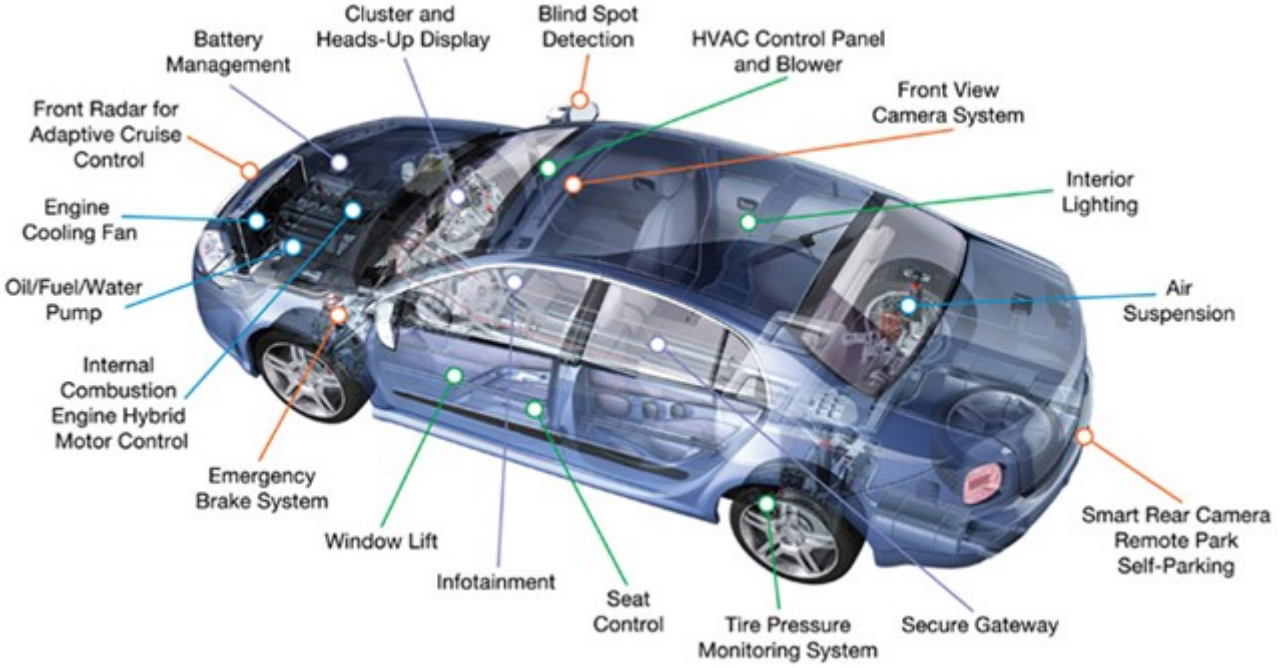
Computer organization and architecture

Lesson 22

Input/Output (I/O) systems

Input/Output (I/O) systems are used to connect a computer with external devices called **peripherals**.

Devices in embedded systems



A processor accesses an I/O device using the address and data busses in the same way that it accesses memory.

A portion of the address space is dedicated to I/O devices rather than memory.

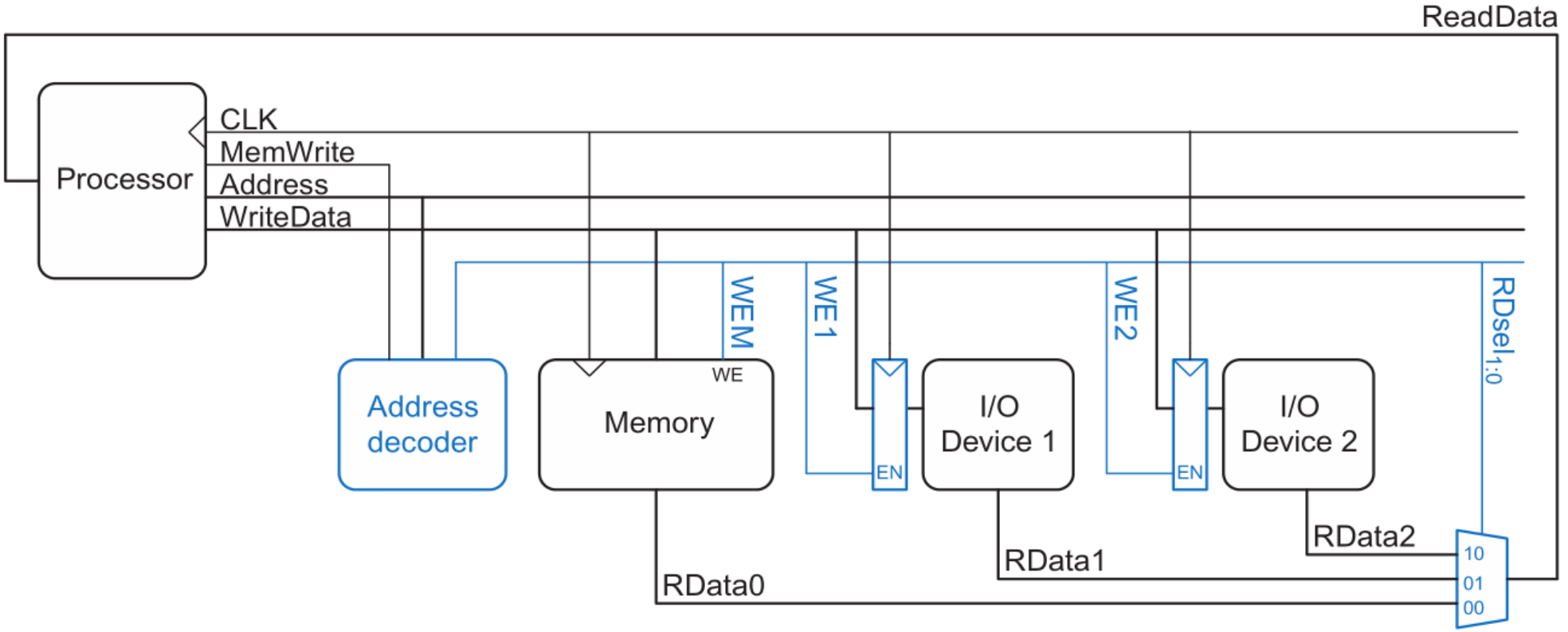
Example: physical addresses in the range 0x20000000 to 0x20FFFFFF may be used for I/O.

