Part 01
-
Gathering Information Hardware

Version: 2022-09-08

What is a Raspberry Pi?

Created by the Raspberry Pi Foundation, the Raspberry Pi is an open-source, Linux based, credit card sized computer board. The Pi is an exciting and accessible means of improving computing and programming skills for people of all ages. By connecting to your TV or monitor and a keyboard, and with the right programming, the Pi can do many things that a desktop computer can do such as surf the internet and play video. The Pi is also great for those innovative projects that you want to try out - newer models are ideal for Internet of Things (IoT) projects due to their processing power.
With Pi 3, Wireless LAN and Bluetooth Low Energy are on-board too.

Here (https://www.raspberrypi.org/products/) you find detailed information about each available model.

What are the differences between the models?

Pi	Soc Family	SoC	CPU	# Cores	Speed in MHz	L2 Cache in KB	GPU	RAM	USB Ports	Ethernet	GPIO	Wireless	Bluetooth	Dimensions in mm
A	BCM2708	BCM2835	ARM11	1	700	128	Videocore IV Sys@400MHz 3D@250MHz	256MB	1 x USB 2.0	No	26	No	No	85 x 56 x 15
B	BCM2708	BCM2835	ARM11	1	700	128	Videocore IV Sys@400MHz 3D@250MHz	512MB	4 x USB 2.0	10/100Base-T	26	No	No	85 x 56 x 17
A+	BCM2708	BCM2835	ARM11	1	700	128	Videocore IV Sys@400MHz 3D@250MHz	512MB	1 x USB 2.0	No	40	No	No	66 x 56 x 14
B+	BCM2708	BCM2835	ARM11	1	700	128	Videocore IV Sys@400MHz 3D@250MHz	512MB	4 x USB 2.0	10/100Base-T	40	No	No	85 x 56 x 17
2B	BCM2709	BCM2836/7	ARM Cortex-A7	4	900	256	Videocore IV Sys@400MHz 3D@250MHz	1GB	4 x USB 2.0	10/100Base-T	40	No	No	85 x 56 x 17
3B	BCM2710	BCM2837	ARM Cortex-A53	2	1200	256	Videocore IV Sys@400MHz 3D@250MHz	1GB	4 x USB 2.0	10/100Base-T	40	802.11n	4.1	85 x 56 x 17
3 A+	BCM2710	BCM2837B0	ARM Cortex-A53		1400	256	Videocore IV Sys@400MHz 3D@250MHz	512MB	1 x USB 2.0	No	40	802.11ac/n	4.2	66 x 56 x 14
3 B+	BCM2710	BCM2837B0	ARM Cortex-A53	4	1400	256	Videocore IV Sys@400MHz 3D@250MHz	1GB	4 x USB 2.0	1000Base-T	40	802.11ac/n	4.2	85 x 56 x 17
4 B	BCM2711		ARM Cortex-A72	4	1500	256	Videocore IV Sys@400MHz 3D@250MHz	1GB	2xUSB2, 2xUSB3	1000Base-T	40	802.11ac/n	5.0	85 x 56 x 17
4 B	BCM2711		ARM Cortex-A72	4	1500	256	Videocore IV Sys@400MHz 3D@250MHz	2GB	2xUSB2, 2xUSB3	1000Base-T	40	802.11ac/n	5.0	85 x 56 x 17
4 B	BCM2711		ARM Cortex-A72	4	1500	256	Videocore IV Sys@400MHz 3D@250MHz	4GB	2xUSB2, 2xUSB3	1000Base-T	40	802.11ac/n	5.0	85 x 56 x 17
Zero	BCM2708	BCM2835	ARM 11	1	1000	128	Videocore IV Sys@400MHz 3D@250MHz	512MB	1 x microUSB	No	40	No	No	67 x 30 x 5
Zero W	BCM2708	BCM2835	ARM 11	1	1000	128		512MB	1 x microUSB	No	40	802.11n	4.1	67 x 30 x 5
Zero WH	BCM2708	BCM2835	ARM 11	1	1000	128		512MB	1 x microUSB	No	40	802.11n	4.1	67 x 30 x 5
Compute	BCM2708	BCM2835	ARM11	1	700	128	Videocore IV Sys@400MHz 3D@250MHz	512MB	No	No	No	NO	No	67.5 x 30

Actually the discrepancy between SoC Family and SoC is due to the designation of the silicon and the chip package. Originally there was the silicon die which is known as BCM2708, all initial development was done around this. In a stacked 9x9 package with 256MB of DRAM it is then known as BCM2763. (Stacked is when you literally bond the DRAM silicon on top of the processor and put bond wires down onto the substrate). But when the memory is POP'd (package on package, the DRAM package is attached to the top of the processor package) then it was known as BCM2835, this is the device that then had the ARM enabled. Later versions of the chip follow the same scheme, there are now several pieces of silicon like BCM2708, BCM2709, BCM2710, ... and several packages BCM2835, BCM2836 , BCM2837, ...

How do I get connected?

To get started with your Pi you will need at minimum:

- A power supply
- An SD card with the latest version of New Out Of Box Software (NOOBS), to install the operating system that you would like to use.
- A wired network and hence a network cable or just a Wireless network
- Another computer

If you want to use the Pi as a real computer you will need this extra stuff

- A monitor or TV screen
- A keyboard
- A mouse

To get sound and video you will need cables to suit what your screen or monitor accepts. For those with monitors that accept VGA, a HDMI to VGA adaptor is needed in addition to a HDMI cable, unless you use the composite video output from the Pi.

For an internet connection, the Pi B+ and Pi 2 B have an ethernet port. You also have the option of adding a WiFi Adapter/dongle which may mean that you need a USB Hub if you have run out of USB ports.
The Pi 3 and above already have 802.11 b/g/n wireless LAN and Bluetooth 4.1

Powering Pi

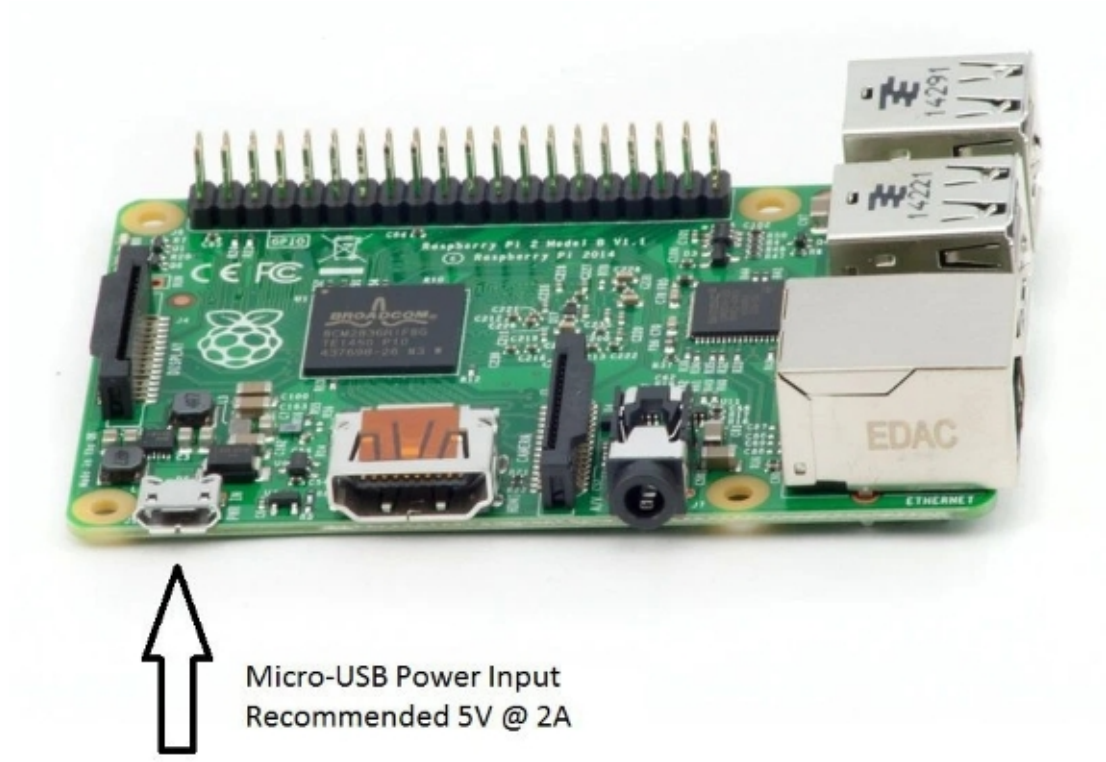
The Pi has a 5 V micro-USB power socket, located on the bottom left hand corner of your Pi board. Although you will always reads '5V power supply' to work without any problems, 5.1V is required. Hence a power supply of Raspberry Pi itself outputs 5.1V. Your Pi will works with 5.0V but may not be a stable as you expect it.

