Adding a Single Board Computer to PlatformDetect for Blinka

Created by Melissa LeBlanc-Williams

https://learn.adafruit.com/adding-a-single-board-computer-to-platformdetect-for-blinka

Last updated on 2024-06-03 03:04:42 PM EDT

Table of Contents

Overview

So you have a new board, that you want to get working with Blinka, our CircuitPython library compatibility layer for Single Board Computers, or SBCs. Adding a board to Blinka is a 2-part process with the first step being to add it to `PlatformDetect` so that Blinka knows what board it is working with.

The way that Blinka works is it combines multiple techniques to access the parts of the board that it needs. For GPIO, libraries such as `RPi.GPIO` or `libgpiod` are used. For I2C, the PureIO python library (https://adafru.it/JEh) is used, and for SPI, we are using the spidev python library (https://adafru.it/JEi).

When it comes to Blinka itself, there are two main components that need to be defined.

- First there is the **chip**, which is the microprocessor definition.
- Second, there is the **board** definition.

The chip defines which pins of the microprocessor are used by the GPIO library and **the board defines which physical pin on the board maps to the pins defined by the chip file**. There can be multiple boards that all use the same chip, but the specific implementation may be slightly different. For example, there are many Raspberry Pi computers, but many share the same Broadcom chips. This is why it is important to detect both the chip and the board.

In this guide, we will be going over what steps are needed to add a board such as the Pine A64 to PlatformDetect.

Parts

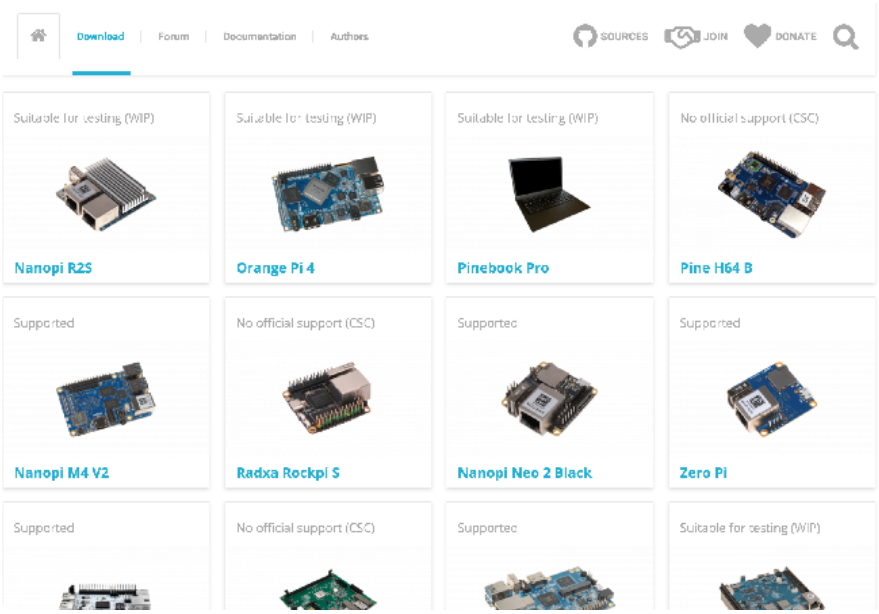
USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi
The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...
https://www.adafruit.com/product/954

FTDI Serial TTL-232 USB Type C Cable
Just about all electronics use a UART serial port with RX and TX pins for debugging, bootloading, programming, serial output, etc. But it's rare for a computer to have a serial...
https://www.adafruit.com/product/4364

Selecting an OS

Many boards will offer a variety of Operating Systems options that run on SBCs, but not all of them provide as many identifying markers as others. Sometimes there is a clear choice such as using Raspbian with the Raspberry Pi and other times it is not so clear. Choosing the right OS is very important because some of them may limit which peripherals you are able to use or interact with.

One OS that we have had some really good luck with getting working is **Armbian** and if it is available for your SBC, it is a solid choice. You can check if your board is supported on the Armbian Downloads page (https://adafru.it/JCk).



