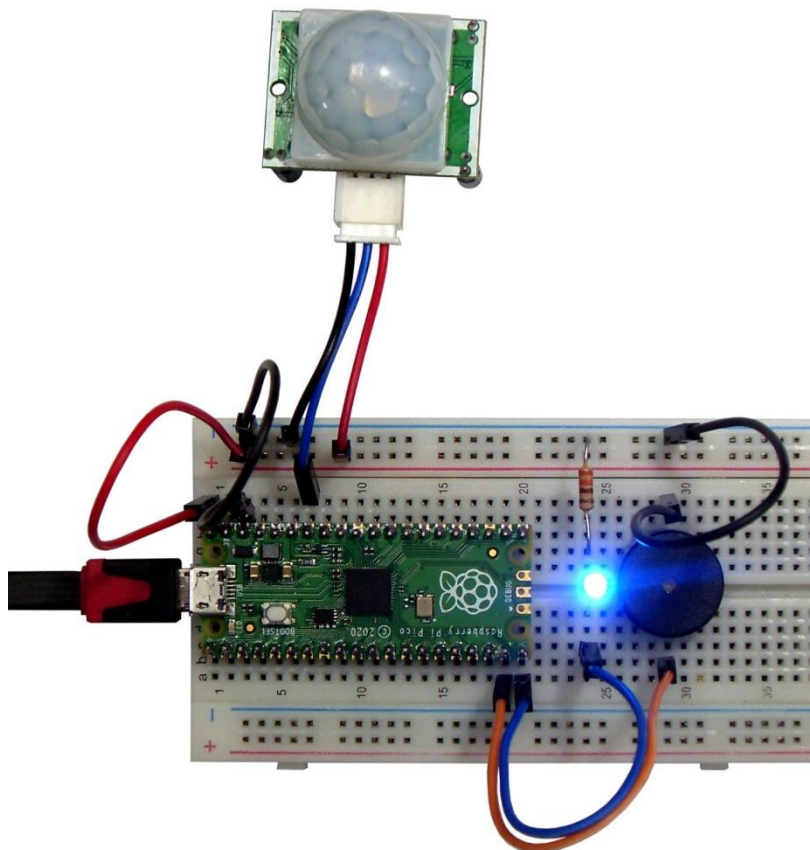


Figure 13. Burglar alarm circuit.

*Experiment 10 PIR Burglar alarm with PIO driven sounder*

The burglar alarm will work with the passive sounder and the code given on page 88 The Book but the pitch of the sounder is a bit low. The following code shows a simple way to implement a 4kHz tone in the code using the “`utime.sleep_us(100)`” function instead of “`utime.sleep_ms(3)`” and increasing the number of iteration in the loop to from 25 to 750.

```
# PIR Burglar alarm with modified software driven sounder "Burglar_alarm_with_passive_sounder.py"
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO" Modified to use a PIO driven sounder
# by Nigel J. Halse 8th April 2021

import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
        print("ALARM! Motion detected!")
        for i in range(50):
            led.value(1)
            for j in range(750):    # 750 cycles
                buzzer.toggle()
                utime.sleep_us(100) # Delay to generate approximately 4kHz
            led.value(0)
            utime.sleep_ms(75)

sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)

while True:
    led.toggle()
    utime.sleep(5)
```

An alternative version of the code using the PIO is also shown in section 3.7 Burglar alarm with PIO driven sounder.

NOTE. Thonny can't stop the code when the processor is executing a long interrupt. Thonny will generate up with a nice warning “**Device is busy or does not respond....**” and should exit the code when the interrupt completes.

## 2.8. Potentiometer and ADC.

### *Experiment 11. Testing the potentiometer.*

The potentiometer we supplied in the kit can be plugged directly into the board.

We can test how the potentiometer works with this simple circuit. See Figure 14. Testing the potentiometer.

Again, we are only using power from the Pi Pico. Wind the knob from low to high and the LED should dim and brighten accordingly. But we find that the LED only turns on when it gets to around the half way mark of the potentiometer. If you replace the Blue LED with a Red Led it will extinguish at a lower position on the potentiometer. This is because the Red LED has a lower “Forward drop” (Vf) than the Blue LED. Roughly 1.8V for the Red LED and 3.4V for the Blue LED. For an LED the forward drop is the voltage at which the diode begins to light up.



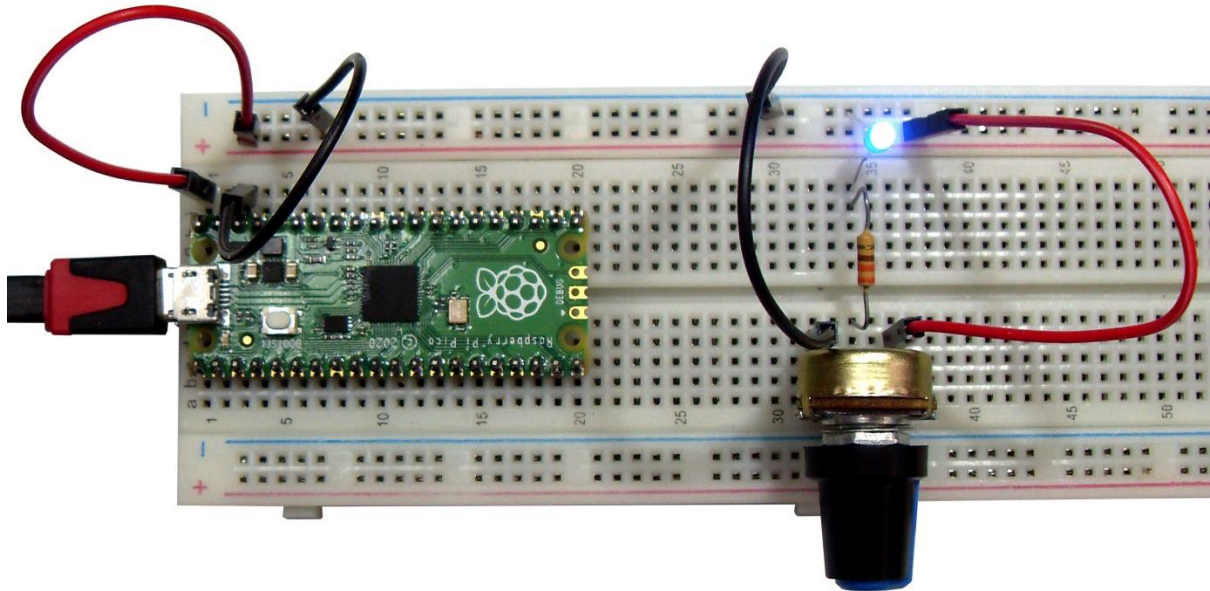
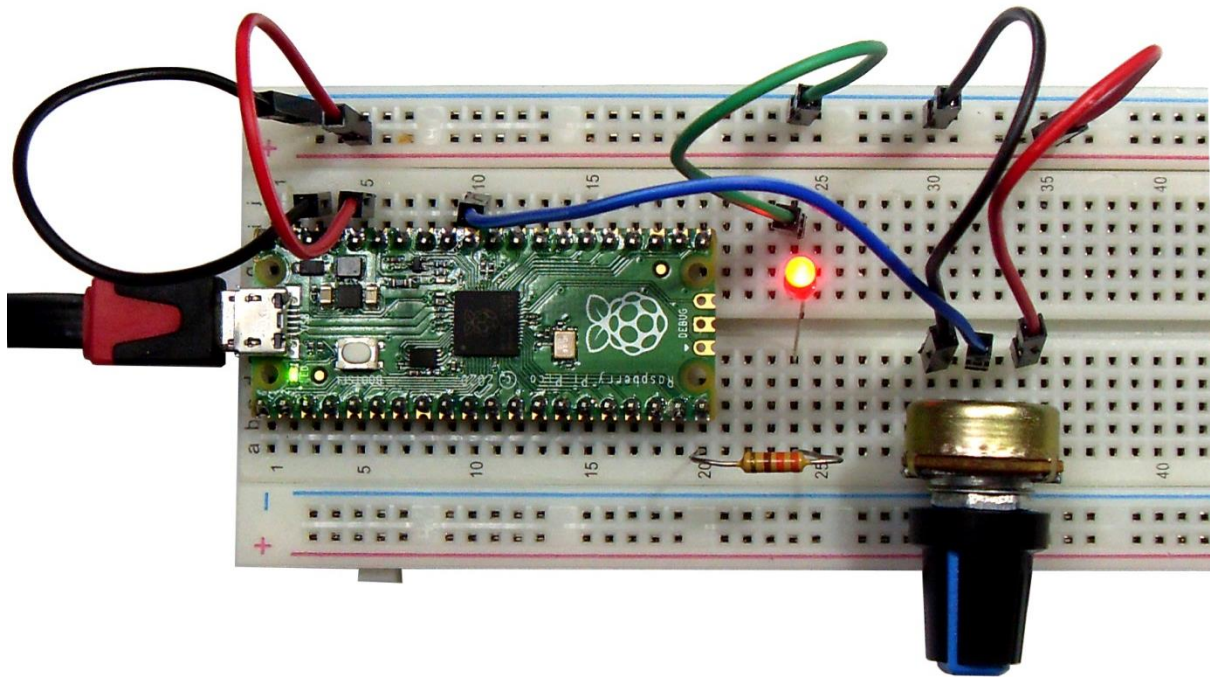


Figure 14. Testing the potentiometer.

