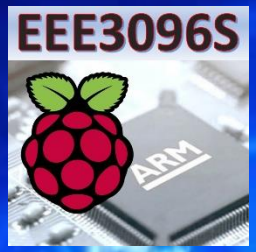
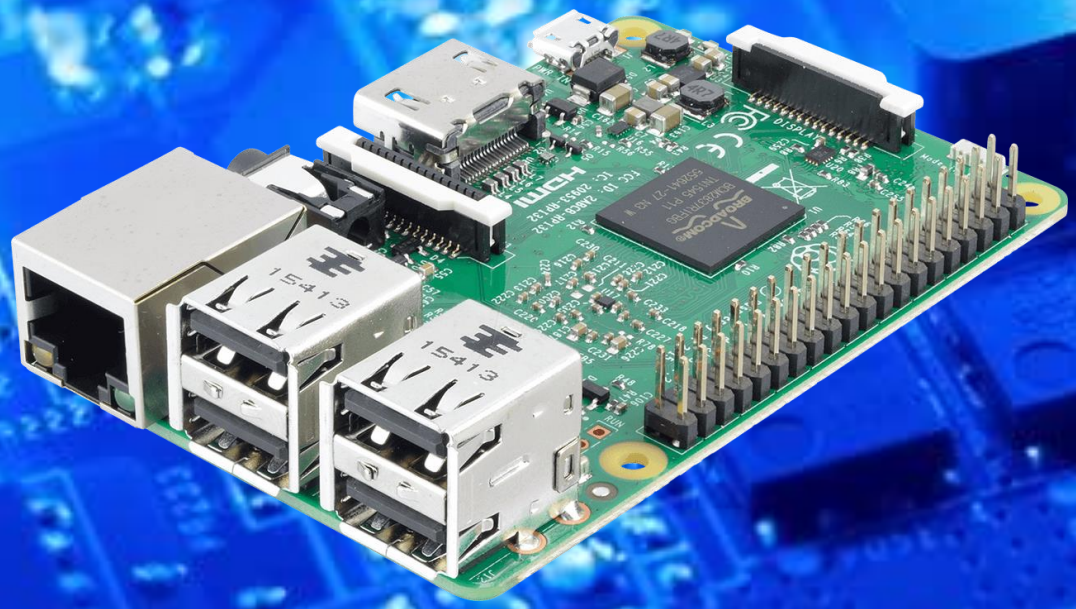




GPIO/Communications on the Raspberry Pi – P03

P3

Embedded Systems II

Dr Simon Winberg

Electrical Engineering
University of Cape Town

Most of this content won't be examined but it could help with the pracs

Test 1 Review

A BRIEF DISCUSSION OF SOME OF THE QUESTIONS

theme:

Comms & I/O Week

Outline of Lecture P02

- Choosing pins
- GPIO libraries for the RPi
- Example code
- Prac3 focus

GPIO

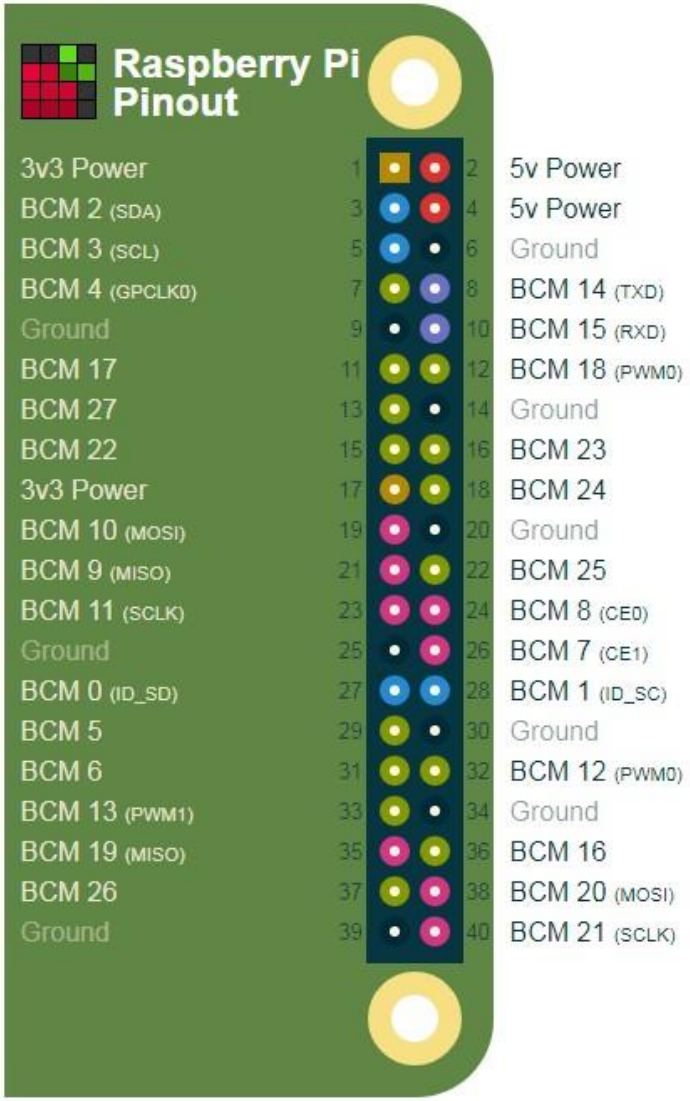
- GPIO = General Purpose I/O
- These are essentially pins that are available on the microcontroller and link to connectors on the board
- By General Purpose it means these can be used for a variety of purposes, e.g. interfacing to peripherals you want to add.