Product	Recommended PSU current capacity	Connector	Maximum total USB peripheral current draw	Typical bare-board active current consumption
A	700mA	MicroUSB 2	500mA	200mA
B	1.2A	MicroUSB 2	500mA	500mA
A+	700mA	MicroUSB 2	500mA	180mA
B+	1.8A	MicroUSB 2	600mA/1.2A (switchable)	330mA
2B	1.8A	MicroUSB 2	600mA/1.2A (switchable)	350mA
3B	2.5A	MicroUSB 2	1.2A	400mA
3A+	2.5A	MicroUSB 2	Limited by PSU, board, and connector ratings only	350mA
3B+	2.5A	MicroUSB 2	1.2A	500mA
4B	3.0A	MicroUSB 3	1.2A	600mA
Zero W/WH	1.2A	MicroUSB 2	Limited by PSU, board, and connector ratings only	150mA
Zero	1.2A	MicroUSB 2	Limited by PSU, board, and connector ratings only	100mA

Generally, the more USB ports and interfaces you use on your Pi, the more power you are going to need.

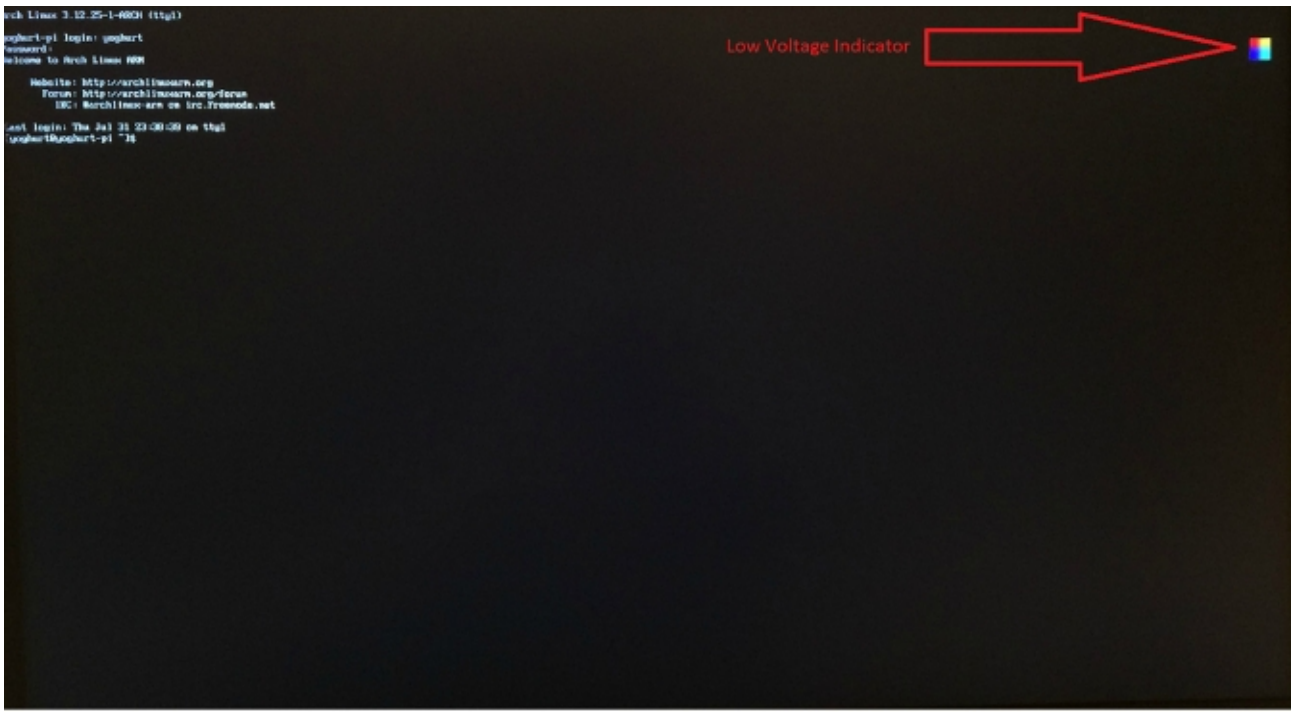
We advise to look at buying a powered USB hub - this means less pressure on your Pi whilst still being able to incorporate all the features and functionality that you want to. When connecting any devices to your Pi, it is advisable to always check the power rating.

How do I power my Raspberry Pi?



The least complicated, but most stringent in terms of power requirement is the Raspberry Pi! As of the Pi 2, it can be powered effectively in a two ways. Please note. Unlike the original Raspberry Pi Model B, the Raspberry Pi 2 Model B cannot be back-powered via the USB ports (or at least can't be booted in this fashion).

Mode 1 – Micro USB Port (5V)
The first, recommended and easiest way to power the Raspberry Pi is via the Micro USB port on the side of the unit. The recommended input voltage is 5.1V, and use the recommended input current.
The Raspberry Pi can function on lower current power supplies e.g. 5V @ 1A. However, any excessive use of the USB ports or even heavy CPU/GPU loading can cause the voltage to drop, and instability during use. The latest versions of the Raspberry Pi have a "low voltage indicator icon" to notify the user if there is a problem with the power. This is demonstrated below:

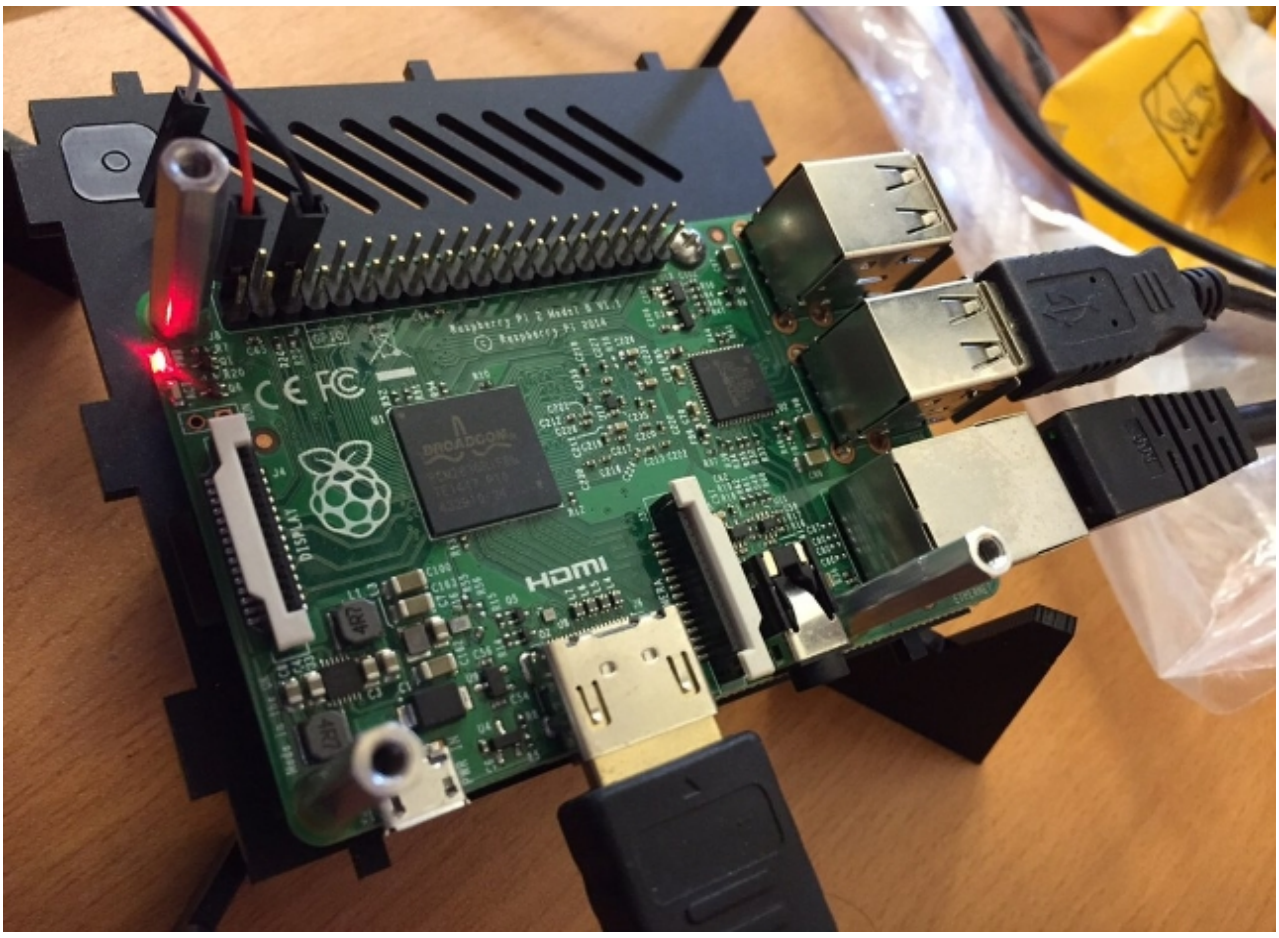


Most "standard" 5V Micro-USB mobile phone, tablet, and digital camera chargers should work with the Raspberry Pi, although we recommend that you utilise a high quality dedicated Raspberry Pi power supply to get the best results!

Mode 2 – Via the GPIO
A more technical (and of course dangerous) way to power the Pi is directly via the GPIO. It should be noted that, unlike the Micro-USB port, there is no regulation or fuse protection on the GPIO to protect from over-voltage or current spikes. If an incorrect voltage is applied, or a current spike occurs on the line you can permanently damage your Raspberry Pi. At best, you'll "burn out" some or all of the GPIO pins, at worst you can fry your Pi! So be careful.
To power via GPIO, you only need to connect 2 pins:

- Connect a 5V source to Pin #2 (5V)
- Connect the ground of that source to Pin #6 (GND).

A simple example of this is demonstrated in the image below. Just hook up the 5V to Pin #2 (Red cable), and the ground to Pin #6 (Black cable).
This process would of course be the same for a spliced USB to Micro USB cable, however we would recommend that you regulate the line voltage and current to negate any nasty stuff!
This method is useful for a range of applications, and a number of breakout boards offer this powering functionality via the GPIO using battery supplies. We therefore recommend that powering via the GPIO only be achieved via a protected source. An excellent example of powering via the GPIO is our own UPS PiCo HAT, an uninterruptible power supply .



Undervoltage warning
If the power supply to the Raspberry Pi drops below 4.63V (+/-5%), the following icon is displayed, if display is attached.

`vcgencmd` is a command line utility that can get various pieces of information from the VideoCore GPU on the Raspberry Pi. One of the command line option is `get_throttled` which returns the throttled state of the system. This is a bit pattern with a bit being set indicates the following meanings:

Bit	Hex value	Meaning
0	0x1	Under-voltage detected
1	0x2	Arm frequency capped
2	0x4	Currently throttled
3	0x8	Soft temperature limit active
16	0x10000	Under-voltage has occurred
17	0x20000	Arm frequency capping has occurred
18	0x40000	Throttling has occurred
19	0x80000	Soft temperature limit has occurred

A value of zero indicates that none of the above conditions is true.
To find if one of these bits has been set, convert the value returned to binary, then number each bit along the top. You can then see which bits are set. For example:

```
vcgencmd get_throttled
```

returns 0x50000

```
0x50000 = 0101 0000 0000 0000 0000
```

Adding the bit numbers along the top we get:

```
19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
 0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

From this we can see that bits 18 and 16 are set, indicating that the Pi has previously been throttled due to under-voltage, but is not currently throttled for any reason.
Alternately, the values can be derived using the hex values above, by successively subtracting the largest value:

```
0x50000 = 0x40000 + 0x10000
```

More on vcgencmd

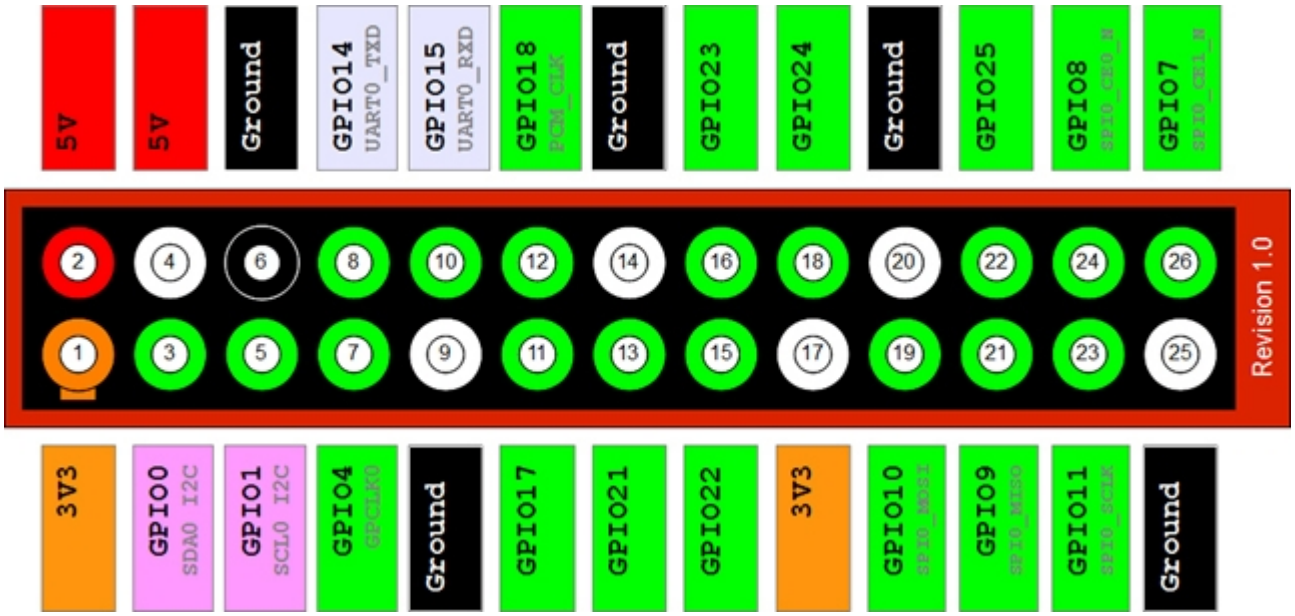
```
vcgencmd measure_volts [block]
```

Displays the current voltages used by the specific block.

Block	Description
core	VC4 core voltage
sdram_c	SDRAM Core Voltage
sdram_i	SDRAM I/O voltage
sdram_p	SDRAM Phy Voltage

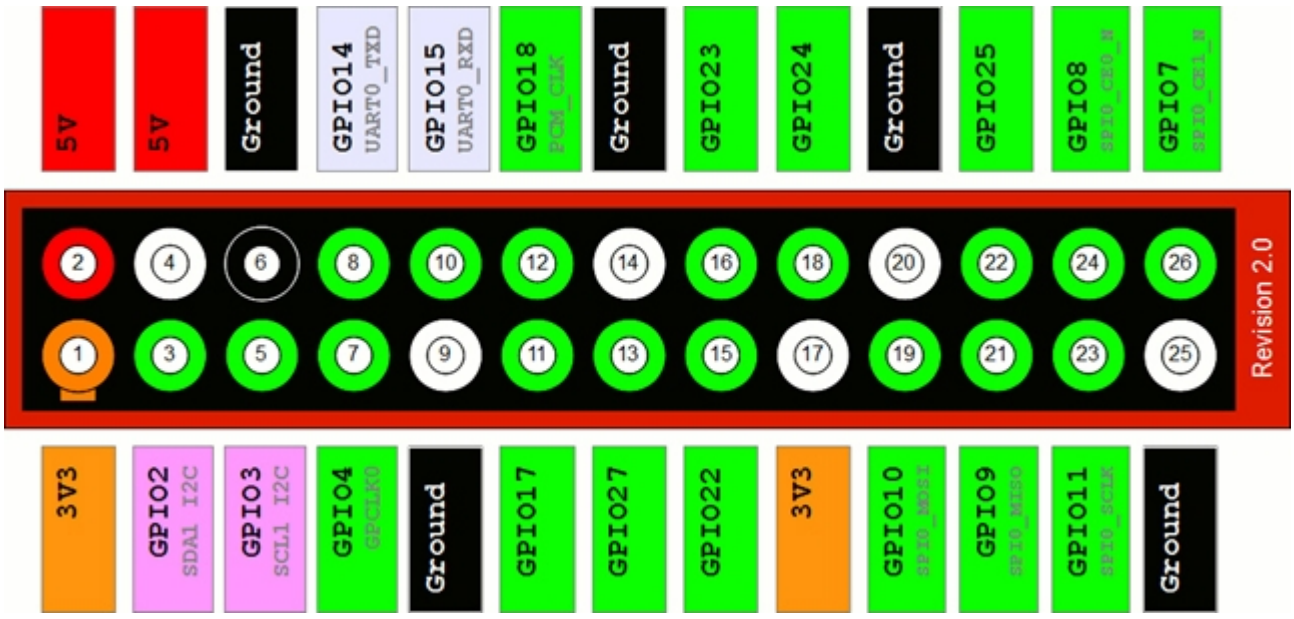
Raspberry Pi GPIO Connector 26 pin layout (Model B Revision 1)

P1 Header - 26 Pins									
Name		Number						Name	
Alt	Base	wPi	BCM	Board	Board	BCM	wPi	Base	Alt
	+3V3			1	2			+5V	
I2C1_SDA	GPIO0	8	0	3	4			+5V	
I2C1_SCL	GPIO1	9	1	5	6			0V-GND	
	GPIO4	7	4	7	8	14	15	GPIO14	UART_TxD
	GND-0V			9	10	15	16	GPIO15	UART_RxD
	GPIO17	0	17	11	12	18	1	GPIO18	PWM
	GPIO27	2	27	13	14			0V-GND	
	GPIO22	3	22	15	16	23	4	GPIO23	
	+3V3			17	18	24	5	GPIO24	
SPI_MOSI	GPIO10	12	10	19	20			0V-GND	
SPI_MISO	GPIO9	13	9	21	22	25	6	GPIO25	
SPI_SCLK	GPIO11	14	11	23	24	8	10	GPIO8	SPI_CE0
	GND-0V			25	26	7	11	GPIO7	SPI_CE1



Raspberry Pi GPIO Connector 26 pin layout (Model A & B Revision 2)

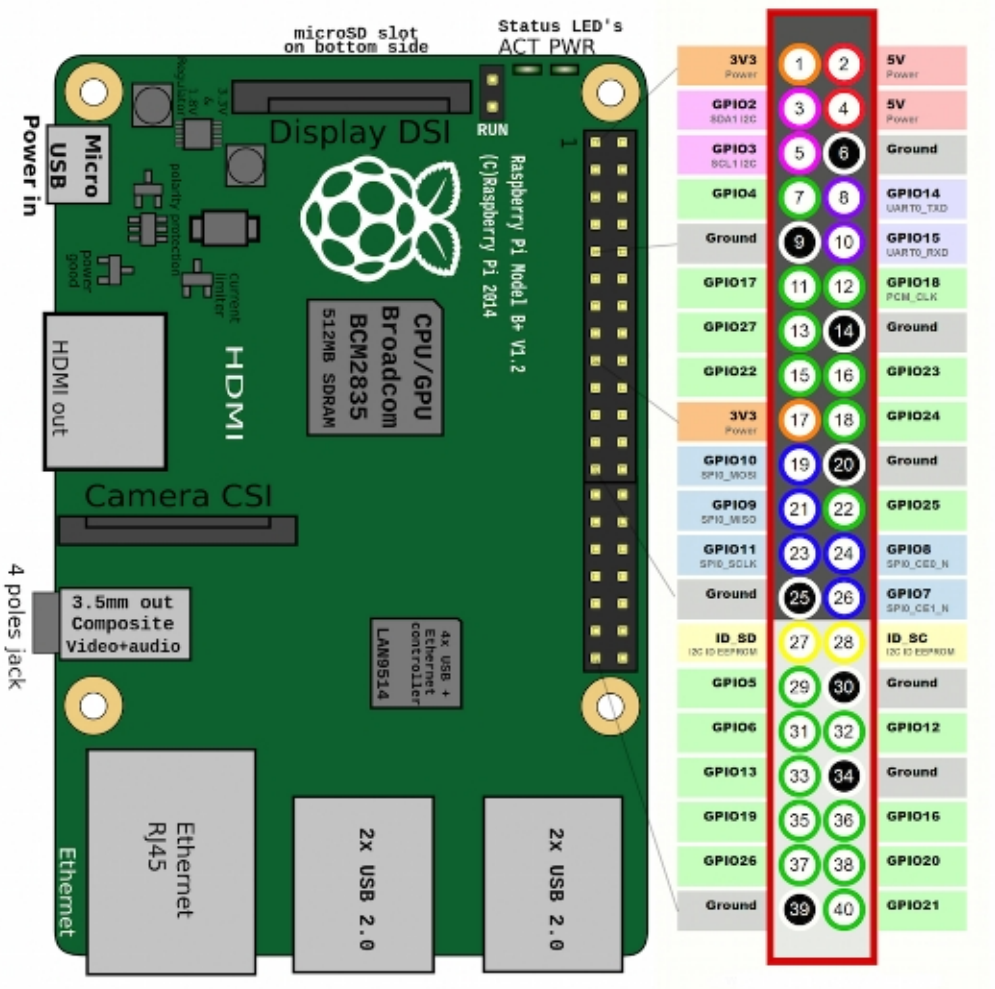
P1 Header - 26 Pins									
Name		Number						Name	
Alt	Base	wPi	BCM	Board	Board	BCM	wPi	Base	Alt
	+3V3			1	2			+5V	
I2C1_SDA	GPIO2	8	2	3	4			+5V	
I2C1_SCL	GPIO3	9	3	5	6			0V-GND	
	GPIO4	7	4	7	8	14	15	GPIO14	UART_TxD
	GND-0V			9	10	15	16	GPIO15	UART_RxD
	GPIO17	0	17	11	12	18	1	GPIO18	PWM
	GPIO27	2	27	13	14			0V-GND	
	GPIO22	3	22	15	16	23	4	GPIO23	
	+3V3			17	18	24	5	GPIO24	
SPI_MOSI	GPIO10	12	10	19	20			0V-GND	
SPI_MISO	GPIO9	13	9	21	22	25	6	GPIO25	
SPI_SCLK	GPIO11	14	11	23	24	8	10	GPIO8	SPI_CE0
	GND-0V			25	26	7	11	GPIO7	SPI_CE1
P5 Header - 8 Pins									
Name		Pin Number						Name	
Alt	Base	wPi	BCM	Board	Board	BCM	wPi	Base	Alt
	+5V			1	2			+3V3	
	GPIO17	17	28	3	4	29	18	GPIO29	
	GPIO19	19	30	5	6	31	20	GPIO31	
	GND-0V			7	8			0V-GND	

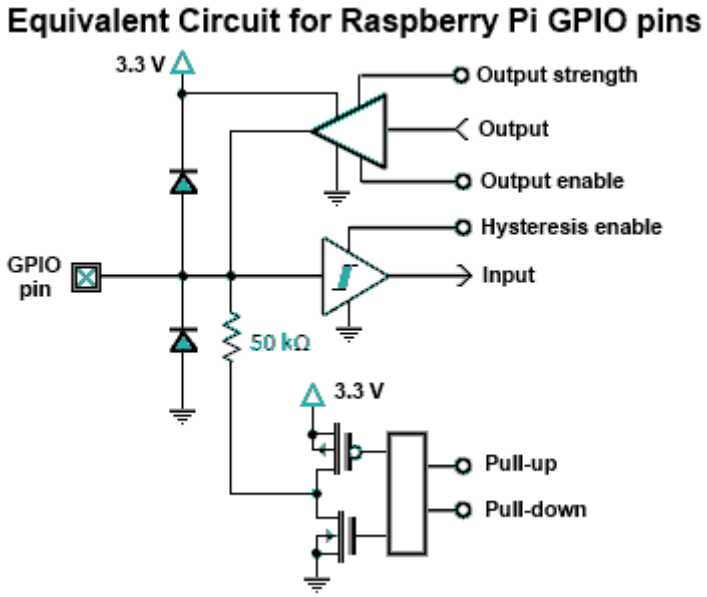


Raspberry Pi GPIO Connector 40 pin layout

P1 Header - 40 Pins									
Name		Number						Name	
Alt	Base	wPi	BCM	Board	Board	BCM	wPi	Base	Alt
	+3V3			1	2			+5V	
I2C1_SDA	GPIO2	8	2	3	4			+5V	
I2C1_SCL	GPIO3	9	3	5	6			0V-GND	
	GPIO4	7	4	7	8	14	15	GPIO14	UART_TxD
	GND-0V			9	10	15	16	GPIO15	UART_RxD
	GPIO17	0	17	11	12	18	1	GPIO18	PWM PCM_CLK
	GPIO27	2	27	13	14			0V-GND	
	GPIO22	3	22	15	16	23	4	GPIO23	
	+3V3			17	18	24	5	GPIO24	
SPI_MOSI	GPIO10	12	10	19	20			0V-GND	
SPI_MISO	GPIO9	13	9	21	22	25	6	GPIO25	
SPI_SCLK	GPIO11	14	11	23	24	8	10	GPIO8	SPI_CE0
	GND-0V			25	26	7	11	GPIO7	SPI_CE1
I2C0_SDA	GPIO0	30	0	27	28	1	31	GPIO1	I2C0_SCL
	GPIO5	21	5	29	30			0V-GND	
	GPIO6	22	6	31	32	12	26	GPIO12	
	GPIO13	23	13	33	34			0V-GND	
PCM_FS	GPIO19	24	19	35	36	16	27	GPIO16	
	GPIO26	25	26	37	38	20	28	GPIO20	PCM_DIN
	GND-0V			39	40	21	29	GPIO21	PCM_DOUT

NB: PCM pins also valid for I2S. Eg: PCM_CLK = I2S_BCLK, PCM_FS = I2S_LRCLK, PCM_DIN = I2S_DIN, PCM_DOUT = I2S_DOUT





You should keep the following limitations in mind when using the GPIO pins:

- 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. A GPIO pin should never be connected to a voltage source greater than 3.3V or less than 0V, as prompt damage to the chip may occur as the input pin substrate diodes conduct. There may be times when you may need to connect them to out-of-range voltages – in those cases the input pin current must be limited by an external resistor to a value that prevents harm to the chip. I recommend that you never source or sink more than 0.5 mA into an input pin.

- To prevent excessive power dissipation in the chip, you should not source/sink more current out of the pin than its programmed limit. So, if you have set the current capability to 2 mA, do not draw more than 2 mA from the pin.

