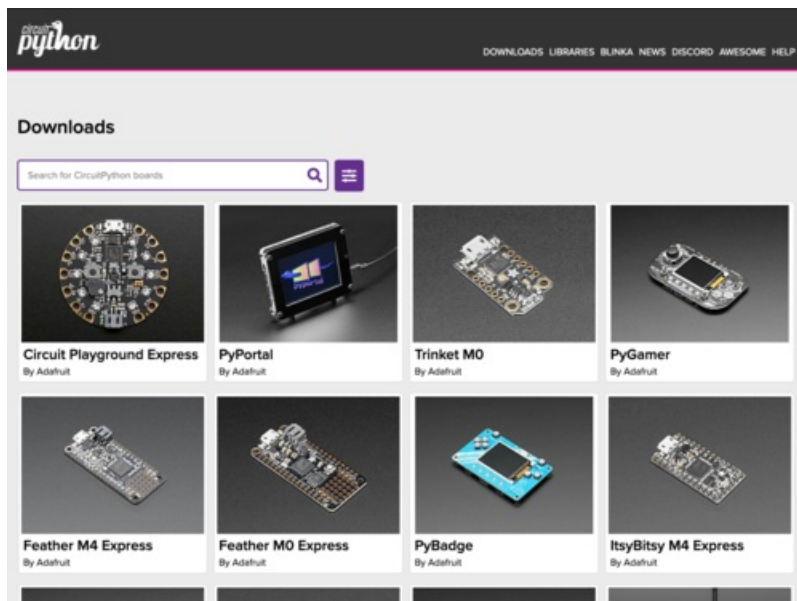


# How to Add a New Board to CircuitPython

Created by Kattni Rembor



Last updated on 2021-01-26 05:47:50 PM EST

## Guide Contents

Guide Contents	2
Overview	3
Get Setup to Add Your Board	4
Build CircuitPython	4
Create a Branch	4
Initial Configuration	5
Initial Steps for Adding Your Board	5
Duplicate a Current Board Directory	5
Board Files	6
Verify the REPL is Working	6
Board Name	6
Chip Variant	7
SPI Flash Configuration	7
Crystal Configuration	7
Initial Build	8
Build CircuitPython for Your Board	8
Customizing the Board Files	10
mpconfigboard.h	10
pins.c	11
Understanding the Schematic to Identify Pins	11
mpconfigboard.mk	13
board.c	13
.github/workflows/build.yml	13
UF2 Bootloader	14
Submit a Pull Request	15
Pull Request to CircuitPython	15
A New CircuitPython Board	15

## Overview



You've designed a circuit board with microcontroller and it's CircuitPython compatible. Amazing! Wouldn't it be great to have CircuitPython automatically built for your board? We can help with that!

Adding a CircuitPython compatible board to CircuitPython means that the firmware will be automatically built for your board every time CircuitPython is updated - on merged pull requests, beta releases as well as final releases. This will enable you to easily use CircuitPython on your board without going through the build process every time, as well as allow for you to promote your board as easy to use with CircuitPython.

Best of all, it's free! We'll do it all for you once you've given us the information we need to do the builds. We'll also build every translated language as well, from Portugese to Pinyin.

This guide will walk through the process of adding a board to CircuitPython. You will need to be familiar with Git and GitHub to do this process. We have a great guide on [Contributing to CircuitPython with Git and GitHub \(https://adafru.it/Fj0\)](https://adafru.it/Fj0) if you're unsure how to get started. This guide assumes you are either already familiar with or have gone through the guide for Git and GitHub.

You will begin by creating your own fork of CircuitPython and verifying you can build successfully. From there, you'll edit a series of files to match your board's configuration, and then create a pull request to have your changes added to CircuitPython. Once added, the firmware for your board will be automatically built.

Let's get started!

