e-ale-lca2019

Embedded Apprentice Linux Engineer LCA2019

Version 1.0

e-ale

The E-ALE (Embedded Apprentice Linux Engineer) is a series of seminars held at existing conferences covering topics which are fundamental to a Linux professional in the field of Embedded Linux.

This seminar will spend equal time on lecture and hands on labs at the end of each seminar which allow you to practice the material you've learned.

> This material makes the assumption that you have minimal experience with using Linux in general, and a basic understanding of general industry terms. The assumption is also made that you have access to your own computers upon which to practice this material.

More information can be found at **https://e-ale.org/**

Contents

Chapter 1

The Floral Bonnet

e-ale

1.1 The Floral Bonnet

Introduction to the Floral Bonnet

- Speaker: Behan Webster <behanw@converseincode.com>
- LCA2019 (2019.01.21)

Brought to you by

- Linux Foundation Training has provided speaker funding
- ARM is subsidizing the manufacturing of the floral bonnet for LCA2019

Raspberry-pi Zero

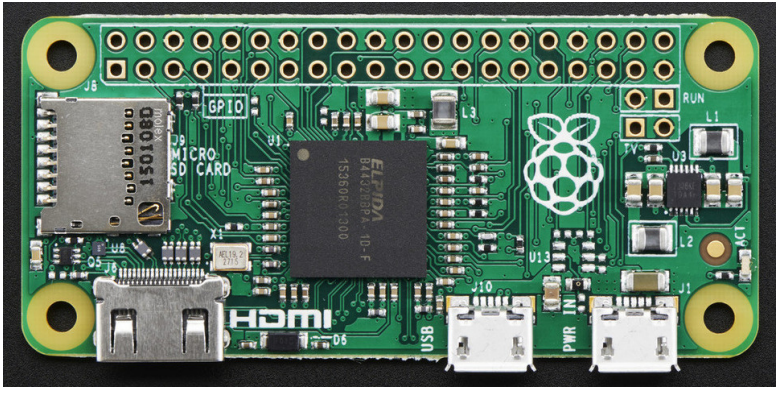


Figure 1.2: **Raspberry-pi Zero**

- Effectively a 65 mm by 30 mm version of the raspberry-pi 2
- 32-bit 1GHz single-core ARM CPU, 512MB RAM, etc

Raspberry-pi Zero Wireless

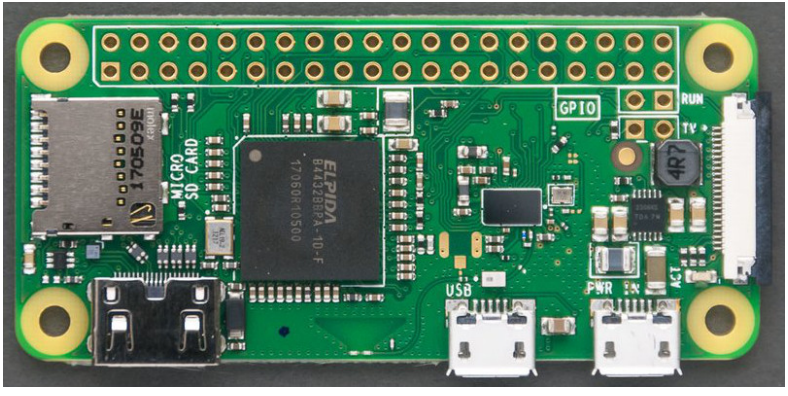


Figure 1.3: **Raspberry-pi Zero Wireless**

- The wireless version of the Zero with wifi and bluetooth

Raspberry-pi Zero Wireless with Headers

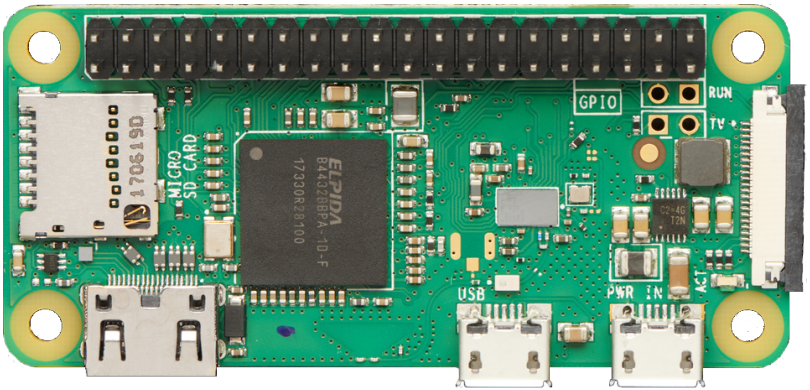


Figure 1.4: **Raspberry-pi Zero WH**

- We need the version with the header already installed for a bonnet

Raspberry-pi bonnet

- Expansion boards for the raspberry-pi are called **hats**
- Since the rpi-zero is smaller, expansion boards for the zero are called **bonnets**

