CircuitPython on Raspberry Pi (Bare Metal / No OS)

Created by Scott Shawcroft

https://learn.adafruit.com/circuitpython-on-raspberry-pi-bare-metal-no-os

Last updated on 2023-08-29 04:45:42 PM EDT
Table of Contents

Overview  
- Status  
- Parts

Installation  
- Raspberry Pi Imager  
- Upgrading

Connecting  
- Pi Zero and Zero W  
- Pi Zero 2 W  
- Pi 4 Model B  
- CM4 IO

Using CircuitPython  
- New to CircuitPython  
- Experienced CircuitPythonistas  
- HDMI

Debug  
- Print Debugging  
- GDB
Overview

The Raspberry Pi single board computers (SBCs for short) are normally used with the Raspberry Pi OS. This is great for gaming and web browsing, but isn’t designed for microcontroller-like tasks. Tasks like reading I2C sensors and SPI devices can be trickier. Furthermore, Linux system maintenance can be tricky for beginners.

Instead of Raspberry Pi OS, one can now use CircuitPython instead. CircuitPython running without another operating system (OS) is typically known as "bare metal". This greatly simplifies system maintenance and makes it easier to treat the Raspberry Pi like a microcontroller. In addition, running CircuitPython on the Raspberry Pi increases the amount of flash and RAM available to CircuitPython programs. Finally, it also adds the ability to use displayio on HDMI displays.

Folks may be running CircuitPython programs from within Raspberry Pi OS currently through Blinka. This is different from CircuitPython standalone because you edit the files using the computer itself and are limited by the underlying OS. See the Blinka guide() for details on how to use a CircuitPython program from within Raspberry Pi OS.

Status

CircuitPython support for the Raspberry Pi SBCs was added in CircuitPython 7.1.0 and is done under the `broadcom` port. The list of known issues is available on the CircuitPython GitHub project under the "broadcom" label().
Parts

Select from the supported boards. Then look for accessories specific to that board (power supply, video cable, etc.). If the Compute Module 4 is used, a suitable carrier board like the official IO Board will be needed.

**Raspberry Pi Compute Module 4 - 4GB RAM, 32GB MMC and WiFi**
The Raspberry Pi Compute Module 4 is based on the Raspberry Pi 4 Model B, but in a smaller form factor - perfect for...
https://www.adafruit.com/product/4982

**Raspberry Pi Compute Module 4 IO Board**
Exposing every interface from Raspberry Pi Compute Module 4, the Compute Module 4 IO Board provides a development platform and reference base-board design for the most...
https://www.adafruit.com/product/4787

**Raspberry Pi 4 Model B - 2 GB RAM**
The Raspberry Pi 4 Model B is the newest Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing
https://www.adafruit.com/product/4292
Raspberry Pi Zero W
If you didn't think that the Raspberry Pi Zero could possibly get any better, then boy do we have a pleasant surprise for you! The new Raspberry Pi Zero W...
https://www.adafruit.com/product/3400

Raspberry Pi Zero 2 W
Raspberry Pi Zero 2 W is the latest product in Raspberry Pi's most affordable range of single-board computers. The successor to the breakthrough Raspberry Pi Zero...
https://www.adafruit.com/product/5291

Installation
CircuitPython builds for Raspberry Pi boards are available on circuitpython.org just like other microcontroller versions, under the Downloads section.

Download from CircuitPython.org
Next, click on the board that you have.

The board page allows you to download the stable or unstable release of CircuitPython. This notion of stability applies to CircuitPython as a whole, not necessarily the "port", meaning support for certain chip family. For the Raspberry Pi builds, always use "unstable" or "absolute latest" while the port is still in an alpha state (documented in the release notes.)
Additionally, you can also select a language for all of the serial messages including Python errors.

Broadcom builds come in two different file types:

- `.disk.img.zip` is full filesystem image that is typically used by "imager" programs like `rpi-imager`.
- `.kernel8.img` or `.kernel.img` is only the CircuitPython portion of the image and can be copied directly onto an existing SD card.

The disk image is best to start with but may overwrite the CircuitPython filesystem if used for updating. (The actual behavior varies based on the imaging program.)

### Raspberry Pi Imager

Raspberry Pi (the company) has created an imager that makes it easy to write disk images to an SD card. We'll use it to write the CircuitPython disk image to a card.

Install the Raspberry Pi Imager based on instructions from [the Raspberry Pi website](https://www.raspberrypi.org/). The Raspberry Pi Imager may erase the entire contents of the microSD card so make sure you've backed up any files you want to keep!

After launching the Imager, click "Choose OS".

![Raspberry Pi Imager](image)

Then, scroll down to the bottom of the popup and select "Use custom".
In the file selection dialog, pick the .disk.img.zip file that you downloaded earlier.
Next, choose the storage device to write the disk image to. Make sure it's correct! It will be overwritten.

After selecting the disk you wish to write, the middle box will change. To initiate the write click "WRITE". Imager will confirm the selection.
After confirming the write, a progress bar will show for writing the SD card and then verifying the contents were written correctly.
After the SD card is written, remove it from the writer and insert it into an unpowered Raspberry Pi. Power the board up to start CircuitPython.