# Get Setup to Add Your Board

There are a number of steps required for adding a board to CircuitPython. You'll need to edit a series of files and put in a pull request with the changes. This guide assumes that you are adding a SAMD21 or SAMD51 microcontroller board to CircuitPython. We also support nRF52840 (and possibly more chips by the time you're reading this!) The process is similar for all of them

The following files must be updated. The `board_name` folder will be your board. The first three must be customized for your board and your board must added to the last:

- `/ports/atmel-samd/boards/board_name/mpconfigboard.h`
- `/ports/atmel-samd/boards/board_name/mpconfigboard.mk`
- `/ports/atmel-samd/boards/board_name/pins.c`
- `./github/workflows/build.yml`

There are a few other files that you may update depending on your board, however, the files listed above are the minimum necessary to add your board support package.

This guide explains how to find and update these files. You will need to be familiar with Git and GitHub. We have an [excellent guide on contributing to CircuitPython with Git and GitHub](https://adafru.it/Dkh) (<https://adafru.it/Dkh>) if you need some help with getting started.

## Build CircuitPython

The first thing you'll need to be able to do, is build CircuitPython successfully. Visit GitHub and create your own fork of CircuitPython. Then, you'll need to get set up to build CircuitPython. We have a [great guide that explains the setup and build process](https://adafru.it/Fj1) (<https://adafru.it/Fj1>). Go through the linked guide before continuing!

Try building for a known-good board such as the Feather M0 Basic to verify that you can build successfully before continuing to the next step!

## Create a Branch

Once you've successfully built CircuitPython, create a branch using git. You'll do all your edits within the branch.

```
10022 kattni@robocrepe:atmel-samd [47m master= f0cf9a4e7]$ git checkout -b pyruler
Switched to a new branch 'pyruler'
```

# Initial Configuration

## Initial Steps for Adding Your Board

To demonstrate, this guide will walk through adding a board similar to the Trinket M0 called PyRuler.

## Duplicate a Current Board Directory

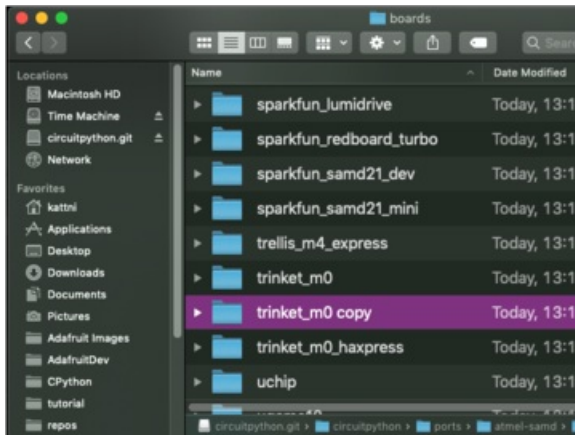
Take a look at the existing boards available for CircuitPython and choose one that is close to yours so you have something to start with to modify.

Finding an existing board that matches your configuration is the easiest way to add your board to CircuitPython! If at all possible, find a board that matches yours to modify, as much of the information will already be accurate and not require updating.

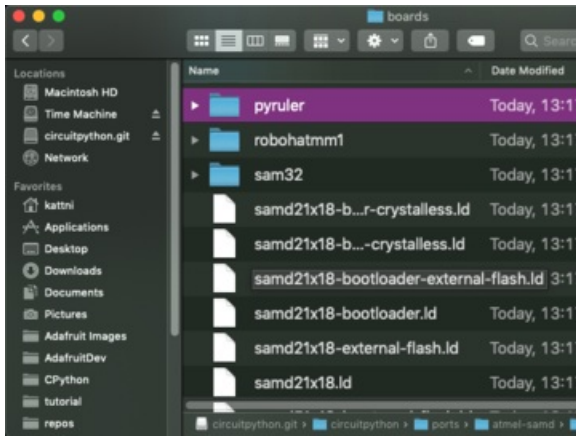
The most important things to match are:

- Which version of the microcontroller are you using? You'll want to match as closely as possible to ensure RAM, size, etc are the same. You can do this by verifying that the full name of the chip, e.g. ATSAM51J19, matches the board you're copying from.
- Does your board have external SPI flash? If so, choose a board to start with that also has external SPI flash, such as the Adafruit Feather M4 Express. Try to find a board that has the same SPI flash chip as you are using. See **SPI Flash Configuration** below for more details.
- Does your board have a crystal? If so, choose a board to start with that also uses a crystal, such as the Adafruit Feather M4 Express.