- Never demand that any output pin source or sink more than 16 mA.

- Current sourced by the outputs is drawn from the 3.3 V supply, which can supply only 50 mA maximum. Consequently, the maximum you can source from all the GPIO outputs simultaneously is less than 50 mA. You may be able to draw transient currents beyond that limit as they are drawn from the bypass capacitors on the 3.3 V rail, but don't push the envelope!

- There isn't a similar limitation on sink current. For sink current, the pertinent limitation is that imposed by maximum chip power dissipation. Even so, you can safely sink up to 16 mA each into any number of GPIO pins simultaneously. In the worst case, the output pins (if configured to the 16mA high current drive capability) have a maximum output low voltage of about 0.4 V, and and their internal circuitry dissipates only 6.4 mW worst case. Even sinking 16 mA into 16 pins simultaneously would produce only 0.1024 W, that is, about a tenth of a watt. However, depending on the source of the current, transient sink currents may make demands on the board's bypass capacitors, so you may not want to switch all the outputs to sink the maximum current synchronously, if you require fast, clean transitions.

- Do not drive capacitive loads. Do not place a capacitive load directly across the pin. Limit current into any capacitive load to a maximum transient current of 16 mA. For example, if you use a low pass filter on an output pin, you must provide a series resistance of at least 3.3V/16mA = 200 Ω.

GPIO input/output pin electrical characteristics	
Output low voltage V_{OL}	< 0.40 V < 0.66 V < 0.40 V < 0.40 V
Output high voltage V_{OH}	> 2.40 V > 2.64 V > 2.90 V
Input low voltage V_{IL}	< 0.80 V < 0.54 V < 1.15 V
Input high voltage V_{IH}	> 2.00 V > 2.31 V > 2.15 V
Hystereses	> 0.25 V 0.66 – 2.08 V
Schmitt trigger input low threshold V_{T-}	1.09 - 1.16 V 0.9
Schmitt trigger input high threshold V_{T+}	2.24 - 2.74 V 0.90 V
Pull-up/down resistance	40 – 65 KΩ 100 KΩ
Pull-up/down current	< 50 uA < 28 uA
Pin capacitance	5 pF
Bus hold resistance	5-11 KΩ

Raspberry Pi's GPIOs Default State

So, you've been playing with your Raspberry Pi and the GPIO panel, and you've noticed that when you read the GPIOs, some of them give you the value HIGH (or 1) by default, some of them give you the value LOW (0). Why is that?

To control the GPIOs we will use the RPi.GPIO Python module.

GPIOs default state for Raspberry Pi
First, let's define exactly what is the "default state". The default state for GPIO applies with the following conditions. You've:

1. Boot the Raspberry Pi.
2. Set a GPIO as input.
3. Read the GPIO's state.

And you've not:

1. Added any external pull up/down resistor.
2. Used the GPIO.cleanup() before.
3. Configured the pin as output before setting it as input.

If you've done any of the previous things, you'll have to reboot your Pi in order to get the default states again.

Default states for Raspberry Pi's GPIOs
So, provided that you've followed the 3 steps and not done any of the "don't do" steps, you will see the following:

- GPIOs up to 8: default state is 1 (HIGH, or close to 3.3V).
- GPIOs 9 to 27: default state is 0 (LOW, or close to 0V).

Here's a Python code you can run on your Raspberry Pi to test yourself.

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIOs = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27]
# Setup all GPIOs to input
for gpio in GPIOs:
GPIO.setup(gpio, GPIO.IN)
# Read state for each GPIO
for gpio in GPIOs:
print("GPIO no " + str(gpio) + ": " + str(GPIO.input(gpio)))
# Cleanup all GPIOs - state will be different after that!
GPIO.cleanup()
```

