# GPIO Usage on Raspberry Pi Devices

# Colophon



| | |
|---|---|
| **Release** | 1 |
| **Build date** | 13/08/2025 |
| **Build version** | 0e6d3b8eb200 |

## Legal disclaimer notice

TECHNICAL AND RELIABILITY DATA FOR RASPBERRY PI PRODUCTS (INCLUDING DATASHEETS) AS MODIFIED FROM TIME TO TIME ("RESOURCES") ARE PROVIDED BY RASPBERRY PI LTD ("RPL") "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN NO EVENT SHALL RPL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE RESOURCES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RPL reserves the right to make any enhancements, improvements, corrections or any other modifications to the RESOURCES or any products described in them at any time and without further notice.

The RESOURCES are intended for skilled users with suitable levels of design knowledge. Users are solely responsible for their selection and use of the RESOURCES and any application of the products described in them. User agrees to indemnify and hold RPL harmless against all liabilities, costs, damages or other losses arising out of their use of the RESOURCES.

RPL grants users permission to use the RESOURCES solely in conjunction with the Raspberry Pi products. All other use of the RESOURCES is prohibited. No licence is granted to any other RPL or other third party intellectual property right.

HIGH RISK ACTIVITIES. Raspberry Pi products are not designed, manufactured or intended for use in hazardous environments requiring fail safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, weapons systems or safety-critical applications (including life support systems and other medical devices), in which the failure of the products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). RPL specifically disclaims any express or implied warranty of fitness for High Risk Activities and accepts no liability for use or inclusions of Raspberry Pi products in High Risk Activities.

Raspberry Pi products are provided subject to RPL's Standard Terms. RPL's provision of the RESOURCES does not expand or otherwise modify RPL's Standard Terms including but not limited to the disclaimers and warranties expressed in them.

# Document version history

| Release | Date | Description |
|---|---|---|
| 1 | 1 Aug 2024 | Initial release |
| 2 | 13 Aug 2025 | Update to Wiring Pi section |

# Scope of document

This document applies to the following Raspberry Pi products:

| Pi 0 | | | Pi 1 | | Pi 2 | | Pi 3 | Pi 4 | Pi 400 | Pi 5 | Pi 500 | CM1 | CM3 | CM4 | CM5 | Pico | Pico2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | W | H | A | B | A | B | B | All | All | All | All | All | All | All | All | All | All |
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | | |

# Introduction

Ever since the very first Raspberry Pi SBC launched in 2012, they have sported a header on which there are several General Purpose Input/Output (henceforth GPIO) pins. The earliest models had a 26-pin header, but on Raspberry Pi 1 Model B+ this was extended to 40 pins and has remained the same since. As the 26-pin header devices are no longer manufactured, this document will only refer to the 40-pin header devices.

Over the last decade, there have been varied options available to drive these GPIO pins, and this document will attempt to explain some of the various libraries that have been and gone, and what is now the recommended way of driving GPIO pins from Linux (using either C/C++ or Python). GPIOs are also available for the many other languages that can be used across the range of Raspberry Pi SBCs, but this document will concentrate on these two languages.

The Raspberry Pi online documentation has a lot of information on GPIOs and should be read in conjunction with this document. See https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#gpio

This document does not attempt to provide information on using the libraries, but is intended solely to help choose an appropriate library matched to the user's requirements.

# What is a GPIO?

A GPIO is a way of connecting your Raspberry Pi device to the outside world. In its simplest form, a GPIO could be used to drive an LED (an output) or to connect a switch (an input).

Raspberry Pi GPIOs work at 3.3V, so when configured as an output they will be set to 3.3V when on, or when set as an input must be connected to a maximum of 3.3V.

