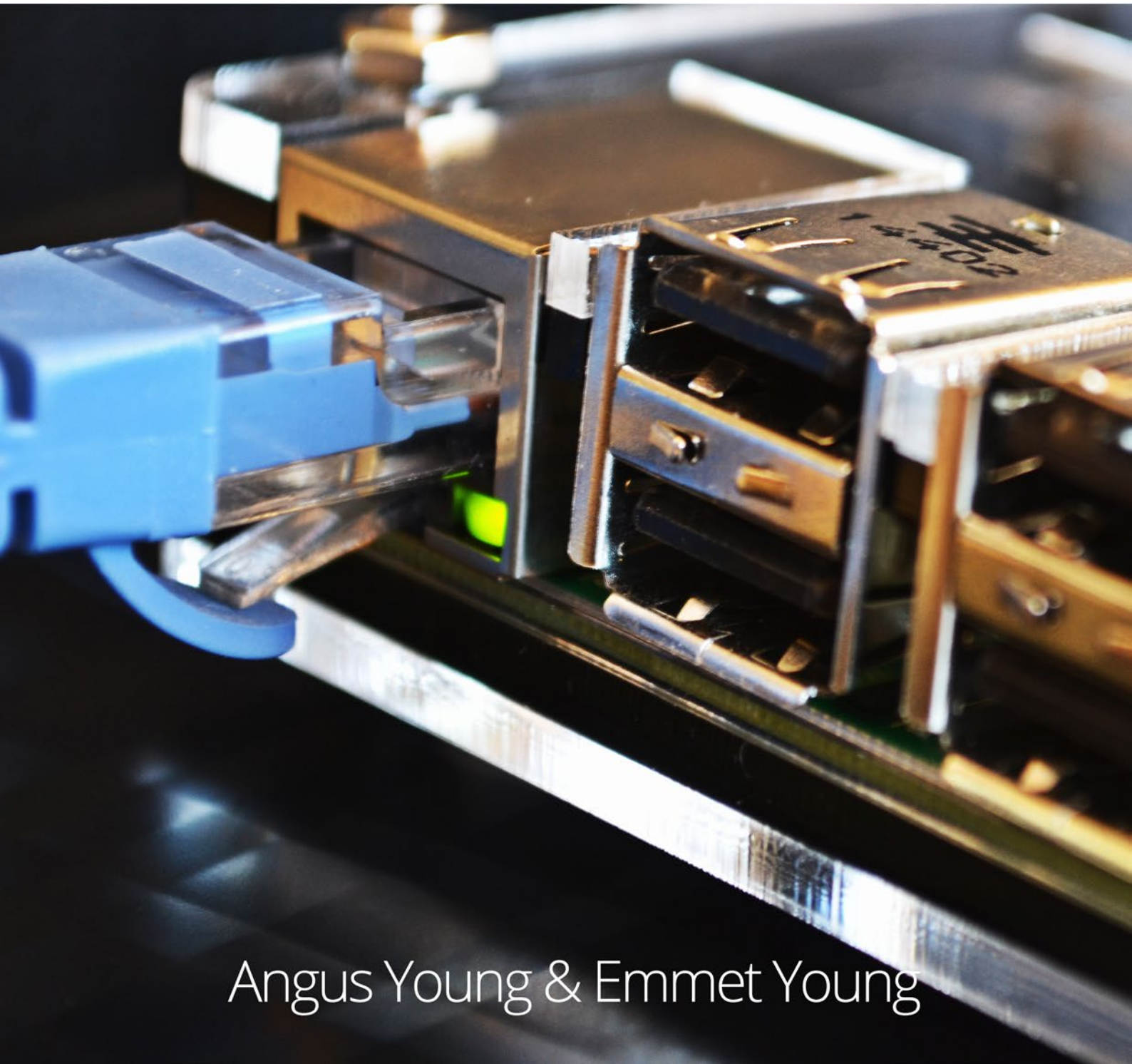
Pi My Life Up's **Advanced Projects** For the Raspberry Pi



Angus Young & Emmet Young

PiMyLifeUp

For permission contact Pi My Life Up at:

support@pimylifeup.com

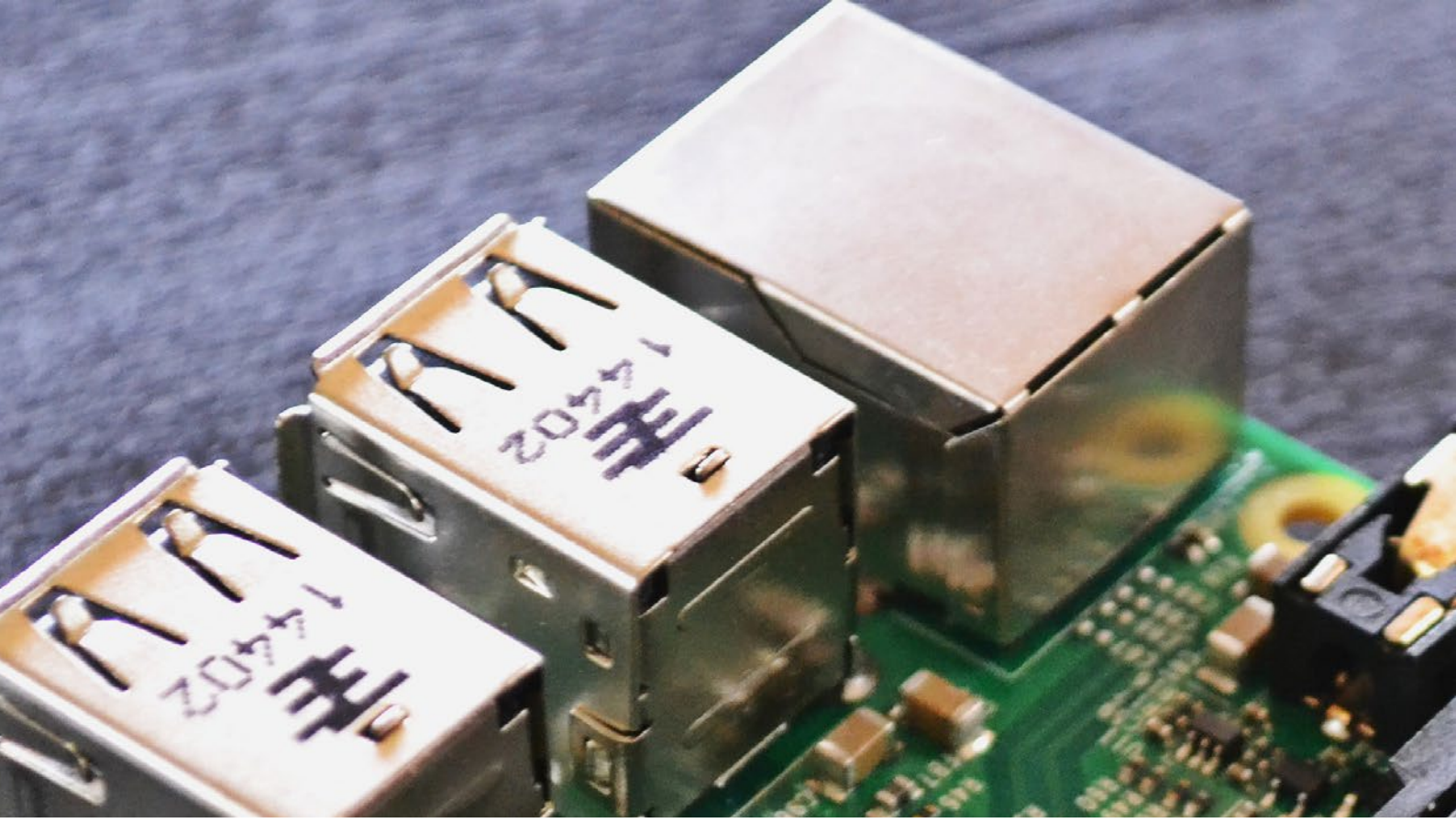
The tutorials in this book are for general information purpose only.