The Floral Bonnet

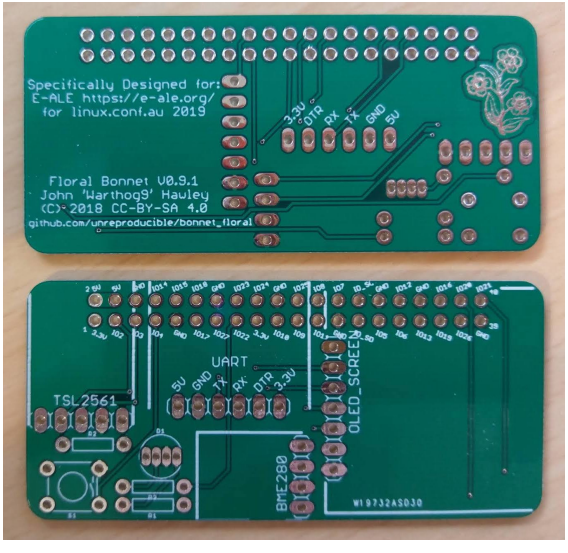


Figure 1.5: **The E-ALE Floral Bonnet for LCA**

- E-ALE has produced the floral bonnet specially for LCA2019

Raspberry-pi Zero Pinouts

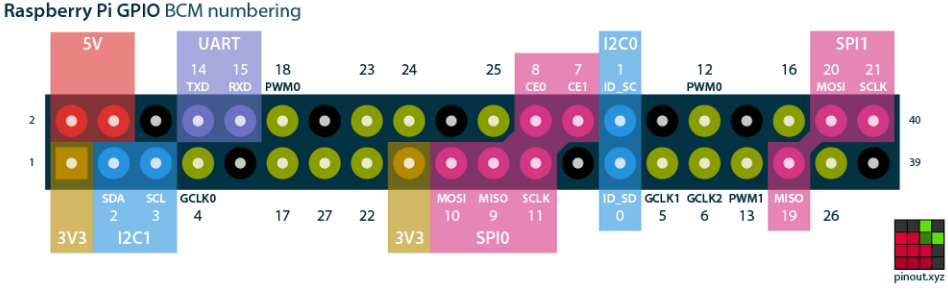
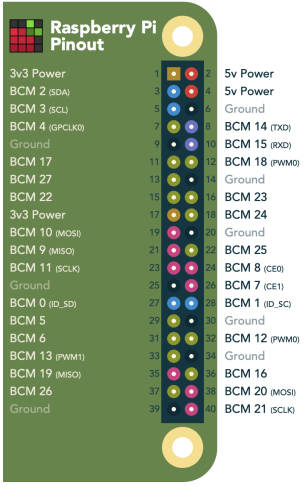


Figure 1.6: **Raspberry-pi Zero Pinouts**

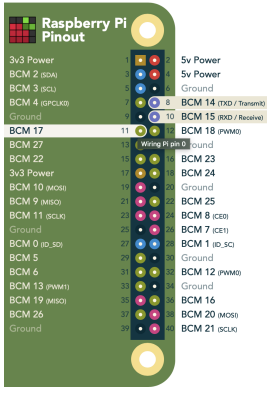
- The pinouts for all recent 20-pin raspberry-pi boards are the same.

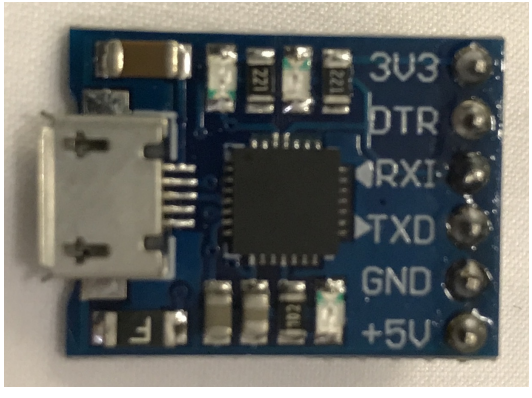
Raspberry-pi Zero Pins



- Pins are shared amongst multiple peripherals
- A pin multiplexer is used to choose the configuration of the pins in use.

