Welcome

Thank you for choosing Freenove products!

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

Copyright

FREENOVE®

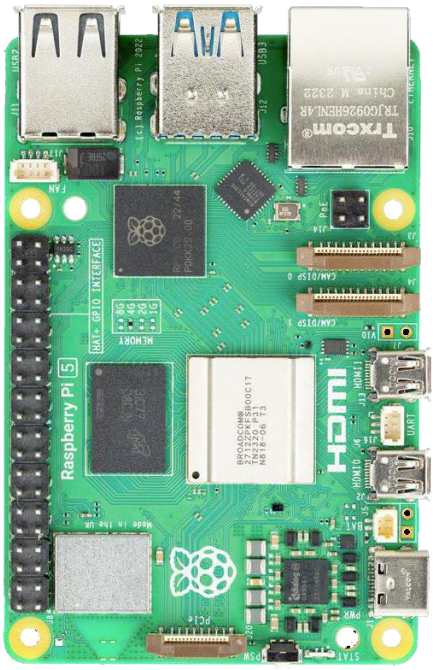
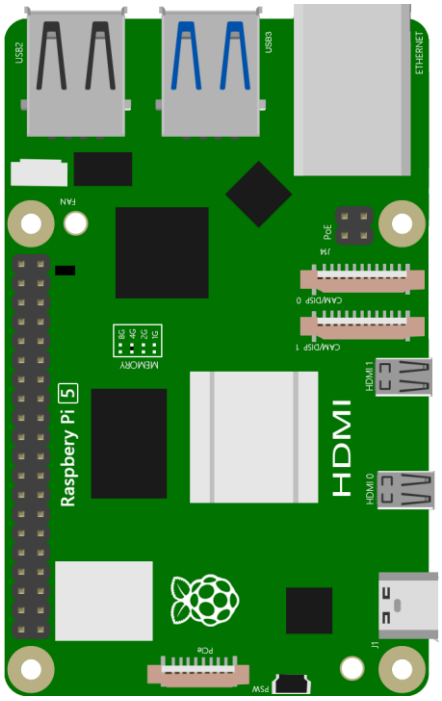
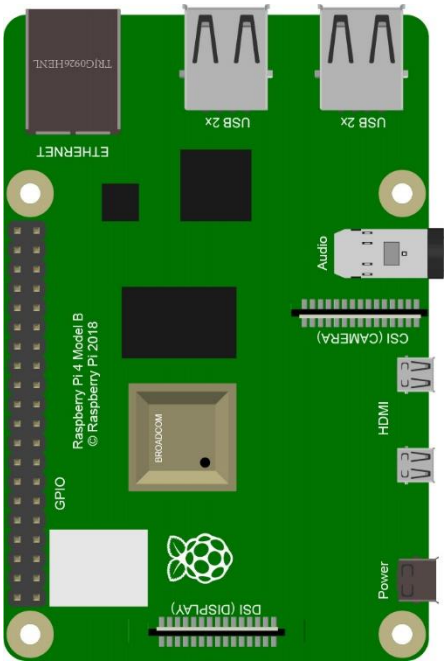
FREE YOUR INNOVATION

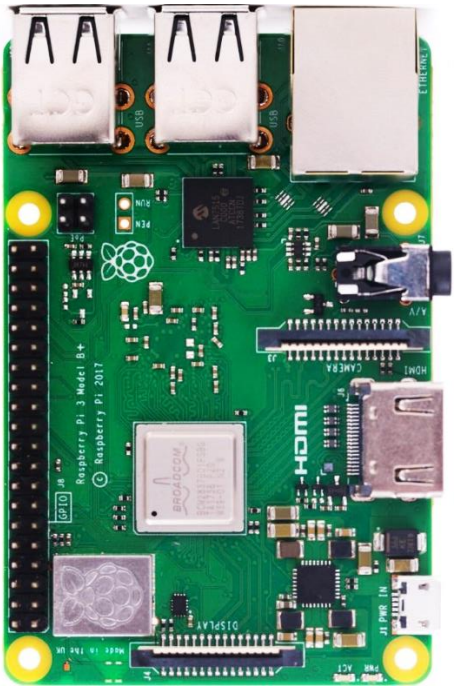
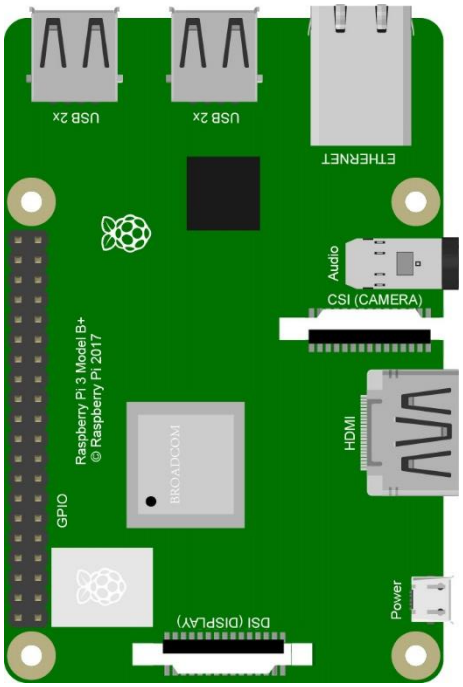
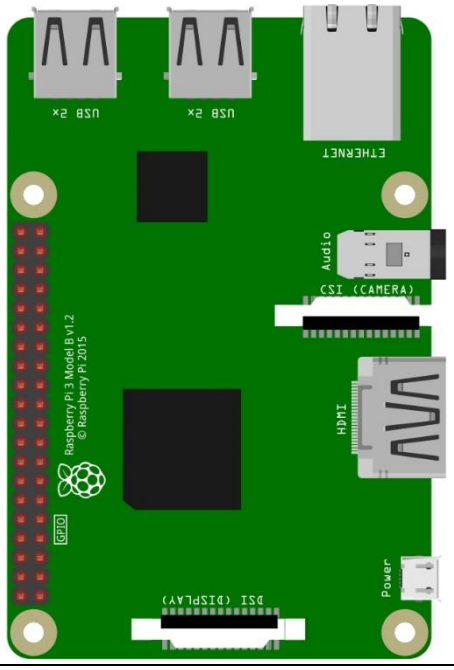
Raspberry Pi® is a trademark of Raspberry Pi Foundation (https://www.raspberrypi.org/).