While we strive to keep our information up to date, we do not make any warranties about the completeness, reliability, and accuracy of this information.

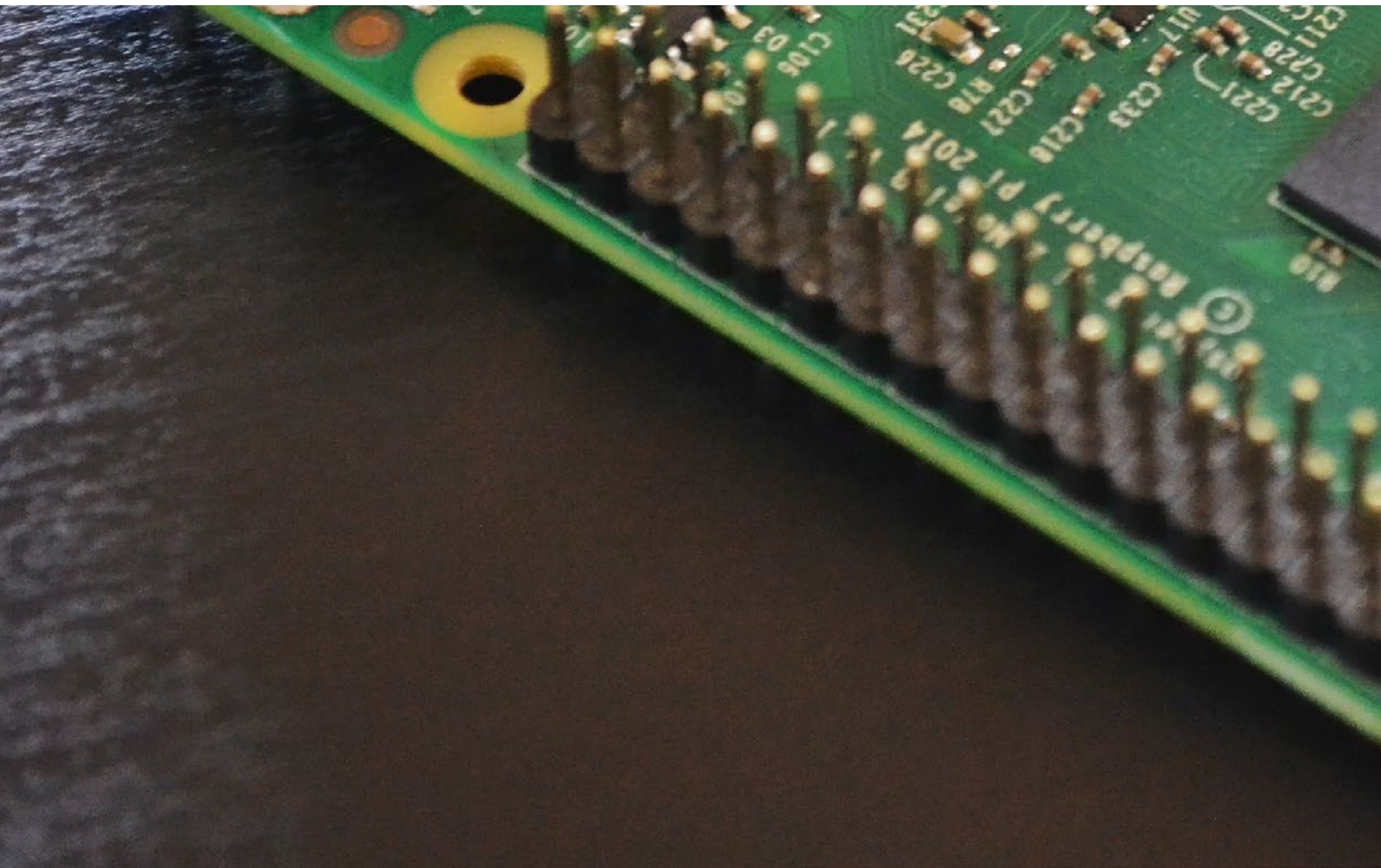
Any action you take upon the information on within this book is strictly at your own risk.

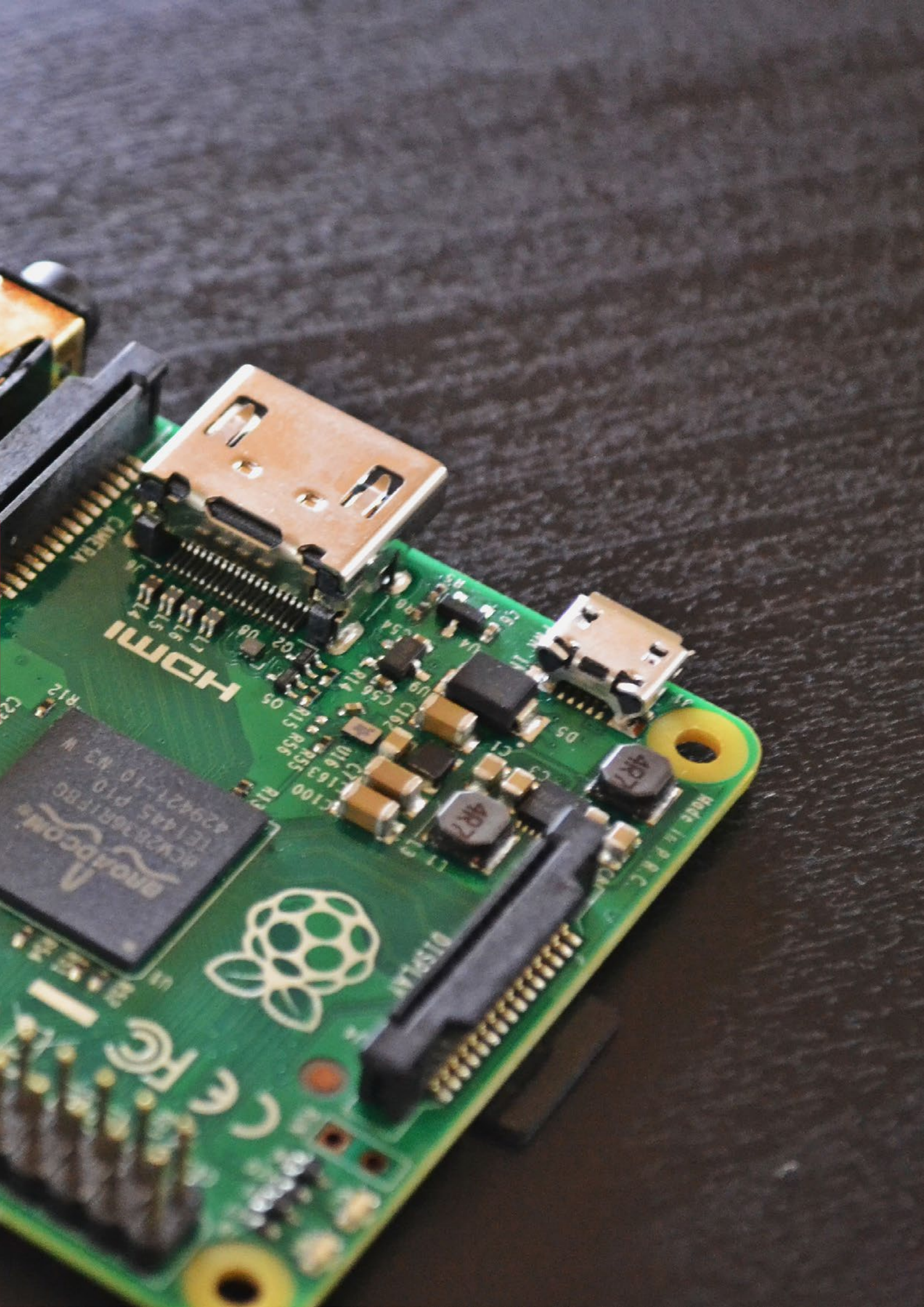
We will not be liable for any losses and damages in connection with the use of the information within this book

Contents



Advanced Projects




HDMI
CAMERA
DISPLAY

Using the PlayStation Buzz Controller on the Pi

Project Description

In this project, we will be showing you how you can utilize a PlayStation Buzz controller with your Raspberry Pi to run a simple quiz game.