Each I/O device is assigned one or more memory addresses in this range.

A store to the specified address sends data to the device.

A load receives data from the device.

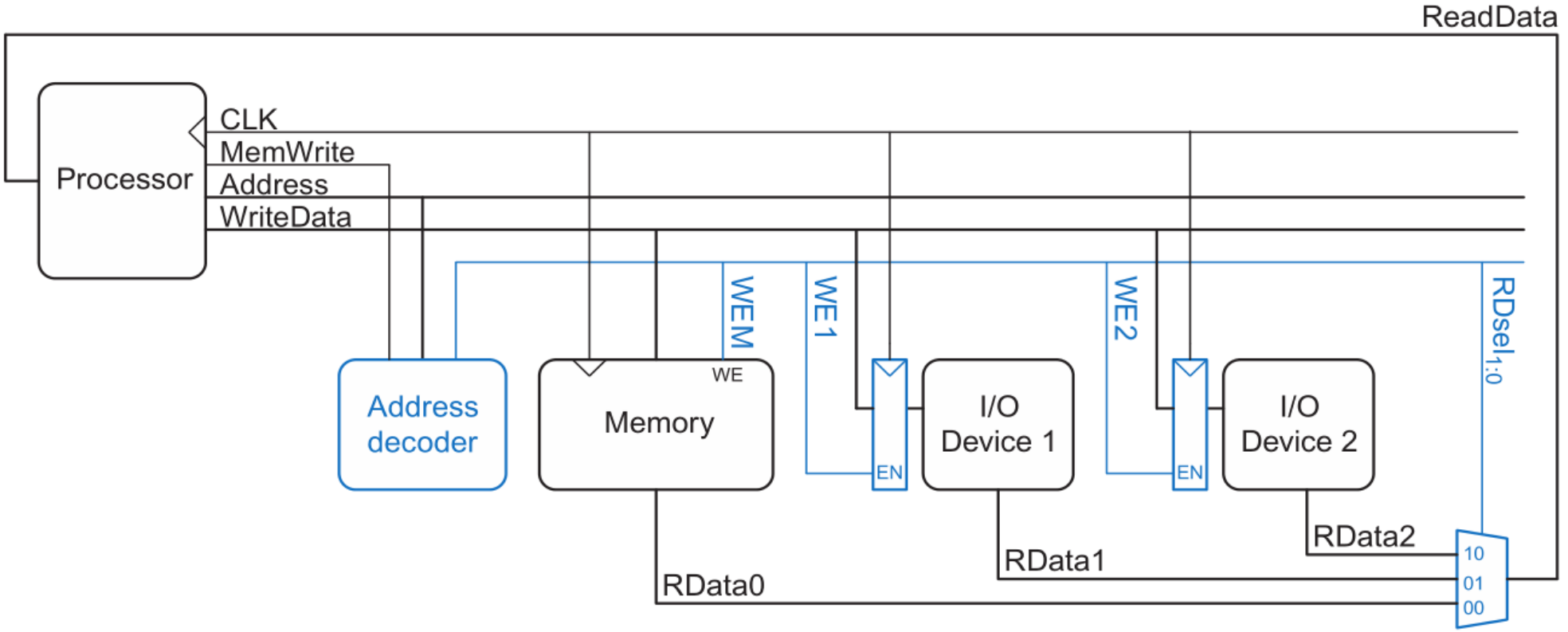
This method of communicating with I/O devices is called **memory-mapped I/O**.



An **address decoder determines** which device communicates with the processor.

It uses the Address and MemWrite signals to generate control signals for the rest of the hardware.

The ReadData multiplexer selects between memory and the various I/O devices.



Write-enabled registers hold the values written to the I/O devices.

The addresses associated with I/O devices are often called **I/O registers** because they may correspond with physical registers in the I/O device.

