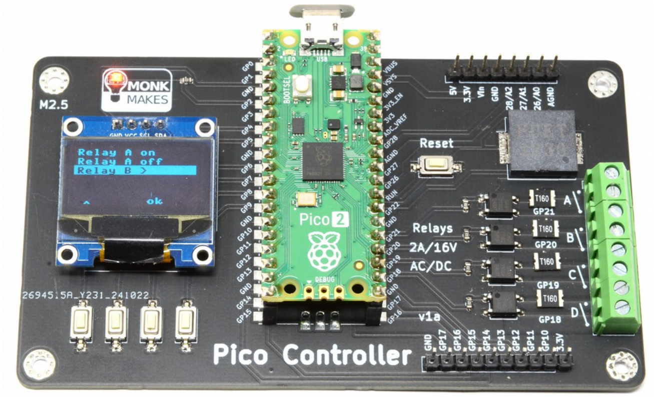
Instructions:

Pico Controller



A 4-channel low-voltage Solid State Relay controller designed for the Raspberry Pi Pico. Ideal for projects involving:

- Model railroads
- RV / caravan
- 12V lighting projects
- Aquarium and greenhouse automation

Features of the product are:

- 4 x 2A (peak) 16V SSRs
- 128x64 pixel monochrome OLED display
- 4 x navigation buttons
- Buzzer
- Reset button
- Access to other GPIO pins
- Extensive MicroPython modules and example code
- Mounting holes

Note: Raspberry Pi Pico not included

Instructions version 1a.

Table of Contents

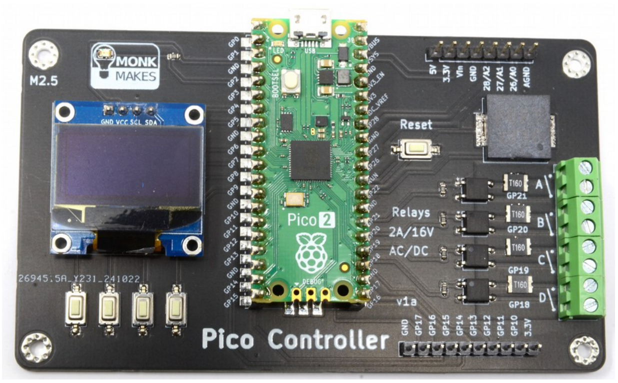
Warning

This product is for switching **low voltage** DC and AC only. Do **NOT** connect to domestic 110 or 220V AC.

Using the Pico Controller

The MonkMakes Pico Controller is for when you need a microcontroller like the Raspberry Pi Pico to control things. That is, to turn things on and off, or control the power to low voltage loads like lights, motors or heaters. It does this using four solid-state relays that can switch low-voltage AC or DC at a peak current of 2 Amps and continuous for 1.5A and up to maximum of 16V. All the relay outputs are protected by self-resetting *polyfuses*.

The Pico Controller also has a small OLED display and four push switches that can provide a simple user interface for your project.

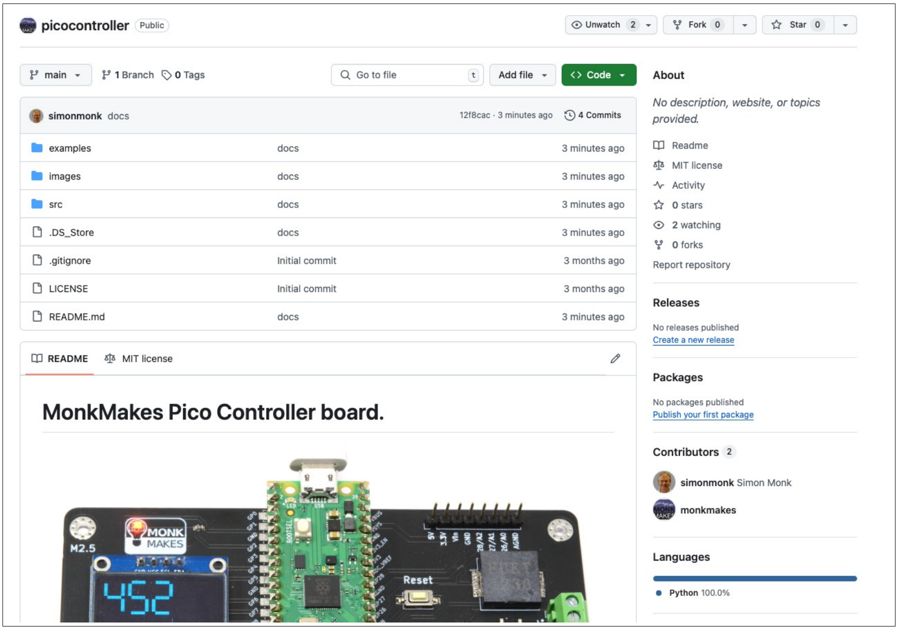


Pico GPIO (General Purpose Input Outputs) that are not used by the display, switches, relays or built-in buzzer can also be accessed using the header pins on the board, allowing sensors and other devices to be attached.

A Raspberry Pi Pico (not included) fits into a socket on the board, allowing different variants of the Pico to swapped in and out. Perhaps a Pico 2 for extra performance, or a Pico 2W for WiFi capability.

Downloading the Examples

To download the ZIP archive containing MicroPython example programs and useful modules, visit https://github.com/monkmakes/picocontroller



picocontroller Public

Unwatch 2 | Fork 0 | Star 0

main | 1 Branch | 0 Tags | Go to file | Add file | Code

simonmonk docs — 12f8cac · 3 minutes ago — 4 Commits

examples	docs	3 minutes ago
images	docs	3 minutes ago
src	docs	3 minutes ago
.DS_Store	docs	3 minutes ago
.gitignore	Initial commit	3 months ago
LICENSE	Initial commit	3 months ago
README.md	docs	3 minutes ago

README | MIT license

MonkMakes Pico Controller board.

About

No description, website, or topics provided.

Readme
MIT license
Activity
0 stars
2 watching
0 forks
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

simonmonk Simon Monk
monkmakes

Languages

Python 100.0%

Click on the Code button, select Download ZIP and then extract the downloaded ZIP archive.