This problem can be overcome by driving the LED in PWM mode. This is covered in Chapter 8 of The Book.

### Experiment 12. PWM controlled LED

The resulting circuit for the PWM controlled LED looks like this:



The program for it is on page 102 of The Book and is wonderfully simple for what it is doing.

For a bit of extra fun check out our example with control of both the LED and the sounder in section 3.8 Potentiometer PWM LED and PIO tone.

We are good now till Chapter 10 of The Book. Have fun with data logging on the way there.

## 2.9. I2C and LCD display.

### Experiment 13. I2C LCD display

At last, we can now do a real “Hello World” program.

The LCD supplied on our kit is only set up for I2C communications and operates at 5V so we need a level shifter to convert between the 3V signals of the Pi-Pico to the 5V signals of the I2C LCD display. Please be careful to wire this circuit correctly as applying a 5V signal to the Pi-Pico could damage it beyond repair. There are two pictures and a table to check the wiring against:

See Figure 15. I2C LCD display. and Figure 16. Level shifter details.

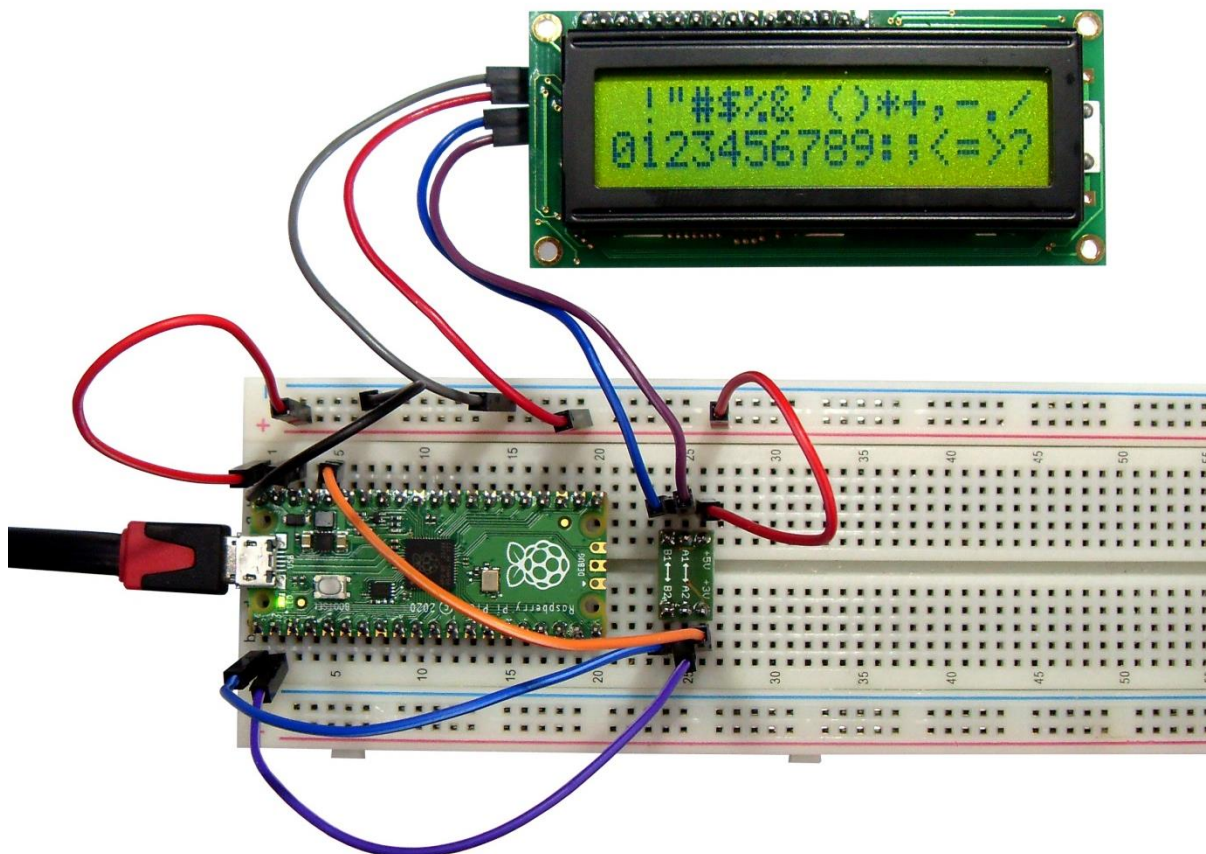


Figure 15. I2C LCD display.

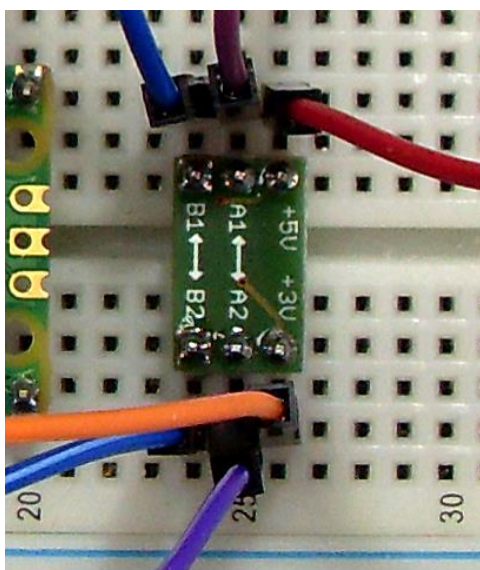


Figure 16. Level shifter details.



Follow the wire colours to wire up the level shifter and display correctly and check against the table below:

**Table 4. I2C LCD Display circuit wiring guide.**

Wire number	Function	Colour	From	To	M-M or M-F
1	0V	Black	Pi-Pico GND [pin 38]	Breadboard -Bus	M-M
2	+USB (5V)	Red	Pi-Pico VBUS [pin 40]	Breadboard +Bus	M-M
3	+3V3 (3.3V)	Orange	Pi-Pico 3V3(OUT) [pin 36]	Level shifter +3V	M-M
4	+5V	Red	Breadboard +Bus	Level shifter +5V	M-M
5	I2C SDA (3.3V)	Blue	Pi-Pico I2C0 SDA [pin 1]	Level shifter B2	M-M
6	I2C SCL (3.3V)	Purple	Pi-Pico I2C0 SCL [pin 2]	Level shifter A2	M-M
7	LCD 0V	Black	Breadboard -Bus	LCD GND	M-F
8	LCD 5V	Red	Breadboard +Bus	LCD Vcc	M-F
9	I2C SDA (5V)	Blue	Level shifter B1	LCD SDA	M-F
10	I2C SCL (5V)	Purple	Level shifter A1	LCD SCL	M-F

To use this display, we need to depart a bit from The Book and use an API and library from <https://github.com/T-622/RPI-PICO-I2C-LCD>.

Follow the instructions given in the link above to load 3 files onto your Pi-Pico then run “pico\_i2c\_lcd\_test.py” to test your display. One of the test screens is shown in Figure 15. I2C LCD display.

If you look through the file “lcd\_api.py” you will find a very comprehensive list of commands for the LCD display. A summary of the LCD commands is given in section 3.11 LCD\_API Commands.

Ok, now we can finally write that “Hello world” program 😊.

```
# Displaying 2 lines of text on a 2 x 16 LCD display "LCD_Hello_World.py"
# Using "lcd_api.py" and "pico_i2c_lcd_test.py" to drive the display using I2C
# controlling a PCF8574 port expander
# Nigel J. Halse 09/04/21

import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

# Get the API
# Library for the PCF8574 I2C port expander
# as a HD44780 LCD display controller

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

# Address of the PCF8574 I2C port expander
# Number of rows (1 or 2)
# Number of columns (16 or 20)

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

# Declare the I2C port and name it
# Declare the LCD and control it using the specified I2C

lcd.clear()
lcd.putstr("Hello World")
lcd.move_to(0,1)
lcd.putstr("I Love You")

# Clear the display and home the cursor position
# Display the message
# Select the second line
# Display another message
```

The Thermometer program becomes:

```
# Reading the internal thermometer and displaying as 1 line of text on a 2 x 16 LCD display "I2C_LCD_Temperature.py"
# Using "lcd_api.py" and "pico_i2c_lcd_test.py" to drive the display using I2C
# controlling a PCF8574 port expander
# Nigel J. Halse 09/04/21

import machine
import utime
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

adc = machine.ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    reading = adc.read_u16() * conversion_factor
    temperature = 25 - (reading - 0.706) / 0.001721
    lcd.clear()
    out_string = "Temp: " + str(temperature)
    lcd.putstr(out_string)
    utime.sleep(2)
```

```
# Get the API
# Library for the PCF8574 I2C port expander
# as a HD44780 LCD display controller
# Address of the PCF8574 I2C port expander
# Number of rows (1 or 2)
# Number of columns (16 or 20)

# Declare the I2C port and name it
# Declare the LCD and control it using the specified I2C

# Define the ADC to use
# Create the conversion factor between the ADC output
# and Deg. C.

# Get the temperature data
# Convert to Deg. C.
# Clear the display and home the cursor position
# Form the message
# Display the message
# Wait 2 seconds before doing it again
```

## 2.10. WS2812 Controllable RGB LED strip

### Experiment 14. PIO controlled RGB LED strip

The WS2812 strip supplied in this kit has been fitted with wires to make it easier to use.