Software that communicates with an I/O device is called a **device driver**.

Writing a device driver requires detailed knowledge about the I/O device hardware including the addresses and behavior of the memory-mapped I/O registers.

Other programs call functions in the device driver to access the device without having to understand the low-level device hardware.

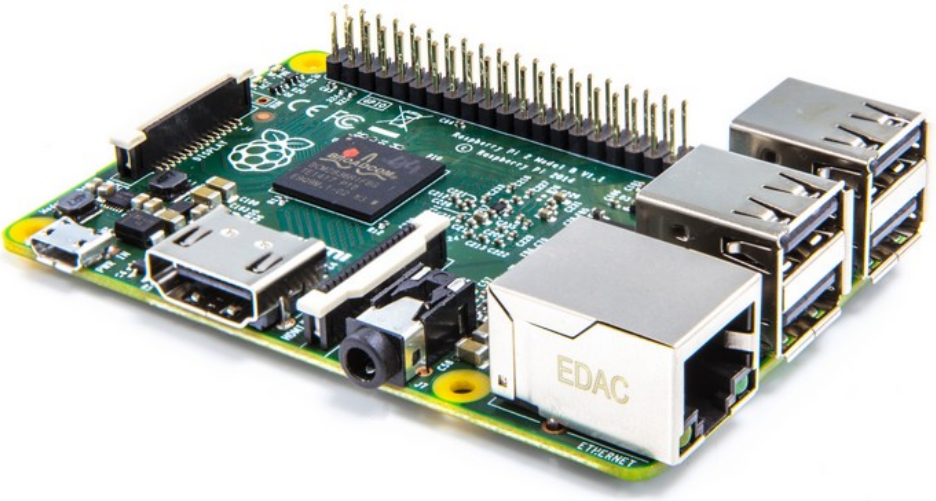
Raspberry Pi

It is family of single board computers.

Raspberry Pi 1, Raspberry Pi Zero are single core, 32 bit, ARMv6.

Raspberry Pi 2 is a 32 bit, quad-core, ARMv7.

Raspberry Pi 3 is a quad-core ARMv8, supporting 32-bit and 64-bit instruction sets.



The book *Sarah Harris, David Harris-Digital Design and Computer Architecture. ARM Edition (2015)* discusses Raspberry Pi Model B+

I have Raspberry Pi 2 Model B – a different board

The peripheral base address in Raspberry Pi Model B+ is 0x20000000

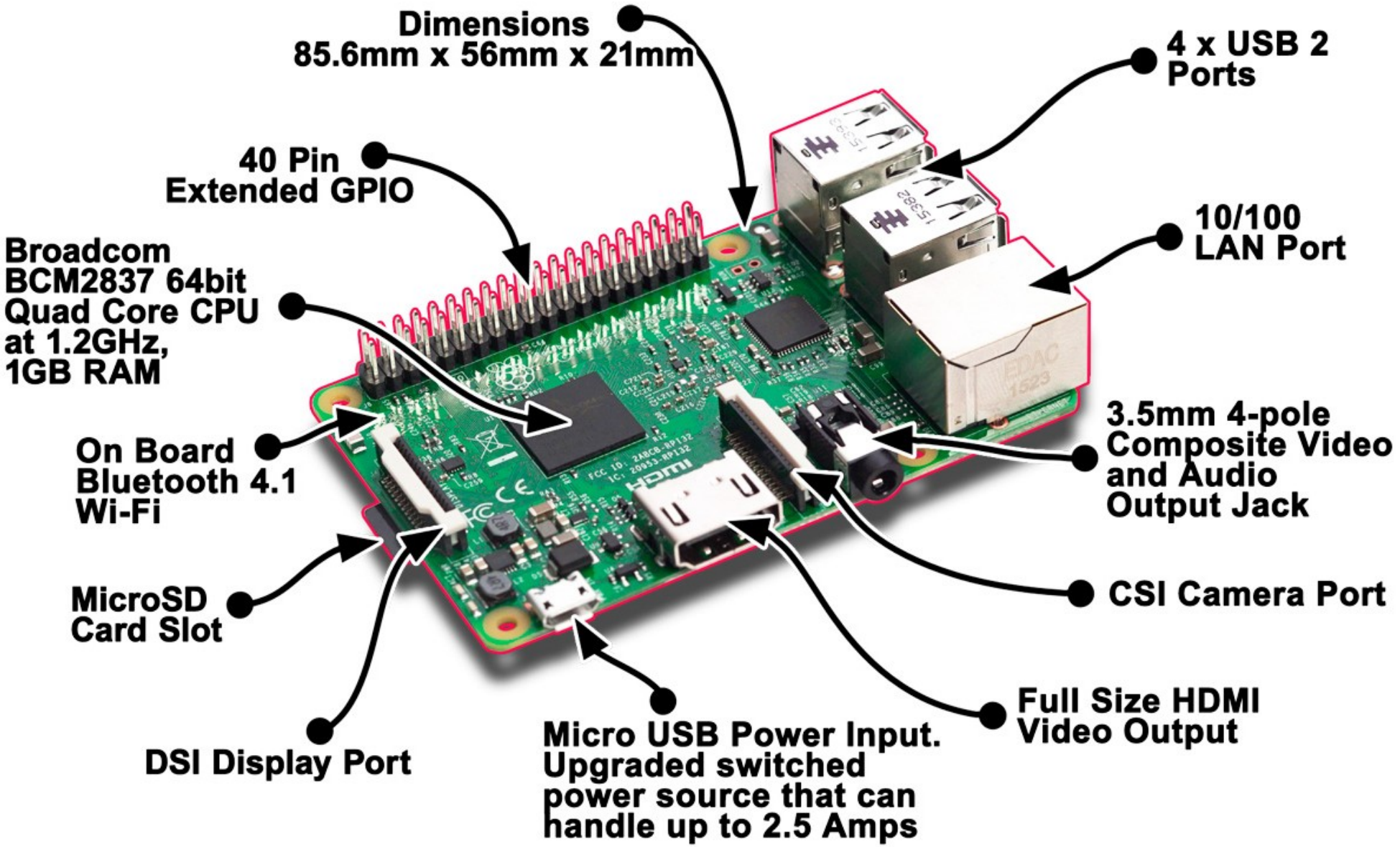
The peripheral base address in Raspberry Pi 2 Model B is 0x3F000000