If you are familiar with git and would prefer to download the examples using the command line, then you can do so with the command:

```
$ git clone https://github.com/monkmakes/picocontroller.git
```

With the extracted archive, you will find folders called:

- examples – MicroPython examples for the Pico Controller
- src – modules in MicroPython to use in projects

Getting Started

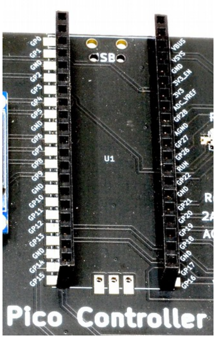
Although you can program Raspberry Pi Picos in a variety of programming languages, in these instructions, we have settled on the most popular of these (MicroPython).

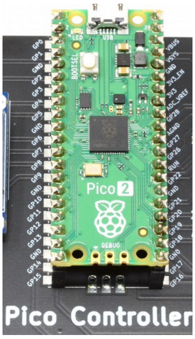
To simplify access to the PicoController board, MonkMakes have developed some code modules.

Plugging and unplugging a Pico

If your Pico was supplied without header pins attached, you will need to solder these on yourself (see Appendix A).

The Pico (or any Pico model) fits into the header sockets with the USB port towards the top.





When pushing the Pico's pins into the socket, line them up carefully, before pushing the Pico into place, to avoid any bending of the pins.

Similarly when removing the Pico, carefully pull it out a few mm on the top and then a few mm on the bottom, repeating until you feel the Pico come free, without bending the pins.

Thonny

Thonny is a Python editor to run on your computer, that allows you to edit MicroPython programs before running them on your Pico. Thonny is available for most operating systems including Raspberry Pi OS. You will need to install it following the instructions here:

https://thonny.org

Installing the Modules

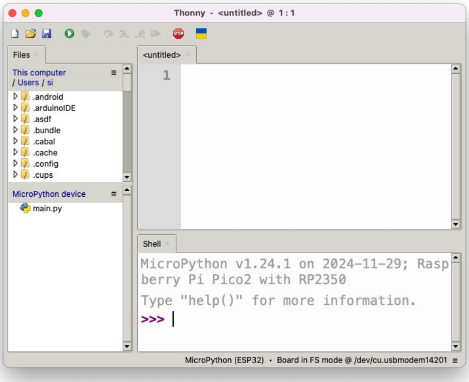
Thonny, allows you to run a program on your computer's file system, as if the program was stored on your Pico. When you press the *Run* button, it sends the code shown in the editor window to the Pico where it is then run. However, if that code uses modules of code, then those modules must actually be installed onto the Pico's file system before the program will work.

To install the modules for the Pico Controller (that you downloaded from Github earlier) onto your Pico, you need to use Thonny.

The steps for this are:

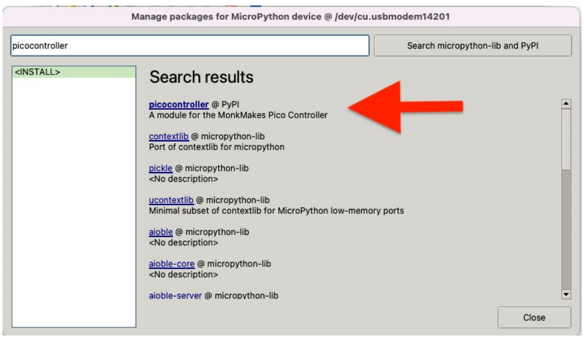
1. Connect your Pico to your computer by USB, then start Thonny and make sure that you can see the >>> prompt in the console. If its not there, press CTRL-c or the Thonny Stop button. If this doesn't work, see Appendix B (trouble-shooting).

2. On the Thonny *View* menu, select *Files*. The Thonny window should now look like this, with the files of the Pico listed on the bottom left (in this case just *main.py*).



```
MicroPython v1.24.1 on 2024-11-29; Raspberry Pi Pico2 with RP2350
Type "help()" for more information.
>>> 
```

3. From Thonny's *Tools* menu, select the *Manage Packages..* option, and type *picocontroller* into the search area and hit Enter.


Manage packages for MicroPython device @ /dev/cu.usbmodem14201

picocontroller

Search results

picocontroller @ PyPI
A module for the MonkMakes Pico Controller

contextlib @ micropython-lib
Port of contextlib for micropython

pickle @ micropython-lib
<No description>

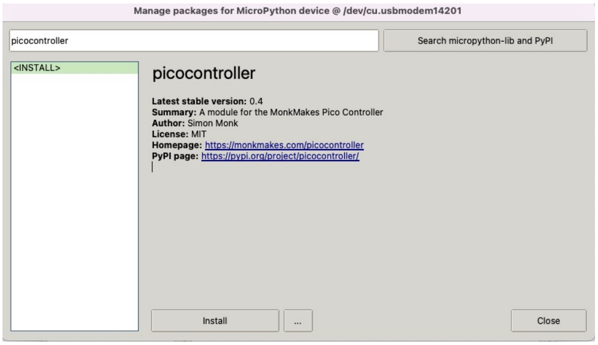
ucontextlib @ micropython-lib
Minimal subset of contextlib for MicroPython low-memory ports

aioble @ micropython-lib
<No description>

aioble-core @ micropython-lib
<No description>

aioble-server @ micropython-lib

Select the result *picocontroller* (for the MonkMakes Pico Controller)

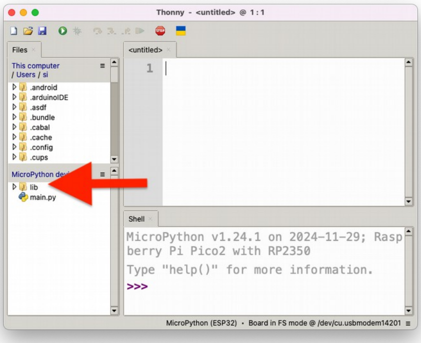

Manage packages for MicroPython device @ /dev/cu.usbmodem14201

picocontroller

picocontroller

Latest stable version: 0.4
Summary: A module for the MonkMakes Pico Controller
Author: Simon Monk
License: MIT
Homepage: https://monkmakes.com/picocontroller
PyPI page: https://pypi.org/project/picocontroller/

