Using the RP2040 Zero with Thonny IDE

By: Richard R. Vasquez

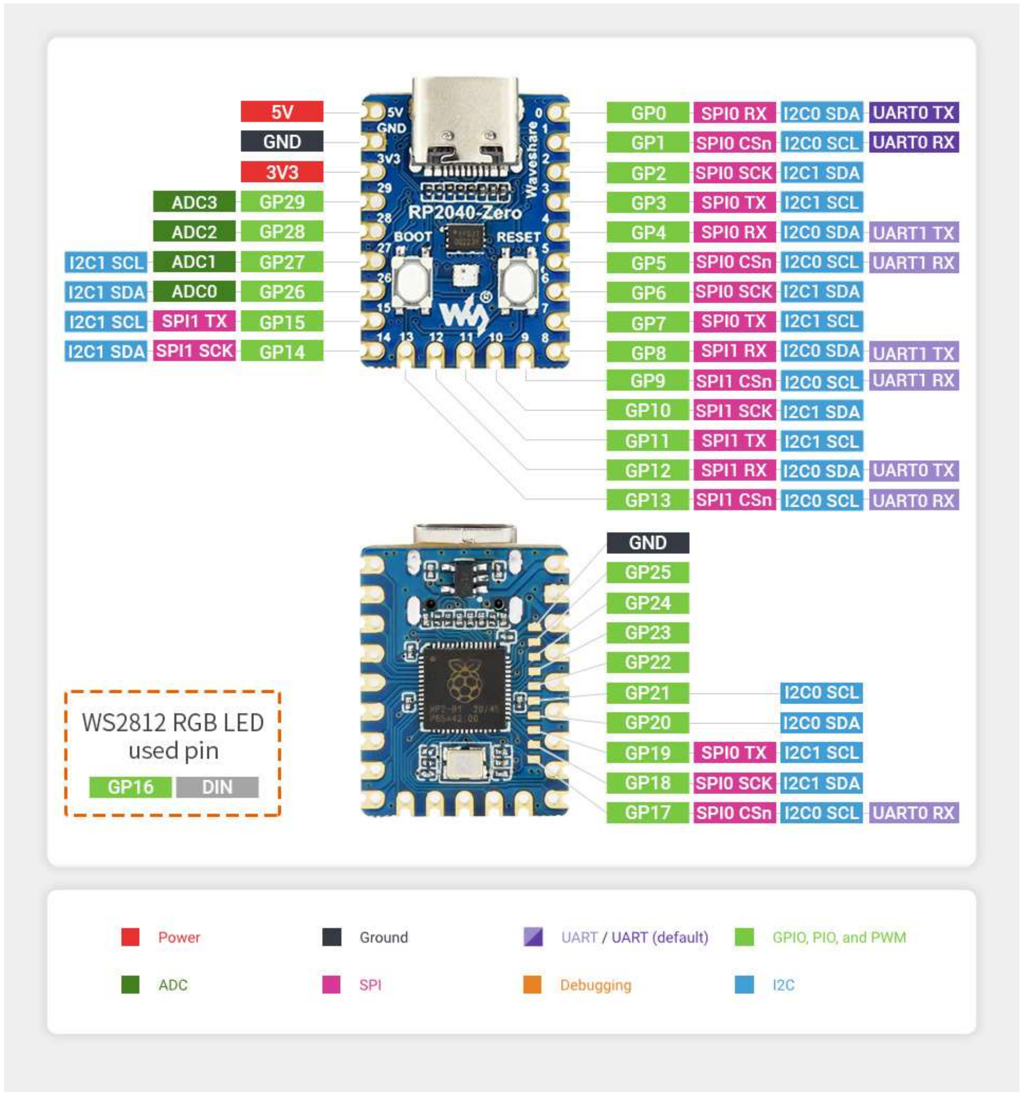


Table of Contents

Scope

This document is a help file for new users to the Waveshare RP2040 Zero; a small, low-cost computer that is capable of running a version of Python called MicroPython. It's a very powerful and easy to use tool.

In this document I use the RP2040 Zero to generate telemetry (albeit false) data, simulating the telemetry output of an F120 ESC (Electronic Speed controller), made by Advanced Power Devices. The simulated data is from an actual F120 ESC.