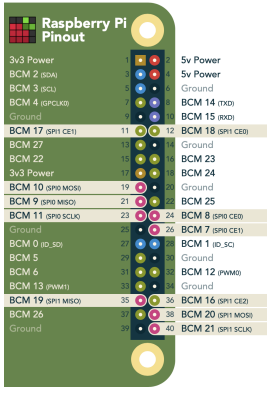
Raspberry-pi Zero Serial

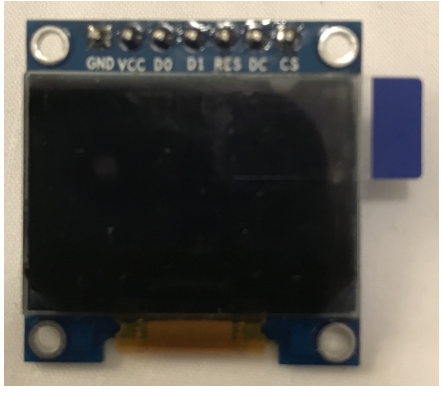




- We have access to the console serial port through the UART pins using a UART-to-USB converter like the CP2102
- We can use this serial port to configure the bootloader and debug kernel issues

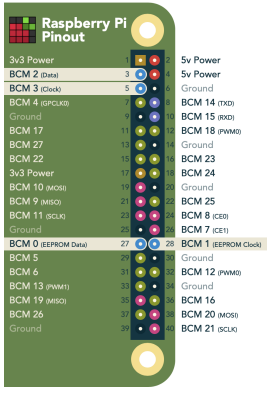
Raspberry-pi Zero SPI

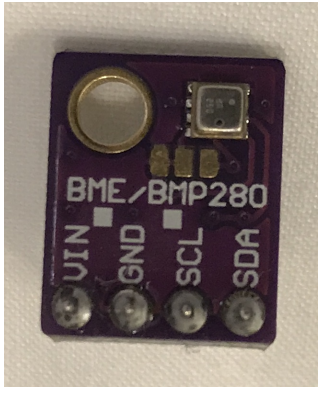




- We can access 2 SPI ports with these pins
- We will use one of the SPI channels to control the **SSD1306 OLED screen** on the floral bonnet

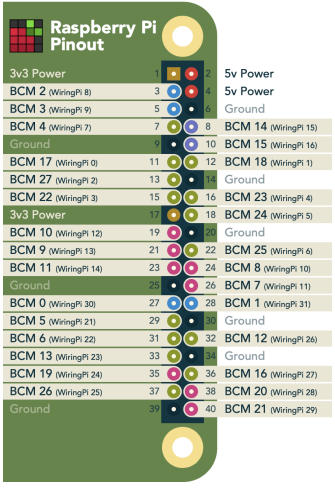
Raspberry-pi Zero I2C





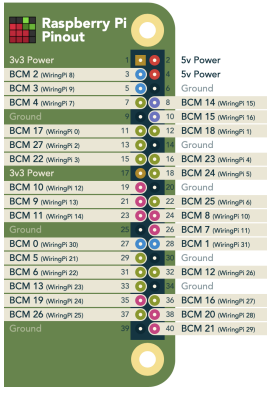
- We can access I2C devices via the I2C pins
- We will use the I2C bus to talk to the **BME280 Environmental sensor** and the **TSL2561 light sensor** on the floral bonnet

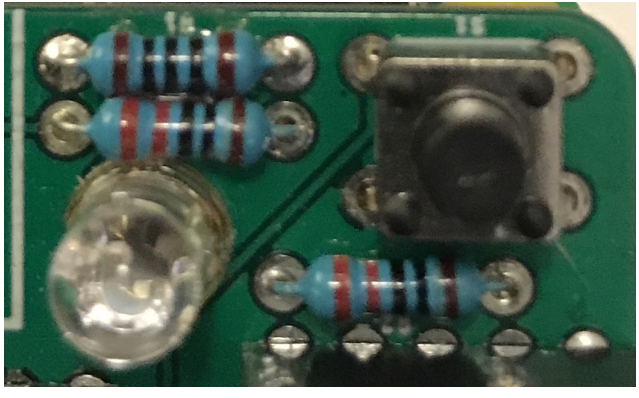
Raspberry-pi Zero GPIO pins



- We have 28 possible GPIOS, though many of these are shared with the previously mentioned buses
- Some of the GPIOs also have PWM capability