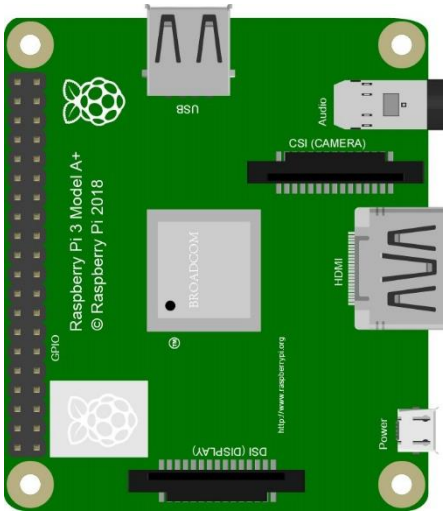
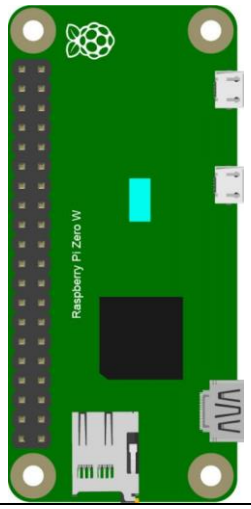
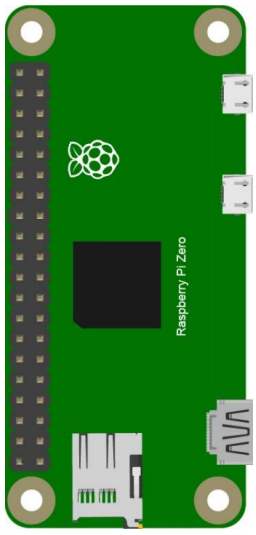
Contents

Raspberry Pi

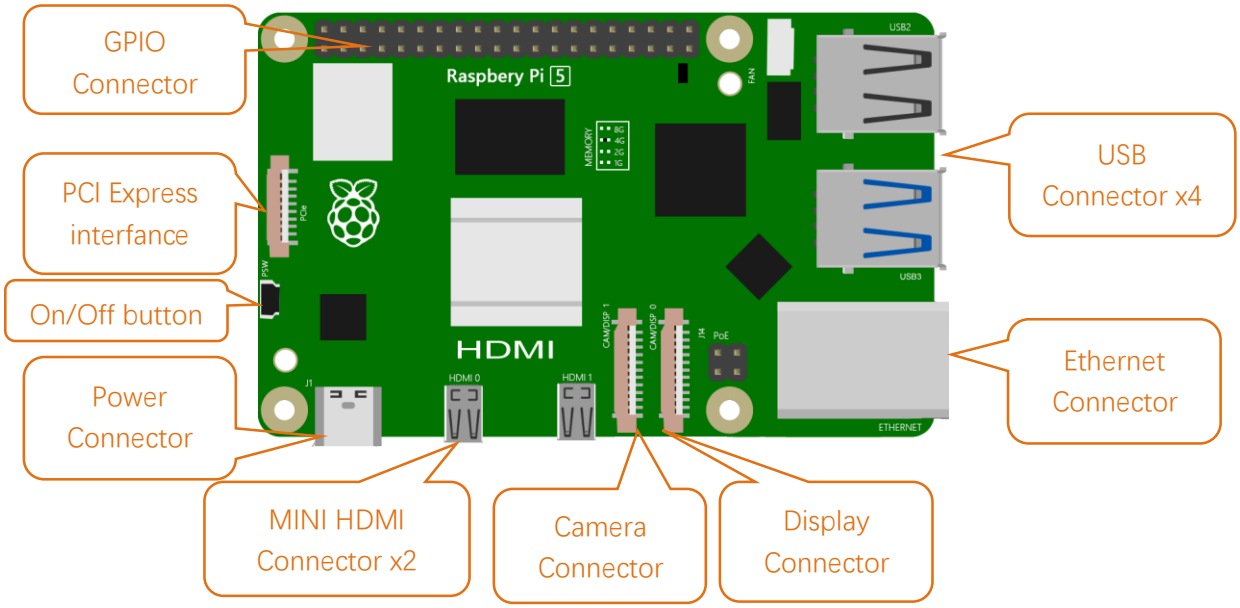
Below are the Raspberry Pi pictures and model pictures supported by this product.