Find the board directory you're choosing to start with in `/ports/atmel-samd/boards`. PyRuler is similar to the Trinket M0.



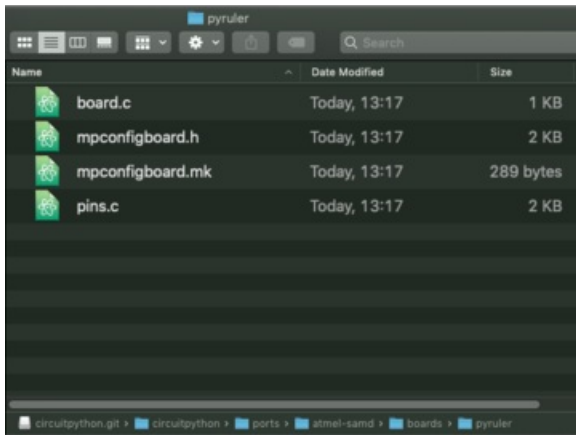
Duplicate the directory of the board that is closest to your board.



Rename the directory to your board name.

## Board Files

Each board directory contains four files. These files currently contain information for the board you chose to duplicate. You will need to go over each file and apply the necessary changes to make the files fit your board. The files are:



1. board.c
2. mpconfigboard.h
3. mpconfigboard.mk
4. pins.c

## Verify the REPL is Working

The first thing you want to do is verify that you can build for your board by making the most basic changes possible, building CircuitPython, loading it on your board, and ensuring that you can get to the REPL. Other things like pin names will be incorrect, but this will verify that the build process works and that you can load the firmware.

These changes include:

- Update the board name to match your board name in **mpconfigboard.h**
- Verify your chip variant is correct in **mpconfigboard.h** and **mpconfigboard.mk**
- Verify that the SPI flash configuration matches in **mpconfigboard.h** and **mpconfigboard.mk**
- Verify the crystal configuration matches in **mpconfigboard.h**

If you've chosen a board that matches your configuration, the last three things on this list should already match. If not, be sure to update them to match.

## Board Name

Updating the board name to match your board name means when you load the REPL, your board name will show up in the initial info line. It's a simple way to know that your changes have been included in your new build. To update the board name, open **mpconfigboard.h**. Change the string in the **BOARD\_NAME**

line to match the full name of your board.

```
#define MICROPY_HW_BOARD_NAME "Adafruit PyRuler"
```

## Chip Variant

If you've chosen a set of board files to modify from a board that uses the exact same chip as yours, then no changes are necessary. However, it's always worth verifying that the chip variant matches.

There are two places where the chip variant on the board is identified: **mpconfigboard.h** and **mpconfigboard.mk**. Start with **mpconfigboard.h** as you should already have it up.

In **mpconfigboard.h**, verify that the string in the **MCU\_NAME** line matches your chip. It can be the full name of the chip - you'll notice that in the PyRuler example, it does not include the last letter of the chip name. This MCU name is used in various strings in CircuitPython, such as the string in the REPL that identifies which board you're using.

```
#define MICROPY_HW_MCU_NAME "samd21e18"
```

In **mpconfigboard.mk**, verify that the **CHIP\_VARIANT** and **CHIP\_FAMILY** match your chip. These affect the compilation of CircuitPython.

The **CHIP\_VARIANT** should be the full name of the chip, e.g. **SAMD21E18A**.

The **CHIP\_FAMILY** should be the beginning of the chip name, including the product family and the product series, e.g. **samd21**.

```
CHIP_VARIANT = SAMD21E18A  
CHIP_FAMILY = samd21
```

## SPI Flash Configuration

You will want to ensure that you've started with an existing Adafruit board that has the same SPI flash configuration. The currently supported chips are listed in **supervisor/shared/external\_flash/devices.h**. There is a lot involved in setting up a new SPI flash chip that will not be covered in this guide at this time. We only guarantee support for the flash chips used on Adafruit boards. This is not to say that other flash chips will not work, it's simply to say we don't guarantee that they will.