They are kind of like an embedded systems developers set of general purpose tools

GPIO and Choosing Pins

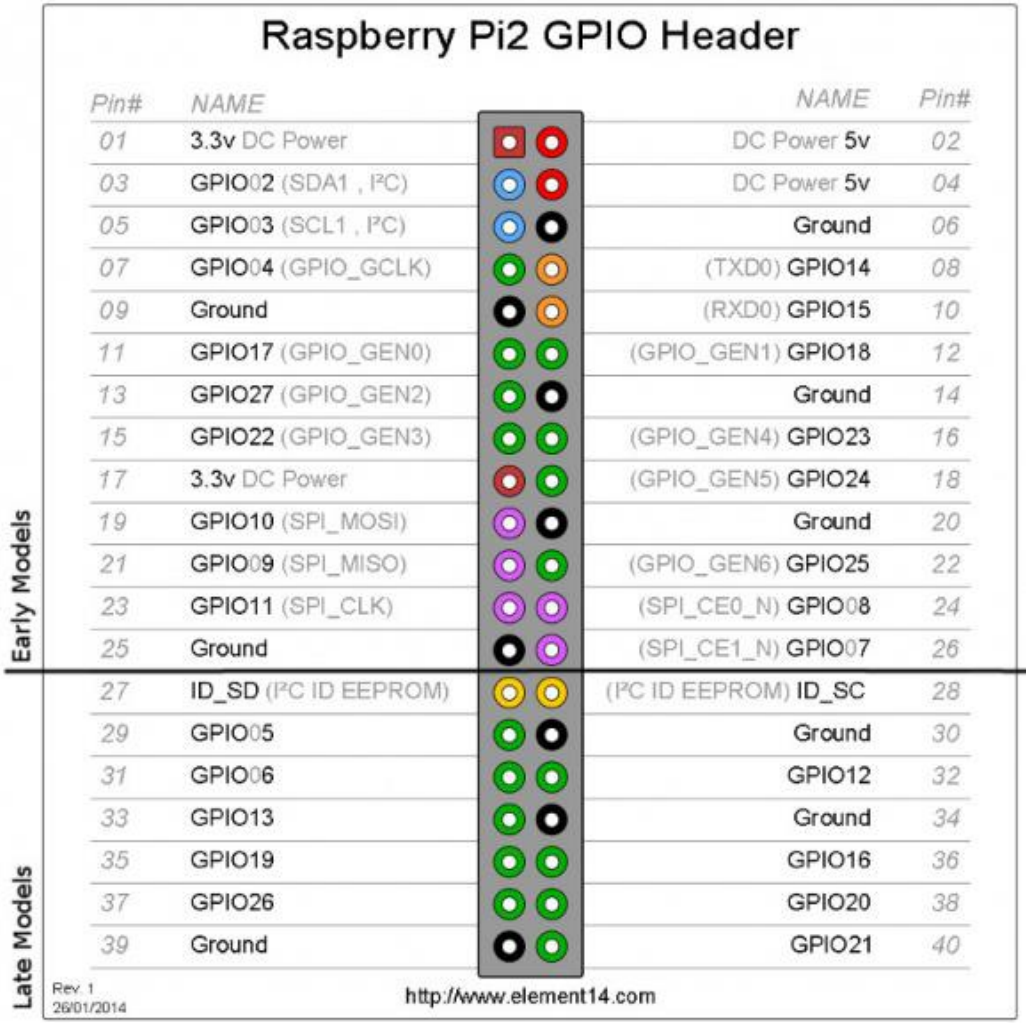
- Not necessarily all the pins on the processor link to connectors on the board.
- When you are planning I/O that your embedded application is going to use you need to make sure there are suitable interfaces and/or GPIO pins that you can make use of.
- **Note:** some pins have multiple purposes, they may be available as GPIO but they might be configurable to other uses too
  e.g. may link to devices on board or in chip, not just to pins on the GPIO header

Raspberry Pi Pinouts



- The Raspberry Pi has a GPIO 40-pin male header
- They are trying to keep this consistent between all models (e.g. right from version 1) to improve the compatibility of code. This is true for the external interfaces and pin names
- Version 1 had a smaller header 26pins + 8, subsequently GPIO headers have been 40pin
- But internally there are some difference between the pin routings

Raspberry Pi Pinouts



- Each pin has a number or a name (but usually numbers are used for efficiency)

The main libraries used for interfacing with the GPIO are:

- Wiring Pi (C code)
- RPi.GPIO (Python)

You can really use either one, I'm not fussed. Most of the pracs use Rpi.GPIO which is a bit simpler

Some basic experiments (library independent)

- Linux allows interfacing with drivers through the file system. These files are generally in /sys or /dev
- As mentioned previously there are char drivers and block drivers. You can send command to these (in root mode) using file command, either in software or via terminal command such as *echo* or *cat*
- This is generally not recommended of course unless you know what you are doing