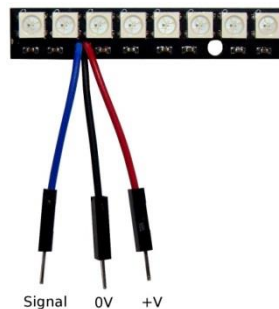


Figure 17. WS2812 RGB LED strip

Each RGB LED draws approximately 50 mA at 5 V with red, green, and blue at full brightness. So when all the LEDs are fully illuminated (all white) the current taken from the USB port is quite considerable (400mA for the 8 RGB LEDs) so you may see flickering or even resetting of the USB port.

The following code has reduced maximum white amplitude and has added code to cycle the display to make it a bit more interesting:

```
# Cycling RGBW Sequence on 8 WS2815 RGB LEDs with reduced amplitude "WS2812 RGB Cycle.py"
# to minimise ripple due to high currents.
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO"
# Modified by Nigel J. Halse 9th April 2021
```

```
import array, utime
from machine import Pin
```

```
import rp2
from rp2 import PIO, StateMachine, asm_pio

# Configure the number of WS2812 LEDs.
NUM_LEDS = 8

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def ws2812():
    T1 = 2
    T2 = 5
    T3 = 3
    label("bitloop")
    out(x, 1) .side(0) [T3 - 1]
    jmp(not_x, "do_zero") .side(1) [T1 - 1]
    jmp("bitloop") .side(1) [T2 - 1]
    label("do_zero")
    nop() .side(0) [T2 - 1]

# Create the StateMachine with the ws2812 program, outputting on Pin(0).
sm = StateMachine(0, ws2812, freq=8000000, sideset_base=Pin(0))

# Start the StateMachine, it will wait for data on its FIFO.
sm.active(1)

# Display a pattern on the LEDs via an array of LED RGB values.
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
while True:
    print("blue")
    for j in range(0, 255):
        for i in range(NUM_LEDS):
            ar[i] = j
            sm.put(ar, 8)
            utime.sleep_ms(10)

    print("red")
    for j in range(0, 255):
        for i in range(NUM_LEDS):
            ar[i] = j << 8
            sm.put(ar, 8)
            utime.sleep_ms(10)

    print("green")
    for j in range(0, 255):
        for i in range(NUM_LEDS):
            ar[i] = j << 16
            sm.put(ar, 8)
            utime.sleep_ms(10)

    print("white")
    for j in range(0, 64):
        for i in range(NUM_LEDS):
            ar[i] = (j << 16) + (j << 8) + j
            sm.put(ar, 8)
            utime.sleep_ms(40)
```

# reduced maximum amplitude

# longer delay time per level

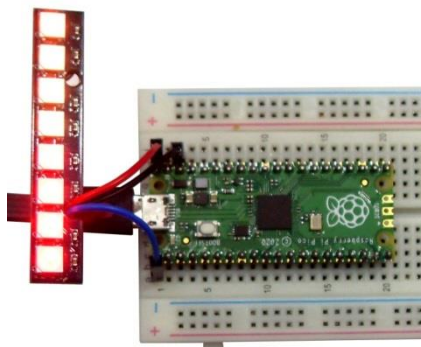


Figure 18. Running WS2812 LED strip.

See section 3.9 More WS2812 examples in More fun. for random and rainbow sequences.

### 3. More fun.

A collection of additional notes, code and ideas for the kit.

#### 3.1. The mathematics of driving LEDs (You can skip this bit if you like)

We can use the Ohm's Law, Kirchhoff's voltage and current laws and the characteristics of LEDs to calculate an appropriate value for the current limiting resistor

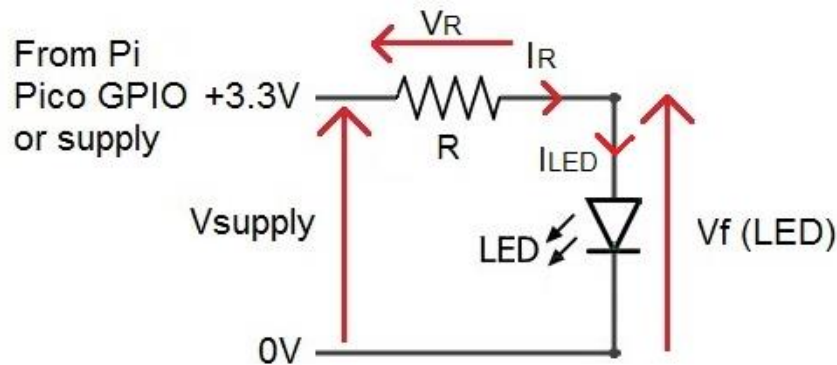


Figure 19. Resistor and LED circuit.

Ohm's law states that The Voltage across a resistor is equal to the value of the resistor x the current going through it.

$$\text{Ohm's Law } V = I \times R$$

$$\text{So, } V_R = I_R \times R$$

Kirchhoff's current law for the above circuit is simply that any current flowing through the resistor will also flow through the LED. None can escape.

$$\text{So, } I_R = I_{LED}$$

Kirchhoff's voltage law for this circuit means that if we add the voltage across the resistor to the voltage across the LED, we will get the voltage of the supply:

$$\text{So, } V_{Supply} = V_R + V_f (LED)$$

For the various LEDs in this kit, we have a maximum forward current ( $I_F$ ) of 20mA and a selection of forward voltages as given in Table 5 below:

Table 5. Forward voltage of LEDs

LED colour	Approximate Forward voltage	Notes
Red	1.8V	626nm
Yellow	2.0V	583nm
Green	2.2V	526nm
Blue	3.4V	470nm
White	3.4V	A Blue LED under a Yellow Phosphor looks White <sup>4</sup>

Note as the frequency of the light emitted from the LEDs goes up (the wavelengths get shorter) then the forward voltage also goes up. They are intimately linked. It takes more voltage to generate a photon blue light than it does to generate a photon of red light.

The current limiting resistor we have supplied for the LEDs is 330R. So, we can work out the worst-case current flowing through the LED:

If we look at the case for the Red LED being operated from the 5V UB supply<sup>5</sup>:

$$V_{\text{Supply}} = V_R + V_{f(\text{LED})}$$

Re arranging gives:

$$V_R = V_{\text{Supply}} - V_{f(\text{LED})}$$

$$V_R = 5V - 1.8V = 3.2V$$

From Ohm's law we can then get the current through both the resistor and the LED:

$$I_R = I_{\text{LED}} = V_R/R = 3.2V/330R = 0.0097A \text{ or } 9.7mA$$

This is well within the capability of the LEDs.

Can you work out the minimum current? i.e., when a blue or white LED is being driven from a 3.3V GPIO pin? Ok, it's a trick question. ☺ The LED does still light and you would need a bit more in-depth information on the LEDs to see why. It is safe to say that the current is quite small but the LEDs are working well<sup>6</sup>.

Just for fun we measured the voltage drop across the resistor when driving a blue LED from 3.3V to be about 0.5V giving 2.8V across the blue LED and a current of 1.5mA.

<sup>4</sup> To the human eye, red light plus green light looks like yellow light. Some of the blue light that is being used to energise the yellow phosphor escapes the device and is mixed with the yellow light of the phosphor. So we see what we think is red plus green and blue light at the same time and perceive this as white.

<sup>5</sup> This is the worst case as it produces the highest voltage across the limiting resistor.