Throughout this tutorial we will walk you through the process of interacting with the Buzz controllers by utilizing the hidapi library for Python. This hidapi library will allow us to communicate directly with the Buzz controllers, including retrieving data back from them and writing data to them to switch on and off their red LEDs.

By the end of this tutorial you will of learned how to read back information about all your USB devices, learnt how to read data that the devices are sending back and also learn how to write a library that will act as a simple wrapper to dealing with the device.

All of this will conclude in writing a simple quiz game that will make use of all the functionality that we program into our Buzz Controller library.

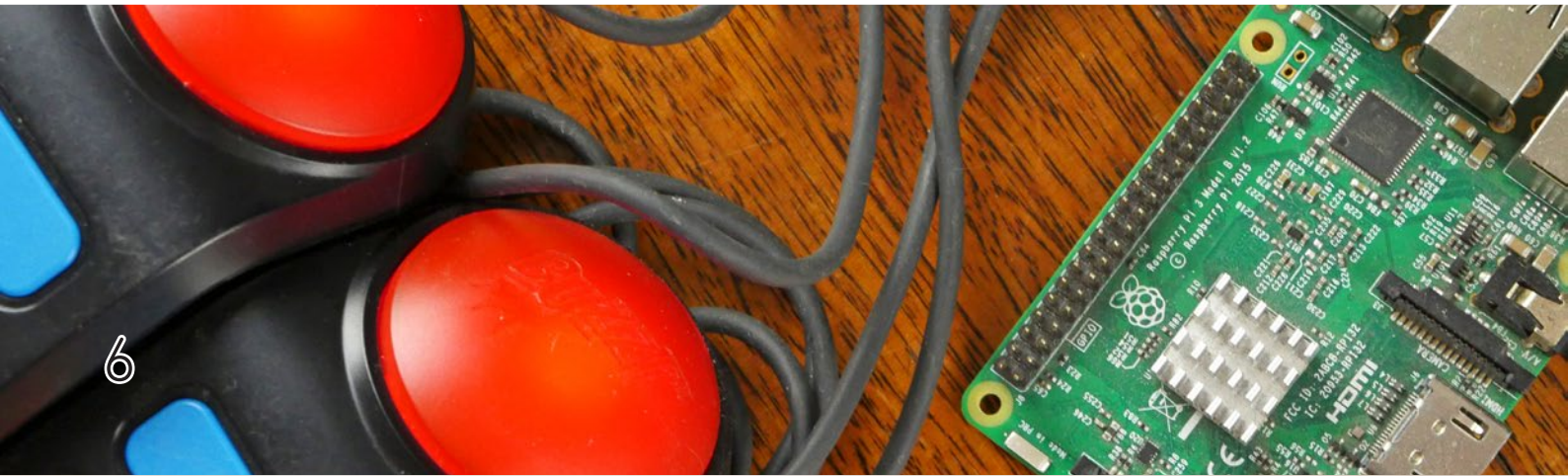
Equipment

Required

- Raspberry Pi (Reccomended to use a Raspberry Pi 3)
- SD Card (8 GB+ Recommended)
- Buzz Controllers

Optional

- Case
- Network connectio



Getting your Pi ready to interact with a Buzz Controller

1. Before we get started, we need first to ensure that the Raspbian operating system is entirely up to date.

We can do this by running the following two commands on the Raspberry Pi.

```
sudo apt-get update
sudo apt-get upgrade
```

2. We will now install some packages that we need for the hidapi package that we will be installing later on in the tutorial. These packages contain the code for accessing USB devices.

Run the command below to install the required packages

```
sudo apt-get install python-dev libusb-1.0-0-dev libudev-dev
```

3. Before we go installing the Python packages we need, let's make sure we are running the latest version of the Python setuptools by running the following command.

```
sudo pip install --upgrade setuptools
```

4. With our Raspberry Pi now up to date let's go ahead and use "pip" to install a particular package named Cython.

For those who do not know what Cython is, it is a version of Python that is compiled to try and achieve C like performance. We mainly need this for a package that we will be using to interface with the USB device.

Now run the following code to install Cython through pip on your Raspberry Pi.

```
sudo pip install cython
```

Please note that Cython can take considerable time to install, so now is a good time to grab a coffee.

5. Let's now go ahead and install the final Python package that we need, that being the hidapi package.

This hidapi package requires Cython that we installed in the previous step. Cython allows the hidapi code to get near C-like performance meaning USB interactions can happen speedily.

Install this package by running the following command on your Raspberry Pi.

```
sudo pip install hidapi
```

6. Now that we have all the packages installed to the system that we require we can now proceed on to our next section.

The next section will go into how we can read data back from our buzz controller and get an idea of the values we can expect to have to deal with.

If you would prefer to go straight into writing the buzz controller library and the quiz game logic you can skip to the end of the next section.

Finding your Buzz Controllers through Python

1. In this section, we will be exploring how we can retrieve the raw data from the Buzz Controllers.

To do this, we will be writing two small Python scripts that utilize the hidapi package that we installed earlier.

One of these scripts will enumerate all connected USB devices so we can pick out our Buzz Controller and grab its vendor id and product id.

The second script that we are writing will read back the raw data from the buzz controller by accessing it through the vendor id and product id that we retrieve from the enumeration.

Let's begin writing our enumeration script by running the following command on our Raspberry Pi.

```
sudo nano enumerate.py
```

2. Within this file write the following lines of code. We will explain each vital line of code to give you an understanding of this all works.

```
#!/usr/bin/env python
```

This line tells the command line interpreter that it should use the environment path to find and utilize python to run the following script.

It's always handy to have the shebang (**#!**) line at the start of a script as it makes it easier to run without having to specify the program to run it every time.

```
from __future__ import print_function

import hid
```

The first line here imports the print function from Python 3 into our current version of Python allowing us to utilize it within our code. This is due to that the print functionality was only just implemented in Python 3, the **__future__** feature enables us to forward that to Python 2.

Our second library imports the hid library that we installed earlier on in this tutorial, this will allow us to interact with our USB devices, and for this script will let us enumerate all available USB devices.

```
for d in hid.enumerate():
    keys = list(d.keys())
    keys.sort()
    for key in keys:
        print("%s : %s" % (key, d[key]))
```

This block of code is the critical part of our short script. It will retrieve an enumeration of all the currently available USB devices by utilizing the USB library.

It will then loop through all the devices provided to it from the **hid** library, creating a list of the keys from each device.

We then sort these keys so that they should print out in a better format.

Finally, we loop through all the keys provided to us and print them back out. The print out will allow us to find some crucial information about our device.

Finding your Buzz Controllers through Python

3. Once you have finished writing out the code, it should look like what we have displayed below.

If you are happy that the code is correct, save the file by pressing **CTRL + X** then **Y** and finally **ENTER**.

```
#!/usr/bin/env python
from __future__ import print_function

import hid

for d in hid.enumerate():
    keys = list(d.keys())
    keys.sort()
    for key in keys:
        print("%s : %s" % (key, d[key]))
```

4. Before we run the Python script that we just wrote you will need to first to plug your Buzz Controllers into your Raspberry Pi.

