

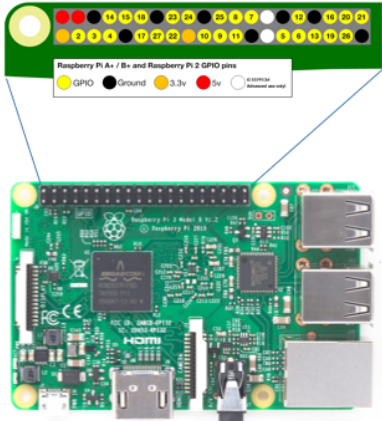


Hello and welcome to the Module 1, third video, GPIO, General Purpose Input Output. Here you learn how to use the ports for connecting devices to your Raspberry, like sensors and actuators.

"Practical IoT with Raspberry Pi"

POLITÉCNICA Universidad Politécnica de Madrid

Presentation of GPIO <http://pinout.xyz>



Alternate Function	BCM Name	Physical Pin#	Physical Pin#	BCM Name	Alternate Function
Power	3.3v	01	02	5v	Power
I ² C, SDA1	GPIO02	03	04	5v	Power
I ² C, SCL1	GPIO03	07	06	GROUND	
	GPIO04	07	08	GPIO14	UART0 TX
	GROUND	09	10	GPIO15	UART0 RX
	GPIO17	11	12	GPIO18	
	GPIO27	13	14	GROUND	
	GPIO22	15	16	GPIO23	
Power	3.3v	17	18	GPIO24	
			20	GROUND	
SPI0 MOSI	GPIO10	19	22	GPIO25	
SPI0 MISO	GPIO09	21	24	GPIO08	SPI0 CS0
SPI0 SCLK	GPIO11	23	26	GPIO07	SPI0 CS1
	GROUND	25	28	ID_SC	I ² C, ID EEPROM
I ² C, ID EEPROM	ID_SD	27	30	GROUND	
	GPIO05	29	32	GPIO12	
	GPIO06	31	34	GROUND	
	GPIO13	33	36	GPIO16	SPI1 CS0
SPI1 MISO	GPIO19	35	38	GPIO20	SPI1 MOSI
	GPIO26	37	40	GPIO21	SPI1 SCLK
	GROUND	39			

Module 1-3: GPIO Speaker: Alberto Brunete

The GPIO are these pins over here. They are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off, as inputs, or that the Pi can turn on or off as outputs. Of the 40 pins, 26 are GPIO pins (in yellow) and the others are power (red and orange) or ground pins (black) (plus two ID EEPROM pins which you should not play with unless you know your stuff!). A more detail view is on the right.

When programming the GPIO pins there are two different ways to refer to them: GPIO numbering and physical numbering.

GPIO NUMBERING, or BCM, are the GPIO pins as the computer sees them. The numbers don't make any sense to humans, they jump about all over the place, so there is no easy way to remember them.

PHYSICAL NUMBERING: refer to the pins by simply counting across and down from pin 1 at the top left (nearest to the SD card).

Beginners may find the physical numbering system simpler, but generally we recommend using the GPIO numbering. It's a good practice and most resources use

this system.

Most PINs can be configured as digital input or output. A digital input means that the Raspberry is able to know if what we have connected to the pin produces 3.3v (high value) or 0V (low value). A digital output means that the raspberry can produce 3.3V (for example to switch on a light) or 0V (to switch it off). Inputs may come from a sensor or other device. Outputs can go to an actuator as a light or motor, or other device.

GPIO voltage levels are **3.3V** and are **not 5V tolerant**. These are 3.3 volt logic pins. A voltage near 3.3 V is interpreted as a logic one while a voltage near zero volts is a logic zero. The input pins are configured by default to sink at most 8mA. Be very careful with these values!!!

Some of the PINs have special purpose, for example, serial communication, I2C communication, etc. We will see them later.

More information about the pinout can be found at this address: <http://pinout.xyz>

"Practical IoT with Raspberry Pi"

POLITÉCNICA Universidad Politécnica de Madrid

GPIO Access

How to write / read to/from pins?

> python yourfile.py

```
import RPi.GPIO as GPIO

Code goes here
...
```

yourfile.py

Module 1-3: GPIO Speaker: Alberto Brunete

So, how do we use the GPIO? It is very easy, especially programming with python. Although we will cover python programming in the next module, we will have a brief introduction here to work with the GPIO.

In order to use the GPIO, you just have to create a file in a text editor, write a header (import RPi.GPIO), write your code, and save the file as .py. Then , execute the following command in a terminal:

> python yourfile.py

And it will execute. That's all!