## Crystal Configuration

This is as simple as whether or not your board has a crystal or not. If your board has a crystal, find an existing board that also has a crystal, as suggested above. Be sure to include **#define BOARD\_HAS\_CRYSTAL 1** in your **mpconfigboard.h** file. If you don't have a crystal, we'll try to sync to the USB 1KHz clock pulses to calibrate the crystal, which save space and component cost, but is less precise when USB isn't connected

# Initial Build

## Build CircuitPython for Your Board

Update the board name to match your board name, and verify that the three other piece of information are correct. Then, build CircuitPython the same way you did initially, however, this time, you build for your board, where `your_board_name` is what you chose to rename the duplicated board folder:

```
make BOARD=your_board_name
```

Don't forget to run that from within the port directory, such as `circuitpython/ports/atmel-samd`

```
ladyada@ladyada301-PC MINGW32 ~/Dropbox/micropython/circuitpython/ports/atmel-samd
$ make BOARD=pyruler -j 4
Use make V=1, make V=2 or set BUILD_VERBOSE similarly in your environment to increase build
verbosity.

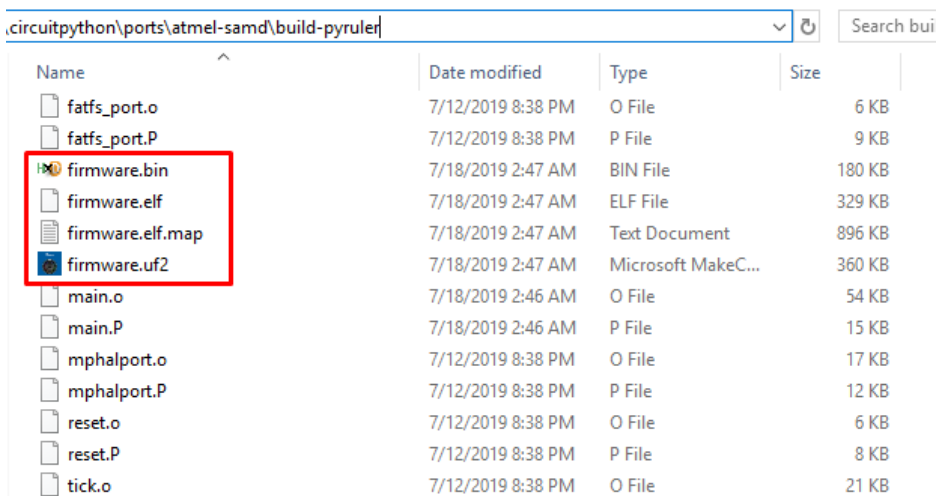
4112 bytes free in flash out of 188160 bytes ( 183.75 kb ).
25916 bytes free in ram for stack out of 32768 bytes ( 32.0 kb ).

Converting to uf2, output size: 368128, start address: 0x2000
Wrote 368128 bytes to build-pyruler/firmware.uf2.

ladyada@ladyada301-PC MINGW32 ~/Dropbox/micropython/circuitpython/ports/atmel-samd
```

Don't forget you have to have set up CircuitPython build environment first. Check out this guide on how to do that (<https://adafru.it/Bfu>)

Once that process is complete, find the firmware file. In this case you see it's in `build-pyruler/firmware.uf2` but there will be a few formats available such as bin and elf, for your convenience



Load the created firmware onto your board using a bootloader or programmer. The **CIRCUITPY** drive should automatically appear over USB if you have a FLASH chip connected properly (or if the internal FLASH is being used).

If you don't get a disk, you should *still* get a USB serial connection to test the REPL, check your OS for the COM or Serial port created. If you aren't getting a REPL check over your work from before - did you select the right chip variant and crystal setup?

Connect to your board via serial, and press enter to start the REPL.

```
Press any key to enter the REPL. Use CTRL-D to reload.
Adafruit CircuitPython 4.1.0-beta.1-14-gf0cf9a4e7 on 2019-07-09; Adafruit PyRuler with samd21e18
>>> |
```



Success! This verifies that the most crucial information is accurate and that the build works on your board.

From the REPL, try the following:

```
import microcontroller
dir(microcontroller.pin)
```

You will see a list of all the pins available to CircuitPython.