Click on *Install*. Don't worry if there is a later version that 0.4. Just install the latest version. Now, when the installation has finished, when you return to the main Thonny window, you should see that there is a *lib* folder in the files area.



```
MicroPython v1.24.1 on 2024-11-29; Raspberry Pi Pico2 with RP2350
Type "help()" for more information.
>>>
```

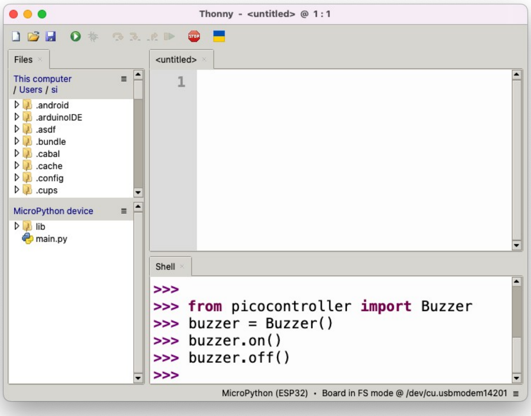
The package installation should now be complete and we are ready to experiment with the PicoController board.

The Python Shell

We can now try out the PicoController from the Python Shell area at the bottom of the Thonny window.

Type in these commands:

```
>>> from picocontroller import Buzzer
>>> buzzer = Buzzer()
>>> buzzer.on()
>>> buzzer.off()
```



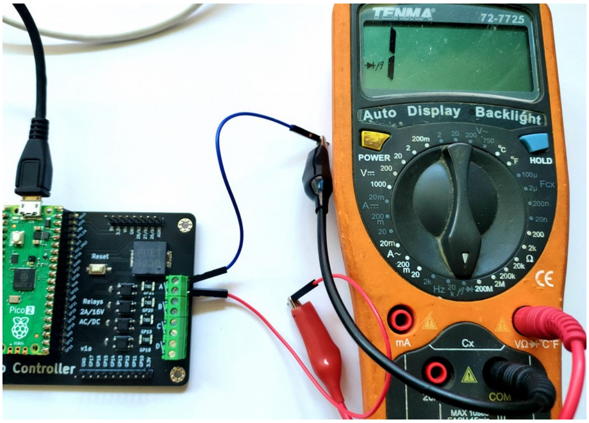
```
>>>
>>> from picocontroller import Buzzer
>>> buzzer = Buzzer()
>>> buzzer.on()
>>> buzzer.off()
>>>
```

MicroPython (ESP32) • Board in FS mode @ /dev/cu.usbmodem14201

When you enter `buzzer.on()` the buzzer will start buzzing, and to turn the buzzer off, type `buzzer.off()`.

Now, let's test out one of the relays. To find out more about relays, see Page XX.

To use the relay we will need something to tell us that the relay has activated. We can use a digital multimeter (DMM) set to its continuity (or buzzer) mode, as shown below.



Notice how I have used short jumper wires and alligator clip leads to connect the relay screw terminals to the DMM. If you DMM has thin probe leads, you may be able to fit them directly into the screw terminal holes.

Now try running the following commands to turn *Relay A* on and off. When the relay is on, the DMM should buzz.

```
>>> from picocontroller import Relay_A
>>> Relay_A.on()
>>> Relay_A.off()
```

Let's try using the OLED display. The *picocontroller.gui* module has a number of useful different ways of displaying things, including: a retro 7-segment simulation, a menu system and a console for displaying messages. It is the latter, that we will use here. You can learn more about the picocontroller.gui module on Page 10.

Run the following commands, and you will see a message appear on the OLED display.

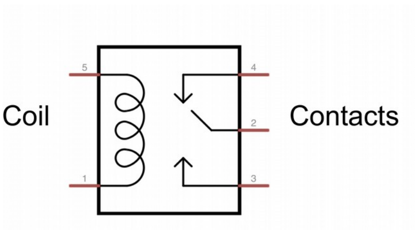
```
from picocontroller.gui import OLEDConsole
>>> console = OLEDConsole()
>>> console.print('Hello World')
```

In the next section we will learn a bit more about using the relays.

Using the Solid State Relays

Warning: Low Voltage ONLY

A traditional relay is an electromechanical device that uses an electromagnet to actually move a pair of contacts.

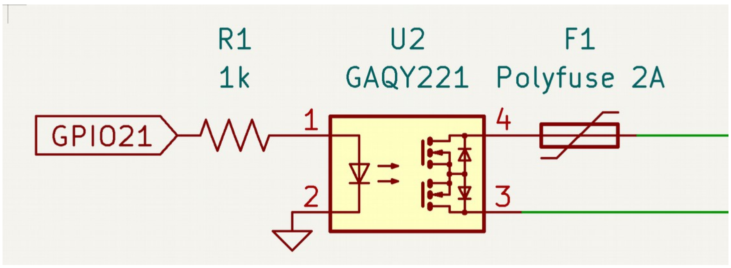


When a current of a few tens of mA (milliamps) flows through the coil, it pulls the contacts together making an electrical connection, as if you had flipped a switch.

The important thing here is that there is NO electrical connection between the coil and the contacts. The contacts are free to operate at a different voltage and switch a completely different circuit to whatever the coil is controlled by (typically a microcontroller like the Pico using some extra circuitry).

Electro-mechanical relays are cheap, but switch slowly and require quite a high current through the coil, to activate them. The Pico Controller uses a type of relay called a solid state relay (SSR). Instead of a coil of wire and contacts, the SSRs used in the Pico Controller use an LED and phototransistors all contained within a small package. The LED only needs about 1mA to switch the phototransistors on, and is still electrically isolated from the 'contacts'.