“Practical IoT with Raspberry Pi”

Example: switch on a LED

```

import RPi.GPIO as GPIO      # Import GPIO library

GPIO.setmode(GPIO.BOARD)    # Use GPIO pin numbering

GPIO.setup(7, GPIO.OUT)     # Setup GPIO Pin 7 to OUT

GPIO.output(7, True)        # Turn on GPIO pin 7

```

GPIO.setmode(GPIO.BOARD)

GPIO.setmode(GPIO.BCM)

Design made with Fritzing

Module 1-3: GPIO
Speaker: Alberto Brunete

Now we will see a couple of examples; switch on a LED, and detect a button pressed.

To switch on a LED, we have to connect one pin of the LED, through a resistor, to a GPIO pin of the Raspberry, and the other pin to ground. Please be very careful anytime you connect something to the Raspberry.

To specify the pin number we have two choices as explained before: GPIO numbering or physical numbering. To select any of them we use:

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


Then, we have to configure the GPIO pin as output, with the instruction " GPIO.setup(7, GPIO.OUT)".

Finally, we switching it on with "GPIO.output(7, True) "

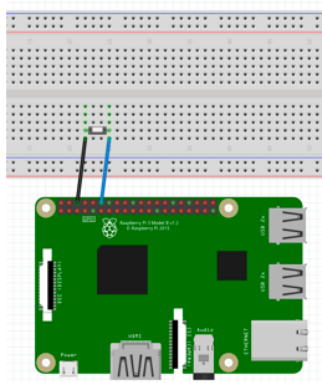
or off with `"GPIO.output(7, False) "`

We can use the same principle with any device that can be switched on and off: ventilator, motor, etc.

"Practical IoT with Raspberry Pi"



Example: connect a button



Design made with Fritzing

```

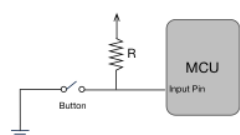
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    input_state = GPIO.input(18)
    if input_state == False:
        print('Button Pressed')
        time.sleep(0.2)

```



Module 1-3: GPIO
Speaker: Alberto Brunete

Now, we are going to use the inputs. We are going to connect a button. In this case, we connect one pin of the button to the GPIO pin, and the other to ground. When we press the button, the GPIO input will read a 0 (ground). When we release, the GPIO input will read a 1. But, will it? No!

Because when the button is not pressed, the GPIO pin is not connected (it is floating). To solve this problem, the raspberry has one thing called pull-up resistors. It connects the pin to VCC through a resistor, as we see in this diagram at the bottom of the page.

The same happens with pull-down resistors. The button is connected to power, and the GPIO pin has a pull-down resistor to ground.

The code that we use is on the screen. In this case we configure pin 18 as input, with a pull up resistor. When we press the button a message is displayed in the terminal.

We use

GPIO.PUD_DOWN for pull-down and
GPIO.PUD_UP for pull-up

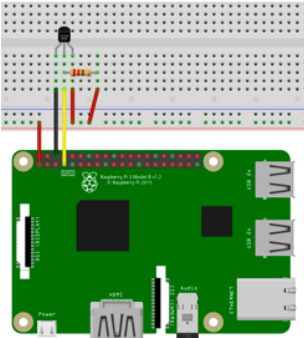
We can use the same principle with a movement detector or a window/door open/close detector.

"Practical IoT with Raspberry Pi"

POLITÉCNICA Universidad Politécnica de Madrid

Special PINs: One Wire

One Wire GPIO pin



Alternate Function	BCM Name	Physical Pin#	Physical Pin#	BCM Name	Alternate Function
Power	3.3v	01	02	5v	Power
I ² C, SDA1	GPIO02	03	04	5v	Power
I ² C, SCL1	GPIO03	07	06	GROUND	
	GPIO04	07	08	GPIO14	UART0 TX
	GROUND	09	10	GPIO15	UART0 RX
	GPIO17	11	12	GPIO18	
	GPIO27	13	14	GROUND	
	GPIO22	15	16	GPIO23	
Power	3.3v	17	18	GPIO24	
	GPIO10	19	20	GROUND	
SPI0 MOSI	GPIO09	21	22	GPIO25	
SPI0 MISO	GPIO11	23	24	GPIO08	SPI0 CS0
SPI0 SCLK	GROUND	25	26	GPIO07	SPI0 CS1
I ² C, ID EEPROM	ID_SD	27	28	ID_SC	I ² C, ID EEPROM
	GPIO05	29	30	GROUND	
	GPIO06	31	32	GPIO12	
	GPIO13	33	34	GROUND	
SPI1 MISO	GPIO19	35	36	GPIO16	SPI1 CS0
	GPIO26	37	38	GPIO20	SPI1 MOSI
	GROUND	39	40	GPIO21	SPI1 SCLK

Module 1-3: GPIO

Speaker: Alberto Brunete

A special pin is used for the 1-wire bus.