Depending on which board you chose to duplicate, the `microcontroller.pin` module may not have access to all of the pins set up on your board. This is determined by an ignored pin list found in `mpconfigboard.h`.

Let's take a look!

# Customizing the Board Files

The next step is customising the board files. This section walks you through each file and how to update it. Let's get started!

## mpconfigboard.h

This file changes how the C code works through macros for the C code. It also tells the board how to set up the file system.

Open **mpconfigboard.h** in your editor of choice. This file contains the following:

- Board name
- MCU name
- Status LED pins
- Board flash size
- Ignored pins
- Default I2C, SPI and UART bus pins

PyRuler does not have an external SPI flash chip, and used the same microcontroller as the Trinket M0, so no changes were needed to NVM or flash sizes. Don't mess with this unless you know what you're doing! You can cause issues with the flash table on your board.

To update this file, you'll need to do the following:

- You should have already updated your board name and verified the MCU name.

```
#define MICROPY_HW_BOARD_NAME "Adafruit PyRuler"  
#define MICROPY_HW_MCU_NAME "samd21e18"
```

- Identify what pin your status LED is on and update the file to reflect the appropriate pin. If you have more than one status LED, e.g. your board also includes a DotStar, you can identify that here as well.

```
#define MICROPY_HW_LED_STATUS    (&pin_PA10)  
  
#define MICROPY_HW_APA102_MOSI  (&pin_PA00)  
#define MICROPY_HW_APA102_SCK  (&pin_PA01)
```

- There will always be at least two pins on the ignored pins list: PA24 and PA25. These pins are used for USB and are always ignored by CircuitPython. Typically, those will be the only two pins on that list. The only time that list is expanded to include more pins is if the board flash is small enough to require tweaking to make the build fit. Trinket M0 and PyRuler are examples of boards where the build must be slimmed down. That is why there is a fairly extensive list of ignored pins in this file for these boards. The pins listed here are pins on the chip that are not connected to anything.

```
// USB is always used internally so skip the pin objects for it.  
#define IGNORE_PIN_PA24      1  
#define IGNORE_PIN_PA25      1
```

- Update the default I2C, SPI and UART busses. These are listed at the bottom of the file. If yours are on different pins, simply change the pin numbers to match what your board uses.

```

#define DEFAULT_I2C_BUS_SCL (&pin_PA09)
#define DEFAULT_I2C_BUS_SDA (&pin_PA08)

#define DEFAULT_SPI_BUS_SCK (&pin_PA07)
#define DEFAULT_SPI_BUS_MOSI (&pin_PA06)
#define DEFAULT_SPI_BUS_MISO (&pin_PA09)

#define DEFAULT_UART_BUS_RX (&pin_PA07)
#define DEFAULT_UART_BUS_TX (&pin_PA06)

```

If you're unsure how to find what pins are correct, you'll want to check the schematic. You'll also need to do that for `pins.c`. Let's take a look.