Some basic experiments

- The Raspberry Pi doesn't have a generous amount of LEDs available – sadly ☹
- But you can control the power LED – you can experiment with using the following commands to turn it on and off and on:

You may need to **su root** for this to work…

Turn off power LED: sudo echo 0 >/sys/class/leds/led0/brightness

Turn on power LED: sudo echo 1 >/sys/class/leds/led0/brightness

Using RPi.GPIO

Using RPi.GPIO

- Should have RPi.GPIO already installed, but you could check for updates as shown in Prac3
- How to use Rpi.GPIO:

  Include the library:

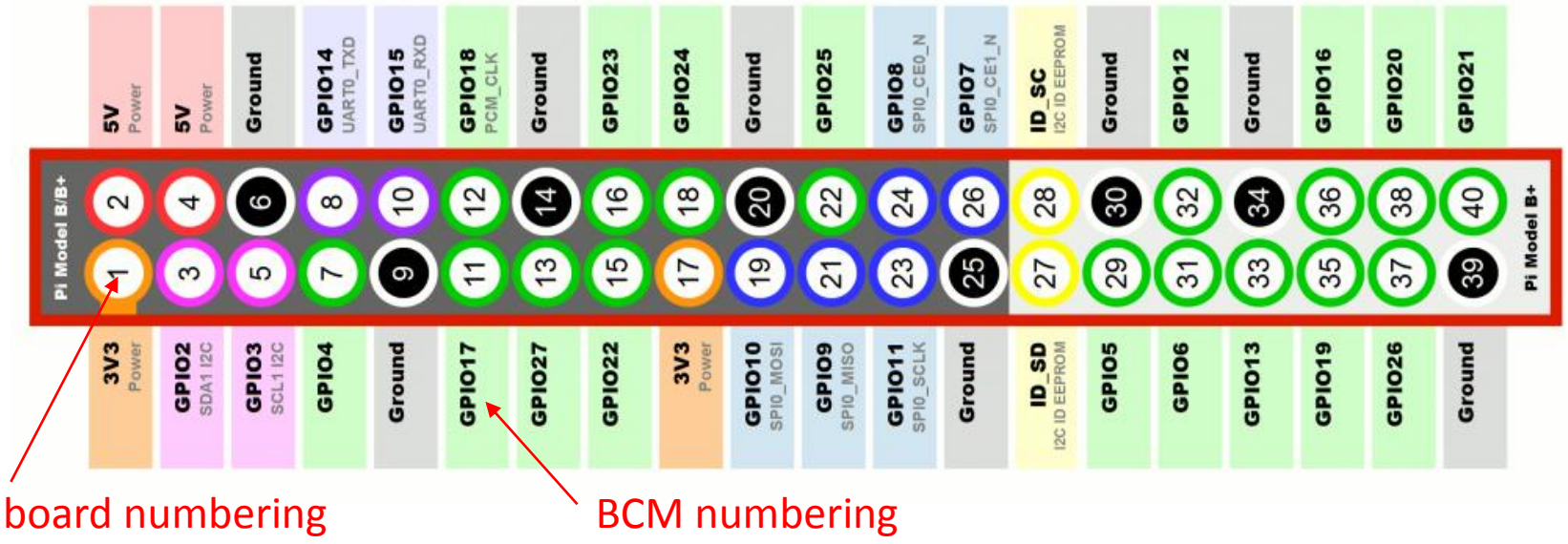
  ```
  import RPi.GPIO as GPIO
  ```

  Set up to use this pin numbering:

  ```
  GPIO.setmode(GPIO.BCM)
  ```

GPIO.BCM pin numbering option

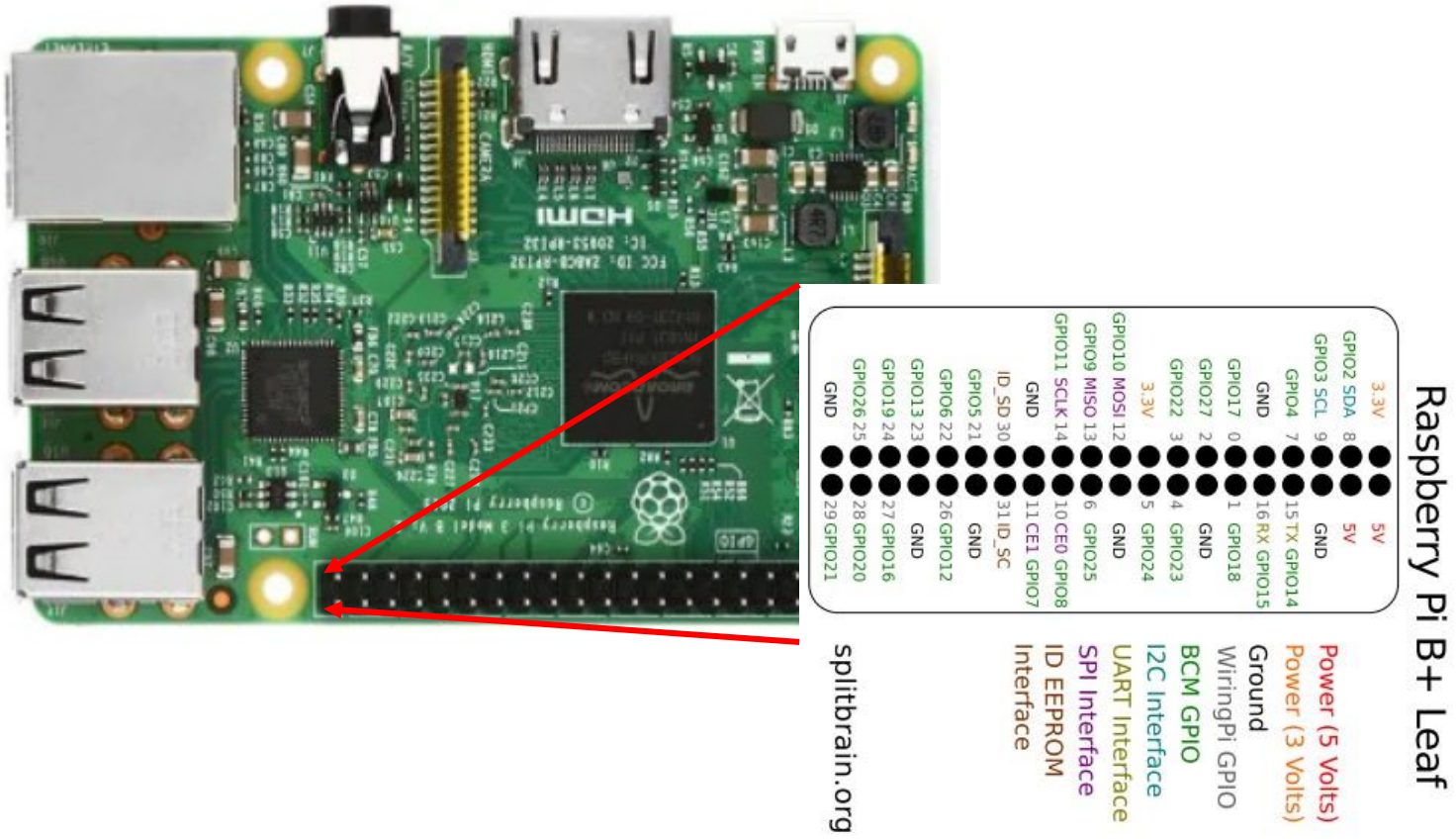
- The GPIO.BCM option indicates you want to refer to pins using their "Broadcom SOC channel" numbers – these are the numbers after "GPIO" in green shaded rectangles along the side of the diagram:



Raspberry GPIO Leaf

- To make reading the pins on the header easier, I suggest creating a PIO Leaf
- See: https://github.com/splitbrain/rpibplusleaf
- You probably want something that is properly sized for A4, so try: https://github.com/splitbrain/rpibplusleaf/blob/master/rpiblusleaf16.pdf
- Print it out, possibly ‘laminate’ it with sticky tape on either side and poke holes in it and put it over the 40-pin header

Orienting the GPIOs