Prerequisites

This document presumes you already have Micropython running. If not, follow the instructions from this website: https://micropython.org/download/?mcu=rp2040

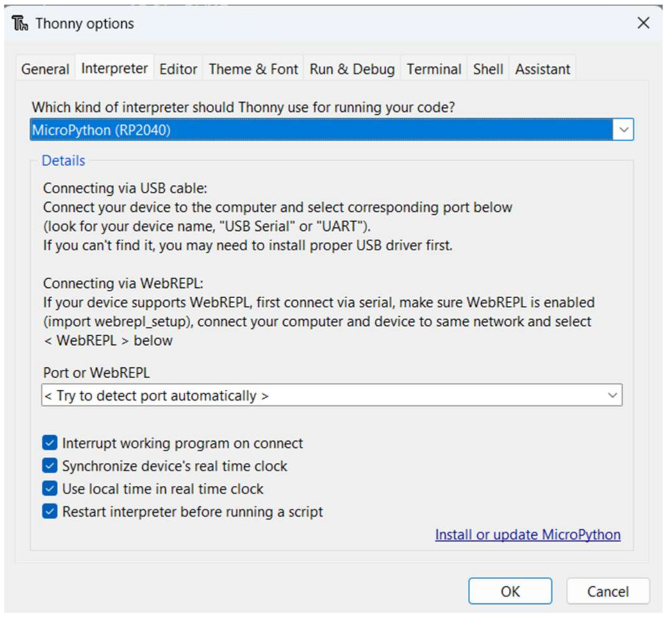
This is the RP2040 Zero: https://www.waveshare.com/w/upload/2/2b/RP2040-Zero-details-7.jpg

Micropython

Help video: Used this video to help get started: **How to Setup a Raspberry Pi Pico and Code with Thonny.** URL: https://youtu.be/_ouzuI_ZPLs?si=EUGX6oWAQq1oDnLq

Here are the steps outlined in this video, customized for the ***RP2040 Zero***:

1) Run Thonny IDE.
2) Hold Boot button and connect to USB port.
3) Press 'Stop' button.
4) In Thonny, select Tools->Options->Interpreter
    a. 'Which kind of interpreter....?': Select MicroPython (RP2040)
5) Click 'Install or update MicroPython' (see image below).



Thonny options

Which kind of interpreter should Thonny use for running your code?
MicroPython (RP2040)

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.

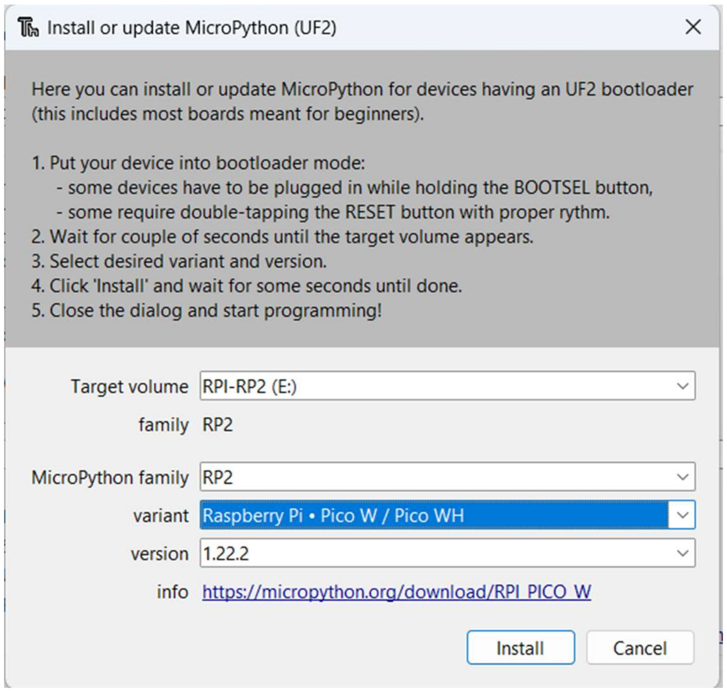
Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL
< Try to detect port automatically >

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

Install or update MicroPython

6) Wait a moment for RP2040 device to be detected (see image below).
7) See figure below. Select:
   a. Micropython Family: RP2
   b. Variant: Raspberry Pi * Pico W/Pico WH
8) Click Install and wait for process to complete.