On first power up, CircuitPython will create a second partition on the SD card to use for the CIRCUITPY drive. On subsequent flashes, some writers may preserve this partition but others may not. So, make sure and back up your code prior to flashing.

**Upgrading**

The CircuitPython disk image includes the CircuitPython code along with supporting code from the Pi folks and potentially erases the whole card.

If you want to just update CircuitPython, then you can download the kernel image alone (called kernel.img for Zeros and kernel8.img for Zero 2, Pi 3 and Pi 4.)
From circuitpython.org/downloads the filename is really long but the Raspberry Pi boot code only knows how to find the exact short name.

So, to update CircuitPython first download the kernel img file following the instructions above and then rename it to remove the circuitpython information.

After renaming the file, drag or copy the file onto BOOT. If your OS asks, confirm that you want to overwrite the existing file.

Unlike most CircuitPython boards, this BOOT partition is a real filesystem on the SD card. It isn't a custom bootloader.

Again, after upgrading, put the SD card back into your Pi when powered off and then power it up.
Connecting

Raspberry Pi boards are typically designed to make it easy to connect USB devices like mice and keyboards. In that case, the Raspberry Pi itself is known as a USB host. The rectangular USB-A connector usually designates the host side of a cable.

However, CircuitPython is usually acts as a device itself. That’s how it presents a CircuitPython drive and/or acts as a keyboard. This is normally done with the small micro-B style plug.

Newer boards have USB-C which can indicate either USB device or host.

For CircuitPython on the Raspberry Pi, the USB device capable connector varies per board design. The board description on CircuitPython.org includes instructions. Below is info for official Raspberry Pi boards.

When the power USB is separate from the indicated USB, insert the power USB before connecting the data USB.

Pi Zero and Zero W

Connect to the micro-B connector marked USB.
Pi Zero 2 W

Connect to the micro-B connector marked **USB**.

---

Pi 4 Model B

The USB-C power connector is also device capable. Use a powered USB hub to both power the Pi and connect it to your computer.
CM4 IO

The Raspberry Pi CM4 IO board has a dedicated micro-B connector for USB device.

Using CircuitPython

The CircuitPython experience on the Raspberry Pi is very similar to the experience on all of the other USB capable CircuitPython boards. One difference is that the CIRCUITPY drive is on removable storage. So please leave the SD card inserted for the whole time CircuitPython is running.

New to CircuitPython

If you are new to CircuitPython, then start by checking out the excellent Welcome to CircuitPython and CircuitPython Essentials guides. They’ll get you started connecting to the CircuitPython serial and editing code.py. The essentials guide will provide basic examples for the common hardware interfaces.
Experienced CircuitPythonistas

The Raspberry Pi port is still very new and is likely missing modules you've used on other boards. Check out the Support Matrix for the lastest details on what is supported.

HDMI

One of the main benefits of CircuitPython on the Raspberry Pi, is the easy use of a connected HDMI display. These are the first CircuitPython boards to support HDMI output. (Audio output over HDMI isn't currently supported.)

When using a display, make sure it is connected on bootup. The GPU firmware will discover the connected display and start the video output. Once CircuitPython starts, it will start a 640x480 frame buffer for displayio's normal terminal output. This is likely the wrong size for your display (but it's a safe bet for support.) So, you can reinitialize the CircuitPython framebuffer with a few lines of code. This code can be put into boot.py if you want to run it once on boot or in code.py if you want to change it later. The example sets the output to 1/4 the size of HDMI (half of each dimension) because large terminal framebuffers are a bit slow for CircuitPython to update.

```python
import displayio
import framebufferio
import videocore

displayio.release_displays()
fb = videocore.Framebuffer(1920 // 2, 1080 // 2)
display = framebufferio.FramebufferDisplay(fb)
```

After this re-initialization, `display` variable or `board.DISPLAY` can be used as normal. Check out the displayio guide for details.

Debug

These instructions are for more advanced debugging methods with external hardware and software and not routine CircuitPython code issues.
Print Debugging

The easiest way to debug is by adding print statements into the CircuitPython C code and viewing the output over UART. UART is a serial connection that is available earlier than serial over USB and tends to be more reliable as well.

```
#include "runtime.h"

...

// Acts like printf.
mp_printf(&mp_plat_print, "hello %d\n", 10);
```

The broadcom builds have the DEBUG UART enabled by default on the default RX and TX pins in the 40 pin header. Connect a USB to Serial converter to view the debug output.

One can also send serial input to CircuitPython over this connection.

GDB

It is also possible to connect to the CPUs over GDB via openocd. See the GDB guide for installation and usage details.

The broadcom-peripherals library includes configuration files for openocd. Here is an example command to run from ports/broadcom from within the CircuitPython repo that uses a JLink to talk to the Pi.

```
openocd -d -f interface/jlink.cfg -f peripherals/openocd/bcm2835.cfg
```

Then, within GDB do this to connect to OpenOCD:

```
tar ext:3333
```