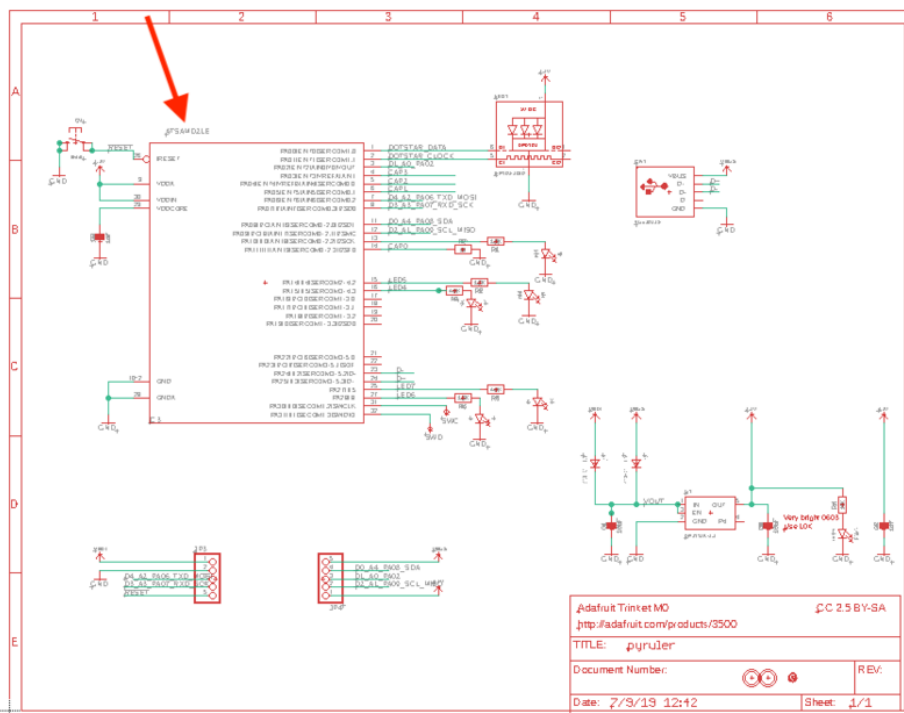
## pins.c

The `pins.c` file contains the board pin names as they will appear to the user associated with the microcontroller (MCU) pin names. This is where the `board` module in CircuitPython gets the board pin names that it makes available to the user for use in code, e.g. `board.A1`, etc. The board pins listed in this file should match the labels on the silkscreen on your board.

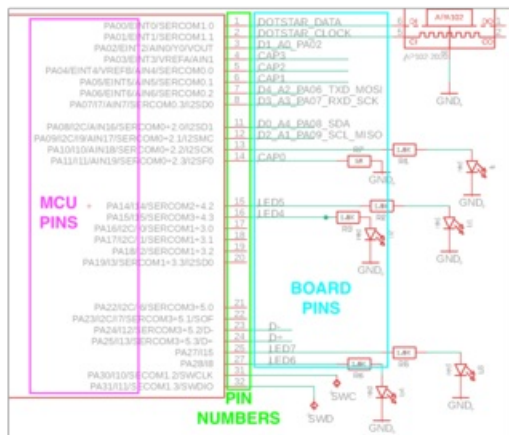
Use the board's schematic to identify what pins are associated with what names and enter all of them into the `pins.c` file, using the same format as is used for the pins already listed in the file. Be sure to verify that all pins present already are accurate, or be certain to delete them or update them to make them accurate. PyRuler has a number of different pin assignments from Trinket MO, and a number of additional pins. These were all updated and added as needed.

## Understanding the Schematic to Identify Pins

Open the schematic for your board and find the symbol for the MCU. It may be labeled with the chip name.



The information you need for identifying which pins names should be associated with which pins is typically contained within this symbol. Let's take a closer look.

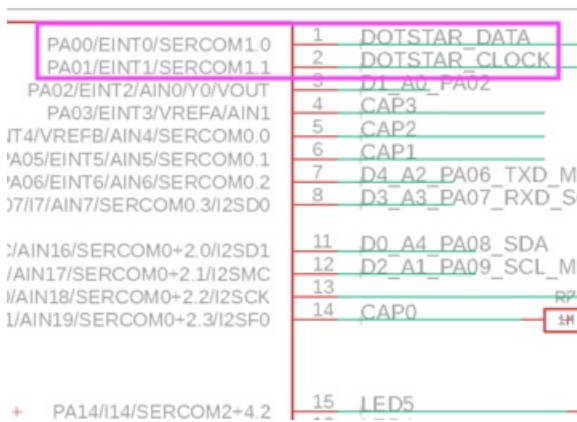


The MCU pins are listed along the right side of the MCU schematic symbol outline (the red box) in the PyRuler schematic, e.g. PA00, PA01, PA03. The MCU pins are outlined in magenta.

The board pin that each MCU pin is connected to is on the outside to the right, e.g. DOTSTAR\_DATA, DOTSTAR\_CLOCK, D1\_A0\_PA02, CAP3. The board pins are outlined in blue.

Note that the pin numbers listed along the edge of the schematic symbol are not the same as the pin names! The names start at 0 and the pin numbers start at 1. The pin numbers are outlined in green.

Now you can begin to identify the associated MCU pins and board pins. Start at the top.



The first two pins are connected to the on-board DotStar LED.

This is relevant to both `mpconfigboard.h` and `pins.c`.