Be very careful with probing the pins!!! As you can see 5V and GND are right next to each other, it is easy to short these pins out, if you do it too often that could be make your Raspberry Pi burned and busted.

Example use of GPIO

Here is some example code testio.py :

```
#!/usr/local/bin/python
# Include the library:
import RPi.GPIO as GPIO
from time import sleep   # to do delays
# Set up to use this pin numbering:
GPIO.setmode(GPIO.BCM)
TESTPO = 26  # pin on top next to GND on far left (see prev. slide)
TESTPI = 21 # pin below GNB on far right to test input
# set up the pins:
GPIO.setup(TESTPO, GPIO.OUT)   # For turning on the pin
GPIO.setup(TESTPI, GPIO.IN)  # For reading the pin
GPIO.output(TESTPO, True)     # turn on the pin
sleep(10) # wait 10s
GPIO.output(TESTPO, False)     # turn on the pin
# read state of the input
if GPIO.input(TESTPI):
        print("Input High")
else:
        print("Input Low")
GPIO.cleanup()  # clean up / revert to previous state
```

You can also use GPIO.HIGH and GPIO.LOW instead of True and False

You can sleep for fractions of a second e.g. sleep(0.1) will sleep for 100ms.

Use IDLE while experimenting with GPIO

- IDLE is the Python IDE that is preinstalled on the Raspberry Pi
- But simply running from the default pi user will be a bit of a hassle, as you won't have permissions to directly use the GPIO, for which super user access is needed
- So, if you want to use IDLE start it with

  gksu idle3

  Or to load a program (e.g. the testio.py) use:

  gksu idle3 testio.py

(a window will pop up to ask for the password, you could just use sudo instead of gksu)