We also recommend you take out any other USB devices to make it easier to find the details for your Buzz Controller.

With your Buzz Controller plugged into your Raspberry Pi you can now run the script by running the following command, make sure you use sudo for this as it needs access to the USB interface.

```
sudo python enumerate.py
```

5. From that script you should see an output similar to what we have showcased below, you should see the manufacturer appear as "**Logitech**" and the product_string as "**Logitech Buzz(tm) controller**".

The two values you want to pay attention to here is both the **product_id** and the **vendor_id** as we will need those in the next section to begin talking with the device.

```
interface_number : 0
manufacturer_string : Logitech
path : 0001:0004:00
product_id : 2
product_string : Logitech Buzz(tm) Controller V1
release_number : 4353
serial_number :
usage : 0
usage_page : 0
vendor_id : 1356
```

6. In the next section, we will be showing you how you can use both the **vendor_id** and the **product_id** to communicate with the Buzz Controllers.

If the **vendor_id** and **product_id** differ to the values, we have in the example above make sure you write them down.

Retrieving data from the Buzz Controllers through Python

1. Now thank's to retrieving the **product_id** and the **vendor_id** from the previous section we can now move onto learning how to read data back from the buzz controller.

Let's begin by writing the script that we will use to read back all the data from the buzz controller by running the following command on the Raspberry Pi.

```
sudo nano read_controller.py
```

2. Within this file enter the following lines of code, we will explain the lines that are important as we go along.

```
#!/usr/bin/env python
```

This line is called a shebang (the **#!** character sequence in, particular, is the **shebang**), the command line interpreter reads this and uses the specified application to interpret the file.

```
import hid
import time
```

Here we are importing the two significant packages we need to get our script up and running and to be able to read from our Buzz Controller.

The first package is called hid and is utilized so that we can interact with USB devices easier through Python. This package will allow us to retrieve input data from the buzz controller.

In addition to the **hid** package we are also importing the time package. We utilize the **time** package so we can put the python script to sleep for a half a second to stop flooding the command line with messages.

```
h = hid.device()

h.open(0x54c, 0x002)

h.set_nonblocking(1)
```

We begin by creating an object from the **hid** library that we imported earlier in the tutorial, this will allow us to access the library and open up an actual USB device using the **vendor_id** and **product_id** that we got earlier in this tutorial.

Now using the **h.open** function we can specify both the **vendor_id** and **product_id** and the library will attempt to make a connection to the device with those specific values.

If you have noticed in our example, we are using the hexadecimal versions of the numbers that we retrieved with **0x54c** being equal to **1356** and **0x002** being equal to **2**.

Once the connection has been successfully made we proceed to enable non-blocking mode for it, this is important to keep the script running smoothly and not having to pause for reads.

Non-blocking mode makes it so that any read calls that do not return immediately will return a value of 0 rather than waiting on data to be returned.

```
while True:
    d = h.read(5)
    if d:
        print(d)
```

In this block of code, we run a while loop that runs infinitely.

On each loop of the while loop we read 5 bytes of data back from the device, if we have data, we print it out. Otherwise, we continue with the loop until the user terminates the script or data is retrieved.

While we should technically run **h.close()** after the loop, there is no way it can reach that thanks to our infinite loop.

3. Once you have finished typing out all the code, it should look something like what we have displayed below.

Once you are happy you have correctly entered the code save the file by pressing **CTRL + X** then **Y** and finally **ENTER**

```
#!/usr/bin/env python
import hid
import time

h = hid.device()

h.open(0x54c, 0x002)

h.set_nonblocking(1)

while True:
    d = h.read(5)
    if d:
        print(d)
    time.sleep(0.5)
```

4. Now while your Buzz controllers are plugged into your Raspberry Pi run the following command to run the script we just wrote. Make sure you use sudo as it is needed to access the hid interface.

```
sudo python read_controller.py
```

5. While running the script, you will notice when you press a button you will see a byte array containing 5 different values. You can ignore the first two of these as they will never change, the last 3 values contain our button presses.

For instance, if you press the red buzz button on the first controller, released it then pressed the blue button you will receive back the following data.

```
[127, 127, 1, 0, 240]
[127, 127, 0, 0, 240]
[127, 127, 16, 0, 240]
[127, 127, 0, 0, 240]
```

Retrieving data from the Buzz Controllers through Python

6. By pressing each button and making a note of the values and the position that it appears in we can work out the values we need to look out for when detecting buttons in Python.

If you write down all the values, you should end up with a table as we have below. We will use this table to work out what we need to utilize to detect different each controllers button presses.

For the last position, we subtract 240 from any result to get the true value as 240 is the resting value. For example, if we get the value 241, we would subtract 240 from that number to get the value of 1.

Buttons	Controller 1	Controller 2	Controller 3	Controller 4
Red	Pos 3 Value 1	Pos 3 Value 32	Pos 4 Value 4	Pos 4 Value 128
Yellow	Pos 3 Value 2	Pos 3 Value 64	Pos 4 Value 8	Pos 5 Value 1
Green	Pos 3 Value 4	Pos 3 Value 128	Pos 4 Value 16	Pos 5 Value 2
Orange	Pos 3 Value 8	Pos 4 Value 1	Pos 4 Value 32	Pos 5 Value 4
Blue	Pos 3 Value 16	Pos 4 Value 2	Pos 4 Value 64	Pos 5 Value 8

7. Now that we have this data table we will move onto the next section of our tutorial.

In the next section, we will begin writing our library that will provide an easily usable point to access the buzz controllers button presses and also will touch on turning on and off the Buzz Controllers LEDs.

Writing a Python library for the Buzz Controller

1. In this section, we will be writing a library that will interact with the buzz controller from our Pi.

This library will make it easier for us to program our small quiz game later on in the tutorial and allow us to easily modify and improve the way we talk with the buzz controllers later on.

To begin writing our Python script that will act as our library run the following command on your Pi.

```
nano BuzzController.py
```

2. Within this file enter the following lines of code.

As we go along, we will explain each important section of the code to try and give you an idea on how everything works so you can modify it for your use.

```
import hid
import time
```

We have explained these imports several times throughout this tutorial so we will go into the basics of what we use them for now.

We import the hid library so that we can access the USB interface and in turn talk with the Buzz Controllers.

Then we import the time library so that we can utilize it to pause the script for any reason, in particular, we will be using this for a controller blink functionality.

```
class BuzzController:
```

This line defines our class name and that all the code below it that is tabbed in is a part of that class. We will be able to access the variables and functions from this class by referencing it from its name "**BuzzController**".

You will see examples of how we access this class later on in the tutorial.

```
    light_array     = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
```

We define a variable named **light_array** that contains an **8-byte** array all set to **0x00**. We define this so that we can keep track of what LED's should be switched on or off at any moment in time.

```
    light_blinking = False
```

This **light_blinking** variable is used to keep track of whether we should be continuing our light blinking loop or not. The only way for it to exit out of its loop is that if this variable is set to False.

```
    buttonState = [
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False}
    ]
```

Our final major variable is our **buttonState** array, this array contains four sets, with each set representing a controller.

Each set then contains a variable for each possible button which can either be **True** for pressed or **False** for not pressed.

```
def __init__(self):
    #instantiate the device class
    self.hid = hid.device()

    #Open up the device
    self.hid.open(0x54c, 0x02)

    #Set the non blocking mode
    self.hid.set_nonblocking(1)

    #Clear the Buzz Controller LEDs
    self.hid.write(self.light_array)
```

Here we define our **__init__** function. This function is called by Python when we initialize our class.

In this function, we set up everything we need to read and write to the buzz controller. We first initialize the hid class, then we make a connection to the buzz controller using the details we retrieved earlier.

Then we proceed to set the nonblocking mode so that reads are returned instantly regardless of whether there is any data to read.