Rather than just being On/Off pins, Raspberry Pi GPIOs can also be assigned to more sophisticated hardware peripherals, provided either in the SoC or, in the case of Raspberry Pi 5, in the RP1 chip. These peripherals include I2C, SPI, DPI, PWM, and serial ports. This document will not go into the details of these peripherals, but there is lots of information on the Raspberry Pi website. When a hardware peripheral is mapped to a GPIO, this is known as an 'alternate function', as the GPIO is now being used for something else. Please refer to the datasheet of the device being used for a list of all alternate functions available, and to which GPIO pins they can be assigned: https://datasheets.raspberrypi.com/

It is strongly recommended that rather than accessing these alternative function devices directly, the appropriate Linux subsystem is used for access. For example, there are specific subsystems for I2C, SPI, DPI, PWM, and serial ports (UARTs) that provide standard Linux APIs, and these should be used to provide the greatest forward and multi-platform compatibility.

## Representation in Linux

The Linux kernel documentation has a good description of the internal representation of GPIOs here: https://docs.kernel.org/driver-api/gpio/driver.html

The following definitions are extracted from that document.

**GPIO**
: General Purpose Input Output

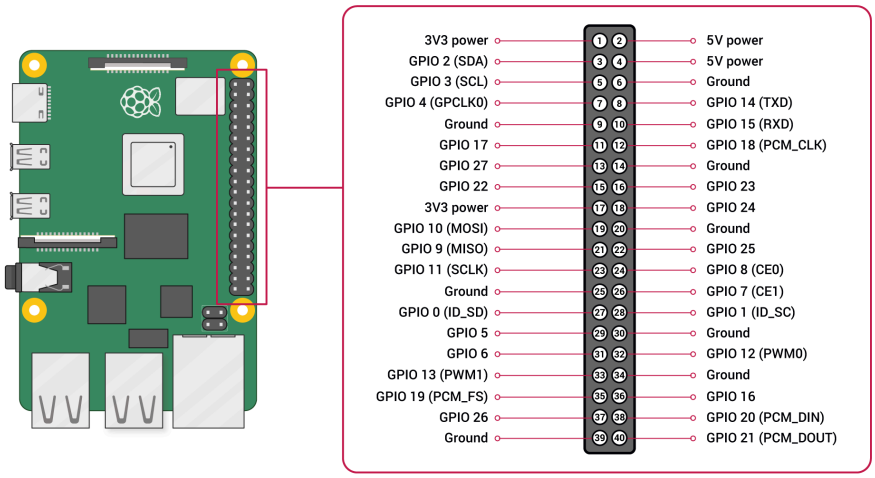
**GPIO Line**
: A single GPIO on one pin.

**GPIO chip**
: A GPIO chip handles one or more GPIO lines. There can be multiple `gpiochips` in a system, usually mapping to distinct GPIO hardware blocks.

**GPIO Offset**
: Individual lines in a GPIO driver are referred to by an `offset` , a unique number between 0 and n-1, where 'n' is the number of GPIOs handled by the GPIO chip. The offset is an internal representation only, and is not the same as the GPIO number presented to the user.

## GPIO on Raspberry Pi SBCs

**Figure 1.**
*GPIO Pinout diagram*




On Raspberry Pi 0-2, all of the GPIO pins are directly driven from the SoC itself, meaning there is a direct connection from the SoC to the physical pins on the boards. From Raspberry Pi 3 onwards, there were not enough GPIOs available on the SoC to drive all the peripherals and still have the same GPIO pinouts as previous models, so a GPIO extender chip attached via I2C is used. This results in two `gpiochips` being instantiated in Linux, one for the SoC-provided GPIO lines and one for the GPIO extender lines.

On Raspberry Pi 5, things are a little more complicated. The SoC itself (The BCM2712) has four distinct GPIO hardware blocks (providing always-on and other features), and the RP1 provides the user-facing GPIO lines. This results in five `gpiochips` being instantiated.