Practicality picture of Raspberry Pi 5:	Model diagram of Raspberry Pi 5:
	
Practicality picture of Raspberry Pi 4 Model B:	Model diagram of Raspberry Pi 4 Model B:
	

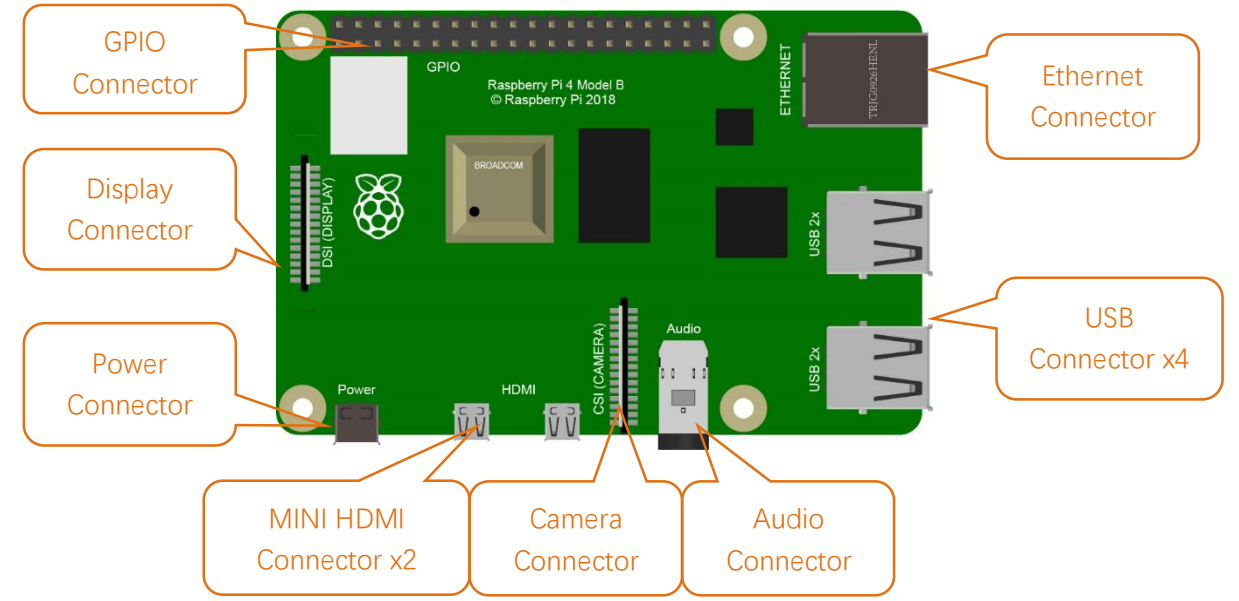
Practicality picture of Raspberry Pi 3 Model B+: 	Model diagram of Raspberry Pi 3 Model B+: 
Practicality picture of Raspberry Pi 3 Model B: 	Model diagram of Raspberry Pi 3 Model B: 

Practicality picture of Raspberry Pi 3 Model A+: 	Model diagram of Raspberry Pi 3 Model A+: 
Practicality picture of Raspberry Pi Zero W: 	Model diagram of Raspberry Pi Zero W: 
Practicality picture of Raspberry Pi Zero: 	Model diagram of Raspberry Pi Zero: 