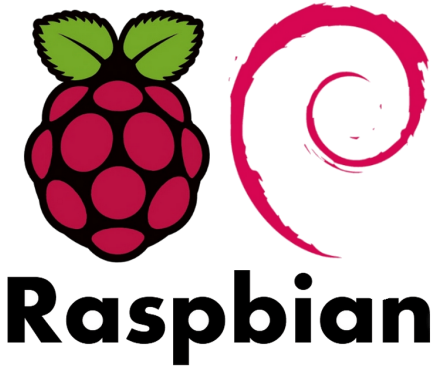
Raspberry-pi Zero GPIO pins





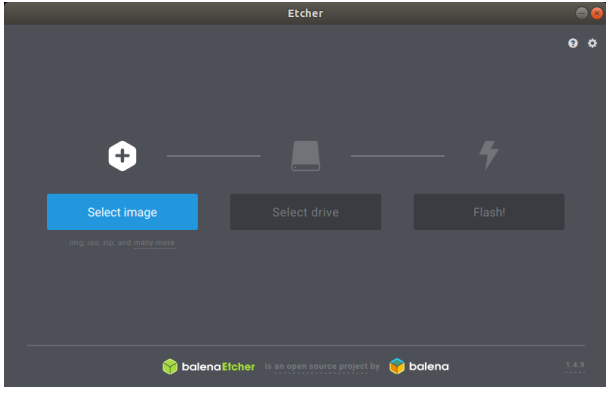
- We will use these pins to interface to a push-button (input) GPIO on the floral bonnet
- We will use 3 of these as output pins (including the 2 PWMs) to drive a tri-color LED

Using Raspbian as the Operating System



- We need to put a Linux based operating system on our board
- Raspbian is one of the many options for OS on the raspberry-pi
- We will be using Raspbian for the remaining labs in these seminars

uSD card



- We will install a Raspbian image directly (instead of using NOOBS) to the uSD card with **Balena Etcher**
- This will allow us to setup our board over a USB cable

Configure Raspian from Host OS

- We need to make some changes to be able to setup Raspbian headless over the USB cables (without a monitor and keyboard)
- With your uSD card still in the SD card reader...
- Use your favourite text editor to modify the following 2 files on the SD card
- Add the following to the end of `boot/config.txt`

```
dtoverlay=dwc2
enable_uart=1
```

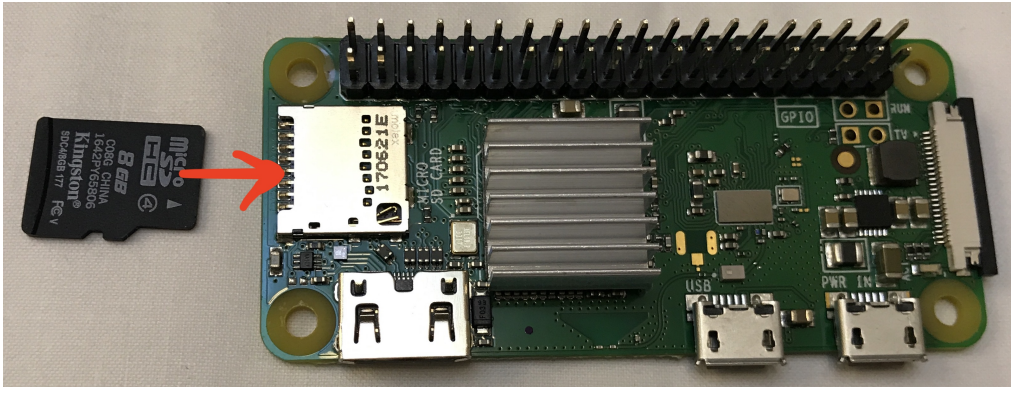
- Add the following after `rootwait` in `boot/cmdline.txt`

```
rootwait modules-load=dwc2,g_ether
```

- (it needs to be immediately after `rootwait` before anything else)
- We also need to start ssh by creating `boot/ssh`