<sup>6</sup> Just for fun we measured the voltage drop across the resistor when driving a blue LED from a GPIO pin at 3.3V to be about 0.5V. This gives 2.8V across the blue LED and a current of 1.5mA.



```
# Loud Tone generator code example "Tone 2.py"
# Using 2 pins to control the sounder. Doubles the voltage and more than doubles the sound level
# Nigel J. Halse 6th April 2021
```

```
from machine import Pin, Timer
```

```
Sounder1 = Pin(12, Pin.OUT)          # Give the ports names
Sounder2 = Pin(13, Pin.OUT)          # The second sounder connection
button = Pin(14, Pin.IN, Pin.PULL_DOWN) # The button input is pulled low
# internally. Pressing the button will
# make the input go high if the other end
# if the switch is at +V
Toggle = 0                           # Declare a variable

timer = Timer()                       # rename "timer()"

def Sounder(timer):                  # Define a function for dealing with the
    # timer interrupt
    global Toggle                   # Indicate that the global variable is
    # required
    if Toggle == 1:
        Sounder1.on()              # Take one side of the sounder high
        Sounder2.off()             # and the other low
        Toggle = 0
    else:
        Sounder1.off()             # Take one side of the sounder low
        Sounder2.on()              # and the other high
        Toggle = 1

# Set up a timer interrupt. # double the frequency required on the sounder
timer.init(freq=8000, mode=Timer.PERIODIC, callback=Sounder)
```

The sounder should be noticeably louder. If in doubt just disconnect either wire from the Pi-Pico and connect it to the 0V rail.

### 3.4. Rising pitch using PIO and sounder

#### *Experiment 16. Rising pitch sounder*

This bit of code uses the PIO to generate a rising pitch on the sounder. Use the wiring configuration given in Figure 8. Sounder wiring.

```
# PIO based tone generator with rising frequency "Rising PIO_tone.py"
# Nigel J. Halse 7th April 2021

import time
from rp2 import PIO, asm_pio
from machine import Pin
Fstart = 2000
Fend = 12000
Finc = 1200
F = Fstart

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW) # Let the MicroPython compiler know that the following is a PIO machine code
definition
def tone():
    wrap_target()
    set(pins, 1)          # 1 instruction
    set(pins, 0)          # 1 instruction plus jump to start
    wrap()

# Instantiate a state machine with the tone program, at 8000Hz, with set bound to Pin(12).
# This will generate a 4kHz square wave on pin 12
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(12))
```



```

def Buzzer(value):
    sm1.active(value)

# Rising tone
while 1:
    sm1 = rp2.StateMachine(1, tone, freq=F, set_base=Pin(12))
    Buzzer(1)
    time.sleep(0.1)
    Buzzer(0)
    F = F + Finc
    if F > Fend:
        F = Fstart

```

# A change of name to make the program easier to read

# A continuous loop

# Declare the state machine

# Turn on the tone

# Wait for a time

# Turn off the tone

# increase the frequency

# restart the sequence

### 3.5. Slightly improved pedestrian crossing

#### *Experiment 17. improved pedestrian crossing*

This version of the code makes sure that the amber and green LEDs are off at the beginning of the sequence and waits for a button press rather than continuously cycling:

```

# Traffic lights with pedestrian crossing sounder version 2 "PIO_Traffic_lights 2.py"
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO" Modified to use a PIO driven sounder
# by Nigel J. Halse 7th April 2021

import machine
import utime
import _thread
from rp2 import PIO, asm_pio
from machine import Pin

led_red = machine.Pin(15, machine.Pin.OUT)
led_amber = machine.Pin(14, machine.Pin.OUT)
led_green = machine.Pin(13, machine.Pin.OUT)
button = machine.Pin(16, machine.Pin.IN, machine.Pin.PULL_DOWN)

global button_pressed
button_pressed = False

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW) # Let the MicroPython compiler know that the following is a PIO machine code
definition
def tone():
    wrap_target()
    set(pins, 1) # 1 instruction
    set(pins, 0) # 1 instruction plus jump to start
    wrap()

# Instantiate a state machine with the tone program, at 8000Hz, with set bound to Pin(12).
# This will generate a 4kHz square wave on pin 12
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(12))

def Buzzer(value): # A change of name to make the program easier to read
    sm1.active(value)

def button_reader_thread():
    global button_pressed
    while True:
        if button.value() == 1:
            button_pressed = True
            utime.sleep(0.01)
_thread.start_new_thread(button_reader_thread, ())

while True:
    if button_pressed == True:
        led_red.value(1)
        for i in range(10):
            Buzzer(1)

```

```

    utime.sleep(0.2)
    Buzzer(0)
    utime.sleep(0.2)
    global button_pressed
    button_pressed = False
    led_red.value(1)
    led_amber.value(0)          # Make sure the amber and green LEDs are off
    led_green.value(0)
    utime.sleep(5)
    led_amber.value(1)
    utime.sleep(2)
    led_red.value(0)
    led_amber.value(0)
    led_green.value(1)
    while button_pressed == False:  # Wait for button press
        pass                       # Do nothing
    utime.sleep(2)
    led_green.value(0)
    led_amber.value(1)
    utime.sleep(5)
    led_amber.value(0)

```

### 3.6. Piezo sounder as a knock sensor

#### *Experiment 18. Energy harvesting with a Piezo sounder*

Can the Piezo drive an input? Yes see <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Knock>

If it can be done on an Arduino™ it can be done by the Pi-Pico.

First, we can try using the sounder as a transducer or sensor. In this mode the sounder will detect sound or vibration and convert it to electrical energy.

Put the sounder, a resistor and 2 back-to-back<sup>7</sup> LEDs in a circuit as shown in Figure 22. If you tap the sounder quite hard with your finger nail the 2 LEDs should light up. The sharper the tap, the brighter the reaction. Note at this point – NO power is being used!! The lights are being lit by the electro-mechanical reaction of the Piezo crystal inside your buzzer!

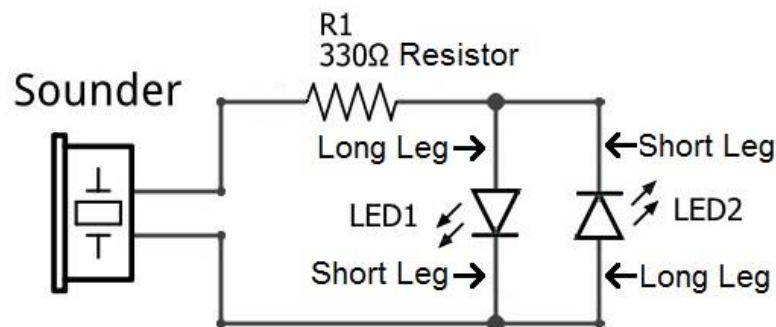


Figure 21 Simple Piezo LED circuit/Energy Harvester.

The resistor is to limit the current to the LEDs and putting 2 LEDs “back-to-back” helps to protect the LEDs from excessive reverse voltages. LEDs can typically handle up to 5V of reverse voltage but will be damaged if forced beyond this. In this arrangement the forward conduction voltage of the first LED protects the second LED from being reverse biased to any significant degree and vice versa. See Figure 22. Sounder driving 2 LEDs below for a suggested arrangement on the breadboard.

<sup>7</sup> This is generally known as reverse parallel. The current generated by the sounder will be oscillating positive and negatively will flow first through one LED and then the other depending on the direction of the current.

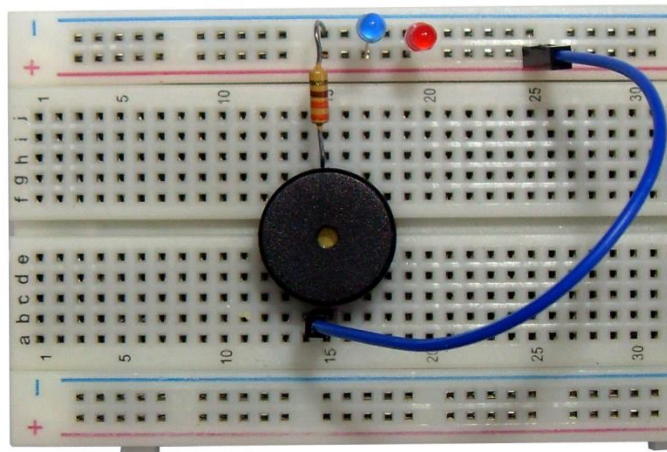


Figure 22. Sounder driving 2 LEDs

This is a form of Energy harvesting - if we attached the sounder to something with a high amount of vibration we could tap off some of the energy to power other things". It is one of the methods used to make kids trainers light up.

### Experiment 19. Using the Piezo sounder as a simple sensor

We can easily detect the voltage generated by the Piezo sounder using the Pi Pico and do something with it...

Connect the sounder to 0V at 1 end and the other via a 330R resistor to GPIO15. Connect an LED to 0V and to GPIO14 via another 330R resistor making sure that the cathode of the LED (Short lead) goes to 0V.

The RP2040 processor has small, reversed biased diodes to +3.3V and 0V supply rails protecting its pins. As long as the current into a pin does not exceed the capabilities of the protection diodes, they will be able to limit the voltages to a safe range for the processor. The 330R between the sounder and the Pi Pico GPIO limits the current into the pin and out through those protection diodes.