If Armbian isn't available, then the next option would be to check if there is a manufacturer recommended OS. The reason why you would want a manufacturer recommended one is because there is likely to be more time put into developing drivers, so getting everything working is much easier.

Once you have selected an OS, go ahead and install it according to the instruction that come with the OS. In this guide, Armbian is available for the Pine64, so we will be using that.

Environment Setup

Once you have the OS loaded, you will need to get to a terminal window and run a few commands that will help you out. Sometimes getting to a terminal is the trickiest part and it really depends on your specific board. Popular options include using SSH, connecting a USB to TTL Serial cable to some GPIO pins, or just connecting a display and keyboard directly to the board.

If you want to connect with a USB to TTL serial cable, we offer a few different cables, but either of these should cover most of your needs.

USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...

https://www.adafruit.com/product/954

FTDI Serial TTL-232 USB Type C Cable

Just about all electronics use a UART serial port with RX and TX pins for debugging, bootloading, programming, serial output, etc. But it's rare for a computer to have a serial...

https://www.adafruit.com/product/4364

Connect to a Network

The next step is to make sure your board is connected to a network, either by an ethernet cable or WiFi.

Ensure SSH is enabled

Some boards such as the Raspberry Pi have SSH disabled by default and you will need to either run a config utility or some other method to enable SSH, such as adding a specific empty file, otherwise all attempts will be denied. This is something you will need to check with the OS or manufacturer if you are not sure. For the Armbian on the Pine64, it is already enabled by default, so we don't need to worry.

Even if you don't plan on using SSH to connect to the terminal, it's still a good idea to enable it so that you can copy your updated PlatformDetect files over easily by FTP if you choose to do so.

Update Your OS and Python

Run the standard updates:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

and

```
sudo pip3 install --upgrade setuptools
```

If above doesn't work try

```
sudo apt-get install python3-pip
```

Enable mDNS

If you prefer to use SSH, you may need to connect with one of the alternative methods or lookup the IP address on your router first. Then once you're in, you can enable Multicast DNS if it is not done already, so you can connect without needing to know the IP address:

- `sudo apt-get install avahi-daemon`

then **reboot**

Install Git

You'll want to make sure you have installed git so that you can clone PlatformDetect right onto your SBC.

- `sudo apt-get install git`

Make sure you're using Python 3!

The default Python on your computer may not be Python 3. Python 2 is officially discontinued and all our libraries are Python 3 only.

We'll be using `python3` and `pip3` in our commands, use those versions of Python and pip to make sure you're using 3 and not 2.

Getting a copy of the code

Step 1 - Fork PlatformDetect to your GitHub account

If you don't already have the code from Adafruit_Python_PlatformDetect (https://adafru.it/Dyb) forked to your local GitHub repository, you'll want to start off by doing that. If you are not sure how to use GitHub, we have an excellent guide available on using Git and GitHub (https://adafru.it/JCR).

Step 2 - git clone the PlatformDetect fork to your single board computer

Once you have forked it, you can clone the repository both onto your local computer and onto your SBC. By using your local repository as a copy point, this makes things really easy. If you'd prefer to just edit right on your SBC and commit from there, you can do that too. **In this guide, we'll assume you cloned it to your home directory on your SBC.**

Some other options for copying over the files include using SFTP and an FTP client such as FileZilla or SCP. However, you decide to do it, the guide will assume that you have a copy of the repository in your home directory and that it is located inside the folder **~/Adafruit_Python_PlatformDetect**.

Adding the Constants

After you have copied the repository, or repo for short, you'll want to go into the **adafruit_platformdetect/constants** folder. Here you will find a couple of files named **boards.py** and **chips.py**. This is where you will add your constants for the board and chip files.