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 RPI-RP2 (E:)
family RP2

MicroPython family RP2
variant Raspberry Pi • Pico W / Pico WH
version 1.22.2
info https://micropython.org/download/RPI_PICO_W

Deploy As Standalone

When your code is ready to deploy as a standalone application without Thonny IDE

1) From Thonny IDE, save developed code at ‘main.py’ (must be lower case) on the RP2040 device.
2) Remove the USB cable.
3) Apply power to the RP2040:
    a. Connect to a USB port as a power source.
    b. Or alternatively apply 5V to the 5V terminals on the board.

Example Code

Note: Save as ‘main.py’ on RP2040 to run as a standalone application. See Deploy As Standalone section of this document.

Example #1

In this example we use pin GP12 as UART0 to send a packet of data, simulating the output of an F120 or F200 ESC. This example also blinks an LED so that the user can see when to expect that data packet once per second.

```
# Filename: TelemetryGenerator.py
# Uses UART0 to simulate and F120 ESC's telemetry output
# 115200 bps, 8 bit, 1 stop, no parity, no flow control
# Data format: 12 bytes

import machine
import utime

# Define UART0 pins
uart0_tx = machine.Pin(12)  # Pin 12 for TX0
uart0_rx = machine.Pin(13)  # Pin 13 for RX0
# Define the GPIO pin connected to the single LED
led_pin = machine.Pin(0, machine.Pin.OUT)  # RP2040 Pico

# Configure UART0 with baud rate 115200, 1 stop bit, no parity
uart0 = machine.UART(0, baudrate=115200, tx=uart0_tx, rx=uart0_rx, bits=8, parity=None, stop=1)
```

```
# Define your packet of 12 bytes

packet = bytes([0x1d, 0x0e, 0x1b, 0x00, 0x04, 0x00, 0x01, 0x00, 0x00, 0x97, 0xff, 0xff])


while True:

   # Turn on the LED

   led_pin.value(1)

   # Write out the packet

   uart0.write(packet)

   # Wait for 0.25 second

   utime.sleep(0.05)

   # Turn off the LED

   led_pin.value(0)

   # Wait for 0.25 second

   utime.sleep(0.95)
```

Example #2

PWM Generator for ESC

This example is used to test an ESC with a motor, for benchtop experiments.

```
# Filename: PWMThrottle.py
# Based upon: ADCtoPWM5.py
# Converts analog voltage from throttle to a PWM signal: 1mS to 2 mS (0 to full speed), 50 Hz.
import machine
import utime

# Configure ADC0 on GPIO pin GP26 (Channel 26)
adc = machine.ADC(machine.Pin('GP26'))

# Configure PWM0 on GPIO pin GP0 (PWM0A)
pwm = machine.PWM(machine.Pin('GP0'))

# Set PWM frequency to 50 Hz (20 ms period)
pwm.freq(50)
sensor_temp = machine.ADC(26)

while True:
    # Read analog value from ADC0
    analog_value = sensor_temp.read_u16()

    # Map the analog value to the PWM duty cycle (0-65535)
    # Assuming the analog range is 0-3.3V (for 3.3V ADC reference)
    # I've learned that the full scale ranges from 5% to 10% of a 50 Hz pulse

    # 10% of 65535 = 6553.5 approximately 6554

    # 5% of 3277 (approximately)

    # Observation of throttle: Full speed = 3.84 volts, Min speed = 0.76 volts

    # Used 0.75 scale down, since 3.8 > 3.3 range of ADC reference voltage.

    # Scaled down 0.75x Throttle: Full speed = 2.88 volts, Min speed = 0.566 volts

    # ADC: 50.355 uV/step

    # Delta steps: 57195-11250 = 45945, Delta PWM = 6554-3277 = 3277

    # Slope: m = 3227/45945 = 0.070237 = 1/14.2377

    # Offet: b = y - m*x = 3227 - (11250/14.2377) = 3277 - 790.156 = 2436.84

    duty_cycle = int(((analog_value)/14.2377) + 2436.84)


    # Check if duty_cycle is below 10% and limit it to 10%

    if duty_cycle < 3277:

       duty_cycle = 3277


    # Check if duty_cycle is above 90% and limit it to 90%

    if duty_cycle > 6554:

       duty_cycle = 6554


    #print(duty_cycle)


    # Set PWM duty cycle

    pwm.duty_u16(duty_cycle)


    # Wait for a short time to update the PWM output (adjust as needed)
```

```
utime.sleep_ms(100)
```