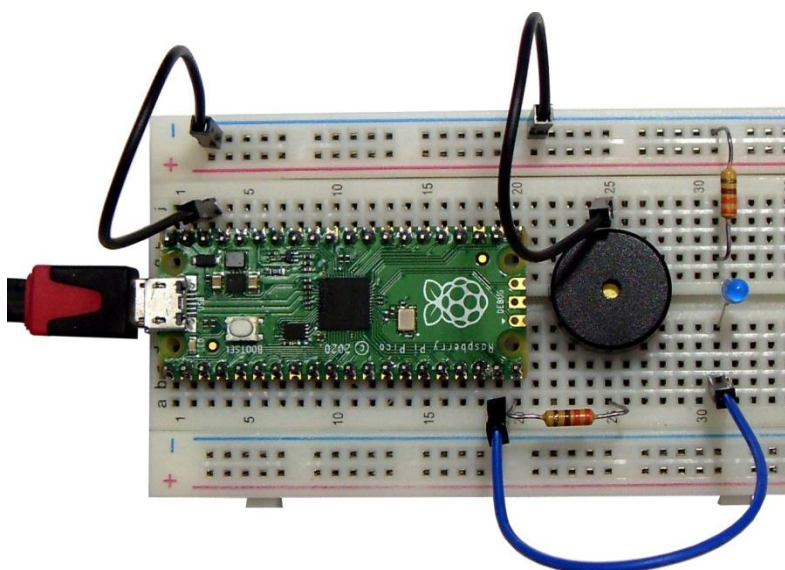


Figure 23. Detecting the sounder on a logic pin.

The following code which will monitor the logic level on GPIO15. If it exceeds the threshold for a 1 then turn on the LED for 1 second. The pull down on GPIO15 is to make the input less sensitive to noise.

```
# Using a Piezo sounder as an input device. "Sounder_sensor_LED.py"  
# If you tap the sounder it will generate a voltage that can easily be detected.  
# by Nigel J. Halse 14th April 2021
```

```
import machine  
import utime
```

```
Sounder = machine.Pin(15, machine.Pin.IN, machine.Pin.PULL_DOWN)  
led = machine.Pin(14, machine.Pin.OUT)
```

```
while True:  
    led.value(0)      # Make sure the LED is off  
    if Sounder.value() == 1: # Check for a voltage  
        led.value(1)    # Turn on the LED  
        utime.sleep(1)  # Wait for a second
```

If you tap the sounder, it will be detected by the processor and turn on the LED for 1 second. It is a very crude way to monitor an analog signal.

### *Experiment 20. Using the ADC to make the Piezo into more sensitive sensor*

If an ADC is used then it is possible to make the input from the sounder more sensitive. This time the signal is taken in via ADC 0 on GPIO26. A 1Meg Ohm resistor from the ADC input to 0V is used to quieten down the signal.

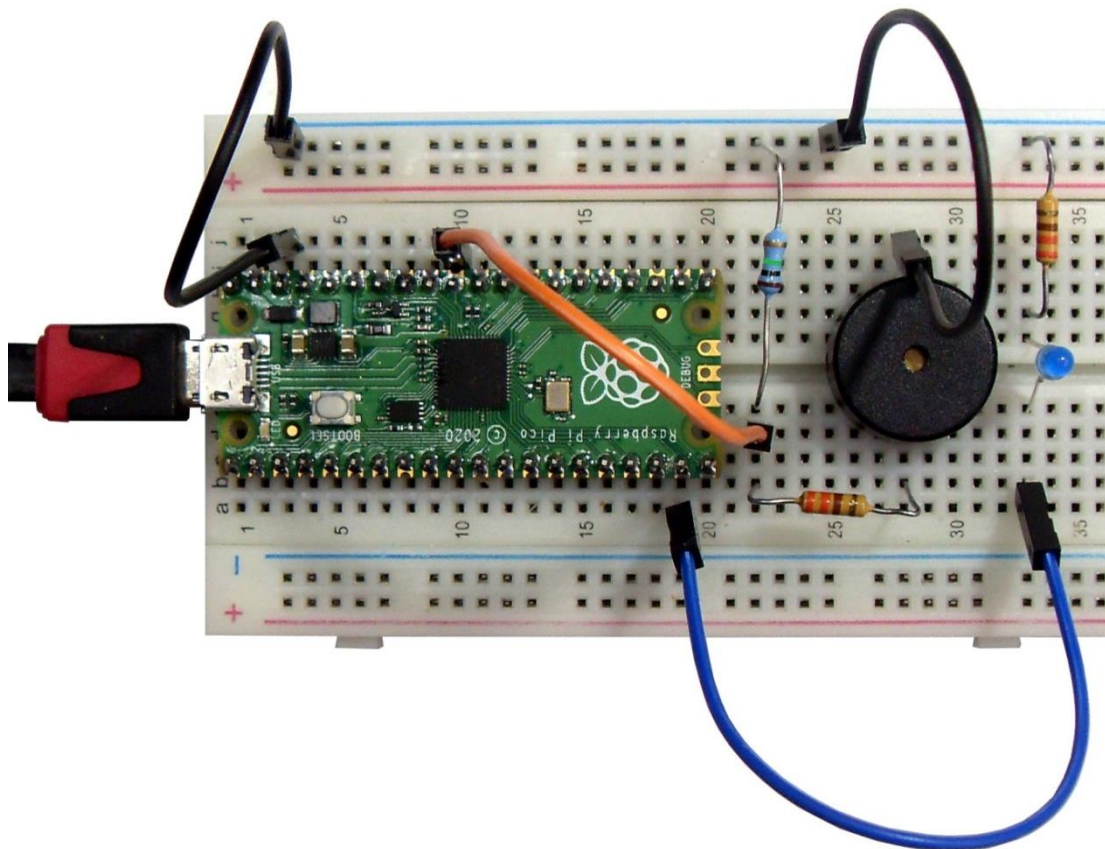


Figure 24. Detecting the sounder using the ADC for more sensitivity.

Enter the code below to improve the performance of the sounder and make a clap sensor.

```
# Using a Piezo sounder as an input device via the ADC. "Sounder_sensor_LED_using ADC.py"  
# If you click your finger above the sounder the sounder
```

```
# it will generate a voltage that can easily be detected.
#
# By Nigel J. Halse 14th April 2021

import machine
import utime

Sounder = machine.ADC(26)
Threshold = 2000

led = machine.Pin(14, machine.Pin.OUT)

while True:
    led.value(0)          # Make sure the LED is off
    if Sounder.read_u16() > Threshold: # Check for a voltage
        led.value(1)      # Turn on the LED
        utime.sleep(1)     # Wait for a second
```

The sounder will now respond to a slight touch instead of a tap and it should respond to a hand clap.

Note. Make sure you may have included the 1Meg Ohm resistor. Also it may be necessary to play around with the value of “Threshold” in the code to allow for differences in the performance of the Piezo and your ambient noise level.

### **3.7. Burglar alarm with PIO driven sounder**

#### *Experiment 21. Burglar alarm with PIO driven sounder*

This version of the code uses the PIO to generate the sounder tone in the same ways as the traffic lights. Use the same circuit as Figure 13. Burglar alarm circuit. In the code “`buzzer = machine.Pin(14, machine.Pin.OUT)`” needs to be deleted and the rest of the code modified as below.

```
# PIR Burglar alarm with PIO driven sounder. "PIR_LED_and_PIO_tone.py"
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO" Modified to use a PIO driven sounder
# by Nigel J. Halse 8th April 2021

import machine
import utime
from rp2 import PIO, asm_pio
from machine import Pin

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
#buzzer = machine.Pin(14, machine.Pin.OUT)

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW) # Let the MicroPython compiler know that the following is a PIO machine code
    # definition

def tone():
    wrap_target()
    set(pins, 1) # 1 instruction
    set(pins, 0) # 1 instruction plus jump to start
    wrap()

# Instantiate a state machine with the tone program, at 8000Hz, with set bound to Pin(12).
# This will generate a 4kHz square wave on pin 14
sm1 = rp2.StateMachine(1, tone, freq=8000, set_base=Pin(14))

def Buzzer(value): # A change of name to make the program easier to read
    sm1.active(value)

def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
```



```

    print("ALARM! Motion detected!")
    for i in range(50):
        toggle()
        Buzzer(1)
        utime.sleep_ms(50)
        Buzzer(0)
        utime.sleep_ms(50)
    LED.value(0)      # Added to ensure LED is off on exit

sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)

while True:
    led.toggle()
    utime.sleep(5)

```

### 3.8. Potentiometer PWM LED and PIO tone

*Experiment 22. Using the potentiometer to control PWM and PIO.*

It just had to be done... This circuit and code will produce a varying LED amplitude and tone frequency on rotating the potentiometer knob.