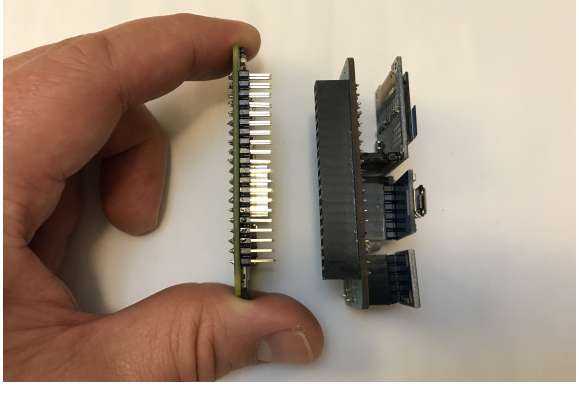
```
$ touch /media/$USER/boot/ssh
```

Move the uSD card into the rpi

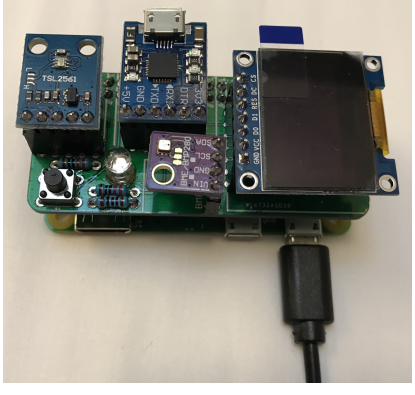


When you apply power once the SD card is inserted, the green LED next to the SD card will blink while the system boots, then stay solid green once booted.

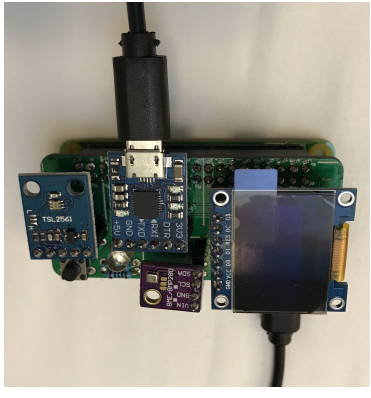
Attach the bonnet to the rpi-0 wh



Attach power to the rpi-0 wh



Attach the serial USB cable to the rpi-0 wh



Connect to the serial port with a terminal program like screen

- There are many terminal programs like **screen**, **minicom** or **putty**
- We will use **screen** in our examples

```
$ screen /dev/ttyUSB0 115200
```

- (You can eventually leave screen with **Ctrl-a, k, y**)

Log into the board

```
[  OK  ] Stopped LSB: Autogenerate and use a swap file.
[  OK  ] Started Show Plymouth Reboot Screen.
[  OK  ] Stopped target Remote File Systems.
[  OK  ] Stopped target Remote File Systems (Pre).
<<<SNIP>>>
[  OK  ] Started /etc/rc.local Compatibility.
[  OK  ] Started Permit User Sessions.
         Starting Hold until boot process finishes up...
         Starting Terminate Plymouth Boot Screen...

Raspbian GNU/Linux 9 raspberrypi ttyS0
raspberrypi login: pi
```

Login as **pi** with password **raspberry**.

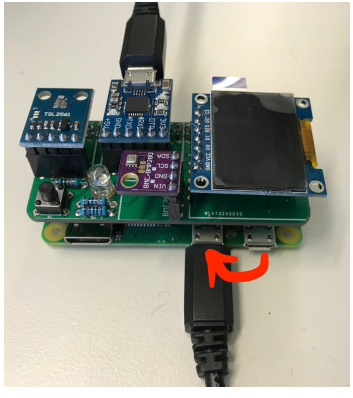
Change your password

You probably want to change your password

```
pi@raspberrypi:~$ passwd
Changing password for pi.
(current) UNIX password: raspberry
Enter new UNIX password: <your password>
Retype new UNIX password: <your password again>
passwd: password updated successfully
pi@raspberrypi:~$
```

Don’t forget this password

Move USB from power to OTG port



In order to get networking going, we have to move the USB cable from the power port to the OTG USB port.

Now connect with ssh

- Get your rpi IP address via the serial port with `screen`

```
pi@raspberrypi:~$ ifconfig usb0
usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
inet 169.254.12.238  netmask 255.255.0.0  broadcast 169.254.255.255
```

- With the RNDIS device configured with a link-local address on your host you can now ssh to the target

```
host$ ifconfig ens160u4u1
ens160u4u1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
inet 169.254.42.138  netmask 255.255.0.0  broadcast 169.254.255.255
host$ ssh pi@169.254.12.238
pi@169.254.12.238's password: *******
```

Now we test the peripherals

- Copy `testkit.tar.xz` over to the rpi

```
host$ rsync -aP testkit.tar.xz pi@169.254.12.238:~/
host$ ssh pi@169.254.12.238
pi@raspberrypi:~ $ tar xvf testkit.tar.xz
pi@raspberrypi:~ $ ls testkit
BME280  RGBLed.py  TSL2561  button.py  commands.txt
:luma.examples  py-rpi-ssd1306
```

Now test GPIO

```
pi@raspberrypi:~ $ cd testkit
pi@raspberrypi:~/testkit $ python RGBLed.py
```

See the LED light up, then stop it with a `Ctrl-C`

```
pi@raspberrypi:~/testkit $ python button.py
Button Pressed
Button Pressed
^C
```

By pressing the button we see messages on button up and down.