`mpconfigboard.h` includes identifying any status LEDs, which includes the DotStar LED built into PyRuler. It is included in this file so CircuitPython can use it for status information.

The entry in `pins.c` is for the user to be able to manipulate the LED using the `board` module.

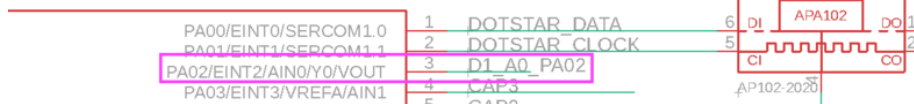
`pins.c` has many lines all with the same format that associate MCU pins with board pins. You can copy the format to add more pins if necessary.

The DotStar clock and data pins are listed towards the bottom of the file.

```
{ MP_ROM_QSTR(MP_QSTR_APA102_MOSI), MP_ROM_PTR(&pin_PA00) },
{ MP_ROM_QSTR(MP_QSTR_APA102_SCK), MP_ROM_PTR(&pin_PA01) },
```

Continue down the list, assigning MCU pins to board pins.

Some MCU pins will have more than one board pin associated with them. Each board pin name requires its own line with the MCU pin name repeated on multiple lines if necessary. For example, MCU pin PA02 on the PyRuler is used for board pins D1 and A0.



So, there are two lines in `pins.c` to assign the board pin names to the MCU pin:

```
{ MP_ROM_QSTR(MP_QSTR_D1), MP_ROM_PTR(&pin_PA02) },
{ MP_ROM_QSTR(MP_QSTR_A0), MP_ROM_PTR(&pin_PA02) },
```

Repeat this process for all the board pins that you would like to expose to the user in the `board` module.

## mpconfigboard.mk

The information in this file changes how the build works, and changes what files are included.

You should have already verified that the `CHIP_VARIANT` and `CHIP_FAMILY` were accurate when you did the initial test build for your board. The `CHIP_VARIANT` should be the full name of the chip, e.g.

`SAMD21E18A`. The `CHIP_FAMILY` will be `samd21` or `samd51` depending on which chip your board uses.

```
CHIP_VARIANT = SAMD21E18A
CHIP_FAMILY = samd21
```

The lines you'll want to update in this file are:

- `USB_VID` - This is the vendor ID.
- `USB_PID` - This is the product ID.
- `USB_PRODUCT` - This is the name of your board.
- `USB_MANUFACTURER` - This is who makes your board.

USB VID and PID are something you *cannot* borrow from another board. So *don't* copy and paste these values! If you don't have a VID/PID pair, [see the how-to page at pid.codes to learn how request one \(https://adafru.it/Qcp\)](https://adafru.it/Qcp), we will assist in the process of assigning a PID for use with CircuitPython.

```
USB_VID = 0x239A
USB_PID = 0x804C
USB_PRODUCT = "PyRuler"
USB_MANUFACTURER = "Adafruit Industries LLC"
```

## board.c

This file handles board-specific initialisation and functionality, such as initialising a display or creating custom ways to get into safe-mode. Generally, it is empty functions. `board.c` is for more advanced usage. Regardless of whether you add anything to this file, it must be present or the build will not occur or be verified, and is also necessary for automatic releases and board inclusion on [circuitpython.org/downloads](https://circuitpython.org/downloads).

A good example of making changes to this file is the [Circuit Playground Express board.c \(https://adafru.it/Fj2\)](https://adafru.it/Fj2) file. It includes a more complex setup, such as optional safe-mode, reset configuration, and resetting the NeoPixels when the board is reset by sending the board state to the NeoPixels to turn them off after the user code is done.

## ./github/workflows/build.yml

The `./github/workflows/build.yml` file is used to determine what firmware needs to be built and released. It is found in the CircuitPython root directory, e.g. `circuitpython/.github/workflows/build.yml` if you cloned the repo into a folder called `circuitpython`.