The averaging and banding in the code is to get around 2 facts. Firstly, When the PIO is re-declared it is initially halted and has to be restarted causing a discontinuity in the tone. Secondly, the driving of the Piezo sounder seems to cause the ADC to produce noisy results so causing the pitch to warble.

The averaging and banding reduce the effects by slowing down changes to the pitch value through a simple low pass filter and then only changing the pitch if it has changed significantly.

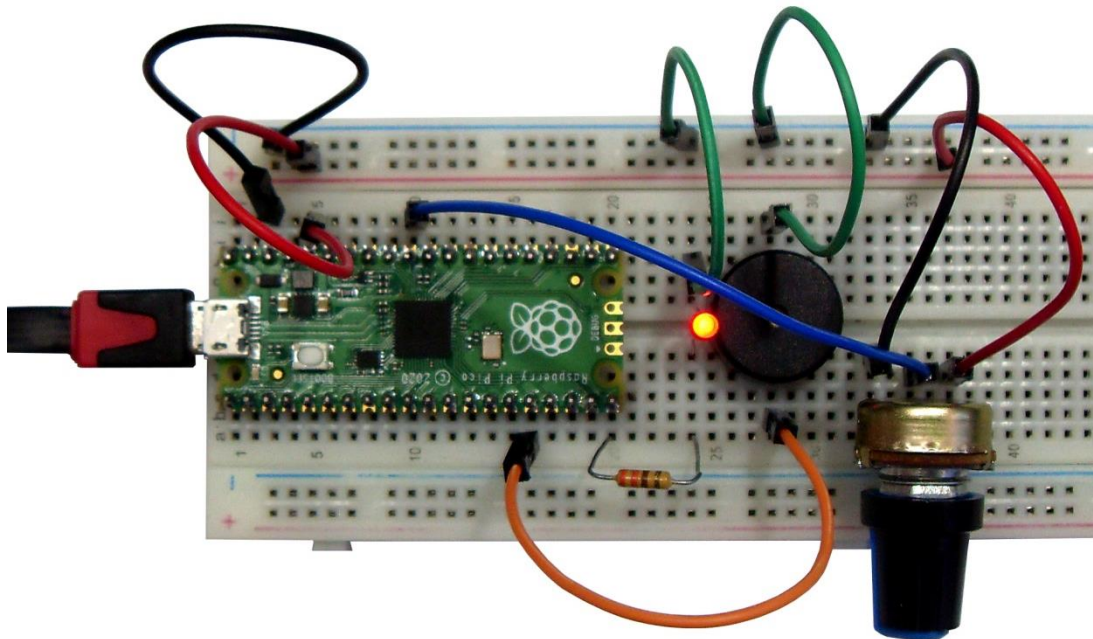


Figure 25. Potentiometer PWM LED and PIO tone circuit

```

# Potentiometer controlling an LED using PWM and the frequency of the sounder using the PIO "Pot_PWM_LED_and_tone.py"
# From "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO" Modified to use a PIO driven sounder
# by Nigel J. Halse 8th April 2021

```

```

import machine
import utime
import time
from rp2 import PIO, asm_pio
from machine import Pin

```

```

F_start = 2000
F_end = 12000
F = 2000

```

```
v_old = 0
v_diff = 500
v_ave = 0

potentiometer = machine.ADC(26)
led = machine.PWM(machine.Pin(15))
led.freq(1000)

# Define the tone program. It has one GPIO to bind to on the set instruction, which is an output pin.
@asm_pio(set_init=rp2.PIO.OUT_LOW) # Let the MicroPython compiler know that the following is a PIO machine code
definition
def tone():
    wrap_target()
    set(pins, 1) # 1 instruction
    set(pins, 0) # 1 instruction plus jump to start
    wrap()

while True:
    v = potentiometer.read_u16()
    led.duty_u16(v)
    F = int(F_start + v * (F_end - F_start)/65535)
    v_ave = v_ave + ((v - v_ave)/100)
    if (abs(v_old - v_ave) > v_diff):
        sm1 = rp2.StateMachine(1, tone, freq=F, set_base=Pin(12)) # redefine the PIO
        sm1.active(1) # Turn on the tone
        v_old = v_ave
        # time.sleep(0.1) # Optional loop delay
```

The commented out last line will change the way the steps due to a change in pitch sound by slowing them down if reinstated.

There is possibly a way to make the frequency of the PIO state machine change without having to restart it but we have not found it yet.

## 3.9. More WS2812 examples

Two more code examples for the WS2812 LED strip. Use the circuit shown in Figure 18. Running WS2812 LED strip.

### 3.9.1. Rainbow on WS2812

#### *Experiment 23. Rainbow on WS2812*

This experiment produces a very pretty moving rainbow of colours. To do this we set up an array of colours and cycle through them incrementing an offset each time we cycle through.

```
# RGBW and other colours RGBW Sequence on 8 WS2815 RGB LEDs "WS2812 RGB rainbow.py"
# Adapted from "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO"
# by Nigel J. Halse 9th April 2021

import array, utime
import random
from machine import Pin
import rp2
from rp2 import PIO, StateMachine, asm_pio

# Configure the number of WS2812 LEDs.
NUM_LEDS = 8
NUM_COLOURS = 8

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def ws2812():
    T1 = 1
    T2 = 4
    T3 = 2
```



```

label("bitloop")
out(x, 1) .side(0) [T3]
jmp(not_x, "do_zero") .side(1) [T1]
jmp("bitloop") .side(1) [T2]
label("do_zero")
nop() .side(0) [T2]

# Create the StateMachine with the ws2812 program, outputting on Pin(0).
sm = StateMachine(0, ws2812, freq=8000000, sideset_base=Pin(0))

# Start the StateMachine, it will wait for data on its FIFO.
sm.active(1)

# Display a pattern on the LEDs via an array of LED RGB values.
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
colours = array.array("I", [0 for _ in range(NUM_COLOURS-1)])
colours = [0,65280,16776960,16711680,16711935,255,65535,16777215] # off,Red,Yellow,Green,Cyan,Blue,Magenta,White
while True:
    for i in range(NUM_COLOURS):          # index offset
        for j in range(NUM_LEDS):
            x=((i + j) % NUM_COLOURS)
            ar[j] = colours[x]
        sm.put(ar,8)
        utime.sleep_ms(50)

```

See Rainbow.AVI in the zip file downloaded with this document for a short video of the display running.

### 3.9.2. Random colours

#### *Experiment 24. Random colours*

This experiment uses a random number to select colours for each of the LEDs in the WS2812 strip.

```

# RGBW and other colours in random Sequence on 8 WS2815 RGB LEDs "WS2812 RGB random.py"
# Modified from "GET STARTED WITH MICROPYTHON ON RASPBERRY PI PICO"
# by Nigel J. Halse 9th April 2021

```

```

import array, utime
import random
from machine import Pin
import rp2
from rp2 import PIO, StateMachine, asm_pio

```

```

# Configure the number of WS2812 LEDs.
NUM_LEDS = 8

```

```

@asm_pio(sideset_init=PIO.OUT_LOW, out_shiftdir=PIO.SHIFT_LEFT, autopull=True, pull_thresh=24)
def ws2812():
    T1 = 1
    T2 = 4
    T3 = 2
    label("bitloop")
    out(x, 1) .side(0) [T3]
    jmp(not_x, "do_zero") .side(1) [T1]
    jmp("bitloop") .side(1) [T2]
    label("do_zero")
    nop() .side(0) [T2]

```

```

# Create the StateMachine with the ws2812 program, outputting on Pin(0).
sm = StateMachine(0, ws2812, freq=8000000, sideset_base=Pin(0))

```

```

# Start the StateMachine, it will wait for data on its FIFO.
sm.active(1)

```

```

# Display a pattern on the LEDs via an array of LED RGB values.
ar = array.array("I", [0 for _ in range(NUM_LEDS)])
colours = array.array("I", [0 for _ in range(7)])

```

```
colours = [0,255,65280,65535,16711680,16711935,16776960,16777215]
while True:
    for i in range(NUM_LEDS):
        x=(random.randint(0, 7)) # Pick a random number between 0 and 7
        ar[i] = colours[x]
    sm.put(ar,8)
    utime.sleep_ms(100)
```