The shape and size of the two boards are the same.

The USB, Ethernet, A/V, HDMI, micro SD and microUSB connectors are int the same exact locations and are the same size

The Camera, Display and 40-pin GPIO connectors are in the same exact locations and are the same size

Raspberry Pi 3



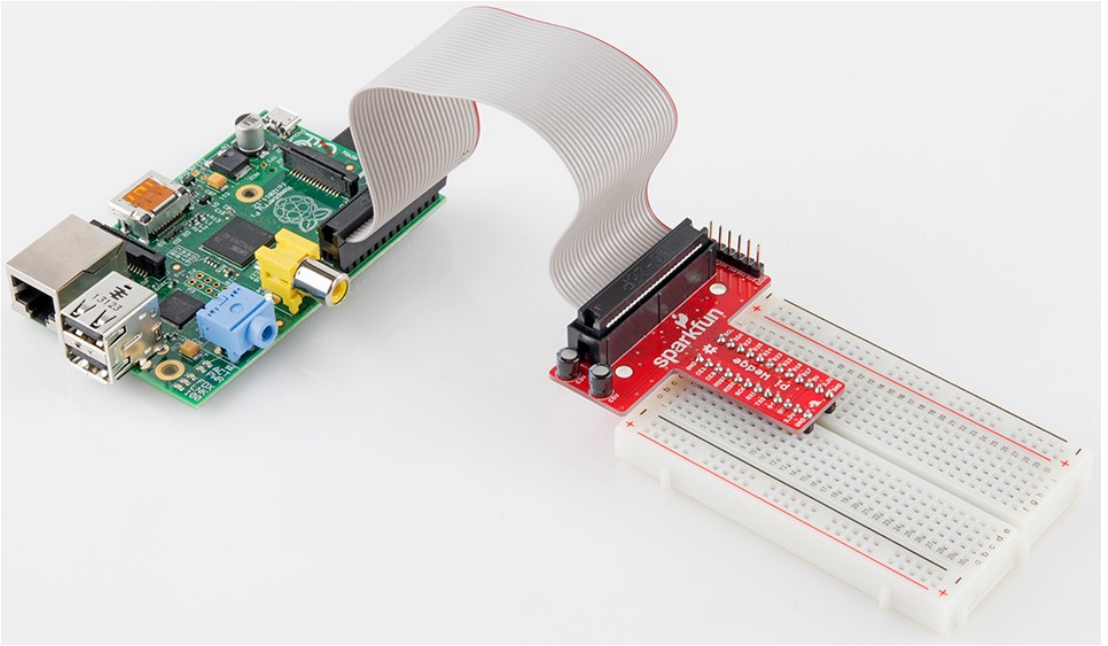
Raspberry Pi can drive HDMI displays, process mouse, keyboard, and camera inputs, connect to the Internet, and run full-featured Linux distributions.

You need an ARM version of Linux.

But it's more than just a small computer, it's a hardware prototyping tool.

The Pi has bi-directional GPIO pins, which you can use to drive LEDs, spin motors, or read button presses.

General-purpose I/O (GPIO) pins are used to read or write digital signals.



GPIO (general-purpose I/O) ports

```
pi@raspberrypi ~/wiringPi/examples $ gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | 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  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | 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      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |  OUT | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 0 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 0 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 | ALT0 | 1 | 27 || 28 | 1 | ALT0 | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 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 2---+---+------+---------+-----+-----+
```

Caution: connecting 5 V to one of the 3.3 V I/Os will damage the I/O and possibly the entire Raspberry Pi.

Programmers can manipulate I/O devices directly by reading or writing the memory-mapped I/O registers.

However, it is better programming practice to call functions that access the memory-mapped I/O.

These functions are called **device drivers**.

The code is easier to read when it involves a clearly named function call rather than a write to bit fields at an obscure memory address.