Finally, we write some data to our device. This data is just our blanked our light array, this is to ensure all of the Buzz Controllers LED's are switched off.

```
def light_blink(self, controller):
    blink_lights_off = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
    self.blink_lights_on = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

    for i in controller:
        self.blink_lights_on[i + 2] = 0xFF

    if (not self.light_blinking):
        self.light_blinking = True
        blink = True
        while self.light_blinking:
            if (blink):
                self.hid.write(self.blink_lights_on)
            else:
                self.hid.write(blink_lights_off)
            blink = not blink
            time.sleep(0.5)

        self.hid.write(self.light_array)
```

Here we define our **light_blink** function. This function will run a loop that will continually switch the LED on and off on our specified controllers.

This code works by first looping through the specified controllers that are handed to it, for each controller we specify we change a value in our **blink_lights_on** array from **0x00** (Off) to **0xFF** (On).

Next, we go into our loop only if we aren't already running it, and toggle our **blink** variable to know whether we need to write **blink_lights_on** or our **blink_lights_off** to the buzz controller.

```
def get_button_status(self):
    data = self.hid.read(5)
    if data:
        self.buttonState[0]["red"] = ((data[2] & 0x01) != 0) #red
        self.buttonState[0]["yellow"] = ((data[2] & 0x02) != 0) #yellow
        self.buttonState[0]["green"] = ((data[2] & 0x04) != 0) #green
        self.buttonState[0]["orange"] = ((data[2] & 0x08) != 0) #orange
        self.buttonState[0]["blue"] = ((data[2] & 0x10) != 0) #blue

        self.buttonState[1]["red"] = ((data[2] & 0x20) != 0) #red
        self.buttonState[1]["yellow"] = ((data[2] & 0x40) != 0) #yellow
        self.buttonState[1]["green"] = ((data[2] & 0x80) != 0) #green
        self.buttonState[1]["orange"] = ((data[3] & 0x01) != 0) #orange
        self.buttonState[1]["blue"] = ((data[3] & 0x02) != 0) #blue

        self.buttonState[2]["red"] = ((data[3] & 0x04) != 0) #red
        self.buttonState[2]["yellow"] = ((data[3] & 0x08) != 0) #yellow
        self.buttonState[2]["green"] = ((data[3] & 0x10) != 0) #green
        self.buttonState[2]["orange"] = ((data[3] & 0x20) != 0) #orange
        self.buttonState[2]["blue"] = ((data[3] & 0x40) != 0) #blue

        self.buttonState[3]["red"] = ((data[3] & 0x80) != 0) #red
        self.buttonState[3]["yellow"] = ((data[4] & 0x01) != 0) #yellow
        self.buttonState[3]["green"] = ((data[4] & 0x02) != 0) #green
        self.buttonState[3]["orange"] = ((data[4] & 0x04) != 0) #orange
        self.buttonState[3]["blue"] = ((data[4] & 0x08) != 0) #blue
    return self.buttonState
```

Here is one of our most important functions, this function reads the button data from the buzz controller. Every read we make from the controller will be 5 bytes as that is all we require to know the current state of all the controller's buttons.

We then proceed to interpret the data that we just read in from the buzz controller. To interpret this date, we will be utilizing information that we wrote down in a table in the previous section of this tutorial.

If you take a look at our handling of the first controller (**self.buttonState[0]**), you will notice that we are using the array position from the table (**data[2]** is the 3rd position in the arrays we printed out), with each button also using the value we obtained from that position in the array (e.g., **data[2] & 0x01**)

To this date value that we grabbed in the last section we utilize a bitwise **and** (**&**) operator, if the result of the data value and the bitwise operator is not equal to 0 then we set the value to True, but if it does, we set the value for that button to False.

Simply put the bitwise "**and**" operation ensures that value is actually within the returned data, by comparing the individual bits of the number. This is, of course, a bit more complicated than that but for now, we will stick with the simple explanation

We then proceed to handle the read-in data for all four controllers. Uou should note with each of them that we are using all the data that we retrieved in the previous section.

```
def get_button_pressed(self, controller):
    buttons = self.get_button_status()
    for key, value in buttons[controller].items():
        if (value):
            return key
```

The **get_button_pressed** function acts as a bit of a helper for interpreting the data we retrieve back using the **get_button_status** function we wrote earlier.

Within this function, we retrieve the result from the **get_button_status** function and loop through it for the values of the specified controller.

Once we have found a value that is not False or empty we return the data immediately otherwise if nothing is found we return nothing at all.

```
def controller_get_first_pressed(self, buzzButton, controllers = [0, 1, 2, 3]):
    while True:
        buttons = self.get_button_status()
        for i in controllers:
            if (buttons[i][buzzButton]):
                return i
```

The **controller_get_first_pressed** function is very much like the **get_button_pressed** function.

Instead of reporting back the buttons pressed by a particular controller the function listens to the specified controllers to see if they have pressed a specified button.

It will wait until it receives a response, once it does it returns the id of the controller that pressed the specified button.

You will see how this can be used when we write the quiz game in the next section of the tutorial.

```
def light_blink_stop(self):
    self.light_blinking = False
```

The **light_blink_stop** function changes the **light_blinking** variable to **False** which stops the loop started by the **light_blink** function. This function needs to be run before **light_blink** can be used again.

```
def light_set(self, controller, status):
    self.light_array[controller+2] = 0xFF if status else 0x00
    self.hid.write(self.light_array)
```

The **light_set** function takes a specified controller and sets its value in our **light_array** to either **0xFF** or **0x00** depending on the value of **status**.

Once we set the value within **light_array**, we proceed to write that data back to the controller.

Writing a python library for the Buzz Controller

3. Once you have written all the code, you can verify that you have entered in everything correctly by looking at our code shown on the page below and over onto the next page.

Once you are happy that the code is correct, you can save the file by pressing Ctrl + X then Y and finally ENTER.

```
import hid
import time

class BuzzController:
  light_array    = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
  light_blinking = False
  buttonState = [
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False},
      {"red": False, "blue": False, "orange": False, "green": False, "yellow": False}
   ]

  def __init__(self):
      #instantiate the device class
      self.hid = hid.device()

      #Open up the device
      self.hid.open(0x54c, 0x02)

      #Set the non blocking mode
      self.hid.set_nonblocking(1)

      #Clear the Buzz Controller LEDs
      self.hid.write(self.light_array)

  def light_blink(self, controller):
      blink_lights_off = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]
      self.blink_lights_on = [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

      for i in controller:
         self.blink_lights_on[i + 2] = 0xFF

      if (not self.light_blinking):
         self.light_blinking = True
         blink = True
         while self.light_blinking:
            if (blink):
               self.hid.write(self.blink_lights_on)
            else:
               self.hid.write(blink_lights_off)
            blink = not blink
            time.sleep(0.5)

         self.hid.write(self.light_array)
```

```
    def get_button_status(self):
        data = self.hid.read(5)
        if data:
            self.buttonState[0]["red"] = ((data[2] & 0x01) != 0) #red
            self.buttonState[0]["yellow"] = ((data[2] & 0x02) != 0) #yellow
            self.buttonState[0]["green"] = ((data[2] & 0x04) != 0) #green
            self.buttonState[0]["orange"] = ((data[2] & 0x08) != 0) #orange
            self.buttonState[0]["blue"] = ((data[2] & 0x10) != 0) #blue

            self.buttonState[1]["red"] = ((data[2] & 0x20) != 0) #red
            self.buttonState[1]["yellow"] = ((data[2] & 0x40) != 0) #yellow
            self.buttonState[1]["green"] = ((data[2] & 0x80) != 0) #green
            self.buttonState[1]["orange"] = ((data[3] & 0x01) != 0) #orange
            self.buttonState[1]["blue"] = ((data[3] & 0x02) != 0) #blue

            self.buttonState[2]["red"] = ((data[3] & 0x04) != 0) #red
            self.buttonState[2]["yellow"] = ((data[3] & 0x08) != 0) #yellow
            self.buttonState[2]["green"] = ((data[3] & 0x10) != 0) #green
            self.buttonState[2]["orange"] = ((data[3] & 0x20) != 0) #orange
            self.buttonState[2]["blue"] = ((data[3] & 0x40) != 0) #blue

            self.buttonState[3]["red"] = ((data[3] & 0x80) != 0) #red
            self.buttonState[3]["yellow"] = ((data[4] & 0x01) != 0) #yellow
            self.buttonState[3]["green"] = ((data[4] & 0x02) != 0) #green
            self.buttonState[3]["orange"] = ((data[4] & 0x04) != 0) #orange
            self.buttonState[3]["blue"] = ((data[4] & 0x08) != 0) #blue
        return self.buttonState
    def get_button_pressed(self, controller):
        buttons = self.get_button_status()
        for key, value in buttons[controller].items():
            if (value):
                return key
    def controller_get_first_pressed(self, buzzButton, controllers = [0, 1, 2, 3]):
        while True:
            buttons = self.get_button_status()
            for i in controllers:
                if (buttons[i][buzzButton]):
                    return i
    def light_blink_stop(self):
        self.light_blinking = False
    def light_set(self, controller, status):
        self.light_array[controller+2] = 0xFF if status else 0x00
        self.hid.write(self.light_array)
```