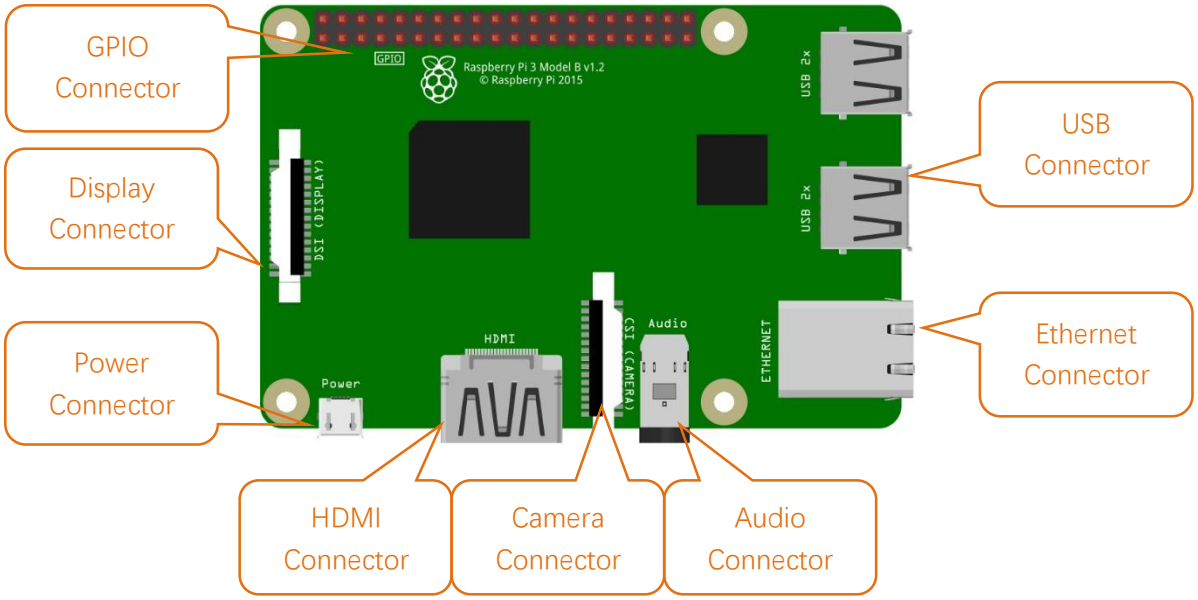
Hardware interface diagram of RPi 5 is shown below:



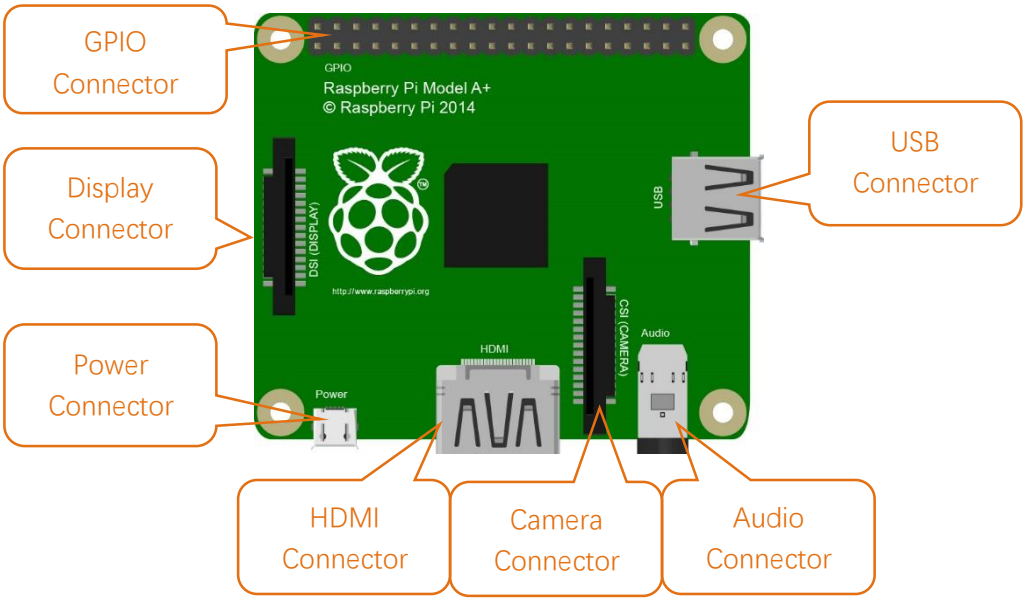
Hardware interface diagram of RPi 4B is shown below:



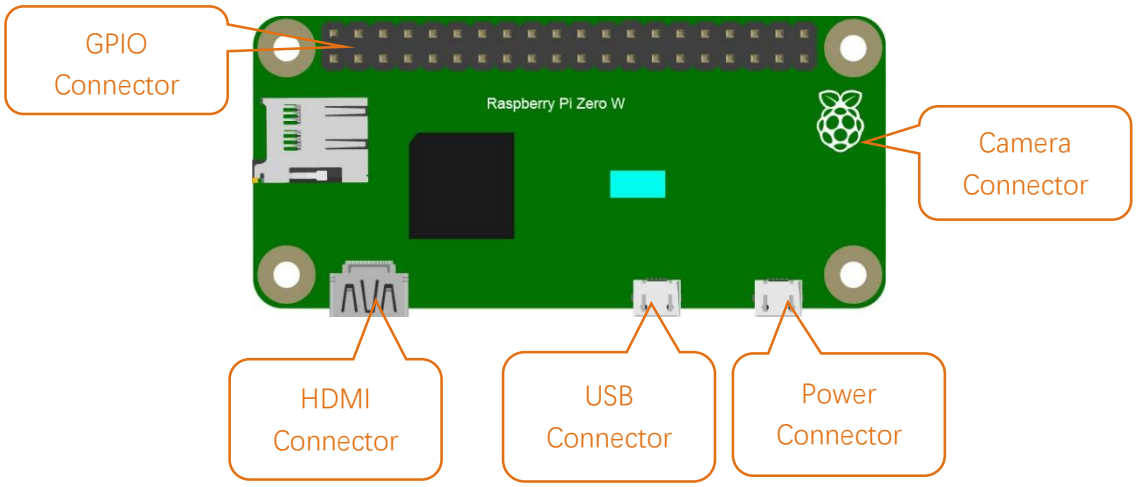
Hardware interface diagram of RPi 3B+/3B/2B/1B+ are shown below:



Hardware interface diagram of RPi 3A+/A+ is shown below:

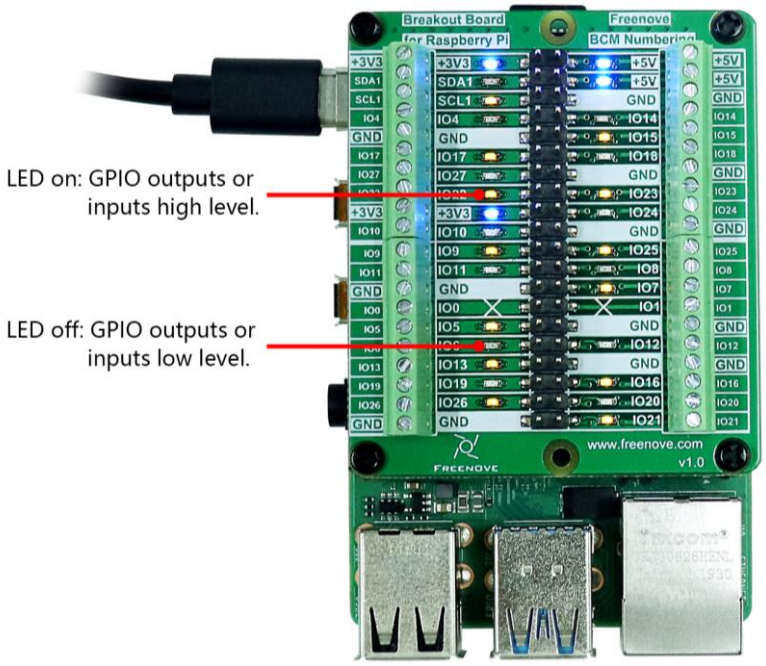


Hardware interface diagram of RPi Zero/Zero W is shown below:

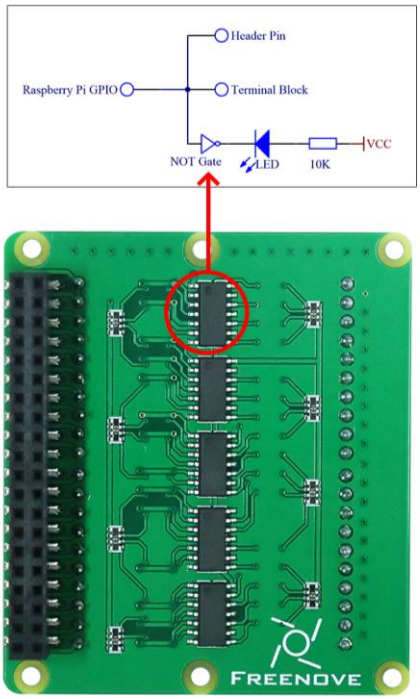


Breakout Board

Led Indicator

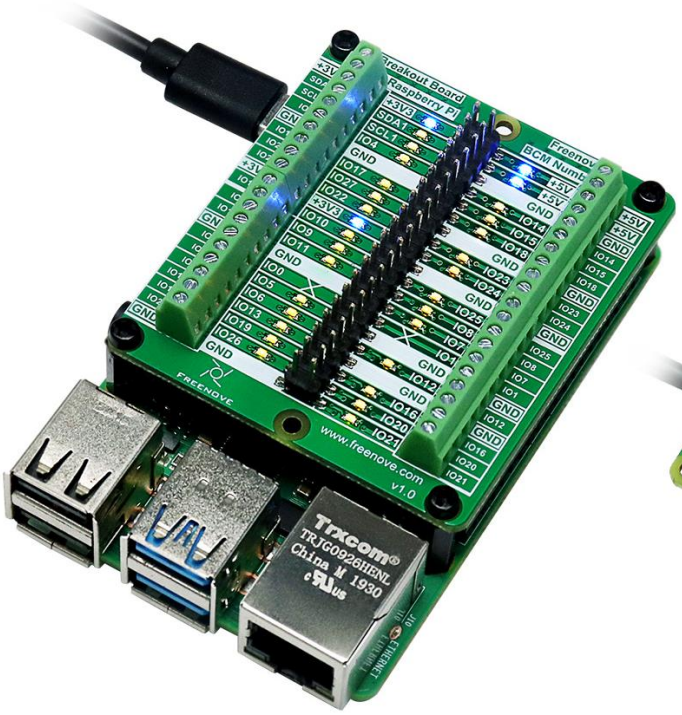


The GPIO will not be affected by the status LED.