Somebody who is familiar with the deep workings of the I/O devices can write the device driver and casual users can call it without having to understand the details.

The code is easier to port to another processor with different memory mapping or I/O devices because only the device driver must change.

We will discuss a simple device driver called EasyPIO to access BCM2835 (Raspberry Pi Model B+) or BCM2836 (Raspberry Pi 2 Model B) devices.

The whole code is in the file EasyPIO.h

Casual users are likely to prefer WiringPi (http://wiringpi.com/), an open-source I/O library for the Pi, which has functions similar to but not exactly matching those in EasyPIO.

The memory-mapped I/O on the BCM2836 is found at physical addresses 0x3F000000-0x3FFFFFFF.

The GPIO physical base address is 0x3F200000

Peripherals have multiple I/O registers starting at their base address.

For example, reading address 0x3F200034 will return the values of GPIO pins 31:0.

The Raspberry Pi typically runs a Linux operating system using virtual memory.

Loads and stores in a program refer to virtual addresses, not physical, so a program cannot immediately access memory-mapped I/O.

The pioInit function in EasyPIO maps the physical addresses of interest to the program's virtual address space.

The pioInit opens /dev/mem , which is a Linux method of accessing physical memory.

Then the mmap function is used to set gpio as a pointer to physical address 0x3F200000, the beginning of the GPIO registers.

Read a manual for mmap in Linux:

```
man mmap
```

The pointer gpio is declared volatile, telling the compiler that the memory-mapped I/O value might change on its own, so the program should always read the register directly instead of relying on an old value.

GPLEV0 accesses the I/O register 13 words past GPIO, e.g. at 0x3F200034, which contains the values of GPIO 31:0.

At a minimum, any GPIO pin requires registers to read input pin values, write output pin values, and set the direction of the pin.

In many embedded systems, the GPIO pins can be shared with one or more special-purpose peripherals, so additional **configuration registers** are necessary to determine whether the pin is general or special-purpose.

Furthermore, the processor may generate interrupts when an event such as a rising or falling edge occurs on an input pin, and configuration registers may be used to specify the conditions for an interrupt.

If a microcontroller needs to send more bits than the number of free GPIO pins, it must break the message into multiple smaller transmissions.

In each step, it can send either one bit (**serial I/O**) or several bits (**parallel I/O**).

Serial I/O is popular because it uses few wires and is fast enough for many applications.