Before adding a new chip constant, check to make sure it doesn't already exist. If it does, that will make adding your board much easier.

chips.py Constants

These should just be in the format with the string value matching the constant name, so for the Pine A64 chip, we would add:

```
A64 = "A64"
```

If you are using a longer name with multiple words, they should be separated by underscores.

boards.py Constants

Board constants should be in the same format as the chip so, for the Pine64 we can add it like:

```
PINE64 = "PINE64"
```

Next, the new board constant may need to be added to a group as well. If the new board you are adding is from the same manufacturer as an existing board, you should either add it to that group, if one exists, or create a new group. For the Pine64, there was already a couple other boards, so we will just make sure the new ID is inside the group.

```
_PINE64_DEV_IDS = (
PINE64,
PINEBOOK,
PINEPHONE
)
```

Adding Detection Code

Now that the constants are all set up, the next step is to add some detection code.

There are several ways that PlatformDetect is able to figure out which board and chip you are running and we will go over some of those methods. We start with adding detection for the chip since that will narrow down the number of boards to check. For detecting the chip, there are a number of available functions built-in that make it much easier.

Chip Detection

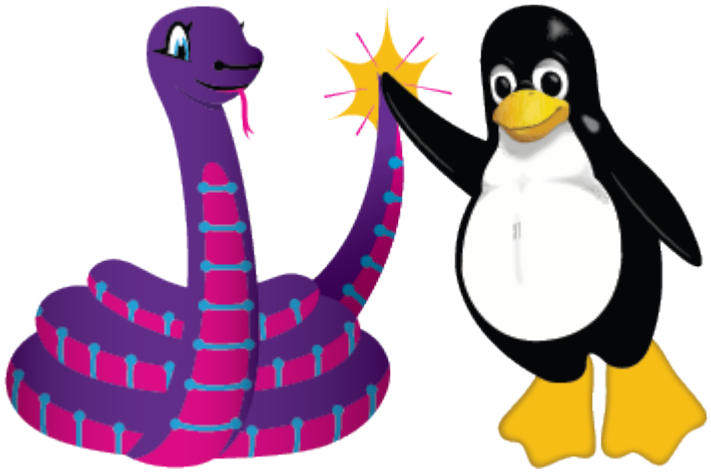
First we'll start with exploring the various chip detection methods.

Environment Variables

The first thing that PlatformDetect does is look for environment variables. This is used for boards that are more OS independent such as the FT232H and MCP2221, which are meant to be connected directly to a desktop. If none of the variables that it is expecting are found, it goes onto the next method.

Checking Linux

The next thing it does is check if Linux is the environment that it is currently running in and if so, it checks a number of locations that are Linux specific.



Using /proc/cpuinfo

One of the main ways of detecting the processor is by using the information available in **/proc/cpuinfo**. There is a built-in function called `get_cpuinfo_field()` that will return the value for a specific field from **/proc/cpuinfo**.

Using /proc/device-tree/compatible

Another location that is used is **/proc/device-tree/compatible**. This will return a string that can be matched against a specific value to check for specific chips. The built-in function `check_dt_compatible_value()` is used checking for specific values easier.

Using /etc/armbian-release

If you are running a release of Armbian, then **/etc/armbian-release** should be available that will check which board and processor that the specific version of

armbian was released for. You will want to use the built-in function `get_armbian_release_field()`.

Using /proc/device-tree/model

If none of those places are an option, another is to get the information from **/proc/device-tree/model** to get the device model. For this, you will want to use the `get_device_model()` function.

Using /proc/device-tree/compatible

Finally, another option is to use information available from **/proc/device-tree/compatible**. To check this, use the built-in `get_device_compatible()` function.

Adding the Chip Detection

Now that you are more familiar with the detection methods, the next step is to start modifying the code so that your chip is correctly detected. It's easiest to start from the top as the **adafruit_platformdetect/chip.py** file will try different things and return the chip once it has been correctly identified. Placing your detection code as close to the top means not needing to do any unnecessary processing.