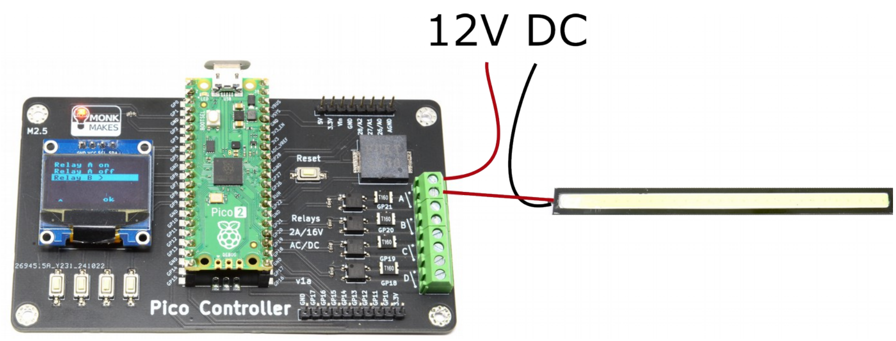
Here is a section of the Pico Controller's schematic diagram showing how the SSR is used, connected directly to one of the Pico's GPIO pins.



The SSR behaves just like an electromechanical relay, except that it switches much faster, and does not have any moving parts in it. The SSR is protected against over-current by a separate self-resetting polyfuse, that cuts of the current if it exceeds 2A for more than a few seconds.

Wiring a Relay Output

The output of each relay goes to a 2-way screw terminal, and can be treated just like a switch that happens to be controlled by the Pico Controller. Here's an example of how you might want to wire up some 12V strip-lighting to be switched by the Pico Controller.



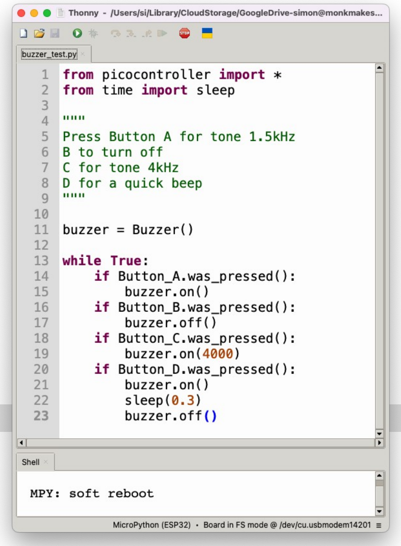
Note that the red connection indicates the positive 12V supply. The Pico Controller is using the USB connection for its own power and is completely separate from the 12V supply for the Lamp.

You will find an example using the *picocontroller.gui* package's menu system to control a relay in Example 2 on Page 13.

Example 1. Buzzer Test

The Pico Controller comes with a selection of example programs designed to illustrate the use of the MicroPython modules and the Pico Controller and to serve as a basis for your own projects. The first if these is called Buzzer Test and can be found in the code download from Github in the examples folder. The file is called *buzzer_test.py*.

Load it up into Thonny and run it.


Thonny - /Users/si/Library/CloudStorage/GoogleDrive-simon@monkmakes...
buzzer_test.py
Shell

```python
from picocontroller import *
from time import sleep

"""
Press Button A for tone 1.5kHz
B to turn off
C for tone 4kHz
D for a quick beep
"""

buzzer = Buzzer()

while True:
    if Button_A.was_pressed():
        buzzer.on()
    if Button_B.was_pressed():
        buzzer.off()
    if Button_C.was_pressed():
        buzzer.on(4000)
    if Button_D.was_pressed():
        buzzer.on()
        sleep(0.3)
        buzzer.off()
```

```
MPY: soft reboot
```

MicroPython (ESP32) • Board in FS mode @ /dev/cu.usbmodem14201

As you will see from the comment string at the top of the code, if you press Button A (the left-most) the buzzer will sound at it's default frequency of 1.5 kHz.

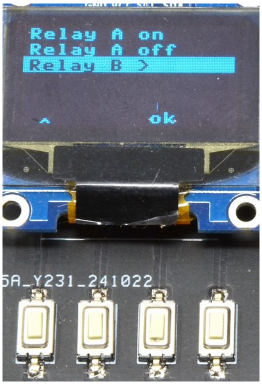
Pressing button B (next right) the buzzing will stop.

You can also change the frequency of the buzzer by supplying a parameter of the frequency in Hz in the the on method, like this:

```
buzzer.on(4000)
```

Pressing Button D shows the technique to use if you just want a short beep. That is, turn the buzzer on, delay for (in this case) 0.3 of a second, and then turn the buzzing off again.

Example 2. Relay Menu



This example (which you will find in the download as *examples/relay_menu.py*) illustrates the *picocontroller.gui* package's menu system to control the relays.

The menu system uses the display and the four push buttons (A to D – left to right). The push buttons acting as navigation buttons to move up and down the menu items or to select one.

So, here on the right, above button A on the display is a ^ symbol indicating that if button A is pressed, at this point, the next item up in the menu (*Relay A off*) will become selected.

The text *ok* on the display above button D, indicates that current selection will be actioned if button D is pressed. In this case, that would open a sub-menu for Relay B.

Here is the code for the project.

```
from picocontroller import *
from picocontroller.gui import *
from time import sleep

relay_b_submenu_data =  [
    {'id' : 'relay_B_on', 'label' : 'On'},
    {'id' : 'relay_B_off', 'label' : 'Off'},
    {'id' : 'back', 'label' : 'Back'}
    ]

menu_data = [
    {'id' : 'relay_A_on', 'label' : 'Relay A on'},
    {'id' : 'relay_A_off', 'label' : 'Relay A off'},
    {'id' : 'relay_B_submenu', 'label' : 'Relay B >'}
    ]

main_menu = Menu(menu_data)
relay_b_submenu = Menu(relay_b_submenu_data)

menu = main_menu
menu.draw_menu()

while True:
    selection = menu.check_keys()
    if selection:
        print(selection)
        if selection == 'relay_B_submenu':
            menu = relay_b_submenu
            menu.draw_menu()
        if selection == 'back':
            menu = main_menu
            menu.draw_menu()
        if selection == 'relay_A_on':
            Relay_A.on()
        if selection == 'relay_A_off':
            Relay_A.off()
        if selection == 'relay_B_on':
            Relay_B.on()
        if selection == 'relay_B_off':
            Relay_B.off()
    sleep(0.1)
```

The menu and submenu are both defined as Python Lists of dictionaries, each dictionary being one menu option, for example:

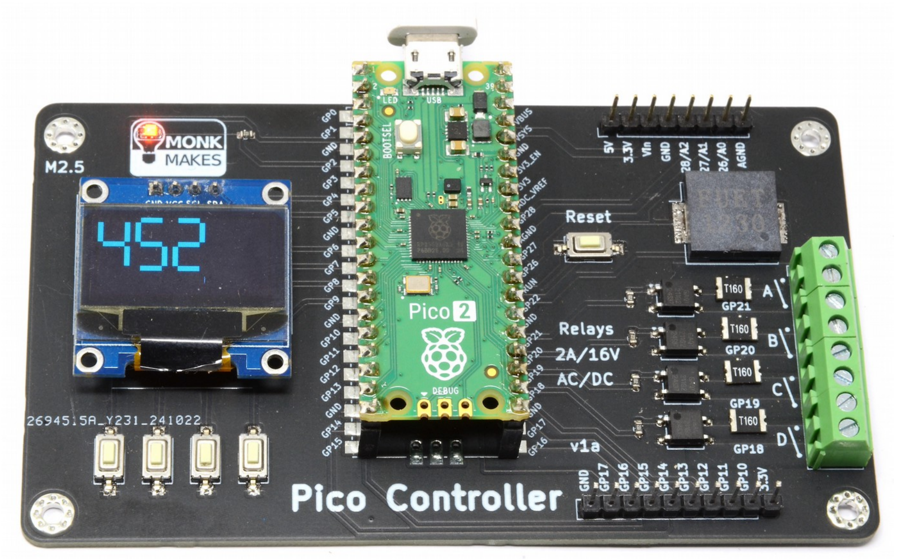
```
{'id' : 'relay_B_on', 'label' : 'On'},
```

The menu has an `id` that is used to identify the menu option and a `label` which is the text actually displayed.

Handling of changes to the menu selections takes place in the main `while` loop. This loop should start my checking whether any selection has been actioned by using `menu.check_keys()`. If it has, then the selection (which will be a menu item id) is compared and the appropriate Python code run. If this action is to switch to a sub-menu, then the variable `menu` (the current menu) is assigned the value of the submenu and the display refreshed to show the new menu using `menu = relay_b_submenu` and `menu.draw_menu()` respectively.

Example 3. Egg Timer

This example illustrates the use of the *picocontroller.gui* package's `SevenSegDisplay` class.



The display shows a time in minutes and seconds which can be increased or decreased by pressing buttons B and A respectively. When button C is pressed the countdown starts and when 0 is reached, the buzzer sounds. Button D resets the timer.

Here is the code:

```
from picocontroller import *
from picocontroller.gui import *
from time import sleep
from machine import Timer

buzzer = Buzzer()
t = Timer()

time_display = SevenSegDisplay(0, 0, digit_w=20, digit_h=40,
num_digits=3)

default_mins = 5
default_secs = 0

mins = default_mins
secs = default_secs
is_running = False
```

```
t.init(mode=Timer.PERIODIC, period=1000, callback=lambda
t:update_time())

def update_time():
    global mins, secs
    if not is_running:
        return
    if secs == 0:
        if mins == 0:
            stop_running()
            buzzer.on()
        else:
            secs = 59
            mins -= 1
    else:
        secs -= 1
    update_display()

def start_running():
    global is_running
    is_running = True

def stop_running():
    global is_running
    is_running = False

def update_display():
    time_display.draw(mins * 100 + secs)

update_display()
while True:
    if Button_A.was_pressed() and not is_running:
        buzzer.off()
        if mins < 9:
            mins += 1
            update_display()
    if Button_B.was_pressed():
        buzzer.off()
        if mins > 1:
            mins -= 1
            update_display()
    if Button_C.was_pressed():
        buzzer.off()
        if is_running:
            stop_running()
        else:
            start_running()
    if Button_D.was_pressed():
        buzzer.off()
        mins = default_mins
        secs = default_secs
        stop_running()
        update_display()
```

There is quite a lot of code here, so you might like to follow along in Thonny.

The 3 digit, 7-segment display is defined by the line:

```
time_display = SevenSegDisplay(0, 0, digit_w=20, digit_h=40, num_digits=3)
```

The first two parameters are the x and y coordinates of the top left of the 7-segemnt display. The parameters `digit_w` and `digit_h` set the width and height of each digit in pixels, and `num_digits` the number of digits to be displayed.

This example uses the MicroPython `Timer` class. You can find out more about this here: https://docs.micropython.org/en/latest/library/machine.Timer

The `update_time` function is called every second by the timer, updating the number of seconds and minutes remaining and calling `update_display` to refresh the time shown on the display.

To set the value to be displayed on the 3 digit 7-segment display, the draw method supplies a 3 digit number made up of the number of minutes times 100 plus the number of seconds.

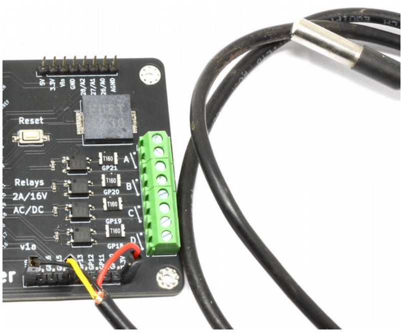
```
time_display.draw(mins * 100 + secs)
```

Example 4. Thermostat

To try out this example, you will need to buy and attach one of the popular DS18B20 temperature sensors and connect it to the Pico Controller's GPIO pins.

This project illustrates the use of the display's *ssd1306* module to display text and graphics on the OLED display, as well as using the readings for the sensor.

The project acts as a simple thermostat, assuming that relay A controls the power to a heater.



This DS18B20 sensor, normally has bare copper leads on the end of its cable. One way to attach these to header pins is to crimp header sockets onto these leads. Alternatively, you can cut female to female jumper leads in half and solder the half-leads onto the leads from the sensor, to achieve the same ends.

The DS18B20 needs to be wired up as follows:

Pico Pin	DS18B80 Pin	Lead color
GND	GND	Black
3.3V	VCC	Red
GPIO 14	Data Out	Yellow

DS18B20 sensors, are often supplied with a resistor to use as a pull-up resistor. Generally, if you are only using one sensor, and you don't have a long lead to it, then you do not need to use this. The code in this example uses the Pico's GPIO pin's built-in pull-up resistor.