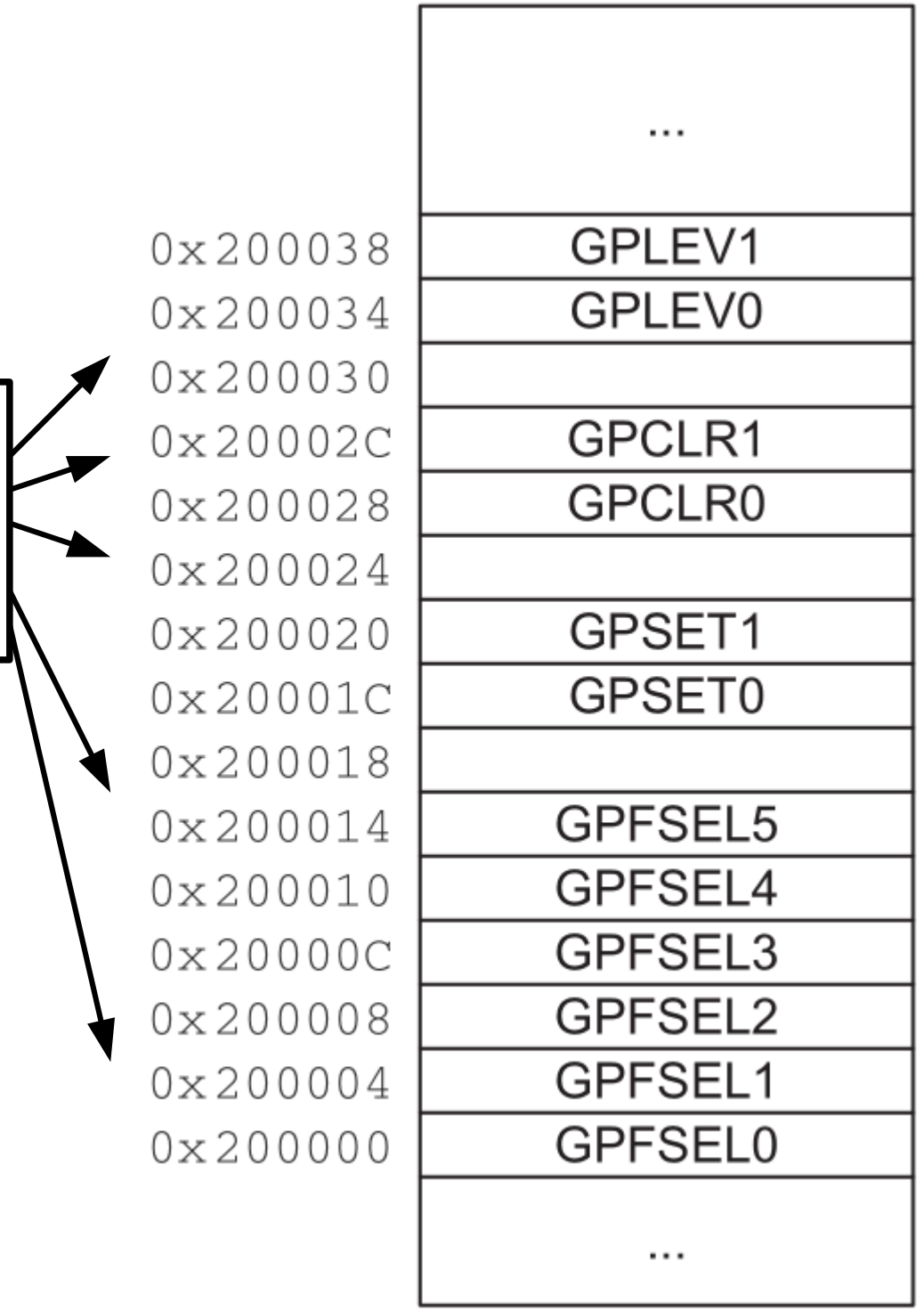
Common serial standards include

- Inter-Integrated Circuit (I^2C)
- Serial Peripheral Interface (SPI)
- Universal Asynchronous Receiver/Transmitter (UART)
- Universal Serial Bus (USB)
- Ethernet

All 5 of these are supported on the Raspberry Pi.

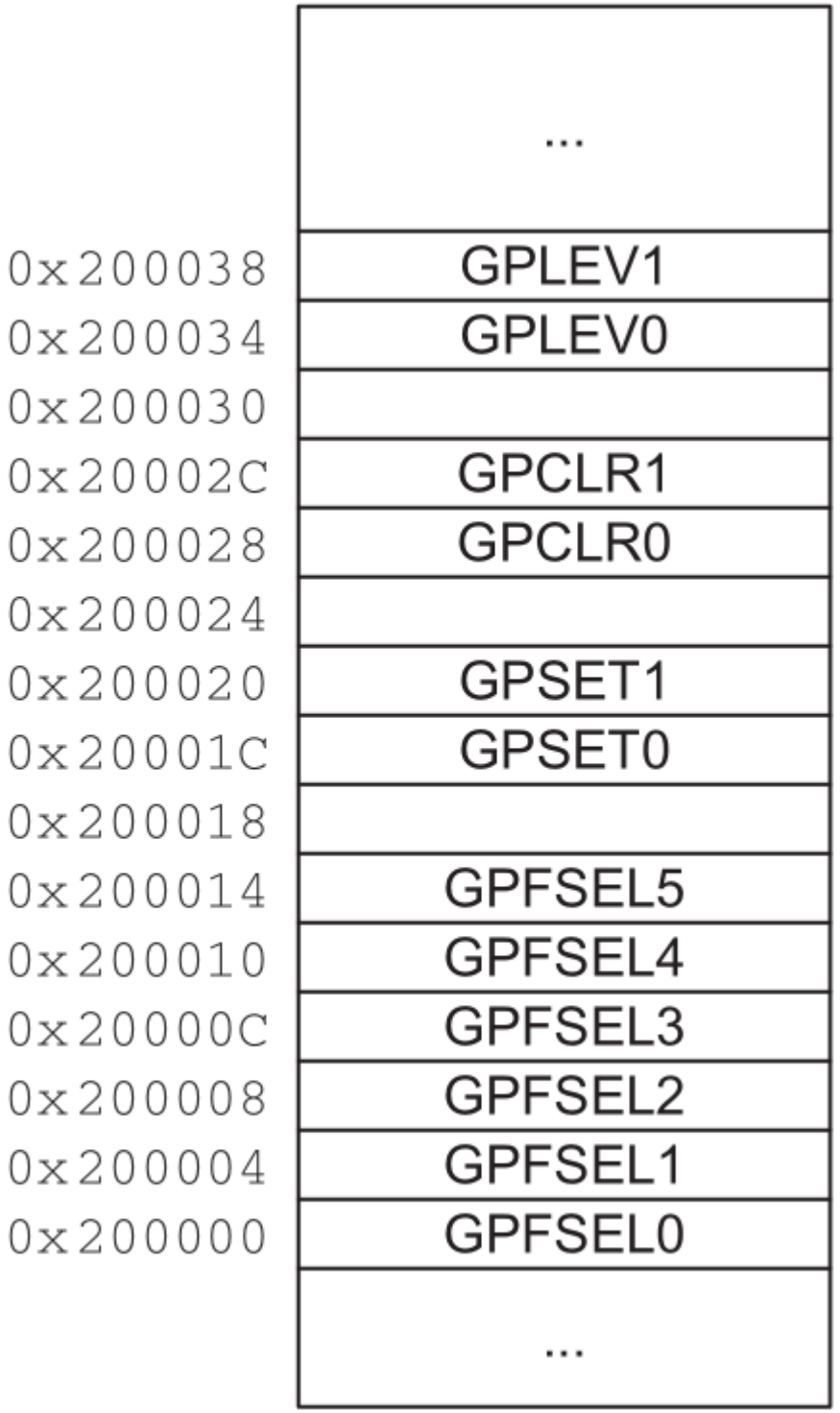
GPIOs are controlled by the GPFSEL, GPLEV, GPSET, and GPCLR registers.