In this code you can see an array containing all the GPIOs we are using.

Then, we set the mode for each GPIO to input – GPIO.IN, and we read the state for the GPIO with GPIO.input(gpio). We print that state along with the GPIO number.
So, the first time we run this program, we get:

```
GPIO no 2: 1
GPIO no 3: 1
GPIO no 4: 1
GPIO no 5: 1
GPIO no 6: 1
GPIO no 7: 1
GPIO no 8: 1
GPIO no 9: 0
GPIO no 10: 0
GPIO no 11: 0
GPIO no 12: 0
GPIO no 13: 0
GPIO no 16: 0
GPIO no 17: 0
GPIO no 18: 0
GPIO no 19: 0
GPIO no 20: 0
GPIO no 21: 0
GPIO no 22: 0
GPIO no 23: 0
GPIO no 24: 0
GPIO no 25: 0
GPIO no 26: 0
GPIO no 27: 0
```

As you can see, GPIOs up to 8 get the state 1 (HIGH, or close to 3.3V), and GPIOs from 9 to 27 get the state 0 (LOW, or close to 0V).

And then, one thing we do at the end of the program is to call the GPIO.cleanup() function. This function will reset the mode of all pins to input, to avoid letting some pins as output, which can be dangerous for the Pi. And – warning! – this will also alter the future state (which will not be "default" anymore) of the GPIOs in input mode.
So, if you run this program a second time, you will get:

```
GPIO no 2: 1
GPIO no 3: 1
GPIO no 4: 1
GPIO no 5: 1
GPIO no 6: 1
GPIO no 7: 1
GPIO no 8: 1
GPIO no 9: 1
GPIO no 10: 1
GPIO no 11: 1
GPIO no 12: 1
GPIO no 13: 1
GPIO no 16: 1
GPIO no 17: 1
GPIO no 18: 1
GPIO no 19: 1
GPIO no 20: 1
GPIO no 21: 1
GPIO no 22: 1
GPIO no 23: 1
GPIO no 24: 1
GPIO no 25: 1
GPIO no 26: 1
GPIO no 27: 1
```

This is because the function GPIO.cleanup() was executed during the first program run. Now if you want to get the default states again, you'll have to reboot your Pi. Also note that if you only plan to use GPIOs as input, you don't need to call GPIO.cleanup(). This function is only useful when you have GPIOs set as output mode.
Another thing: if you have set your GPIOs to output mode, and then to input, chances are that you're going to read 1 (HIGH) for all GPIOs.

As you can see, the default state is only after you boot your Pi, under a certain set of circumstances. Then, it's quite probable that this "default" state will be modified in your future programs.

Why is the GPIO default state like that?
First, on page 102 of the Broadcom 2835 datasheet (Broadcom 2835 is the more technical name for the GPIO header), check the second column named "Pull". In this column, for each GPIO, you will see either HIGH or LOW.
This value is also the same you found when running the code: HIGH for GPIOs up to 8, and LOW for GPIOs starting from 9.
Here, HIGH means that there is a default pull up resistor on the pin, and LOW means there is a default pull down resistor on the pin. I'm not going to explain everything about pull up and down resistors, just this:

- When you add a resistor between a component and the ground, this is a pull down resistor, which will make sure the default voltage you read is close to 0V.
- When you add a resistor between a component and Vcc (power supply), this is a pull up resistor, which will make sure the default voltage you read is close to Vcc (3.3V for the Raspberry Pi)

So, what this tells you is that for GPIOs 0 to 8, there is an internal pull up resistor, which will make you read 1 (HIGH) by default. For GPIOs 9+, there is an internal pull down resistor, which will make you read 0 (LOW) by default.

Now, what is the value of this pull up/down resistor?
Unfortunately, it's not possible to know exactly. The actual Raspberry Pi hardware is not open source (most of the software is, not the hardware), so there is no way to know directly from the Raspberry Pi Foundation. However, with some tests published by users on the Internet, we can approximate this internal resistor to be 50kOhm.

How to "override" the default state for Raspberry Pi's GPIOs
Now let's have a look at 2 different ways you can override this default state for the GPIOs. You may be fine with the default state, but you could also want to set the default state to something else (all HIGH, all LOW, or a mix of both).

Override the default state with the code
Using the RPi.GPIO Python module, you can choose to make the internal resistor as a pull up, or a pull down resistor, for any given GPIO.
Here's a code example with just one GPIO.

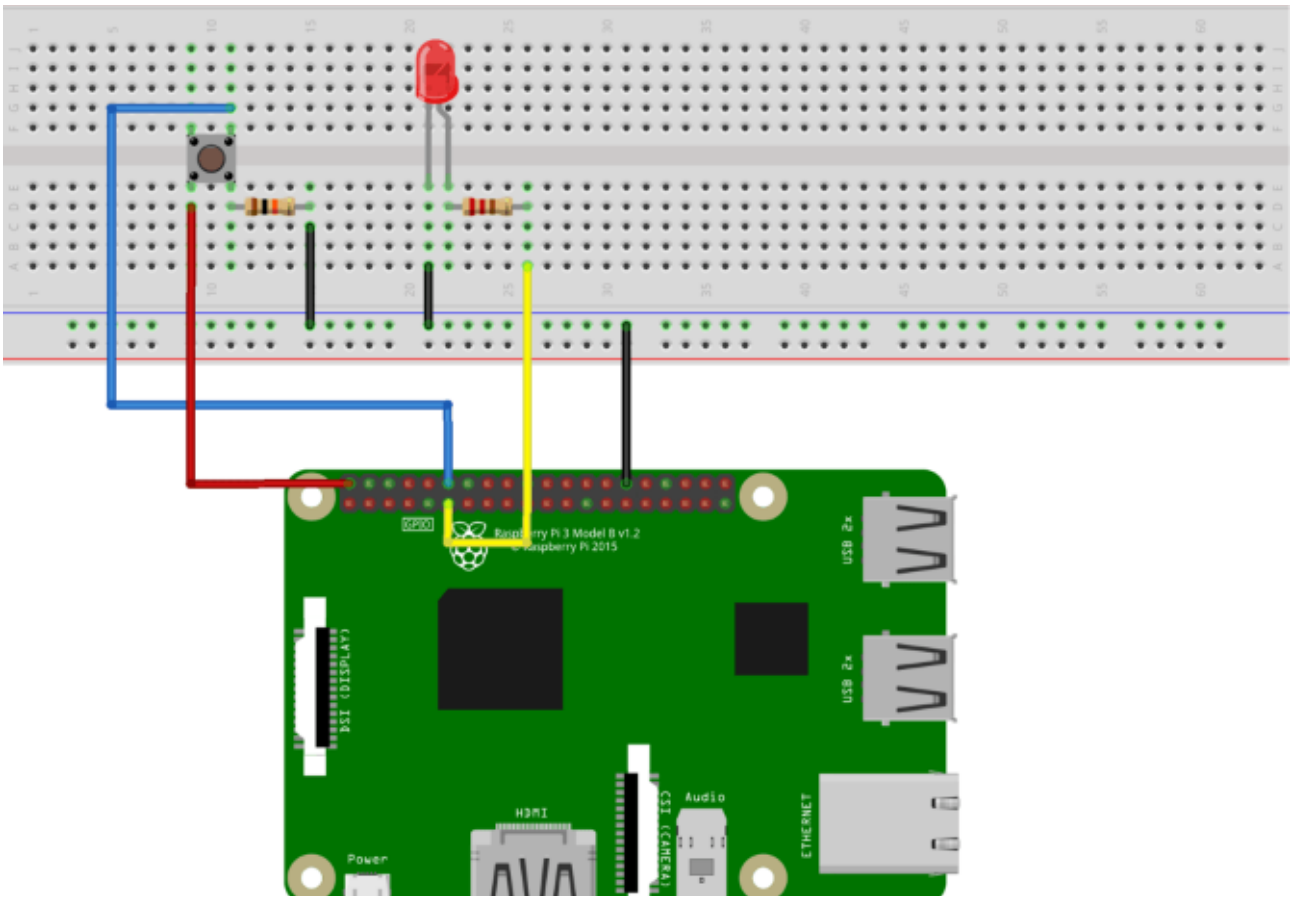
```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
gpio_number = 8
GPIO.setup(gpio_number, GPIO.IN, pull_up_down=GPIO.PUD_UP)
print("GPIO no " + str(gpio_number) + ": " + str(GPIO.input(gpio_number)))
```