Using `gpiodetect` (see later), we get the following results for various Pi models:

Raspberry Pi 4 example

```
pi@pi4:~ $ gpiodetect
gpiochip0 [pinctrl-bcm2711] (58 lines)
gpiochip1 [raspberrypi-exp-gpio] (8 lines)
```

Raspberry Pi 5 example

```
pi@pi5:~ $ gpiodetect
gpiochip0 [gpio-brcmstb@107d508500] (32 lines)
gpiochip1 [gpio-brcmstb@107d508520] (4 lines)
gpiochip2 [gpio-brcmstb@107d517c00] (17 lines)
gpiochip3 [gpio-brcmstb@107d517c20] (6 lines)
gpiochip4 [pinctrl-rp1] (54 lines)
```

Careful examination of the above outputs will show that the user-facing GPIOs, those with `pinctrl` in their description, actually appear on a different `gpiochip` : 0 on models prior to Raspberry Pi 5, and 4 on Raspberry Pi 5. This is a problem for GPIO libraries that are intended to run over the entire model range, as they need to determine the model of Pi being used, then decide which `gpiochip` to access.

To counter this problem, as of July 2024, a modification has been made to the Linux kernel and the device tree system on Raspberry Pi 5 to reorder the `gpiochips` to match those of earlier models (see https://github.com/raspberrypi/linux/pull/6144#issuecomment-2234013076) As a result, `gpiodetect` will return the following on Raspberry Pi 5.

```
pi@pi5:~ $ gpiodetect
gpiochip0 [pinctrl-rp1] (54 lines)
gpiochip10 [gpio-brcmstb@107d508500] (32 lines)
gpiochip11 [gpio-brcmstb@107d508520] (4 lines)
```

```
gpiochip12 [gpio-brcmstb@107d517c00] (17 lines)
gpiochip13 [gpio-brcmstb@107d517c20] (6 lines)
```

The user-facing GPIOs are now to be found on `gpiochip0` , as with earlier models. Internal system GPIOs have been reallocated to 10 and above to show that they are system-level devices.

# A history of GPIO libraries

When Raspberry Pi first launched, there were no specific software libraries to drive the GPIOs provided by Raspberry Pi, so third parties stepped into the gap. There are many different libraries out there, but this document will concentrate on the more popular ones. Over the years, not only have the libraries evolved, but so have the Linux subsystems used for GPIO. A major change is the deprecation of the `sysfs` access to GPIO, which is due to be removed completely from the Linux kernel. Since some libraries used this interface, they are no longer usable.

Some libraries use direct access to registers. Whilst this is the fastest way of accessing GPIO information, the register's location (and sometimes content) varies from device to device. In addition, Raspberry Pi 5 and its use of the RP1 to provide its I/O, including GPIO, means these direct access approaches no longer work. The appropriate method to access GPIO features is now via the standard Linux `libgpiod` library, which will be discussed more fully later. Using a hardware abstraction like this means the Linux kernel handles any differences in hardware that may be encountered between different models, and also products from different manufacturers, leaving the user space libraries to provide features without having to worry about the underlying implementation of the GPIOs.

## Wiring Pi

https://github.com/WiringPi/WiringPi

A library written in C, first developed in around 2013 and one of the first available. The original developer found he did not have the time to develop the library further, so maintenance of the library is now led by the Grazer Computer Club (https://github.com/GrazerComputerClub)

Wiring Pi has now been updated to work with the Raspberry Pi 5, and is actively maintained.

NOTE

On the Raspberry Pi 5, the `GCLK` functionality is currently not supported.

There are several language wrappers available for Wiring Pi. These are linked from the Wiring Pi GitHub page.

## pigpio

https://github.com/joan2937/pigpio

Another C library that uses DMA to provide high performance. Other language wrappers are available. `pigpio` is heavily tailored to the Broadcom SoCs used on Raspberry Pi devices, to the extent that it does not work on any other SoCs. This means that it can fully exploit the hardware capabilities of the Broadcom devices, and so very high sampling rates are available (e.g. 1 million samples per second on GPIO0-27, including timestamping). This makes 'pigpio' very useful for programs such as `piscope` , a software oscilloscope.