Here is the code for this example:

```
from machine import Pin
from picocontroller import *
from time import sleep

import onewire
import ds18x20

ONE_WIRE_PIN = 14

ow = onewire.OneWire(Pin(ONE_WIRE_PIN, pull=Pin.PULL_UP))
ds = ds18x20.DS18X20(ow)
```

```
relay = Relay_A

MIN_T = 0
MAX_T = 40

thermometer = None
t_set = 20.0
t = 0.0

def refresh_display(output_on):
    text_color = 1
    background_color = 0
    if output_on:
        text_color = 0
        background_color = 1
    display.fill(background_color)
    display.text('Measured', 5, 20, text_color)
    display.text(str(t), 90, 20, text_color)
    display.text('Set', 5, 40, text_color)
    display.text(str(t_set), 90, 40, text_color)
    display.show()

def connect_thermometer():
    global thermometer
    try:
        devices = ds.scan()
        thermometer = devices[0]
    except:
        print("Couldn't detect thermometer - check wiring")

def read_temp():
    global t
    ds.convert_temp()
    t = ds.read_temp(thermometer)

connect_thermometer()
refresh_display(False)
while True:
    read_temp()
    if t < t_set:
        # power on if measured temp too low
        relay.on()
        refresh_display(True)
    else:
        relay.off()
        refresh_display(False)
    if Button_A.was_pressed() and t_set > MIN_T:
        t_set -= 1
        refresh_display(False)
    if Button_B.was_pressed() and t_set < MAX_T:
        t_set += 1
        refresh_display(False)
```

The DS18B20 uses the *Onewire* bus interface, which, along with the protocol of the sensor itself are provided with interfaces with the `onewire` and `ds18x20` modules included with MicroPython. So, no special installation of modules is needed.

The Onewire interface is initialised using the line:

```
ow = onewire.OneWire(Pin(ONE_WIRE_PIN, pull=Pin.PULL_UP))
```

This takes a `Pin` as it's parameter, and its here that the pin is defined with it's internal pull-up resistor enabled so that the Onewire bus will work.

The variable `t_set` should contain the desired temperature in degrees C and `t` will contain the measured temperature.

The function `refresh_display` uses the SSD1306 library to display both the set and measured temperatures, inverting the text into black on white if the `output_on` variable indicates that the heater controlled by relay A is on.

The picocontroller Module

The picocontroller module provides a convenient way of interacting with the physical hardware of the Pico Controller. It provides some variables as well as two classes, `Button` and `Relay`.

Variables

`Button_A`, `Button_B`, `Button_C` and `Button_D`	Instances of the class `Button` for each of the four navigation buttons.
`Relay_A`, `Relay_B`, `Relay_C` and `Relay_D`,	Instances of the class Relay for each of the four relays.
`display`	The instance of the `SSD1306` class created when the picocontroller module is imported.
`i2c`	A link to the I2C bus object
`W`	The screen width in pixels (128)
`H`	The screen height in pixels (64)

Button Class

Method	**Example**	
`is_pressed()`	`Button_A.is_pressed()`	Returns true if button A is currently pressed.
`was_pressed()`	`Button_A.was_pressed()`	Returns true when button A is released after being pressed.

Relay Class

Method	**Example**	
`on()`	`Relay_A.on()`	Turn the relay on
`Off()`	`Relay_A.off()`	Turn the relay off
`value(new_state)`	`Relay_A.value(1)`	1 for on, 0 for off
`on_for(duration)`	`Relay_A.on_for(10000)`	Turn the relay on for the number of

		milliseconds in the parameter. (Non-blocking)
`off_for(duration)`	`Relay_A.off_for(10000)`	Turn the relay off for the number of milliseconds in the parameter. (Non-blocking)
`oscillate(period)`	`Relay_A.oscillate(1000)`	Turn the relay on and off with a delay of the number of milliseconds in the parameter. (Non-blocking)

`cancel_timer(value= on/off)`	`Relay_A.cancel_timer()`
	`Relay_A.cancel_timer(1)`
		Cancel any repeating timer, optionally setting the Relay to on or off.

Buzzer Class

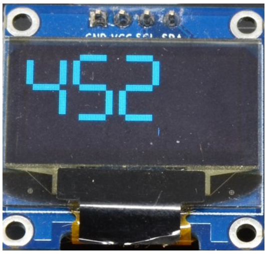
Method	**Examples**	

`on(f=frequency)`	`buzzer = Buzzer()`
	`buzzer.on()`
	`buzzor.on(3000)`
		Start the buzzer at a default frequency or a specified frequency in Hz

`off()`	`buzzer = Buzzer()`
	`buzzer.off()`
		Silence the buzzer

THE PICOCONTROLLER.GUI MODULE

The `gui` module within the `picocontroller` module contains a number of classes, for building user interfaces using the OLED display and four navigation buttons.



SevenSegDisplay Class

This class allows you to simulate 7-segment displays on the display. The position, size and number of digits of the display can all be set.

Method	**Example**	
`SevenSegDisplay(x, y, digit_w=20, digit_h=40, num_digits=4)`	`d1 = SevenSegDisplay(0, 0, digit_w=10, digit_h=20, num_digits=2)`	Constructor, defining x,y or origin, digit width and height in pixels and number of digits.
`draw(value)`	`d1 = SevenSegDisplay(0, 0, digit_w=10, digit_h=20, num_digits=2)` `d1.value(99)`	Display 99 on a 2 digit display.

OLEDConsole Class

The `OLEDConsole` class provides a way of sending output text to the display. Once the display is full, it will scroll up automatically.

The example *oled_console_example.py* illustrates the use of this class.

Method	Examples	
`OLEDConsole(line_spacing=10)`	`log = OLEDConsole()` `log = OLEDConsole(line_spacing=12)`	Create a new instance, optionally setting the line spacing in pixels.
`print()`	`log.print('Counting ' + str(x))`	Print a string to the console.
`clear()`	`log.clear()`	Clear the display.

Menu Class

The `Menu` class is pretty well illustrated by the *relay_menu.py* example. It provides a data-driven way of expressing menus and sub-menus using the OLED display.