PWM Pin

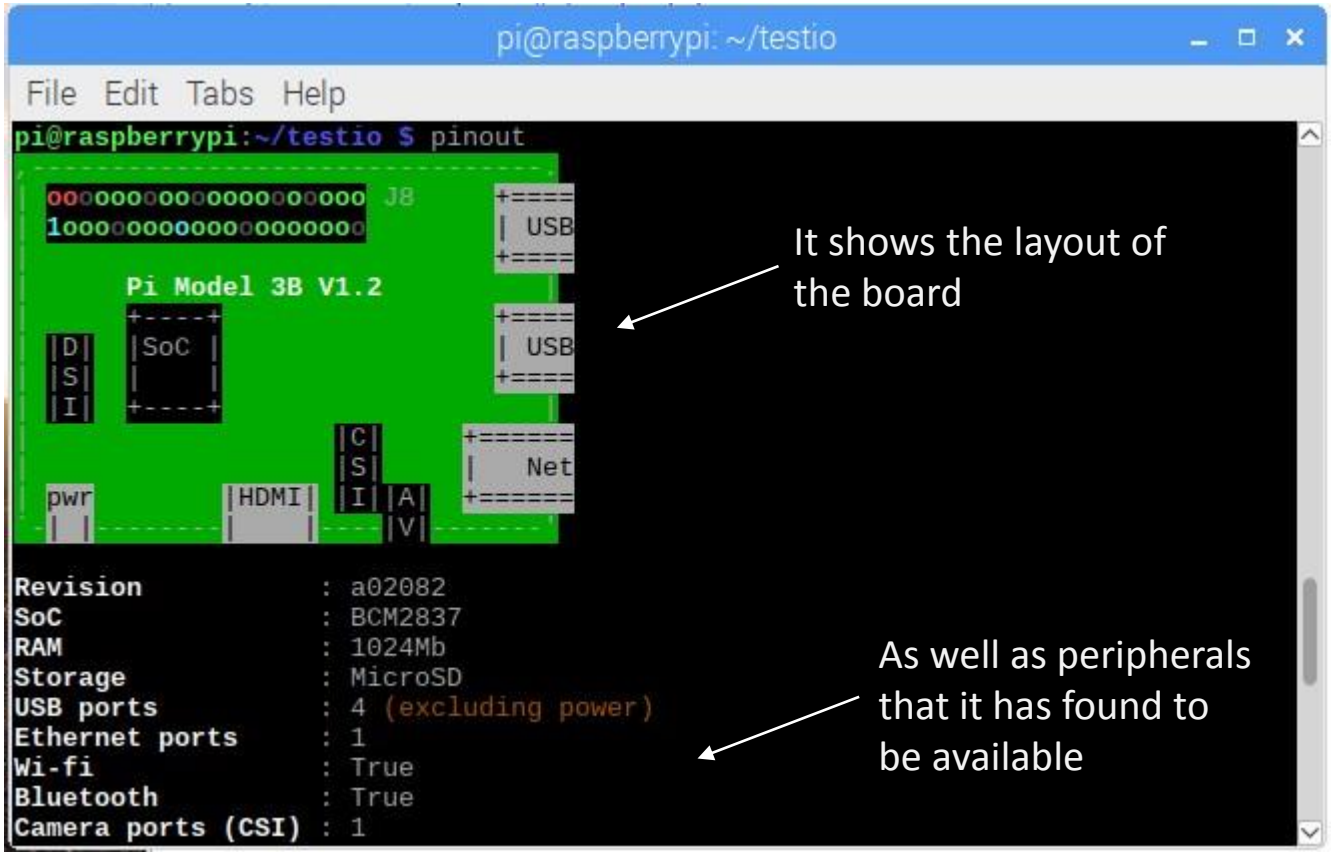
- Pin 18 (in BCM numbering) can perform automatic PWM
- It is easy to use:
  - Use GPIO.PWM([pin], [frequency]) to create a PWM object and then call start on that object to start PWM and specify the duty cycle, e.g.

```
pwm = GPIO.PWM(18, 1000) # create PWM object
pwm.start(50)    # start 50% duty cycle
```

RPi pinout

- Conveniently enough Raspbian has a pinout map that can be displayed from the shell:

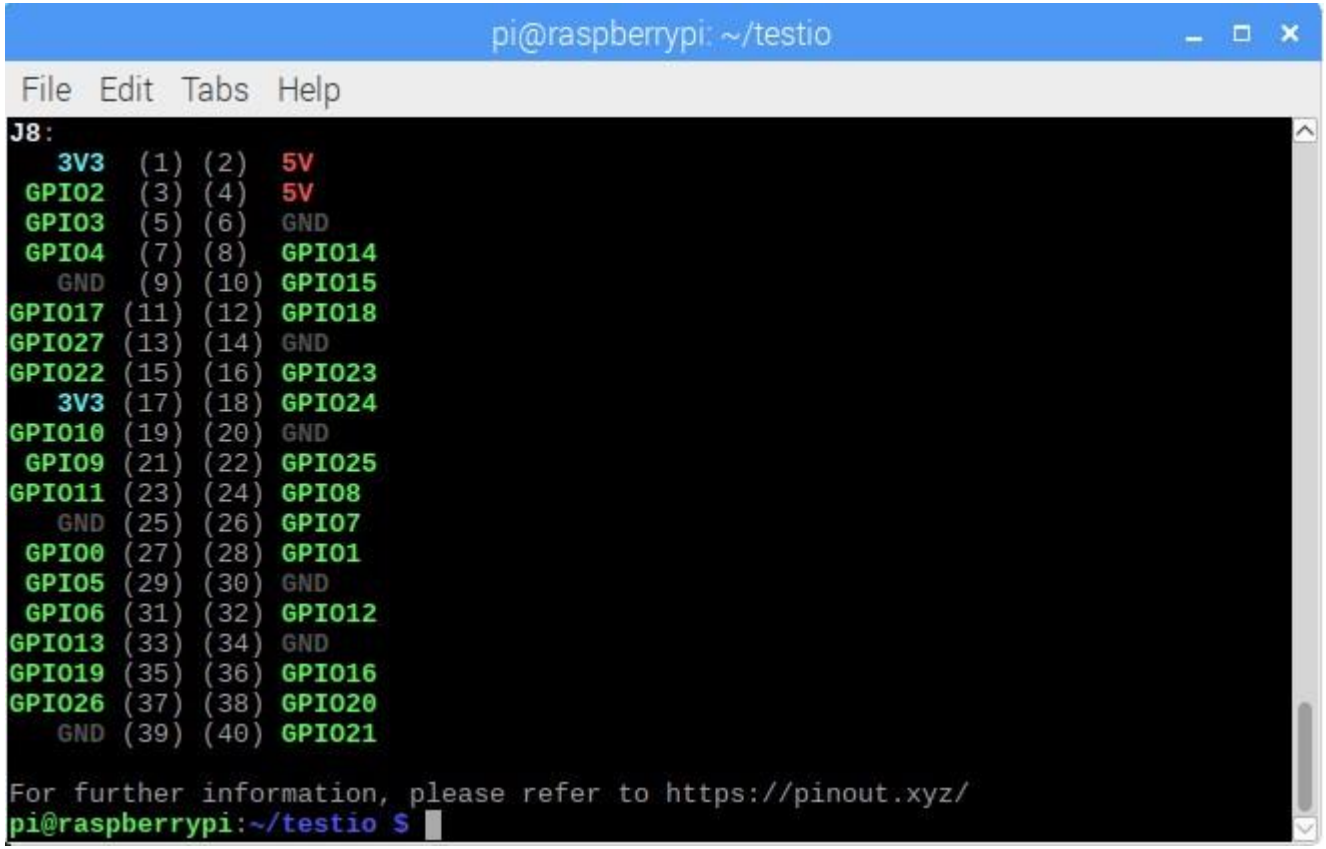
Simply type **pinout** at the command like and you will be presented this useful info:

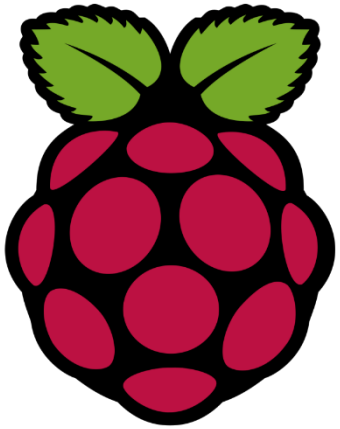


… PTO …

RPi pinout

- If you scroll further down it shows the board mapping (in grey brackets) and the BCM mappings (in green)





You’re now all set for Prac3 !!

WiringPi

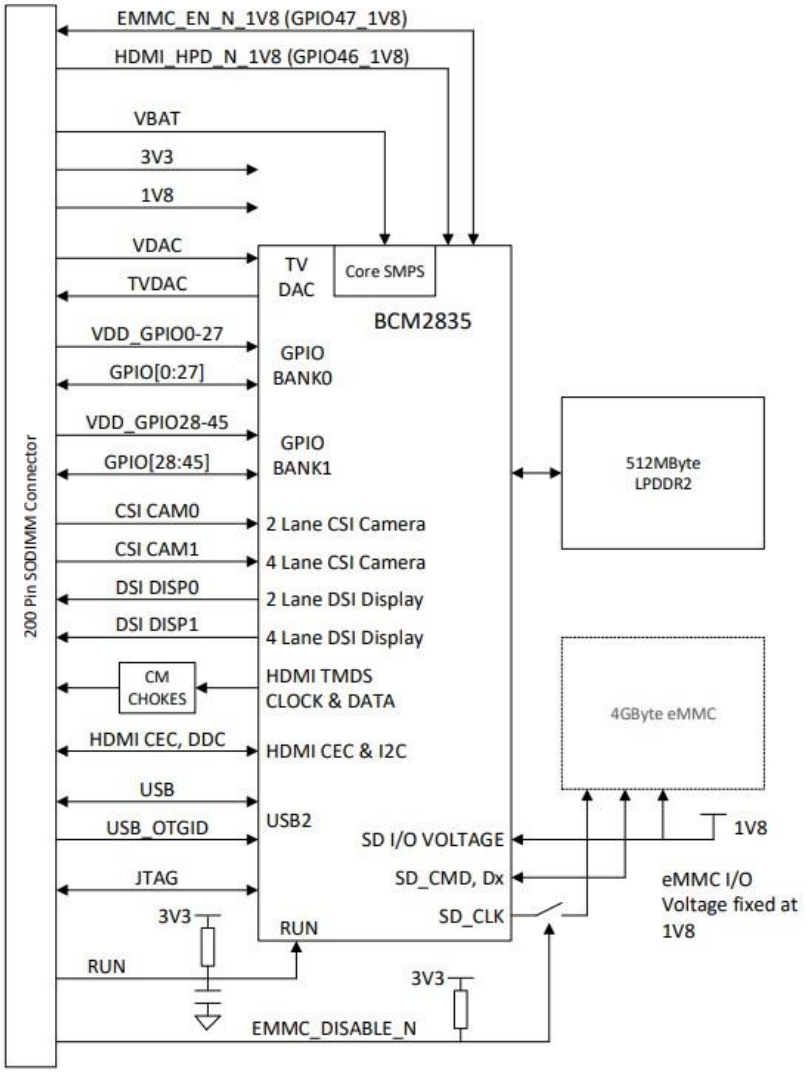
- The WiringPi library is a C based GPIO library for the Raspberry Pi
- It is technically a bit more sophisticated but works much the same as RPi.GPIO, i.e. you need to set up the pins to use first etc.
- But the advantage of WiringPi is primarily that it is C and is not interpreted. This makes the timing much more predictable than Python that sometimes decides to do other things (like garbage collection*).
- Nice simple example available at: https://learn.sparkfun.com/tutorials/raspberry-gpio/c-wiringpi-example but needs a breadboard circuit to be built to test the blinking LEDs