1-Wire is a device communications bus system designed by Dallas Semiconductor that provides low-speed data, signalling, and power over a single conductor. 1-Wire is similar in concept to I²C, but with lower data rates and longer range. It is typically used to communicate with small inexpensive devices such as digital thermometers and weather instruments.

In 1-Wire sensors, all data is sent through one wire, which makes it great for microcontrollers such as the Raspberry Pi, as it only requires one GPIO pin for sensing. In addition to this, most 1-Wire sensors will come with a unique serial code, which means you can connect multiple units up to one microcontroller without them interfering with each other.

In the raspberry, One Wire works in pin 7, GPIO04.

We will use it for connecting temperature sensors, as in the image. We will learn more on this in module 4.

"Practical IoT with Raspberry Pi"

POLITÉCNICA Universidad Politécnica de Madrid

Special communications PINs: I2C, SPI, UART

Alternate Function	BCM Name	Physical Pin#	Physical Pin#	BCM Name	Alternate Function
Power	3.3v	01	02	5v	Power
I ² C, SDA1	GPIO02	03	04	5v	Power
I ² C, SCL1	GPIO03	07	06	GROUND	
	GPIO04	07	08	GPIO14	UART0 TX
	GROUND	09	10	GPIO15	UART0 RX
	GPIO17	11	12	GPIO18	
	GPIO27	13	14	GROUND	
	GPIO22	15	16	GPIO23	
Power	3.3v	17	18	GPIO24	
SPI0 MOSI	GPIO10	19	20	GROUND	
SPI0 MISO	GPIO09	21	22	GPIO25	
SPI0 SCLK	GPIO11	23	24	GPIO08	SPI0 CS0
	GROUND	25	26	GPIO07	SPI0 CS1
I ² C, ID EEPROM	ID_SD	27	28	ID_SC	I ² C, ID EEPROM
	GPIO05	29	30	GROUND	
	GPIO06	31	32	GPIO12	
	GPIO13	33	34	GROUND	
SPI1 MISO	GPIO19	35	36	GPIO16	SPI1 CS0
	GPIO26	37	38	GPIO20	SPI1 MOSI
	GROUND	39	40	GPIO21	SPI1 SCLK

Module 1-3: GPIO

Speaker: Alberto Brunete

Other special pins are used for serial communications buses: I2C, SPI and UART. All of them are used to send data to other devices.

Serial Communication is an umbrella word for all that is "Time Division Multiplexed". It means that the data is sent spread over time, most often one single bit after another.

We have

UART, for Universal Asynchronous Receiver Transmitter, one of the most used serial protocols.

SPI (Serial Peripheral Interface), another very simple serial protocol. Synchronous and Master-Slave.

I2C (Inter-Integrated Circuit, pronounced "I squared C") also a synchronous protocol, and master slave, but it uses only 2 wires.

You will use one or another depending on the device you are connecting to.

The BCM2837 on the Raspberry Pi3 has 2 UARTs (as did its predecessors), however to support the Bluetooth functionality the fully featured PL011 UART was moved from the header pins to the Bluetooth chip and the mini UART made available on header pins 8 & 10.

This has a number of consequences for users of the serial interface.

The `/dev/ttyAMA0` previously used to access the UART now connects to Bluetooth.

The miniUART is now available on `/dev/ttyS0`.

In the latest operating system software there is a `/dev/serial0` which selects the appropriate device so you can replace `/dev/ttyAMA0` with `/dev/serial0` and use the same software on the Pi3 and earlier models.

Unfortunately there are a number of other consequences:-

The mini UART is a secondary low throughput UART


intended to be used as a console.



The mini Uart has the following features:

- 7 or 8 bit operation.
- 1 start and 1 stop bit.
- No parities.
- Break generation.
- 8 symbols deep FIFOs for receive and transmit.
- SW controlled RTS, SW readable CTS.
- Auto flow control with programmable FIFO level.
- 16550 like registers.
- Baudrate derived from system clock.

There is no support for parity and the throughput is limited, but the latter should not affect most uses.

There is one killer feature "Baudrate derived from system clock" which makes the miniUART useless as the this clock can change dynamically e.g. if the system goes into reduced power or in low power mode.



"Practical IoT with Raspberry Pi"
Universidad Politécnica de Madrid

Conclusion

- GPIO Access
- General Purpose IO Pins
- Protocol Pins (I2C, 1-wire)
- Demos

Module 1-3: GPIO

Speaker: Alberto Brunete

SO, that's all. In this lesson we have learned how to use the input and output ports of our Raspberry, the GPIO. How to access them, how to configure them, the different protocols that are available and that we will use in Module 4, and we have made 2 examples to introduce the GPIO.

I hope you like it and see you in next lesson!