4. Now that we have finished writing our Buzz Controller library we will move onto showing you how you can use this library in practice.

To show the library off we will be writing a short quiz game that implements the controller library. This little game will read questions in from a set, randomize the buttons the answers are assigned to and implement a simple scoring system.

Utilizing our Buzz Controller library in a quiz game

1. In this final section of our Buzz Controller tutorial, we will be showing you how to put everything that we have done over the past two sections together to create a simple quiz game.

To begin writing the quiz game script run the following command on your Raspberry Pi.

```
nano quiz_game.py
```

2. Within this file, you will need to enter the following lines of code, after each important block of code we will try and explain how it works.

```
from __future__ import print_function
```

We have used this import a fair bit throughout this tutorial. It allows us to use the print function in Python 2 despite it being a feature of Python 3.

```
import BuzzController
```

Here we import the library that we just wrote. This import will allow us to access the **buzzcontroller** class we created in the previous section.

```
import time
```

The time library is imported just so that we can pause the script for a fraction of a second.

```
import thread
```

We utilize the thread library so that we can start blinking the controllers LED's without holding up the entire script by starting it in its own separate thread.

```
from random import shuffle
```

Our last import for the script loads in the shuffle functionality, this will allow us to shake up the possible answers so that the correct answer is not always assigned to the same button.

```
import_questions = [
    {"question":"What is the capital of Australia", "answers" : ["Canberra", "Sydney", "Hobart", "Melbourne"]},
    {"question":"What is the capital of Japan", "answers" : ["Tokyo", "Horoshima", "Osaka", "Kyoto"]},
]
questions = []
score = [0, 0, 0, 0]
```

Here we start by defining our **import_questions** array, each element of this array will be a new question and set of answers.

Each set needs to have "**question**" and a "**answers**" array defined for it. For the "**answers**" array the first element must be the correct answer with the other three answers meant to be an incorrect alternative.

For example in our first question, we define "**question**" as "**What is the capital of Australia**" and set the "**answers**" variable to **[**"**Canberra**", "**Sydney**", "**Hobart**", "**Melbourne**"**]** with "**Canberra**" being the correct answer to the question.

We then define an empty array named questions, to this array we will append a processed version of our **import_questions** array. These prepared questions will make it easier for game logic to operate.

Finally, we create an array named **score** that will keep track of all the user's correct answers.

```
    for question in import_questions:
        buttons = ["blue", "orange", "green", "yellow"]
        new_answer = {}
```

This block of code loops through our imported questions and converts it to something more manageable for our program to handle.

We start by defining an array of possible buttons, these being, blue, orange, green, yellow.
Then we define an empty set that will contain our modified question.

```
        shuffle(buttons)
        new_answer['question'] = question['question']
```

After we define our buttons array, we finally put the shuffle library to use by utilizing it to shuffle our buttons array.

By shuffling the array, we ensure that the correct answer can be set to any of the possible buttons and remove the ability for someone to memorize the right button for each answer.

Once we have shuffled the array, we then proceed to copy our actual question to the new_answer variable. Our quiz game logic will output this question alongside the possible answers.

```
        for i in range(4):
            if i == 0:
                new_answer["correct"] = buttons[i]
            new_answer[buttons[i]] = question["answers"][i]
        questions.append(new_answer)
```

Finally, we utilize a for loop to loop through our **imported_questions** '**answers**' array.

As we know that the first element in the array is the correct answer, we check if we are on the first loop then set the "**correct**" element of the **new_answer** set to the button value for position 0 in the array.

Thanks to our shuffling earlier this value could either be, "**blue**", "**orange**", "**green**" or "**yellow**".

We then again assign another value to the **new_answer** set, this time we set the answer value to the button value for that iteration. For example, **new_questions["blue"]** would have a specific answer set to it.

We continue to do this until all four possible buttons have an answer set to it.

Finally, we append our formatted **new_answer** value to our **questions** variable.

```
    buzz = BuzzController()
```

Here we fire up our buzz controller library by instantiating it to our buzz variable. By doing this, we are calling the **__init__** function that we defined within our library.

If you recall, the **__init__** function is what initializes the hid library and makes the connection with your Buzz Controller. This will allow us to access various elements of our BuzzController library in our game logic loop.

Utilizing our Buzz Controller library in a quiz game

```
    for question in questions:
        question_answered = False
        available_answers = ["Blue", "Orange", "Green", "Yellow"]
        available_controllers = [0, 1, 2, 3]
```

Here we begin a loop that goes through all our available questions. This loop can run for as long as you want as we are just relying on Pythons "**for**" to iterate through the array.

For each loop we set a few values, **question_answered** we utilize to know whether a question has been answered correctly.

Next, we set the **available_answers**, the value for this on every loop should be all possible buttons on the buzz controller. We use this variable to track what answers are still available to choose from.

Finally, we set our **available_controllers** array. This array contains the ids of each possible controller that is allowed to answer.

```
        while not question_answered:
            print(question["question"])

            for i in available_answers:
                print(i + " " + question[i.lower()])
```

Here we start another while loop. This loop will keep running until one of the four possible controllers gets the correct answer.

On each loop we print the question being asked to the user, then we proceed to loop through the available buttons and print out each of the possible answers alongside the buttons color.

```
            thread.start_new_thread(buzz.light_blink, (available_controllers,))
            controller = buzz.controller_get_first_pressed("red", available_controllers)
            buzz.light_blinking = False
```

In this code block, we utilize the thread library that we imported at the start of the script to start a new thread that runs our **light_blink** function from our buzz controller library. To this, we also pass in the available controllers.

What this does is allow only the users who can answer the questions controllers to flash, while not affecting the current scripts loop.

We then utilize the **controller_get_first_pressed** function to check for which of the available controllers is the first to press the flashing red Buzz button on their controller. We then store the returned controllers id in our controller variable.

Once a user has pressed their red button we then stop all the controllers from blinking continually by setting the **light_blinking** variable in the Buzz Controller library to **False**. In turn, this also ends the thread we spun up.

```
            buzz.light_set(controller, True)
            time.sleep(0.5)
```

Now that a controller has been chosen we switch the red light on their controller on to signal that it is their turn to answer the question.