* Although if you aren't using fancy OO code and data structures then Python will generally behave well, but still won't match the speed of C.

BCM2835 Connections on RPi

Looking at more detail into how the BCM2835 SoC is connected up…
This is just one of multiple aspects showing the connections. You can inspect the complete IO descriptions in the “Raspberry Pi Compute Module” *

All GPIO pins have at least two alternative functions in the SoC. When not used for the alternate peripheral function, each GPIO pin may be set as an input (optionally an interrupt) or an output. The alternate functions are usually peripheral I/Os, and most peripherals links appear twice to allow flexibility on the choice of I/O voltage.

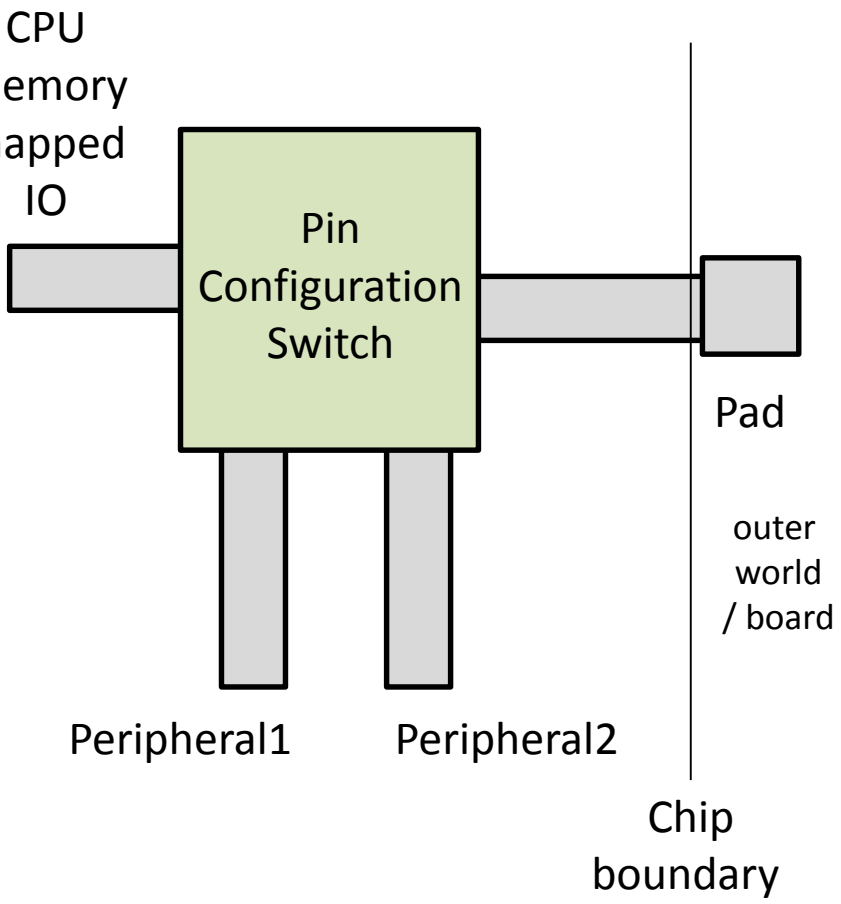


BCM2835 connections on Raspberry Pi

* source: https://www.alliedelec.com/m/d/1988f69b72864ea60d5867861de53e42.pdf

Pin Sharing

- In terms of pin sharing, this often concerns peripherals inside the chip. This is essentially a trade-off between:
  - Pins linking to the CPU (via memory)
  - Pins connected to the *pad* (and 'outer world')
  - Pins connected to internal peripherals