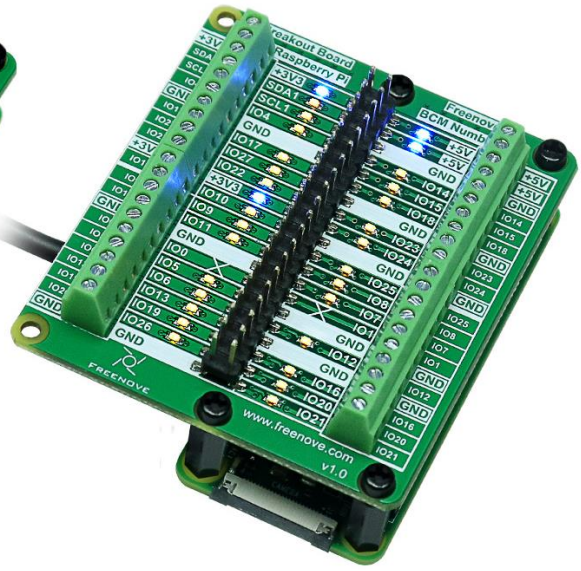


The status LED is driven by the NOT gate chip instead of the GPIO.

Asssembly



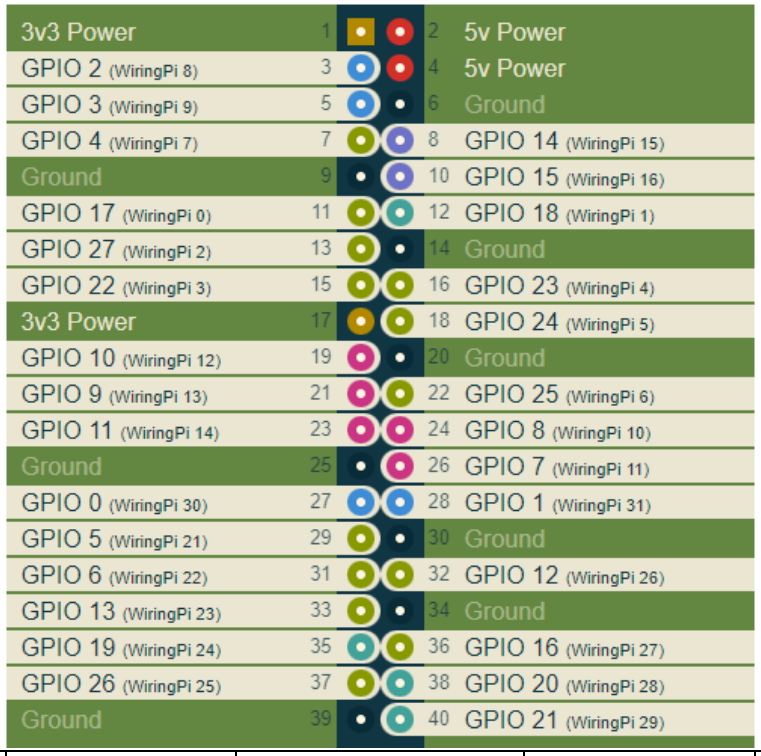
Raspberry Pi 4B



Raspberry Pi Zero

GPIO

GPIO Numbering Relationship



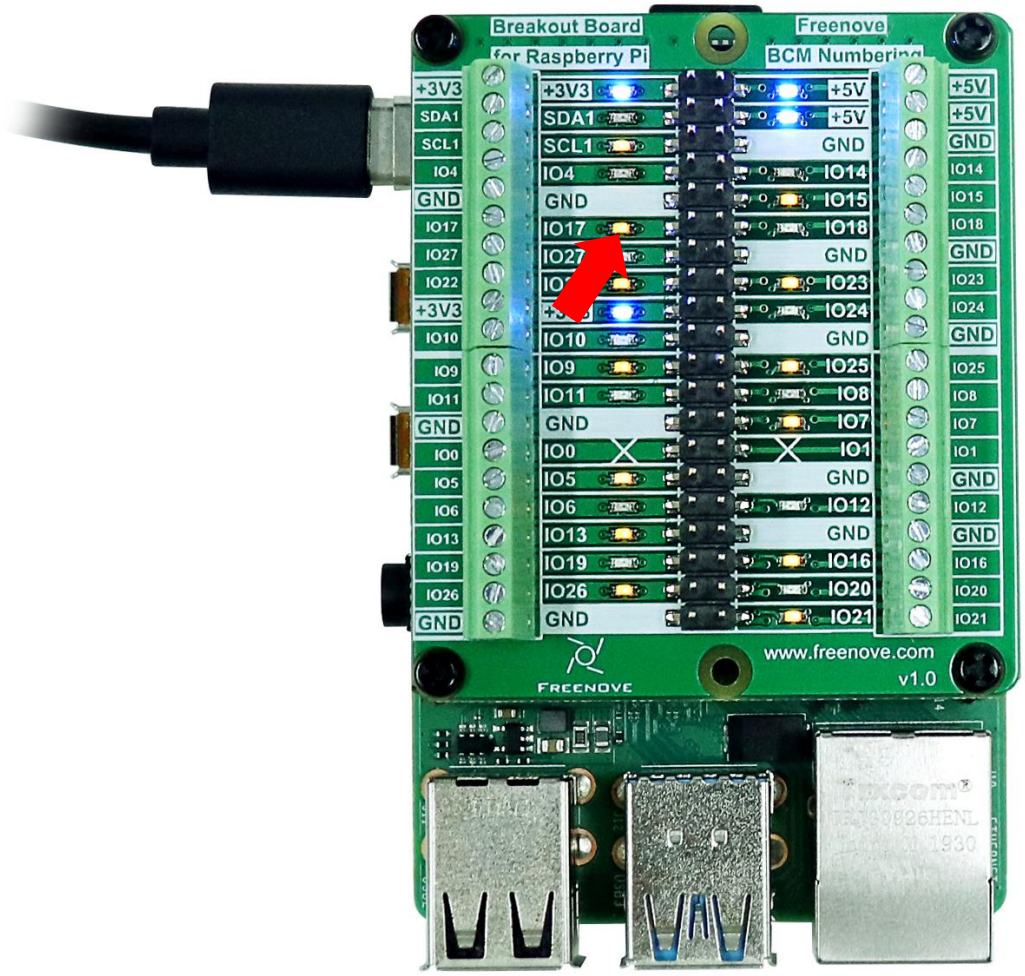
WingPi	BCM	Physical		BCM	WingPi
3.3V	**3.3V**	1	2	5V	**5V**
8	SDA1	3	4	**5V**	**5V**
9	SCL1	5	6	**GND**	**GND**
7	GPIO4	7	8	GPIO14/TXD0	15
GND	**GND**	9	10	GPIO15/RXD0	16
0	GPIO17	11	12	GPIO18	1
2	GPIO27	13	14	**GND**	**GND**
3	GPIO22	15	16	GPIO23	4
3.3V	**3.3V**	17	18	GPIO24	5
12	GPIO10/MOSI)	19	20	**GND**	**GND**
13	GPIO9/MOIS	21	22	GPIO25	6
14	GPIO11/SCLK	23	24	GPIO8 /CE0	10
GND	**GND**	25	26	GPIO7 CE1	11
30	GPIO0/SDA0	27	28	GPIO1 /SCL0	31
21	GPIO5	29	30	**GND**	**GND**
22	GPIO6	31	32	GPIO12	26
23	GPIO13	33	34	**GND**	**GND**