Offsets from the peripheral base address (0x20000000 in Raspberry Pi; 0x3F000000 in Raspberry Pi 2)



GPFSEL5…0 determine whether each pin is a general-purpose input, output, or special-purpose I/O.

Each of these **function select registers** uses 3 bits to specify each pin and thus each 32-bit register controls 10 GPIOs.



	GPFSEL0	GPFSEL1	GPFSEL2	GPFSEL3	GPFSEL4	GPFSEL5
[2:0]	GPIO0	GPIO10	GPIO20	GPIO30	GPIO40	GPIO50
[5:3]	GPIO1	GPIO11	GPIO21	GPIO31	GPIO41	GPIO51
[8:6]	GPIO2	GPIO12	GPIO22	GPIO32	GPIO42	GPIO52
[11:9]	GPIO3	GPIO13	GPIO23	GPIO33	GPIO43	GPIO53
[14:12]	GPIO4	GPIO14	GPIO24	GPIO34	GPIO44	
[17:15]	GPIO5	GPIO15	GPIO25	GPIO35	GPIO45	
[20:18]	GPIO6	GPIO16	GPIO26	GPIO36	GPIO46	
[23:21]	GPIO7	GPIO17	GPIO27	GPIO37	GPIO47	
[26:24]	GPIO8	GPIO18	GPIO28	GPIO38	GPIO48	
[29:27]	GPIO9	GPIO19	GPIO29	GPIO39	GPIO49	

Example: GPIO13 is configured by GPFSEL1[11:9].

GPFSEL configuration

GPFSEL	Pin Function
000	Input
001	Output
010	ALT5
011	ALT4
100	ALT0
101	ALT1
110	ALT2
111	ALT3

Reading GPLEV1…0 returns the values of the pins.

Example:

GPIO14 is read as GPLEV0[14]

GPIO34 is read as GPLEV1[2]

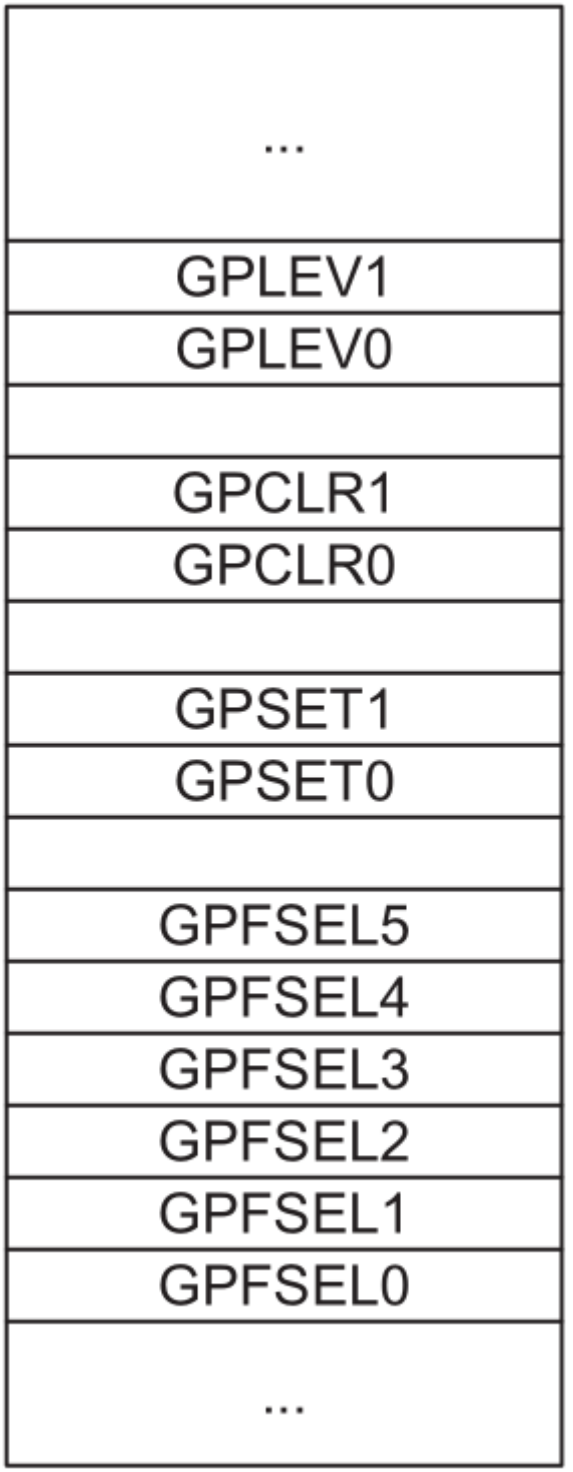
The pins cannot be directly written.