Utilizing our Buzz Controller library in a quiz game

```
            while True:
                button = buzz.get_button_pressed(controller)
```

Now we enter yet another while loop. This loop will run until that controller either selects the correct answer or an incorrect one.

On every loop, we utilize our buzz controller libraries **get_button_pressed** function to retrieve any buttons being pressed by the specified controller.

```
            if button and button != "red":
                if button == question["correct"]:
                    print("Controller " + str(controller) + " was correct")
                    question_answered = True
                    score[controller] += 1
                    break
```

We then proceed to check if a button has been returned and ensure that it isn't the red button to stop the user accidentally pressing the red button too long and automatically triggering an "**Incorrect Answer**" response.

We check if the button the user pressed on the controller matches our correct answer button.

If the user selected the right button we first print our a message notifying them of this, then we set our **question_answered** variable to **True**, increase the controllers **score** by 1 and then utilize **break** to exit out of the current while loop.

```
            elif button.capitalize() in available_answers:
                print("Sorry incorrect answer")
                available_controllers.remove(controller)
                available_answers.remove(button.capitalize())
                break
```

If the user did not select the correct answer, we see if that button is in our **available_answers** array.

If the value is in the array we notify the user that they have selected a wrong answer, we then remove this controller from the **available_controllers** array, and also remove this answer from the **available_answers** array so another user cannot select it.

```
        buzz.light_set(controller, False)
    time.sleep(1)

print("Final score")
print(score)
```

Once the user has selected an answer, we go ahead and switch off their light by using the buzz controller libraries **light_set** function.

Then we sleep the script for a second before it continues looping through all available questions.

Now if the game has run out of questions to ask the users, it will print out the text "**Final Score**" followed by the score array that we set up earlier in the script.

Utilizing our Buzz Controller library in a quiz game

3. With our quiz game script now written, you can verify that you have entered the code correctly by looking over the next two pages.

If you are happy the code is correct you can save the file by pressing **CTRL + X** then **Y** and finally **ENTER**.

```
from __future__ import print_function
import BuzzController
import time
import thread
from random import shuffle

import_questions = [
    {"question":"What is the capital of Australia", "answers" : ["Canberra", "Sydney", "Hobart", "Melbourne"]},
    {"question":"What is the capital of Japan", "answers" : ["Tokyo", "Horoshima", "Osaka", "Kyoto"]},
]
questions = []
score = [0, 0, 0, 0]

for question in import_questions:
    buttons = ["blue", "orange", "green", "yellow"]
    new_answer = {}
    shuffle(buttons)
    new_answer['question'] = question['question']
    for i in range(4):
        if i == 0:
            new_answer["correct"] = buttons[i]
        new_answer[buttons[i]] = question["answers"][i]
    questions.append(new_answer)

buzz = BuzzController()

for question in questions:
    question_answered = False
    available_answers = ["Blue", "Orange", "Green", "Yellow"]
    available_controllers = [0, 1, 2, 3]

    while not question_answered:
        print(question["question"])

        for i in available_answers:
            print(i + " " + question[i.lower()])

        thread.start_new_thread(buzz.light_blink, (available_controllers,))
        controller = buzz.controller_get_first_pressed("red", available_controllers)
        buzz.light_blinking = False
        buzz.light_set(controller, True)
        time.sleep(0.5)

        while True:
            button = buzz.get_button_pressed(controller)
```

```
            if button and button != "red":
                if button == question["correct"]:
                    print("Controller " + str(controller) + " was correct")
                    question_answered = True
                    score[controller] += 1
                    break
                elif button.capitalize() in available_answers:
                    print("Sorry incorrect answer")
                    available_controllers.remove(controller)
                    available_answers.remove(button.capitalize())
                    break
        buzz.light_set(controller, False)
    time.sleep(1)

print("Final score")
print(score)
```

4. Now that we have written our script let's go ahead and start it by running the following command on your Raspberry Pi.

Remember to have your Buzz Controllers connected to your Raspberry Pi before you do this otherwise you will run into errors.

```
sudo python quiz_game.py
```

5. You should now be able to play through the quiz game if you were using the sample questions you should end up with a result somewhat like what we have shown below.

```
What is the capital of Australia
Blue Sydney
Orange Melbourne
Green Hobart
Yellow Canberra
Sorry incorrect answer
What is the capital of Australia
Blue Sydney
Green Hobart
Yellow Canberra
Controller 3 was correct
What is the capital of Japan
Blue Horoshima
Orange Kyoto
Green Osaka
Yellow Tokyo
Controller 2 was correct
Final score
[0, 0, 1, 1]
```

6. By now you should have the script running successfully and talking with your Buzz Controllers without any issues. Hopefully, you now have some understanding of how everything works and be able to expand on this yourself.

You can challenge yourself by expanding this script further, first off would be to make the questions imported through a text file, the other would be to give better feedback on scores and incorrect answers.


BUZZ!
Raspberry Pi 3 Model B V1.2
© Raspberry Pi 2015
GPIO
DISPLAY

References

Linux Cheat Sheet

File System

ls	Directory Listing
tree	Indented file/directory listing
cd	Change directory
pwd	Display current directory
mkdir newDir	Creates a directory
rmdir oldDir	Removes a directory
cp file newfilename	Copies a file to a new location
mv file.txt ./newDir	Moves a file to a new location
touch file	Sets the last modified timestamp
cat	List contents of a file
head file	Outputs first 10 lines of a file
tail file	Outputs last 10 lines of a file
chmod 777 file	CHMOD is used to alter the permissions of a file or files.
chmod u=rw file	You can use symbols or numbers depending on what you prefer
chown pi:root file	CHOWN will change the user and/or the group that owns a file.
unzip archive.zip	Unzip will extract the files and directories from a compressed zip archive.
tar -cvzf **tar** -xvzf	Compress and extract the contents of an archive in the tar format. -c to compress & -x to extract

Users Management

id	Get the id of a user or group
who	List users that are logged in
last	List of users recently logged in
groupadd name	Adds a new group
useradd name	Adds a new user
userdel name	Deletes a user & all files related to that user
deluser name	Deletes user with options to remove certain data
usermod	Modify a user account
passwd	Change your password or a password of another account

Process Management

ps	Show snapshot of current processes
top	Show real time processes
kill pid	Kill process with id pid
pkill name	Kill process with name
killall name	Kill processes with name

Networking

ping host	Pings a host
hostname	Hostname of the system
ifconfig	Shows network configuration details for the interfaces on current system
ssh	Connect to another computer using SSH
scp	Transfer files over SSH
wget {URL}	Downloads a file directly to the device
netstat -tupl	Active connections to/from device

Shortcuts

!!	Repeats last command
ctrl+z	Stops command, resume using **fg** for foreground or **bg** for background
ctrl+c	Halts the current command
ctrl+w	Deletes 1 word on the current line
ctrl+u	Deletes entire line
ctrl+r	Search previous commands
exit	log out of current session

General Commands

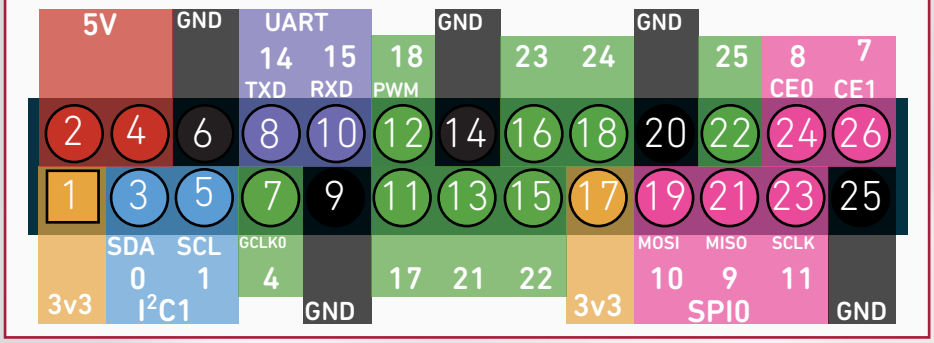
Man ls	Brings up the manual page for ls
ls **|** head	Pipes	allows you to direct output from one command into another
date	Shows system date
whoami	Shows your username
uptime	Shows uptime
uname -a	Show system & kernel