24	GPIO19	35	36	GPIO16	27
25	GPIO26	37	38	GPIO20	28
GND	**GND**	39	40	GPIO21	29

For more details about pin definition of GPIO, please refer to http://pinout.xyz/

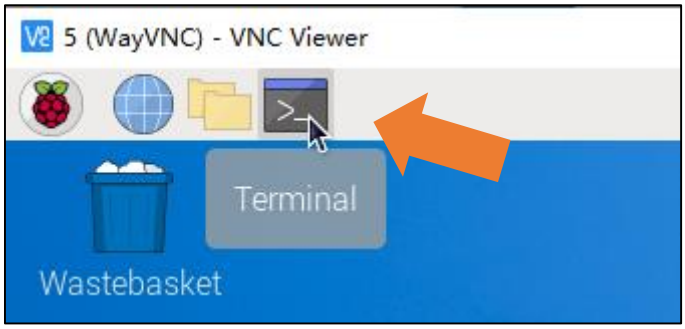
Project Example

LED Blink

We will make the LED indicator of GPIO 17 blink.

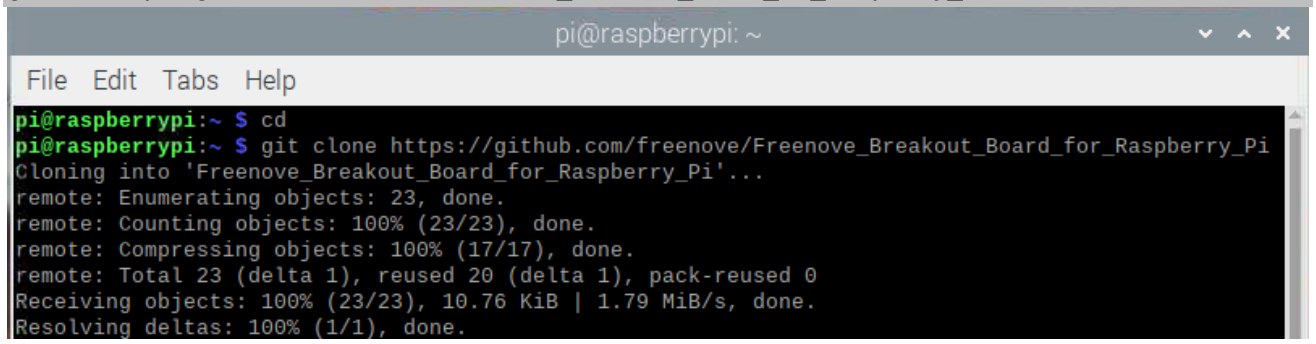


Geting the Code



Run following commands in terminal:

```
cd
git clone https://github.com/freenove/Freenove_Breakout_Board_for_Raspberry_Pi
```



Run the Code

C Code 01.1.1 Blink

If you have any concerns, please contact us via: support@freenove.com

1. If you did not update wiring pi, please execute following commands **one by one**.

```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```

2. Use cd command to enter 01.1.1_Blink directory of C code.