Find the `board` list below the `matrix` line. Add your board name in the same format as the folder you renamed in the beginning. In the case of PyRuler, the folder is `pyruler`. So, `pyruler` has been added to a new line in alphabetical order.

```
- "pyportal_titano"  
- "pyruler"  
- "robohatmm1_m0"
```

## UF2 Bootloader

We highly recommend using a UF2 bootloader. If you are not, however, you will need to update another file to ensure that the correct type of firmware is built for your board. The file is `/circuitpython/tools/build_board_info.py` and the relevant section begins on [line 28 \(https://adafru.it/Fj3\)](https://adafru.it/Fj3).

There are three options for building firmware.

- **BIN** means build only a .bin firmware
- **BIN\_UF2** means build both .bin and .uf2 firmware
- **HEX** means build only .hex firmware

Add your board to the appropriate section in the same format if you need an alternative format of firmware built.

# Submit a Pull Request

## Pull Request to CircuitPython

Now that you've updated all the necessary files, it's time to put in a pull request to the CircuitPython repo with your changes. First, verify that you've updated all the necessary files. In many cases, you will have edited only the following files:

- `mpconfigboard.mk`
- `mpconfigboard.h`
- `pins.c`
- `../../github/workflows/build.yml`

Your list may include more files depending on your setup, but the files listed above are the minimum likely needed. Now it's time to use Git to get your changes pushed to your fork.

```
18038 kattni@robocrepe:atael-saad [3h pyru1er *% f8cf9a4e7]$ git status
On branch pyru1er
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

   modified:   ../../.travis.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)

   boards/pyru1er/

no changes added to commit (use "git add" and/or "git commit -a")
18039 kattni@robocrepe:atael-saad [3h pyru1er *% f8cf9a4e7]$
```

Use `git status` to verify that the necessary files have been edited. Note that since you created a new directory, the directory (not the individual files) is what shows up in `git status` until you `add` the directory.

```
18043 kattni@robocrepe:atael-saad [3h pyru1er *% f8cf9a4e7]$ git add .
18044 kattni@robocrepe:atael-saad [3h pyru1er *% f8cf9a4e7]$ git add ../../.travis.yml
18045 kattni@robocrepe:atael-saad [3h pyru1er * f8cf9a4e7]$ git status
On branch pyru1er
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

   modified:   ../../.travis.yml
   new file:   boards/pyru1er/board.c
   new file:   boards/pyru1er/mpconfigboard.h
   new file:   boards/pyru1er/mpconfigboard.mk
   new file:   boards/pyru1er/pins.c

18046 kattni@robocrepe:atael-saad [3h pyru1er * f8cf9a4e7]$
```

Use `git add` to stage your files for commit. Check `git status` to verify that everything has been added. Your file list should include at least every file shown in the image.

Now commit your changes, and push your branch to your fork. Use GitHub to create a pull request (PR) to CircuitPython with your changes. If you need assistance with creating a PR, follow the process explained in the [Git/GitHub guide in the Create Your Pull Request \(https://adafru.it/EvC\)](https://adafru.it/EvC) and the [Open Pull Request \(https://adafru.it/Fj4\)](https://adafru.it/Fj4) pages.

## A New CircuitPython Board

Once your PR is created, it's now up to us! We'll take a look at it and make sure everything is in order before merging your changes into CircuitPython. If anything was missed, we'll let you know so you can get it taken care of. Once the changes are merged, your board is officially part of CircuitPython!

When a board is merged into CircuitPython, it will be built and pushed to [the CircuitPython S3 bucket \(https://adafru.it/AjW\)](https://adafru.it/AjW). New firmware is built every time there's a commit to the CircuitPython repo, which happens when a PR is merged. Find your board, and then choose the language you'd like, and you'll find all the available builds.

When CircuitPython is released, all the boards that have been added since the last release will be included in the release. At this point, the [firmware builds will be available on GitHub below the release \(https://adafru.it/tBa\)](https://adafru.it/tBa). CircuitPython builds firmware in every available language for every available board, meaning there are MANY assets found on GitHub releases. It's often easier to obtain the firmware from S3 as it's organised by board and language in subfolders.

The next step will be getting your board added to [circuitpython.org/downloads \(https://adafru.it/Em8\)](https://adafru.it/Em8). To do that, be sure to check out our [How to add a New Board to the circuitpython.org website \(https://adafru.it/Fnw\)](https://adafru.it/Fnw) guide on how to do that!