Testing I2C and SPI

There are a few more things that need to be installed in order to run the I2C and SPI demos. In order to install them we need to get wifi working and install a number of Raspbian packages.

These are beyond the scope of this class and we'll cover them in the upcoming I2C and SPI classes.

Finish setup

We now should have a setup **raspberry-pi zero WH** with a working **Floral Bonnet**

Chapter 2

GPIOs

e-ale

2.1 GPIO

Introduction to the Floral Bonnet

- Speaker: Behan Webster <behanw@converseincode.com>
- LCA2019 (2019.01.21)

Brought to you by

- Linux Foundation Training has provided speaker funding
- ARM is subsidizing the manufacturing of the floral bonnet for LCA2019

GPIO

- GPIO stands for **General Purpose Input/Output**
- GPIO pins can be programmed to be used as inputs or outputs
- As an input you can read back values of 1 or 0
- As an output you can write values of 1 or 0 to the GPIO pin

Active high or low

- Further they can be configured **Active high** or **Active low**
- These settings designate whether a high or low voltage is considered a 1 or a 0

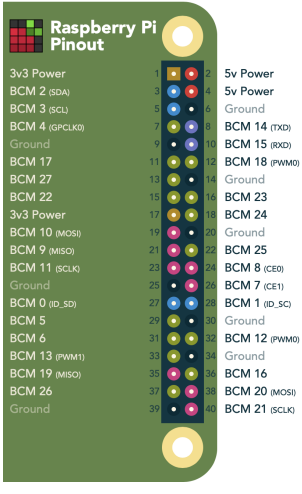
Open Drain or Open Source

- Furter more some SoCs have internal pull up and/or pull down resistors which can be used to force the value of the pin up or down when the pin isn't being driven
- **Open Drain** refers to the situation where a signal usually floats high unless driven low by the value of the GPIO pin
- **Open Source** refers to the situation where a signal usually floats low unless driven high by the value of the GPIO pin

Bit Banging Protocols

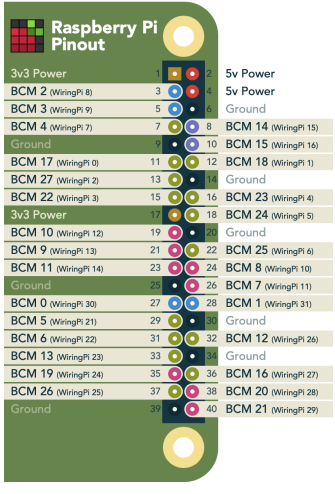
- The best situation is one where dedicated HW can be used to implement HW protocols in the most efficient manner possible
- However in situations where you don't have a HW bus controller, or you don't have enough buses, one can elect to sythesize a protocal using SW to drive GPIOS appropriately to emulate the bus protocol
- This SW implementation of a HW protocol is affectionately known as **bit banging** in the industry

Raspberry-pi Zero Pins



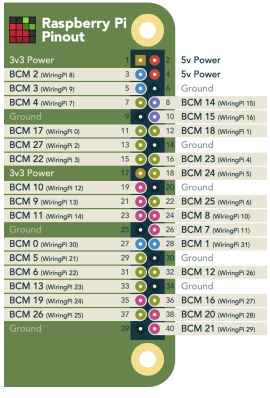
- Pins are shared amongst multiple peripherals
- A pin multiplexer is used to choose the configuration of the pins in use.