Searching

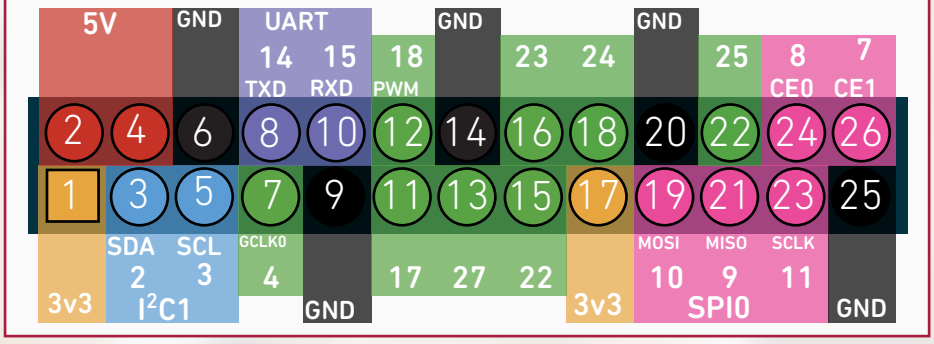
grep "search" *.txt	Search for a pattern inside files
find . -name 'help'	Will search for directories & files that match a pattern
whereis ls	Displays documentation, binaries & source files of a command

Raspberry Pi GPIO Pins

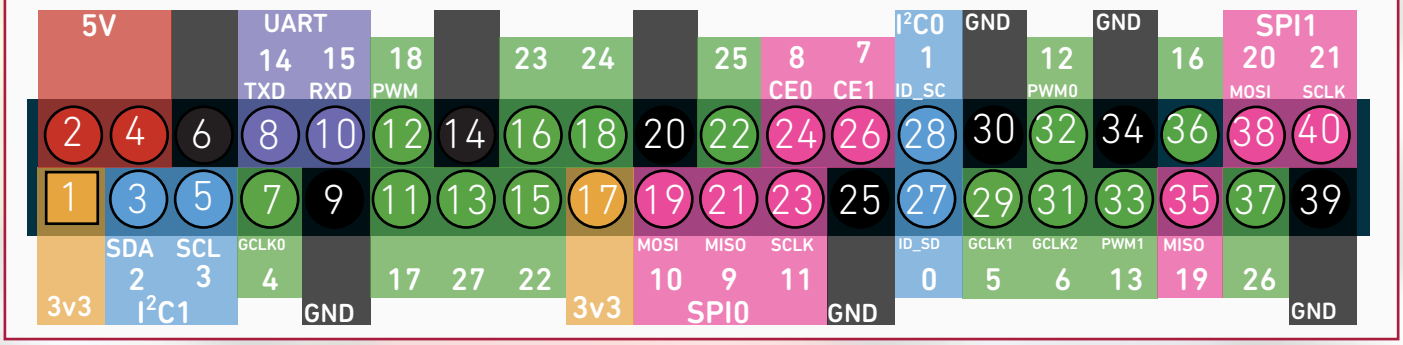
Raspberry Pi (Revision 1)



Raspberry Pi (Revision 2)



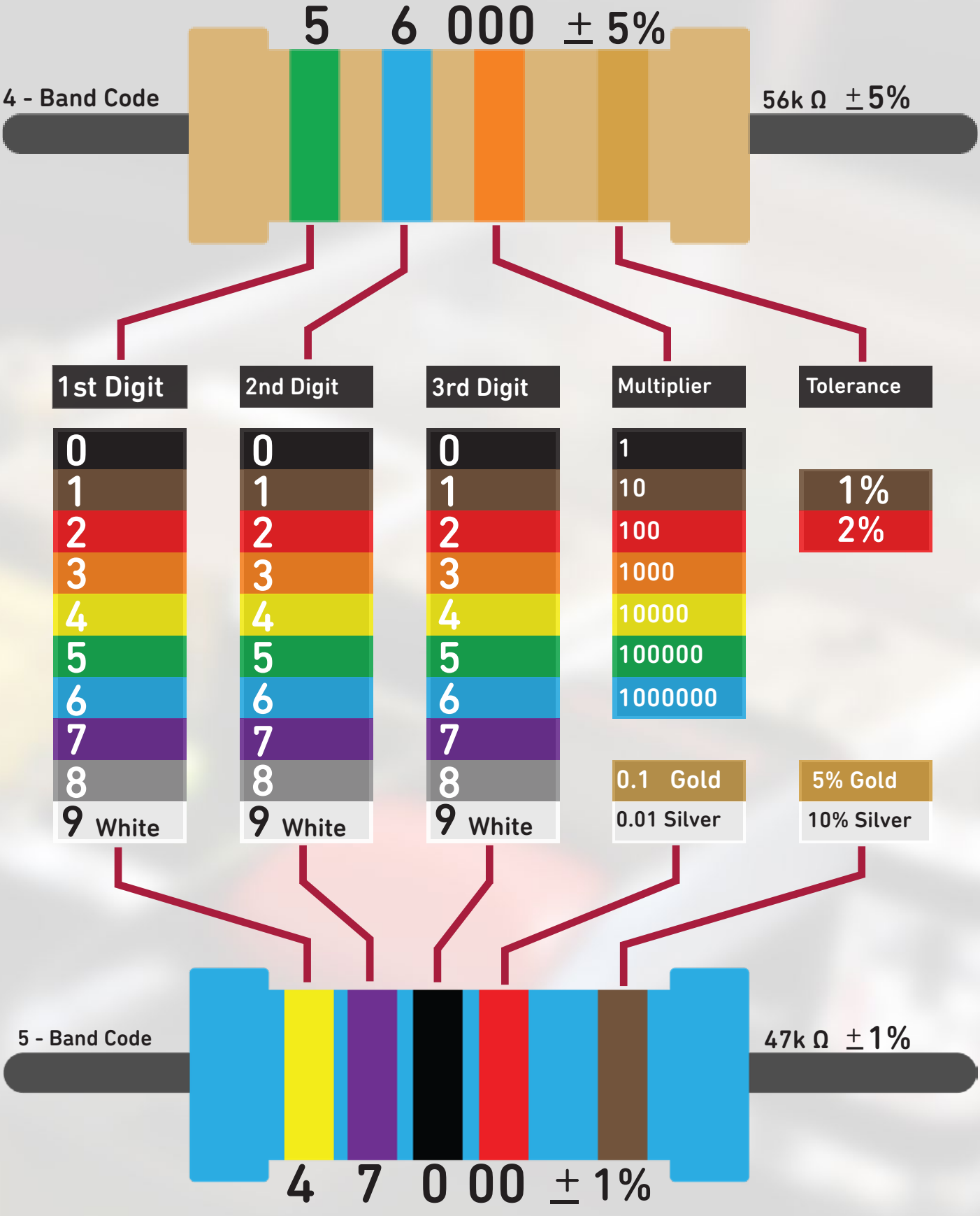
Raspberry Pi B+, 2, 3 & Zero



Key

- GPIO Pins
- Ground
- Serial Peripheral Interface (SPI)
- Universal asynchronous receiver/transmitter (UART)
- Inter-Intergrated Circuit (I²C)
- 5v Power Pins
- 3v3 Power Pins

Resistor Colour Guide



Voltage Divider Equation

Calculating the Voltage Out

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

Calculating the R2 Value

$$R_2 = \frac{V_{out} * R_1}{V_{in} - V_{out}}$$