Here's a little gotcha with the chip in hardware. The **sun50iw1p1** value that appears in **/proc/cpuinfo** looks like it should be a unique value, but is a common among several AllWinner chips. The unique value is inside of **/proc/device-tree/compatible**. We recently ran into this trap and after attempting to add other chips with the same ID, we switched detection methods. The place that worked best for the Pine64 was inside the **_linux_id()** function where we end up adding the following lines:

```
if compatible and "sun50i-a64" in compatible:
    linux_id = chips.A64
```

Board Detection

Once you have the chip being correctly detected, it's time to move onto the board. For board detection, we start with taking the result of the chip detection and run it through a giant if/else if statement. If there are multiple boards using the same chip, we will narrow it down further. If there's only one board, which is often the case, then it's easy and we will just return that board.

Adding Multiple Board Detection

If there are multiple boards you want to create a new function in **adafruit_platformdetect/board.py** down near the bottom. It should be called something that starts with an underscore, represents the board or chip manufacturer that they have in common, and ends in `_id()`. In the case of the Pine64, we will go with **_pine64_id** and since this is part of a class, we need to pass the self parameter, so it ends up being:

```
def _pine64_id(self):
```

After that, the function should employ some sort of detection to narrow down which board it is and should return the board constant that you defined earlier. Take a look at some of the other functions for detection ideas. The functions used in chip detection are also available for board detection and we will use `get_device_model()` for checking which board we have. For the Pine64, here is the full function:

```
def _pine64_id(self):
    """Try to detect the id for Pine64 board or device."""
    board_value = self.detector.get_device_model()
    board = None
    if 'pine64' in board_value.lower():
        board = boards.PINE64
    elif 'pinebook' in board_value.lower():
        board = boards.PINEBOOK
    elif 'pinephone' in board_value.lower():
        board = boards.PINEPHONE
    return board
```

In the giant if/else if statement, we want to add the following lines near the bottom right before the return statement:

```
elif chip_id == chips.A64:
    board_id = self._pine64_id()
```

The bottom of the statement then becomes:

```
elif chip_id == chips.A64:
    board_id = self._pine64_id()
return board_id
```

Adding an Any ID Property

If this is a new ID group, even if it contains a single ID, you want to add a convenience property that we can use to check if it is any Pine64 board. This makes things much

easier in Blinka. All we need to do is check if the identified board id matches any of our Pine64 board IDs. Here's the code for that function:

```
@property
def any_pine64_board(self):
    """Check whether the current board is any Pine64 device."""
    return self.id in boards._PINE64_DEV_IDS
```

Adding to any_embedded_linux()

It's very important to make sure the board can be found in `any_embedded_linux`. This is because **busio.py** now uses this to determine if the board is a linux board or MicroPython in order to simplify adding the right modules.

Testing Detection

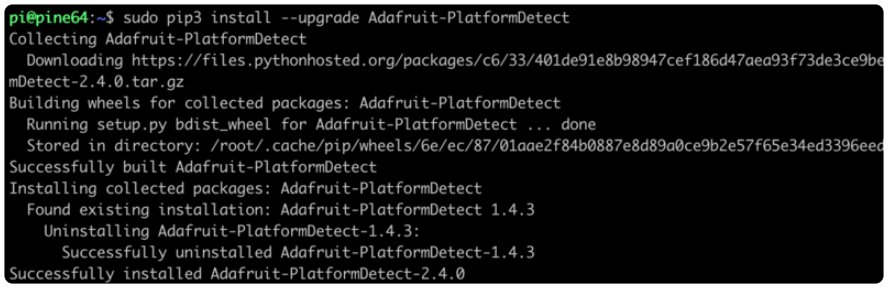
Once you have finished adding your chip and board detection code, or if you just want to see what the board and chip are currently being detected as, you can run the detection test script.

If you followed the instruction in the environment setup, you should have the **Adafruit_Python_PlatformDetect** folder in your home directory. Inside of there is a **bin** folder, and inside of that there is a file called **detect.py**. This is a useful script that can be used to see what PlatformDetect is seeing.