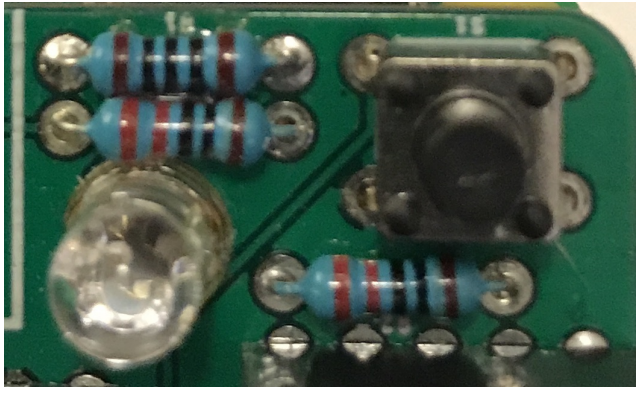
Raspberry-pi Zero GPIO pins



- We have 28 possible GPIOS, though many of these are shared with the previously mentioned buses
- Some of the GPIOs also have PWM capability

Raspberry-pi Zero GPIO pins





- We will use these pins to interface to a push-button (input) GPIO on the floral bonnet
- We will use 3 of these as output pins (including the 2 PWMs) to drive a tri-color LED

Using the sysfs gpio interface for input

```
# Set up GPIO 7 and set to input
echo "7" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio7/direction
# Read from input
cat /sys/class/gpio/gpio7/value
# Clean up
echo "7" > /sys/class/gpio/unexport
```

Using the sysfs gpio interface for output

```
# Set up GPIO 4 and set to output
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction
# Write output
echo "1" > /sys/class/gpio/gpio4/value
# Clean up
echo "4" > /sys/class/gpio/unexport
```

sysfs gpio interface is deprecated

- The sysfs gpio mechanism is actually deprecated in favour of libgpiod
- Although the libgpiod set of tools are now the preferred way of handling gpios, platforms like the raspberry-pi have their own utilities like `gpio`

libgpiod

- The libgpiod library and set of tools is now the official way of dealing with gpios.

```
pi@raspberrypi:~$ gpiodetect
gpiochip0 [pinctrl-bcm2835] (54 lines)
pi@raspberrypi:~$ gpioinfo
gpiochip0 - 54 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
...
        line  46:      unnamed       unused   input  active-high
        line  47:      unnamed       "led0"  output  active-high [used]
...
```

gpioset and gpioget

- **gpioset** and **gpioget** allow for a shell programatic interface to gpios without directly manipulatin sysfs

```
$ gpioset -m wait gpiochip0 4=1
```

The Raspberry-pi gpio utility

```
pi@raspberrypi:~$ gpio readall
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | ALT5 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
...
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
```

Now test GPIO

```
pi@raspberrypi:~ $ cd testkit
pi@raspberrypi:~/testkit $ python RGBLed.py
```

See the LED light up, then stop it with a `Ctrl-C`

```
pi@raspberrypi:~/testkit $ python button.py
Button Pressed
Button Pressed
^C
```

By pressing the button we see messages on button up and down.

Finish setup

We now should have a setup **raspberry-pi zero WH** with a working **Floral Bonnet**

Chapter 3

I2C

e-ale

3.1 I2C

I2C from Userspace

- Speaker: Grant Likely <grant.likely@arm.com>
- LCA2019 (2019.01.21)

Brought to you by

- Linux Foundation Training has provided speaker funding
- ARM is subsidizing the manufacturing of the floral bonnet for LCA2019

I2C

- **I2C** (Inter-Integrated Circuit), pronounced I-squared-C is a bus designed to communicate between chips on a board
- It is also known by the name **IIC**, **TWI** and **smbus** (although smbus is strictly a subset of the I2C specification)
- I2C was initially developed by Philips, however today owned by NXP Semicondictors
- Other companies, who have used I2C-like protocals have used the other listed names (for various reasons)

Two Wire Interface

- The common alternate name of **TWI** (Two Wire Interface) literally describes the physical layout of the bus
- I2C is made up of 2 wires:
  - A data line called SDA
  - A clock line called SCL

Synchronous bus

- I2C is a synchronous, multi-master/slave, packet based bus
- The 2 wires (SDA/SCL) are bidirectional **open collector** or **open drain** lines
- I2C tyipcally runs at 5V or 3.3V, though other voltages are permitted

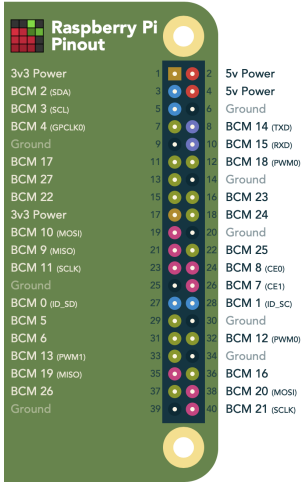
I2C speeds

- It was initially imagined for low speed inter-chip communications
- Although implemented in HW, the lowspeed version of I2C can still be implemented via GPIO using **bit-banging**, although the HW version is vastly preferred.
- HW based versions can run at speeds up to 5 Mbps