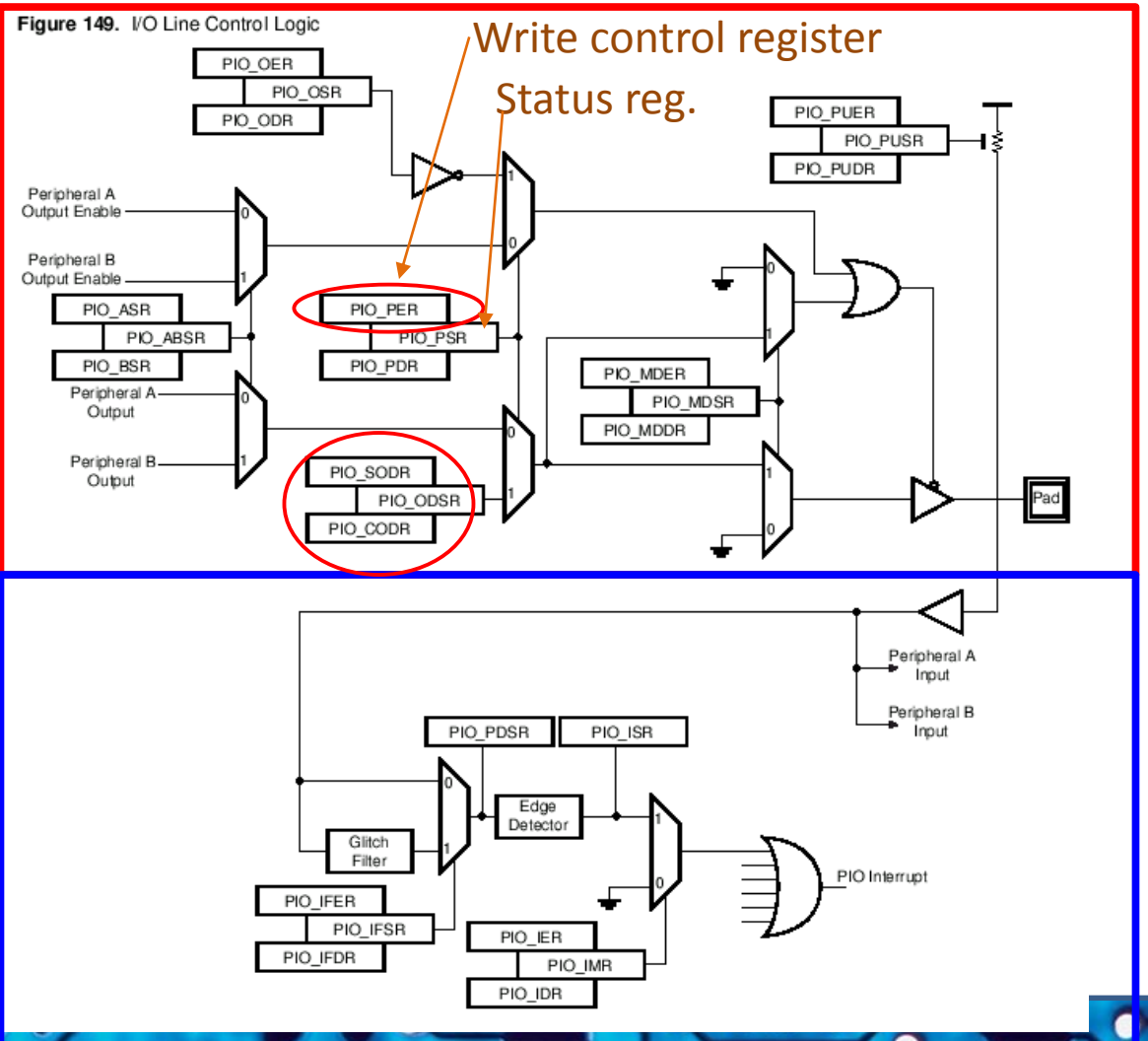
This is essentially what the GPIO.setup command controls… but how?

ARM I/O Control Logic

Functional Description

The PIO Controller features up to 32 fully-programmable I/O lines. Most of the control logic associated to each I/O is represented in Figure 149.

Figure 149. I/O Line Control Logic



The ARM processor have IO Line Control Logic block for each of its IO Lines (including GPIO lines) this shows the general design used.

Output Control Logic

PER – Enable Peripheral
SODR – Set bit of data register
CODR – Clear bit of data register

Interrupt Control Logic

These are generally accessed via certain memory addresses, so

So, RPi.GPIO provides a convenient user-friendly facility…

Reflect back on what was presented, somehow memory registers need to be written…

How does RPi.GPIO access low level port addresses,

e.g. how does the GPIO.output function write an address to set the output value of a pin?

Enable GPIO pin 1 as output by writing 1 to address 0xFFFF0100…

Which way do you think RPi.GPIO does this:

A) Interfacing C to Python
   e.g. memwrite(4294902016,1)
B) Accessing memory addresses directly from Python
   e.g. mem[int('0xFFFF0100',16)] = 1;
C) Use sys file
   e.g. open('/sys/class/gpio/gpio%s/value'%id, 'w') as f:
            f.write('1')

The standard implementation is just using a system file, there are various other alternatives and versions that abound, that support the same API.

Rpi.GPIO – low level I/O code

Doing output:

```
def output(channel, value):
    """Write to a GPIO channel"""
    id = _GetValidId(channel)
    if id not in _ExportedIds or _ExportedIds[id] != OUT:
        raise WrongDirectionException
    with open('/sys/class/gpio/gpio%s/value'%id, 'w') as f:
        f.write('1' if value else '0')
```

This is pretty much good enough, at least if your timing isn't too stringent, and it is fine for e.g. just setting up parameters etc. But it is not necessarily the best and most reliable means for long-term IO operations, the frequent opening of the file for example is

The Next Episode…

Lecture P12

Presented by Dr Yunus Gaffar

References

- Raspberry Pi Pinouts! https://pinout.xyz/
- Gordons Projects : WiringPi, available: https://projects.drogon.net/raspberry-pi/wiringpi/
- Raspberry Pi Compute Module Data Sheet https://www.alliedelec.com/m/d/1988f69b72864ea60d5867861de53e42.pdf