To choose what is going to be the internal resistor, you have to add the option "pull_up_down=" to the GPIO.setup() function, with:

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

If you run this code (using GPIO.PUD_UP), you'll get the result:
GPIO no 8: 1
.
And if you change the code (using GPIO.PUD_DOWN instead of GPIO.PUD_UP), you'll get:
GPIO no 8: 0
.
This way, with just one small addition in your code, you can decide yourself of the default state for any GPIO.

Override the default state with an external pull up/down resistor
This method requires you to add a resistor to your circuit, for each GPIO you want to use. So basically you'll have both the internal pull up/down resistor, and your own external pull up/down resistor. To make your external resistor "take over", you just have to provide a stronger value, for example 10kOhm. 10kOhm is stronger than 50kOhm, it works in reverse for the resistors.
Here is an example with a circuit containing a push button and an LED.



For the LED, you have a resistor, but this is different. The GPIO for the LED will be configured as output, and the resistor is here simply to limit the current that goes through the LED. As we've seen before, the default state issue is only when you read a GPIO in input mode.
For the push button, you can see a 10kOhm resistor between one leg (same side as the connection with a GPIO), and the ground (GND). This pull down resistor will make sure that when you read the value from the push button, you will get 0 (LOW) when the push button is not pressed, and 1 (HIGH) when the button is pressed.
With this external pull down resistor, you are sure to get the same value every time. The internal pull up/down resistor is too weak compared to this external resistor, it won't have any effect. So, in this case using the option pull_up_down=GPIO.PUD_UP or GPIO.PUD_DOWN will have no effect.

Conclusion – Raspberry Pi's GPIOs default state – What to do

As you've seen in this tutorial, each GPIO on the Raspberry Pi has a 50kOhm internal resistor, which can be used as a pull up or pull down resistor in your code.

And by default, before you set anything, the resistors for GPIOs up to number 8 will automatically be set as pull up, and for GPIOs after number 9, pull down.

This default state will only be activated after you boot the Pi. If you use the GPIOs in your code you might change that "default" state for any other program you run after that. The only way to come back to the "true" default state is by rebooting your Raspberry Pi.

So, a recommendation here is to choose yourself (or "override") the default state when you use a GPIO as input. You don't want to get values that are hard to predict.

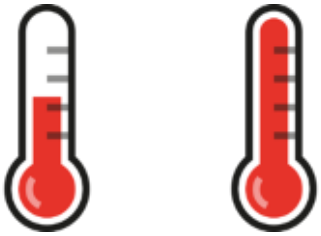
The first option is to set the internal pull up/down resistor in your code, for example with the RPi.GPIO module. But this option is not optimal. The value of the resistor – 50kOhm – is quite weak actually, and if you have longer wires, you might experience weird behaviors.

The best option is to add an external 10kOhm resistor by yourself, either in pull up (connected to Vcc) or pull down mode (connected to ground). This is a bit more work, but with this you will be sure to get predictable and stable results when you read data from a component.

Raspberry Pi Thermal Management

All Raspberry Pi models perform a degree of thermal management to avoid overheating under heavy load. The SoCs have an internal temperature sensor, which the software on the GPU polls to ensure that temperatures do not exceed a predefined limit. This is 85°C on all models. It is possible to set this to a lower value, but not to a higher one. As the device approaches the limit, various frequencies and sometimes voltages used on the chip (ARM & GPU) are reduced. This reduces the amount of heat generated, keeping the temperature under control.

When the core temperature is between 80°C and 85°C, a warning icon showing a red half-filled thermometer will be displayed, if a display is attached. The ARM cores will be progressively throttled back. If the temperature reaches 85°C, an icon showing a fully filled thermometer will be displayed, and both the ARM cores and the GPU will be throttled back.



For Raspberry Pi 3 Model B+, the PCB technology has been changed to provide better heat dissipation and increased thermal mass. In addition, a soft temperature limit has been introduced, with the goal of maximising the time for which a device can "sprint" before reaching the hard limit at 85°C. When the soft limit is reached, the clock speed is reduced from 1.4GHz to 1.2GHz, and the operating voltage is reduced slightly. This reduces the rate of temperature increase: we trade a short period at 1.4GHz for a longer period at 1.2GHz. By default, the soft limit is 60°C, and this can be changed via the `temp_soft_limit` setting in `config.txt`.

On Raspberry Pi 4 Model B, firmware from late November 2019 onwards implements Dynamic Voltage and Frequency Scaling. This technique allows Raspberry Pi 4B to run at lower temperatures whilst still providing the same performance.

Various clocks (e.g. ARM, Core, V3D, ISP, H264, HEVC) inside the SoC are monitored by the firmware, and whenever they are not running at full speed, the voltage supplied to the particular part of the chip driven by the clock is reduced relative to the reduction from full speed. In effect, only enough voltage is supplied to keep the block running correctly at the specific speed at which it is running. This can result in significant reductions in power used by the SoC, and therefore in the overall heat being produced.

In addition, a more stepped CPU governor is also used to produce finer-grained control of ARM core frequencies, which means the DVFS is more effective. The steps are now 1500MHz, 1000MHz, 750MHz, and 600MHz. These steps can also help when the SoC is being throttled, and mean that throttling all the way back to 600MHz is much less likely, giving an overall increase in fully loaded performance.

Whilst heatsinks are not necessary to prevent overheating damage to the SoC (the thermal throttling mechanism handles that), a heatsink or small fan will help if you wish to reduce the amount of thermal throttling that takes place. Depending on the exact circumstances, mounting the Pi vertically can also help with heat dissipation, as doing so can improve air flow.

Due to the architecture of the SoCs used on the Raspberry Pi range, and the use of the upstream temperature monitoring code in the Raspberry Pi OS distribution, Linux-based temperature measurements can be inaccurate. There is a command that can provide an accurate and instantaneous reading of the current SoC temperature, as it communicates with the GPU directly: `vcgencmd measure_temp`

More on vcgencmd

```
vcgencmd measure_clock [clock]
```

This returns the current frequency of the specified clock. The options are:

clock	Description
arm	ARM cores
core	VC4 scaler cores
H264	H264 block
isp	Image Signal Processor
v3d	3D block
uart	UART
pwm	PWM block (analogue audio output)
emmc	SD card interface
pixel	Pixel valve
vec	Analogue video encoder
hdmi	HDMI
dpi	Display Peripheral Interface