I2C addressing

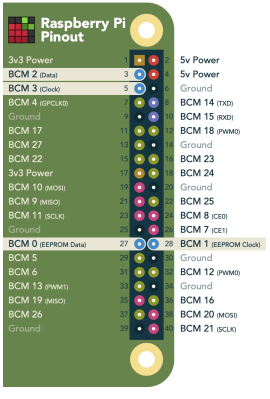
- I2C uses a 7-bit address space (though there is a rarely used 10-bit extension)
- I2C based devices usually have a range of selectable addresses which allow you to have more than one I2C device on the same bus, or more than one of the same kind of I2C device (with a different address)

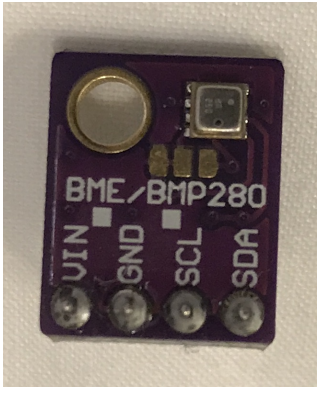
Raspberry-pi Zero Pins

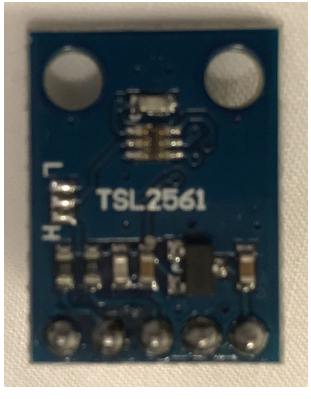


- Pins are shared amongst multiple peripherals
- A pin multiplexer is used to choose the configuration of the pins in use.

Raspberry-pi Zero I2C







- We can access I2C devices via the I2C pins
- We will use the I2C bus to talk to the **BME280 Environmental sensor** and the **TSL2561 light sensor** on the floral bonnet

I2C devices on Linux

- From the Linux userspace, you can access the I2C bus from the `/dev/i2c-*` device files

```
pi@raspberrypi:~$ ls -l /dev/i2c-1
crw-rw---- 1 root i2c 89, 1 Jan 20 19:47 /dev/i2c-1
```

Linux packages for I2C

- You will have to install some more packages on your raspberry-pi in order to easily access the i2c device files

```
pi@raspberrypi:~$ sudo apt-get install python-smbus i2c-tools
```

List I2C devices on the bus

```
pi@raspberrypi:~$ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- 39 -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@raspberrypi:~$
```

- Notice we can see the 2 devices which are on our I2C bus

Now we test the i2c devices

- Copy `testkit.tar.xz` over to the rpi (from the initial floral-bonnet class)

```
pi@raspberrypi:~ $ tar xvf testkit.tar.xz
pi@raspberrypi:~ $ ls testkit
BME280  RGBLed.py  TSL2561  button.py  commands.txt
:luma.examples  py-rpi-ssd1306
```

Running the BME280 over I2C

```
pi@raspberrypi:~$ cd testkit/BME280/Python/
pi@raspberrypi:~/testkit/BME280/Python$ python BME280.py
Temperature in Celsius : 24.27 C
Temperature in Fahrenheit : 75.69 F
Pressure : 1006.74 hPa
Relative Humidity : 49.45 %
pi@raspberrypi:~/testkit/BME280/Python$ watch -n 1 python BME280.py
```

- Try putting your finger on the BME280 and watch the temperature rise
- Blow on the BME280 and watch the temperature fall

Running the TSL2561 over I2C

```
pi@raspberrypi:~$ cd src/testkit/TSL2561/Python/
pi@raspberrypi:~/src/testkit/TSL2561/Python$ python TSL2561.py
Full Spectrum(IR + Visible) :108 lux
Infrared Value :16 lux
Visible Value :92 lux
pi@raspberrypi:~/src/testkit/TSL2561/Python$ watch -n 1 python TSL2561.py
```

- Try covering the TSL2651 with your hand and watch the values fall
- Shine a light on the TSL2651 and watch the values rise

Finish setup

We now should have a setup **raspberry-pi zero WH** with a working **Floral Bonnet**