The easiest way to test your edited files on your board is by installing the latest version of PlatformDetect from PyPI using pip and then overwriting the files. You can install the latest version by typing:

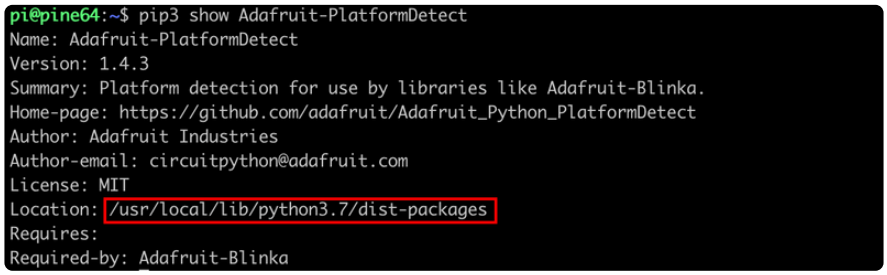
```
pip3 install --upgrade Adafruit-PlatformDetect
```

Keep in mind that if you have already installed PlatformDetect, this will upgrade it to the latest version if a newer one is available.



After installing, you can see where the files are located by typing the following command:

```
pip3 show Adafruit-PlatformDetect
```



In this case, the files were located in the **/usr/local/lib/python3.7/dist-packages** folder. You can just use the **cp** command to overwrite them. For instance, if you uploaded your modified files to a folder named adafruit_platformdetect inside your home directory and your files were in the same location as ours, you could just type:

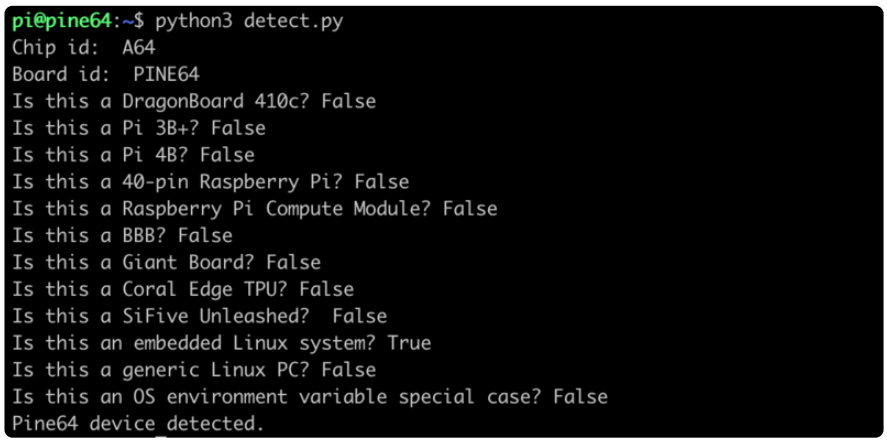
```
sudo cp -R ~/Adafruit_Python_PlatformDetect/adafruit_platformdetect /usr/local/lib/python3.7/dist-packages
```

The **-R** flag will recursively copy the folder and all of the files over.

Once you have everything copied over, make sure you are in the bin folder and then run the detect script by typing the following:

```
cd ~/Adafruit_Python_PlatformDetect/bin
python3 detect.py
```

This should output something like the following:



At the top, it displays the Chip ID and Board ID that it has detected and then it will check a few other boards to make sure that it isn't incorrectly detecting something

else. Once you have your board being correctly detected, you may want to add to this file so that your board is detected by others who want to add a new board in the future.

Next Steps

Now that you have your board being correctly detected, the next step is to create a Pull Request to the Adafruit_Python_PlatformDetect (https://adafru.it/Dyb) repo. If you are not sure how to create a Pull Request, check out this page from the GitHub guide (https://adafru.it/EvC) we mentioned earlier. You'll want to be sure to run Black (https://adafru.it/MF4) to format your code as well as Pylint (https://adafru.it/BI5) so that it passes the automated code check.

This guide just covers the first part of adding a new board to Blinka, which is correctly detecting the board and chip so that Blinka knows what to do. The second part of adding a board to Blinka is covered in the guide Adding a Single Board Computer to Blinka (https://adafru.it/KEF).