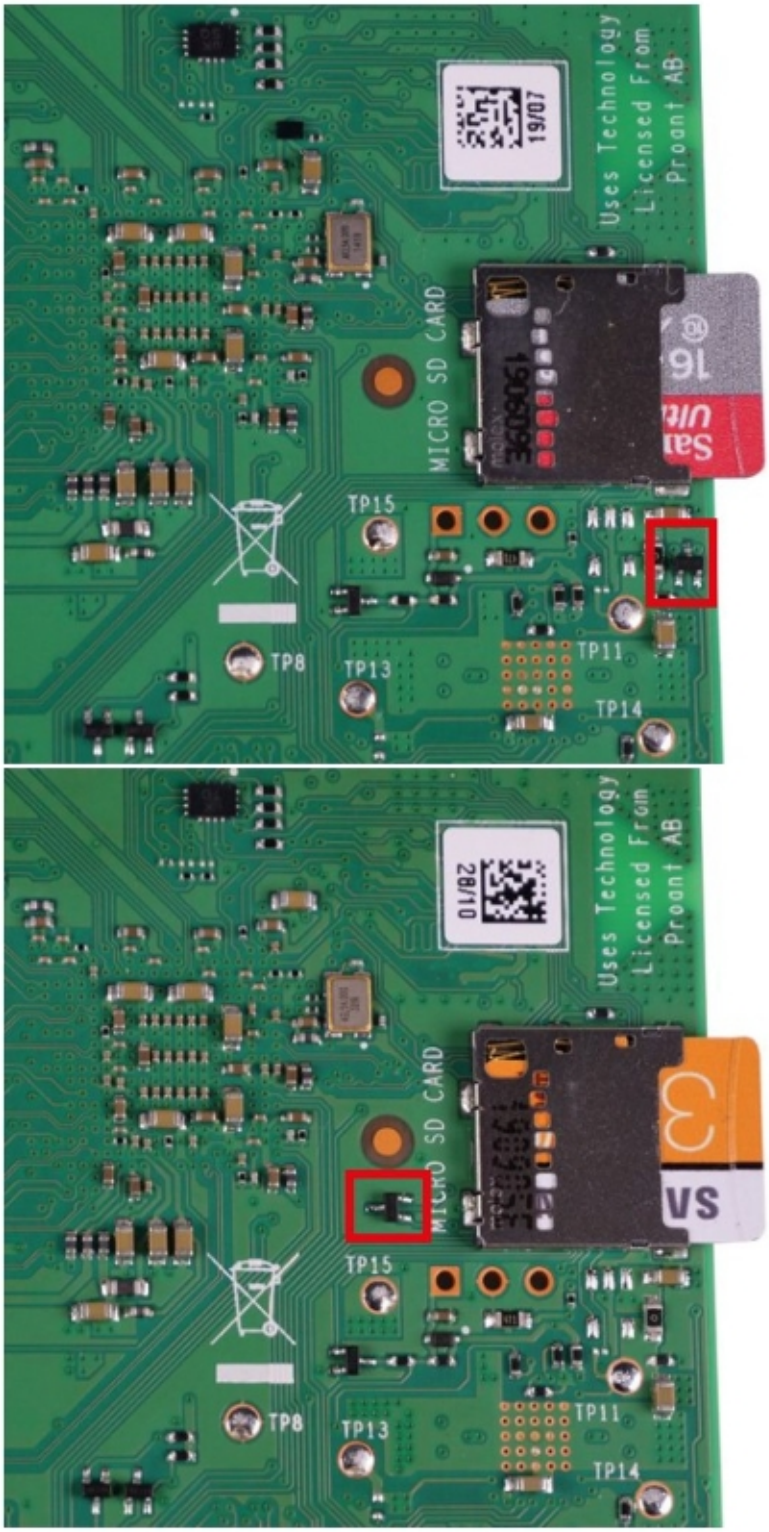
Understand LED warning flash codes

Every Raspberry Pi has LED lights on the board. The LED lights are a visual indicator of what Raspberry Pi is doing. The on-board red LED indicates that your Raspberry Pi is on, signifying power (PWR). This will flash if the voltage drops too low too, so check your power supply if this happens.
The on-board green LED, labelled ACT, indicates microSD card activity and flashes during a read or write, but on Raspberry Pi Zero this indicates power and usually flashes a lot during boot. There are LED lights within the Ethernet ports too – the green LNK LED will be on when an Ethernet cable is connected, and flashing indicates network activity. The 100/1000 LED will be yellow if gigabit networking is working, otherwise it will be off.
If Raspberry Pi fails to boot, in many cases an LED will be flashed a specific number of times to indicate what happened. Usually, the pattern will repeat after a two-second gap.

Long flashes	Short flashes	Status
0	3	Generic failure to boot
0	4	start*.elf not found
0	7	Kernel image not found
0	8	SDRAM failure
0	9	Insufficient SDRAM
0	10	In HALT state
2	1	Partition not FAT
2	2	Failed to read from partition
2	3	Extended partition not FAT
2	4	File signature/hash mismatch - Pi 4
3	1	SPI EEPROM error - Pi 4
3	2	SPI EEPROM is write protected - Pi 4
4	4	Unsupported board type
4	5	Fatal firmware error
4	6	Power failure type A
4	7	Power failure type B

Appendix

Difference between Raspberry Pi 4 Rev 1.1 and Rev 1.2



Een blik op de onderkant van de Raspberry Pi 4 laat zien welke revision het is. Als er een transistor direct naast het opschrift 'MICRO' van de MicroSD-kaartsleuf zit(onder), dan is het de nieuwe board-revision 1.2 zonder USB-C-bug. Bij de oude Raspberry Pi 4 (boven) zit de transistor nog op de rand van het board.

Other vcgencmd command line options

`otp_dump`
Displays the content of the One Time Programmable (OTP) memory, which is part of the SoC. These are 32 bit values, indexed from 8 to 64. See the OTP bits page for more details.

`get_mem type`
Reports on the amount of memory allocated to the ARM cores vcgencmd get_mem arm and the VC4 vcgencmd get_mem gpu.

Note: On a Raspberry Pi 4 with greater than 1GB of RAM, the arm option is inaccurate. This is because the GPU firmware which implements this command is only aware of the first gigabyte of RAM on the system, so the arm setting will always return 1GB minus the gpu memory value. To get an accurate report of the amount of ARM memory, use one of the standard Linux commands, such as `free` or `cat /proc/meminfo`

`codec_enabled [type]`
Reports whether the specified CODEC type is enabled. Possible options for type are AGIF, FLAC, H263, H264, MJPA, MJPB, MJPG, MPG2, MPG4, MVC0, PCM, THRA, VORB, VP6, VP8, WMV9, WVC1. MPG2, WMV9 and WVC1 currently require a paid for licence, except on the Pi4, where these hardware codecs are disabled in preference to software decoding, which requires no licence. Note that because the H265 hardware block on the Raspberry Pi4 is not part of the VideoCore GPU, its status is not accessed via this command.

`get_config type | name`
This returns all the configuration items of the specified type that have been set in config.txt, or a single configuration item. Possible values for type parameter are int, str, or simply use the name of the configuration item.

`get_lcd_info`
Displays the resolution and colour depth of any attached display.

`mem_oom`
Displays statistics on any Out Of Memory events occuring in the VC4 memory space.

`mem_reloc_stats`
Displays statistics from the relocatable memory allocator on the VC4.

`read_ring_osc`
Returns the curent speed voltage and temperature of the ring oscillator.

`hdmi_timings`
Displays the current HDMI settings timings. See Video Config for details of the values returned.

`dispmanx_list`
Dump a list of all dispmanx items currently being displayed.

`display_power [0 | 1 | -1] [display]`
Show current display power state, or set the display power state. `vcgencmd display_power 0` will turn off power to the current display. `vcgencmd display_power 1` will turn on power to the display. If no parameter is set, this will display the current power state. The final parameter is an optional display ID, as returned by tvservice -l or from the table below, which allows a specific display to be turned on or off.
Note that for the 7" Raspberry Pi Touch Display this simply turns the backlight on and off. The touch functionality continues to operate as normal.

`vcgencmd display_power 0 7` will turn off power to display ID 7, which is HDMI 1 on a Raspberry Pi 4.

Display	ID
Main LCD	0
Secondary LCD	1
HDMI 0	2
Composite	3
HDMI 1	7

To determine if a specific display ID is on or off, use -1 as the first parameter.
`vcgencmd display_power -1 7` will return 0 if display ID 7 is off, 1 if display ID 7 is on, or -1 if display ID 7 is in an unknown state, for example undetected.