All the examples in the table below assume the menu_data below:

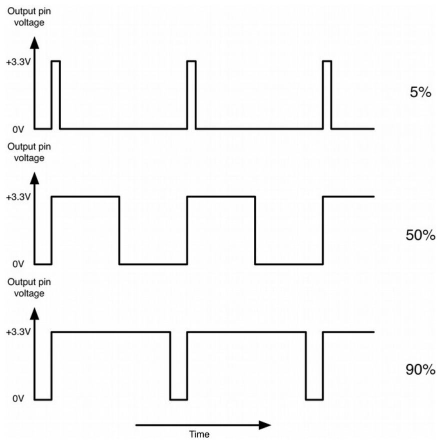
```
menu_data = [
    {'id' : 'relay_A_on', 'label' : 'Relay A on'},
    {'id' : 'relay_A_off', 'label' : 'Relay A off'},
    {'id' : 'relay_B_submenu', 'label' : 'Relay B >'}
    ]
```

Method	**Example**	
`Menu(menu_data, menu_sep=10)`	`main_menu = Menu(menu_data)`	Construct an instance of Menu, optionally with a line spacing.
`draw_menu()`	`main_menu.draw_menu()`	Render the menu on the display, with the current selection highlighted.
`check_keys()`	`main_menu.check_keys()`	Check if buttons A B or C have been pressed. Pressing A move the menu up one place, B – down a place and C actions the selection returning the id of the current selelection.

Pulse Width Modulation

The SSRs used in the PicoController are capable of more than just on/off control. They can also be used say to control the brightness of an LED or the speed of a motor using a process called Pulse Width Modulation or PWM.

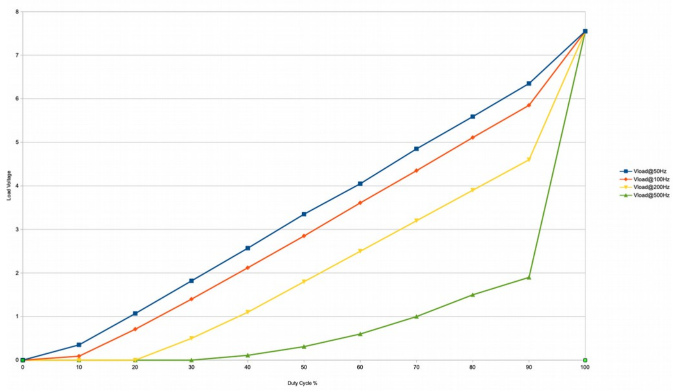
A PWM output supplies a train of pulses to the control pin. The duration of these pulses is varied to vary the apparent brightness of a lamp or the speed of a motor. So a short pulse will make the lamp appear dim or a motor slow, but if the pulse is long, so that it is 3.3V for most of the cycle, then the lamp will appear bright or the motor rotate quickly.



Although the SSRs used in the Pico Controller are much faster than electromechanical relays, they are still relatively slow in electronics terms.

The graph below shows how the response of the relay (at full load of 1.5A) to PWM at various frequencies.

The ideal SSR would have a completely straight line, and at lower currents than the test current here, you get close to that ideal. But here, you can see that you get quite acceptable results up to a frequency of about 100Hz.



MicroPython has built-in support for PWM. You will find an example program for it in *examples/pwm_test.py*.

```
from machine import PWM
from picocontroller import Relay_A

pwm_Relay_A = PWM(Relay_A._pin)

while True:
    f = int(input('f (Hz):'))
    d = int(input('duty (0..65535):'))
    pwm_Relay_A.freq(f)
    pwm_Relay_A.duty_u16(d)
```

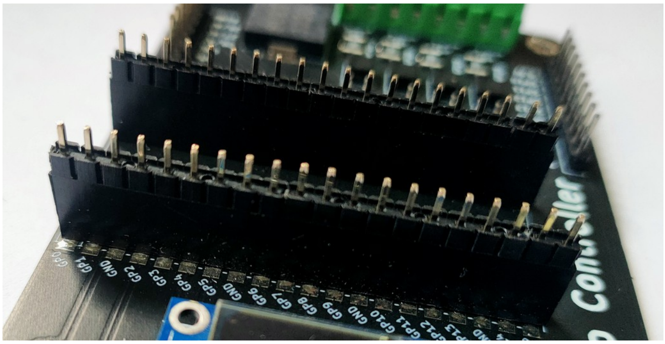
You will need to attach a load to your Pico Controller, such as the LED lamp of Page 11.

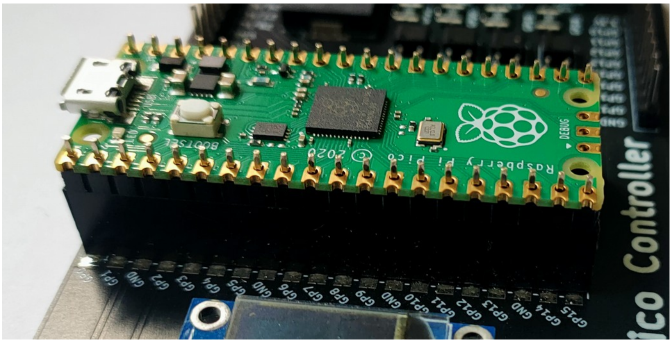
When you run the program, you will be prompted for first for the PWM frequency (I suggest using 100. Then for the duty. Try the values: 0 (off) 32767 (half brightness) and 65535 (full brightness).

Appendix A. Soldering Header Pins

If your Pico does not have header pins attached, you will have to solder them on. To do this, you will need a soldering iron and some solder wire. Don't worry if you are new to soldering its not so hard.

To keep the pins straight, it is a good idea to push them into the Pico Controller long ends first. This has the added advantage that once soldered, the Pico will be in the right place, and won't need to be relocated.





If its your first attempt at soldering, watch a few tutorials before you start to get an idea of what you are trying to do. Here's a good one:

https://www.youtube.com/watch?v=37mW1i_oEpA

Appendix B. Troubleshooting

Problem: The orange power LED in the MonkMakes logo on the Pico Controller does not light.