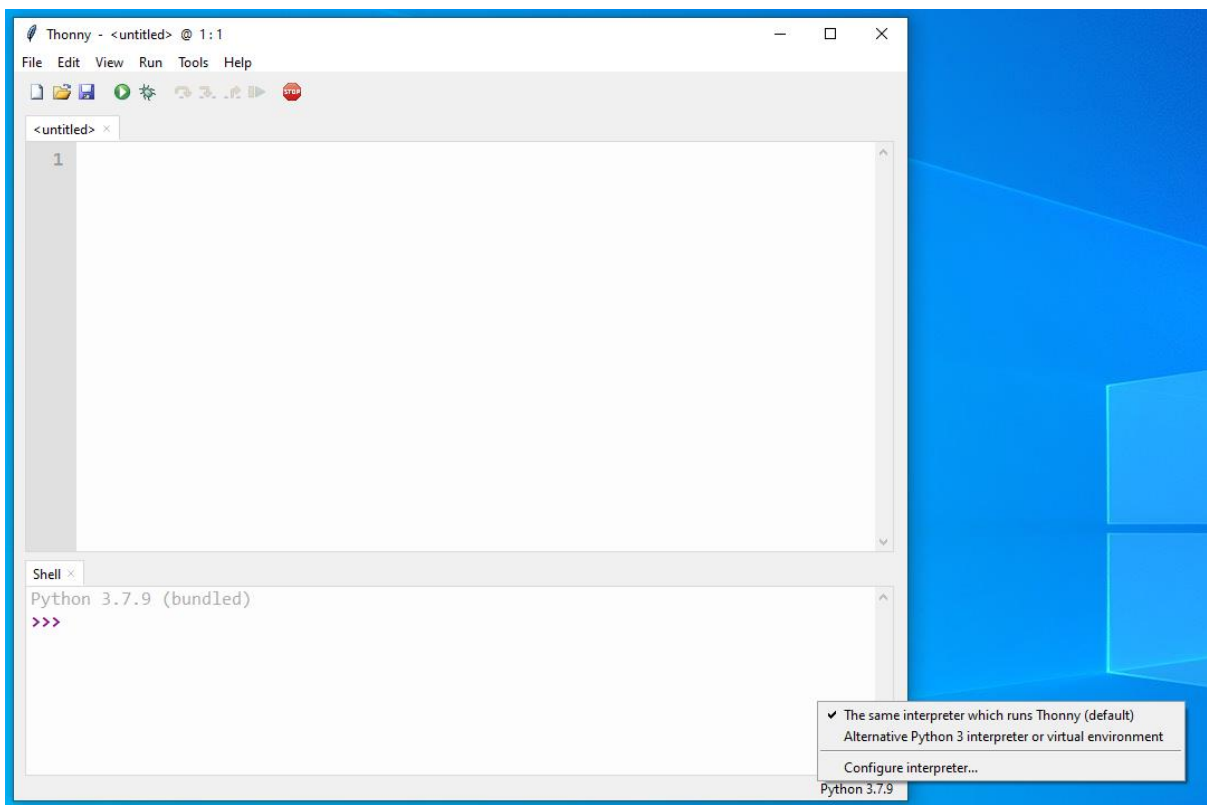
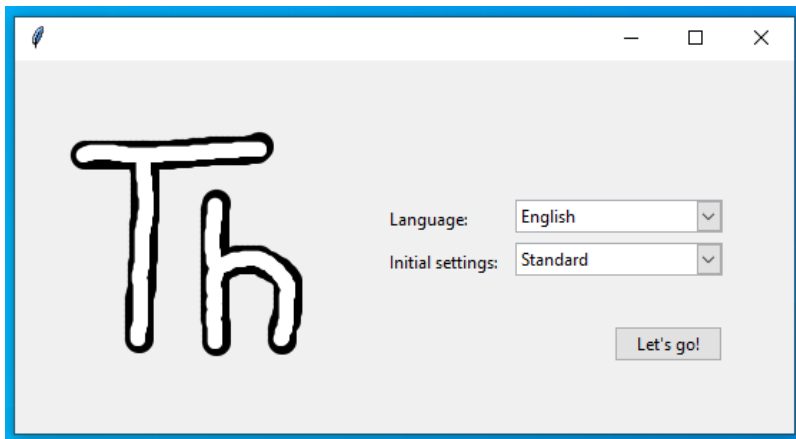
### 3.10. Installing Thonny on Windows™

The best way we found to install Thonny on a Windows 7 PC was to use the instructions given at <https://projects.raspberrypi.org/en/projects/getting-started-with-the-pico/1>

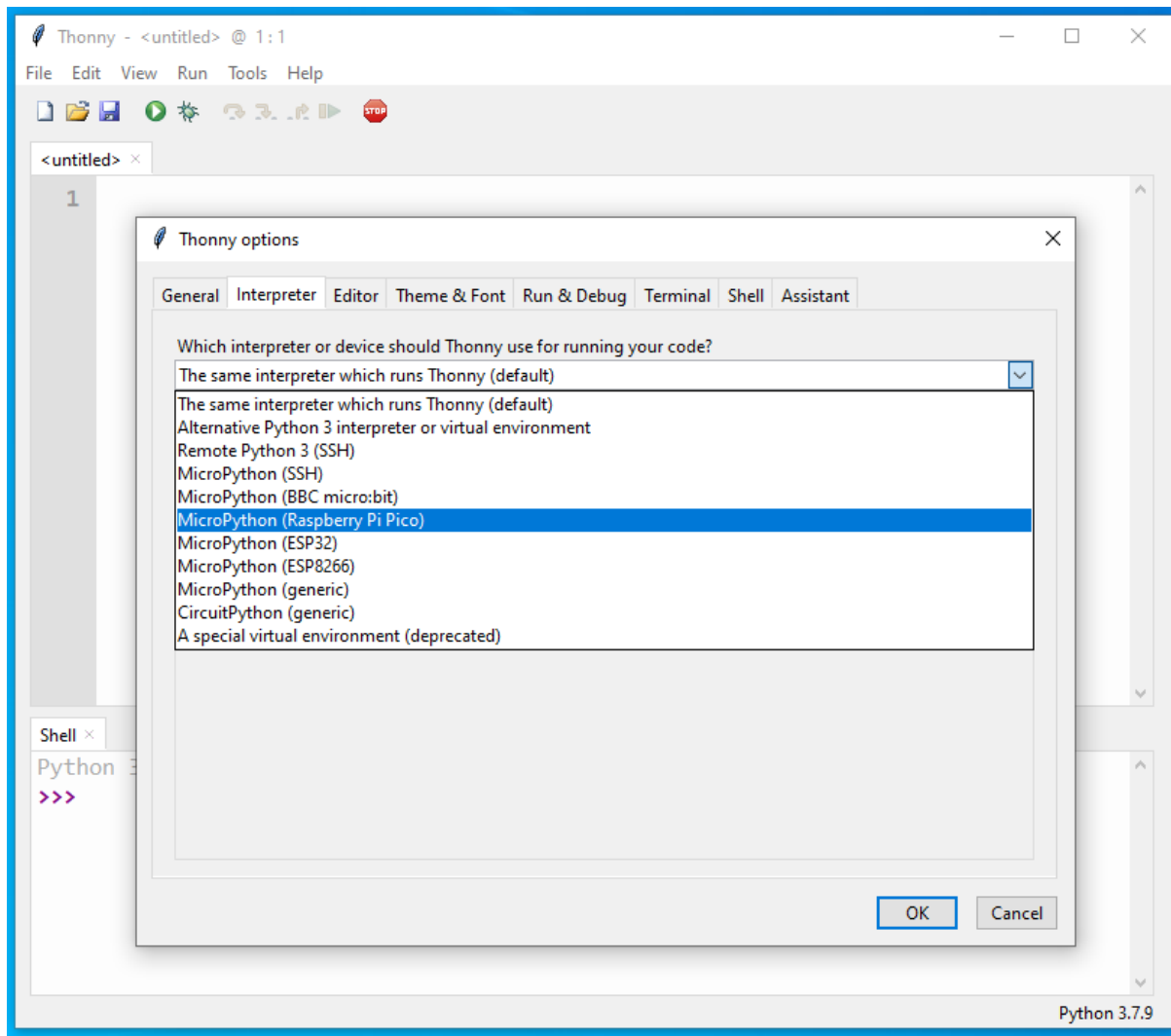
Follow the instructions to “Install Thonny” and select “Install Thonny on other operating systems” at the top of the second page.

Navigate to [thonny.org](https://thonny.org) as suggested, download the Windows™ version and install it.

After installing, run Thonny, set you language, leave initial settings as Standard and click on “Let’s go”.



Click on the interpreter in the lower right-hand corner. Select “Configure interpreter”...



Choose “MicroPython (Raspberry Pi Pico)” as the interpreter.

This is the point where MicroPython will be downloaded to the Pi Pico.

Thonny will ask you to plug in your Pi Pico whilst the onboard button is pressed, and try to connect to it. When it does, download MicroPython to it.

However, you may find that Thonny can’t see the Pi Pico port. If so, then there is probably a driver missing.

We found a solution given by “One Transistor” <https://www.onetransistor.eu/2021/02/set-up-raspberry-pi-pico-for-micropython.html> that suggests adding a driver.

Download Zadig from <https://zadig.akeo.ie/> and run it.