Instead, bits are forced high or low by asserting the corresponding bit of GPSET1…0 or GPCLR1…0.

Example:

GPIO14 is forced to 1 by writing GPSET0[14] =1

GPIO14 is forced to 0 by writing GPCLR0[14] =1



The functions pinMode , digitalRead , and digitalWrite in EasyPIO configure a pin's direction and read or write it.

Because multiple registers are used to control the I/O, the functions compute which register to access and what bit offset to use within the register.

pinMode then clears the 0 bits and sets the 1 bits for the intended 3-bit function.

digitalWrite handles writing either 1 or 0 by using GPSET or GPCLR.

digitalRead pulls out the value of the desired pin and masks off the others.

Some other microcontrollers use a single register to configure whether each pin is input or output and another register to read and write the pins.

The BCM2835 datasheet does not specify the logic levels or output current capability of the GPIOs.

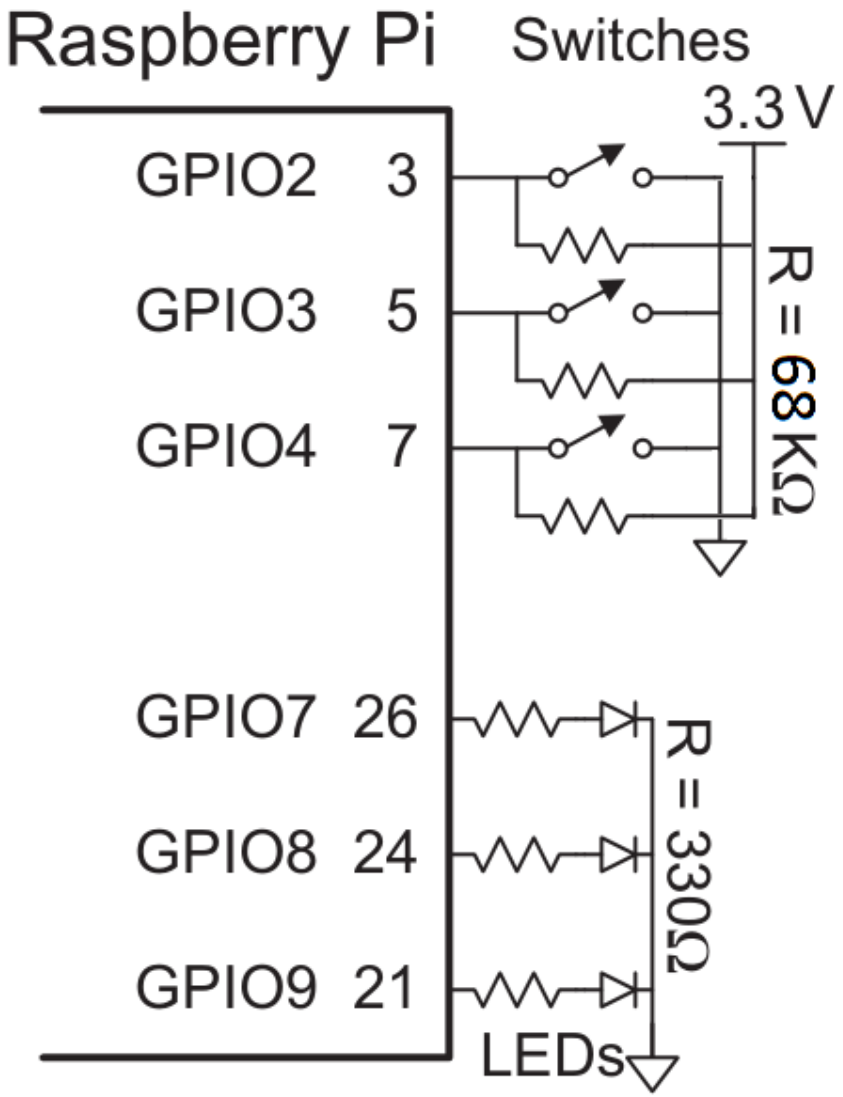
However, users have determined empirically that one should not try to draw more than 16 mA from any single I/O or 50 mA total from all the I/Os.

Thus, a GPIO pin is suitable for driving a small LED but not a motor.

The I/Os are generally compatible with other 3.3 V chips but are not 5 V-tolerant.

For security reasons, Linux only grants the superuser access to memory-mapped hardware.

To run a program as the superuser, type sudo before the Linux command.




```c
#include "EasyPIO.h"

#define PB1 2 // 8
#define PB2 3 // 9
#define PB3 4 // 7
#define LED1   9 // 13
#define LED2   8 // 10
#define LED3   7 // 11

int main (void)
{
  pioInit();

  pinMode (PB1, INPUT);
  pinMode (PB2, INPUT);
  pinMode (PB3, INPUT);
  pinMode (LED1, OUTPUT);
  pinMode (LED2, OUTPUT);
  pinMode (LED3, OUTPUT);

  For (;;) {
    digitalWrite (LED1, !digitalRead(PB1));
    digitalWrite (LED2, !digitalRead(PB2));
    digitalWrite (LED3, !digitalRead(PB3));
  }
  return 0;
}
```