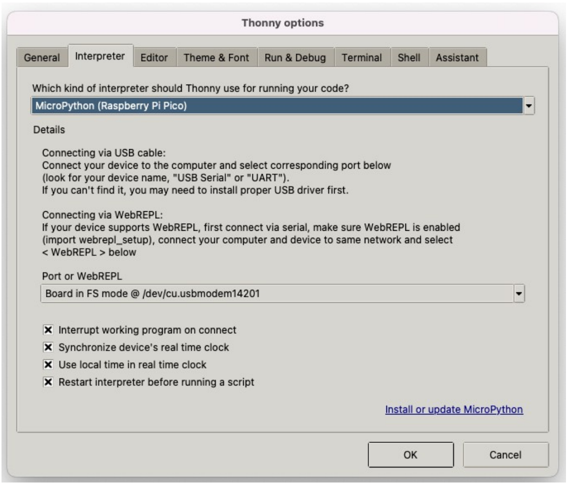
Solution: Check the USB connection, and try a different USB port on your computer.

Problem: I can't upload anything onto my Pico, even through the Pico Controller has power indicated by the orange LED being lit.

Solution: There are a number of possible reasons that this might happen.

1. The program running on the Pico is still running, this must be halted before running a new one. Try clicking the *Stop* button repeatedly and in the Shell area typing CTRL-c

2. Thonny may not be configured to work with your Pico board. Open the Thonny *Preferences* panel, select the *Interpreter* tab and make sure that *MicroPython (Raspberry Pi Pico)* is selected and the correct port is selected in the drop-down *Port or WebREPL*.



Which kind of interpreter should Thonny use for running your code?

MicroPython (Raspberry Pi Pico)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL

Board in FS mode @ /dev/cu.usbmodem14201

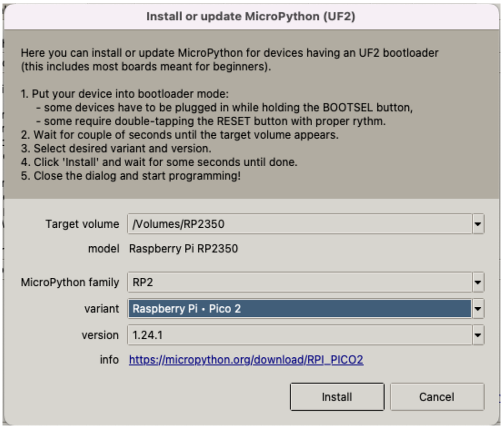
- [x] Interrupt working program on connect
- [x] Synchronize device's real time clock
- [x] Use local time in real time clock
- [x] Restart interpreter before running a script

3. MicroPython is not installed on your Pico.

It may be that your Pico has been used with a different programming environment such as Arduino and does not have the MicroPython firmware. To reinstall MicroPython, put your Pico into *Upload* mode by holding down the *Boot* switch while you plug it in and then on the *Preferences* panel click on *Install or update*

MicroPython.

Set the options as shown below, in the variant drop-down, choose your particular type of Pico and then click *Install*.



Install or update MicroPython (UF2)

Here you can install or update MicroPython for devices having an UF2 bootloader
(this includes most boards meant for beginners).

1. Put your device into bootloader mode:
   - some devices have to be plugged in while holding the BOOTSEL button,
   - some require double-tapping the RESET button with proper rythm.
2. Wait for couple of seconds until the target volume appears.
3. Select desired variant and version.
4. Click 'Install' and wait for some seconds until done.
5. Close the dialog and start programming!

Target volume: /Volumes/RP2350
model: Raspberry Pi RP2350
MicroPython family: RP2
variant: Raspberry Pi • Pico 2
version: 1.24.1
info: https://micropython.org/download/RPI_PICO2

Problem: I can't tell if the Relay is working.

Solution: Attach a DMM in continuity mode as described on Page 9.

Support

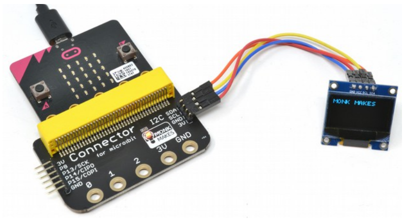
You can find the Product's information page here: https://monkmakes.com/picocontroller including a datasheet for the product.

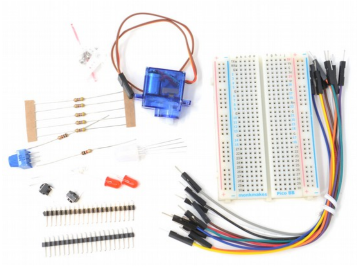
If you need further support, please email support@monkmakes.com.

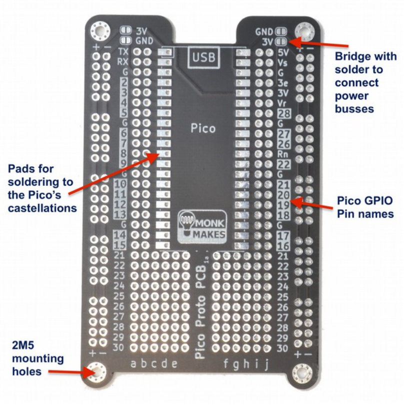
MonkMakes

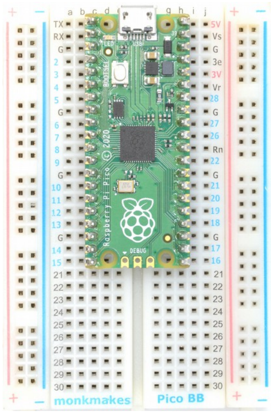
As well as this kit, MonkMakes makes all sorts of kits and gadgets to help with your electronics projects. Find out more, as well as where to buy here: https://monkmakes.com you can also follow MonkMakes on X as @monkmakes and on Instagram as @monk_makes_ltd

A selection of products: Connector for micro:bit, Electronics Kit 1 for Pico, Pico Proto PCB and Breadboard for Pico.






Bridge with solder to connect power busses
Pads for soldering to the Pico's castellations
Pico GPIO Pin names
2M5 mounting holes



Book by Simon Monk

If you like this product and want to learn more about programming in MicroPython, then you might like Simon Monk's book on the subject.