However, due to the underlying hardware changes on Raspberry Pi 5, `pigpio` does not yet work on this device. For this reason, it is not yet recommended for new projects.

## pigpiod

A daemon/resident version of `pigpio` which accepts commands from pipe and socket interfaces.

## lgpio

By the same author as `pigpio` , `lgpio` is a more general-purpose library than 'pigpio', and will run on other SBCs and Linux devices. It uses the `/dev/gpiochip` device kernel interface rather than accessing registers directly. The library is written in C, but there is also a Python library and a version that provides remote control of GPIOs using the `rpgiod` daemon in both C and Python.

Whilst not as performant as `pigpio` , it should run on all models of Raspberry Pi devices.

# RPi.GPIO

https://pypi.org/project/RPi.GPIO/

A Python module to control the GPIO on a Raspberry Pi. Because Python uses garbage collection, this can affect timings, so this library is not recommended for real-time or time-critical applications. It does not support SPI, I2C, Hardware PWM, or serial ports.

This library uses direct hardware access under the hood, so it does not support Raspberry Pi 5. However, there is a new library with the same API that uses `libgpiod` for the backend, called `rpi-lgpio` , which should be functional on Raspberry Pi 5: https://rpi-lgpio.readthedocs.io/en/release-0.4/

There is a useful article on the reasoning behind the development of `rpi-lgpio` here: https://waldorf.waveform.org.uk/2022/the-one-where-dave-breaks-stuff.html

# Current libraries

## libgpiod

https://libgpiod.readthedocs.io/en/latest/index.html https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/

`libgpiod` is a C library and set of tools for interacting with the Linux GPIO character devices (/dev/gpiochipX). Note that `gpiod` stands for GPIO device, not daemon. There are two versions of `libgpiod` , version 1 and version 2. At present, Raspberry Pi OS includes the version 1 library, so you should take that into account when using the various tools as their behaviour does change between versions.

By default, `libgpiod` follows the Linux ethos of taking control of a peripheral, allowing the peripheral to be used, and, when the process ends, returning the peripheral to its previous state. This means that there is no guarantee that when a program to set a GPIO to a particular value exits, the value will remain as set. It may or may not be returned to the state from before the set program was run. In order to make the move from the persistent operations of the previous `sysfs` to `libgpiod` easier, the Raspberry Pi kernel modifies this behaviour so that the state of a GPIO **IS** retained when a program that has changed the value (e.g. `gpioset` , see below) exits.

You can enable the automatic reversion behaviour of `libgpiod` by adding the following to the `config.txt` file.

```
dtparam=strict_gpiod
```

Alternatively, add the following to the kernel command line ( `cmdline.txt` ).

```
pinctrl_bcm2835.persist_gpio_outputs=n
```

One related feature of `libgpiod` is that it guarantees exclusivity whilst the GPIO is claimed, which none of the other libraries do. That means if you open a GPIO line via `libgpiod` , no other applications will be able to access it until you release it.

## Tools

`libgpiod` includes several tools that are available in Raspberry Pi OS. Please see the man pages for the commands for more details. Note that the behaviour and parameters for the commands do vary between v1 and v2 of `libgpiod` .

**gpiodetect**
: List all GPIO devices (gpiochips) in the system. Displays their names, labels, and number of GPIO lines supported.

**gpioinfo**
: List all the GPIO lines, their `gpiochip` , offset, name, and direction. Any lines in use will also display the consumer name and configured attributes, for example the active state, bias, drive, edge detection, and debounce period.

**gpioget**
: Read values of specified GPIO lines.

**gpioset**
: Set the value of specified GPIO lines. By default, `gpioset` in `libgpiod v1` will exit immediately, which may not be useful. Use `--mode=wait` on the command line to force it to wait until `Ctrl-C` is pressed. The default behaviour for `gpioset` in `libgpiod v2` is **NOT** to exit, which will maintain the GPIO line at the requested value.