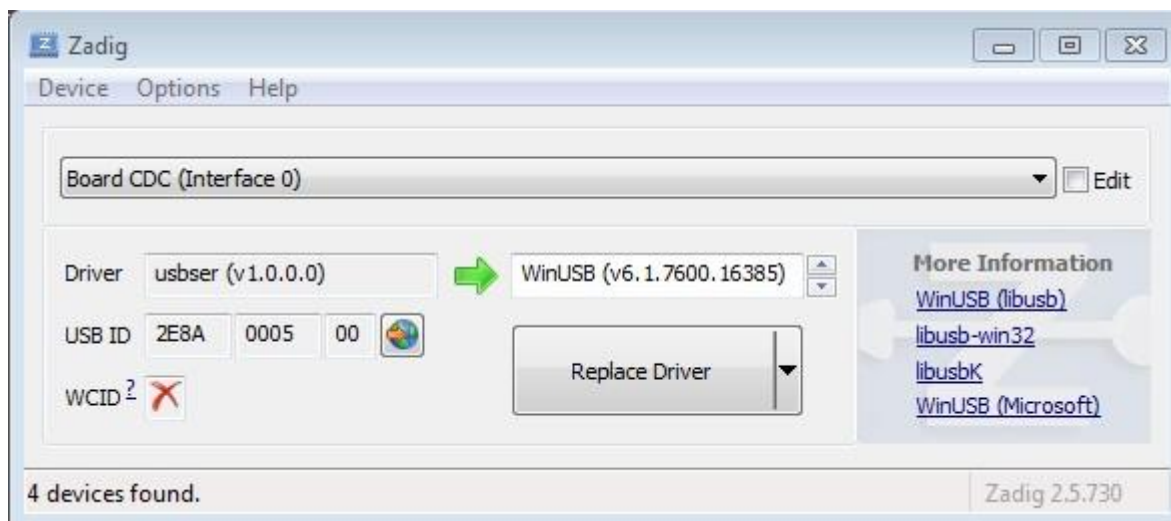


Figure 26. Zadig

It may find that the Pi Pico is missing a driver if so, then select USB Serial (CDC) and install the driver. You should find that Thonny can now see the required port and you are good to go.

If there is a driver installed but it is the wrong one then you will have to use Options>List All Devices. This will show all connected USB objects. Select the “Board CDC(Interface 0)” device then select “USB Serial (CDC)”. If this works then Thonny will suddenly be able to see the Pi Pico.

**Remember.** Once the MicroPython has been loaded onto the Pi Pico then you will no longer need to hold down the Boot button when plugging in the Pi Pico.

Also, if you buy your Pi Pico from us it will have the pins soldered on and MicroPython pre loaded for testing.

### 3.11. LCD\_API Commands

Table 6. LCD\_API commands

Instruction	Description
clear():	Clears the LCD display and moves the cursor to the top left corner
show_cursor():	Causes the cursor to be made visible
hide_cursor():	Causes the cursor to be hidden
blink_cursor_on():	Turns on the cursor, and makes it blink
blink_cursor_off():	Turns on the cursor, and makes it no blink (i.e. be solid)
display_on():	Turns on (i.e. unblanks) the LCD
display_off():	Turns off (i.e. blanks) the LCD
backlight_on():	Turns the backlight on.  This isn't really an LCD command, but some modules have back light controls which are activated by 1 bit of the PCF8574 parallel to serial converter chip in the I2C interface.
backlight_off():	Turns the backlight off.

	This isn't really an LCD command, but some modules have backlight controls which are activated by 1 bit of the PCF8574 parallel to serial converter chip in the I2C interface.
<code>move_to(cursor_x, cursor_y):</code>	Moves the cursor position to the indicated position.  The cursor position is zero based (i.e. <code>cursor_x == 0</code> indicates first column).
<code>putchar(char):</code>	Writes the indicated character to the LCD at the current cursor position, and advances the cursor by one position.
<code>putstr(string):</code>	Write the indicated string to the LCD at the current cursor position and advances the cursor position appropriately.
<code>custom_char(location, charmap):</code>	Write a character to one of the 8 CGRAM locations, available as <code>chr(0)</code> through <code>chr(7)</code> .



## 4. Pi-Pico ideas

For project ideas for Pi-Pico check out and search for Raspberry Pi Pico on:

- <https://www.instructables.com>
- <https://hackaday.com>
- <https://www.tomshardware.com>
- <https://www.electronicshub.org>
- <http://www.penguintutor.com>
- <https://www.hackster.io/>
- <https://www.youtube.com/>
- <https://makersportal.com>

The list is growing all the time. Just type “Raspberry Pi Pico™” (Without the “™” ☺) into a web browser and lots of fun stuff will pop up.

## 5. The small print

### Please note:

1. This is a collection of individual electronic components for a responsible adult to use, following specific instructions for experiments, and using only a Raspberry Pi Pico™ on a computer USB connection for the power. "No liability can be accepted for injury, fire or damage, howsoever caused etc."
2. Users are reminded not to use any damaged goods, and are reminded not to expose components or wires to any substances or situations which may cause them to break or degrade.
3. The components of this kit are RoHS compliant to the best of our knowledge and the soldering we have completed is compliant to RoHS.
4. GreenHalse Electronics Ltd. are fulfilling their Distributor Obligations as a member of the WEEE Distributor Takeback Scheme through Valpak WEEE Retail Services - Registration ID : 416634

Recycling your old electricals is easy!

Recycle your electrical and electronic devices free at your local recycling centre. To find your nearest centre, visit the Recycle More [www.recycle-more.co.uk](http://www.recycle-more.co.uk) website and type in your postcode.



5. This is not a toy. This kit contains small parts – keep out of the reach of young children. Choking hazard.
6. Software Disclaimer

The software provided with this kit is free to use but please leave the author details in place. Our software is covered by this disclaimer: While GreenHalse Electronics Ltd makes every effort to deliver high quality products; we do not guarantee that our products are free from defects. Our software is provided "as is," and you use the software at your own risk. We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied. No oral or written communication from or information provided by GreenHalse Electronics Ltd shall create a warranty. Under no circumstances shall GreenHalse Electronics be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if GreenHalse Electronics has been advised of the possibility of such damages.

## 6. Glossary

ADC	Analog to Digital Converter. The Pi Pico has 3 of them available.
Anode	The positive side of a diode such as an LED.
API	Application Programming Interface. In this case it is generally chunks of pre-written software embedded within MicroPython or a library used to control and configure the RP2040 microcontroller.
Compiler	In the computing world a compiler will examine a program written in a computer Language such as Arduino, basic, C, Python, Pascal and many others and generate machine code specifically for the target processor. This machine code file is then made available for the processor to run as often as required. The compiler is no longer needed and the resultant machine code is very fast and compact.
Cathode	The negative side of a diode such as an LED.
IDE	Integrated Development Environment i.e. Thonny, Arduino, PyCharm and many others. These programs bring together tools and compilers to help with the process of developing software for various hardware such as the Pi-Pico, Raspberry Pi, Arduino™ boards and may other boards and processors.
Interpreter	In the computing world, an Interpreter will execute a program directly from text written in a programming language such as MicroPython or Basic. Although modern interpreters are very clever they are unlikely to be as fast as in operation as compiled code. They also are required to be installed on the target processor taking up valuable memory space.
Pk-Pk	Peak to Peak. The maximum voltage across a device. Generally used for AC signals.
PIO	Programmable Input/Output. The Pi Pico has 8 independent IO processors that can be reprogrammed on the fly to perform high speed communications and control functions and off load otherwise intensive tasks from the processor.
Pi Pico	Short for “Raspberry Pi Pico™” in this document.
PWM	Pulse Width Modulation. Rapidly turning a signal off and on where the ratio of the off to on time changes to convey information or control power.
STEM	STEM Learning operates the National STEM Learning Centre and Network, alongside other projects supporting STEM education <a href="http://www.stem.org.uk">www.stem.org.uk</a> The term STEM is worldwide to promote the Science, technology, engineering, and mathematics disciplines.
The Book	“Getting started with MicroPython on Raspberry Pi Pico” published by Raspberry Pi™ Press. ISBN: 978-1-912047-86-4
Thonny	A nice, easy and Free Python Integrated Development Environment (IDE) developed by University of Tartu, Estonia, and supported by Cybernetica and the Raspberry Pi Foundation.
URL	Uniform Resource Locator otherwise known as a web address.
USB	Universal Serial Bus. This port on a PC, MAC or Raspberry Pi™ is used by this kit to provide power and data to the Raspberry Pi Pico™ and the experiments.

Table 7. Glossary

Version number	Date	Notes
1.0	23 <sup>rd</sup> April 2021	First release
1.1	28 <sup>th</sup> April 2022	Corrected pin number in I2C LCD Display circuit wiring guide. Added rear view of I2C, 2x16 char LCD Display in contents photographs. Lots of grammar and spelling corrections! 😊