**gpiomon**
: Wait for edge events on GPIO lines. Specify which edges to watch for, how many events to process before exiting, or if the events should be reported to the console.

**gpionotify**
: Wait for changes to the info for GPIO lines. Specify which changes to watch for, how many events to process before exiting, or if the events should be reported to the console.

**gpiofind**
: Determine the location of the specific named GPIO ( `gpiochip` and offset). Use the results as input to the set/get operations.

## Examples

Display all the GPIO chips on a Raspberry Pi 5.

```
pi@pi5:~ $ gpiodetect
gpiochip0 [pinctrl-rp1] (54 lines)
gpiochip10 [gpio-brcmstb@107d508500] (32 lines)
gpiochip11 [gpio-brcmstb@107d508520] (4 lines)
gpiochip12 [gpio-brcmstb@107d517c00] (17 lines)
gpiochip13 [gpio-brcmstb@107d517c20] (6 lines)
```

```
pi@pi4:~ $ gpiofind RGMII_MDIO
gpiochip0 28
```

Set GPIO3 high for three seconds, then return to default.

```
pi@pi4:~ $ gpioset --mode time --sec 3 gpiochip0 3=1
gpiochip0 28
```

# gpiozero

https://gpiozero.readthedocs.io/en/stable/

`gpiozero` is a Python library providing a simple-to-use interface to Raspberry Pi GPIOs.

The library includes interfaces to many simple everyday components, as well as complex devices like sensors, analog-to-digital converters, full-colour LEDs, robotics kits, and more.

The underlying access to the hardware GPIO lines is via a `pin factory` , and this can be set to whatever underlying system is needed. At the time of writing, `libgpiod` is not yet available as a pin factory, so `lgpio` is suggested as an alternative. More information on pin factories can be found here: https://gpiozero.readthedocs.io/en/stable/api_pins.html#module-gpiozero.pins

To use `lgpio` as the underlying pin control, you can either set an environment variable before running Python (on the command line, exported, or in a login script), or select it in the Python code.

```
pi@raspberrypi:~ $ GPIOZERO_PIN_FACTORY=lgpio python3
```

```
from gpiozero.pins.lgpio import LGPIOFactory
from gpiozero import Device, LED

Device.pin.factory = LGPIOFactory(chip=0)

led = LED(12)
```

The chip parameter to the factory constructor specifies which `gpiochip` device to open. It defaults to 0, but be careful on Raspberry Pi 5 to get the right device. See the 'GPIO on Raspberry Pi SBCs' section for details.

# pinctrl

`pinctrl` is a more powerful replacement for the older `raspi-gpio` , a tool for displaying and modifying the GPIO and pin muxing state of a system. It is not a library but a single application that can be used to manipulate GPIO states.

It accesses the hardware directly, bypassing the kernel drivers, and as such requires root privilege (run with 'sudo'). This application is the only officially supported mechanism for baremetal access to the GPIOs, and it works on all models of Raspberry Pi SBC. It is designed as a debugging tool for working with GPIOs during product development, rather than for use in production software.

The source for `pinctrl` can be found here: https://github.com/raspberrypi/utils/tree/master/pinctrl

## Examples

Display all the GPIO chips on a Raspberry Pi 5.

```
pi@pi5:~ $ pinctrl get 26-42
```

# Conclusions

Raspberry Pi Ltd make considerable effort to maintain backward compatibility over much of their software stack, however GPIO has always been problematic due to the plethora of third-party libraries, along with the evolution of the underlying Raspberry Pi hardware and the associated Linux GPIO subsystem.

It is hoped that the most recent move to `libgpiod` will stabilise the GPIO environment, meaning better compatibility for future products, and less or no work needed when porting software to them.

## Contact Details for more information

Please contact applications@raspberrypi.com if you have any queries about this whitepaper.

Web: www.raspberrypi.com

Raspberry Pi



**Raspberry Pi Ltd**