```
cd ~/Freenove_Breakout_Board_for_Raspberry_Pi/Code/C_Code/01.1.1_Blink
```

3. Use the following command to compile the code "Blink.c" and generate executable file "Blink".

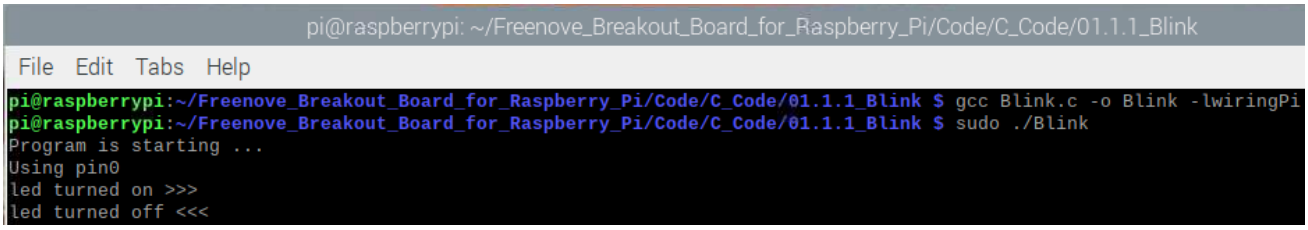
"l" of "lwiringPi" is low case of "L".

```
gcc Blink.c -o Blink -lwiringPi
```

4. Then run the generated file "blink".

```
./Blink
```

Now your LED should start blinking!



You can press "**Ctrl+C**" to end the program. The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define  ledPin    17  //define the led pin number
5
6  void main(void)
7  {
8      printf("Program is starting ... \n");
9
10     wiringPiSetupGpio();    //Initialize wiringPi. Use BCM Number.
11
12     pinMode(ledPin, OUTPUT);//Set the pin mode
13     printf("Using pin%d\n",%ledPin);    //Output information on terminal
14     while(1){
15         digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
16         printf("led turned on >>>\n");      //Output information on terminal
```

```
17          delay(1000);                        //Wait for 1 second
18          digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
19          printf("led turned off <<<\n");         //Output information on terminal
20          delay(1000);                        //Wait for 1 second
21      }
22  }
```

In the code above, the configuration function for GPIO is shown below as:

void pinMode(int pin, int mode);

This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes. This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program

void digitalWrite (int pin, int value);

Writes the value HIGH or LOW (1 or 0) to the given pin, which must have been previously set as an output.

GPIO connected to ledPin in the circuit is GPIO17 and GPIO17 is defined as 0 in the wiringPi numbering. You can refer to the corresponding table.

```
#define ledPin    17 //define the led pin number
```

GPIO Numbering Relationship

WingPi	BCM(Extension)	Physical		BCM(Extension)	WingPi
3.3V	3.3V	1	2	5V	5V
8	SDA1	3	4	5V	5V
9	SCL1	5	6	GND	GND
7	GPIO4	7	8	GPIO14/TXD0	15
GND	GND	9	10	GPIO15/RXD0	16
0	GPIO17	11	12	GPIO18	1
2	GPIO27	13	14	GND	GND
3	GPIO22	15	16	GPIO23	4
3.3V	3.3V	17	18	GPIO24	5
12	GPIO10/MOSI)	19	20	GND	GND
13	GPIO9/MOIS	21	22	GPIO25	6
14	GPIO11/SCLK	23	24	GPIO8 /CE0	10
GND	GND	25	26	GPIO7 CE1	11
30	GPIO0/SDA0	27	28	GPIO1 /SCL0	31
21	GPIO5	29	30	GND	GND
22	GPIO6	31	32	GPIO12	26
23	GPIO13	33	34	GND	GND
24	GPIO19	35	36	GPIO16	27
25	GPIO26	37	38	GPIO20	28
GND	GND	39	40	GPIO21	29

In the main function main(), initialize wiringPi first.

```
    wiringPiSetupGpio();    //Initialize wiringPi. Use BCM Number.
```

After the wiringPi is initialized successfully, you can set the ledPin to output mode and then enter the while loop, which is an endless loop (a while loop). That is, the program will always be executed in this cycle, unless it is ended because of external factors. In this loop, use digitalWrite (ledPin, HIGH) to make ledPin output high level, then LED turns ON. After a period of time delay, use digitalWrite(ledPin, LOW) to make ledPin output low level, then LED turns OFF, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
    pinMode(ledPin, OUTPUT);//Set the pin mode
    printf("Using pin%d\n",%ledPin);    //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
        printf("led turned on >>>\n");      //Output information on terminal
        delay(1000);                        //Wait for 1 second
        digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
        printf("led turned off <<<\n");         //Output information on terminal
        delay(1000);                        //Wait for 1 second
    }
```

Python Code 01.1.1 Blink

Now, we will use Python language to make a LED blink.

If you have any concerns, please contact us via: support@freenove.com

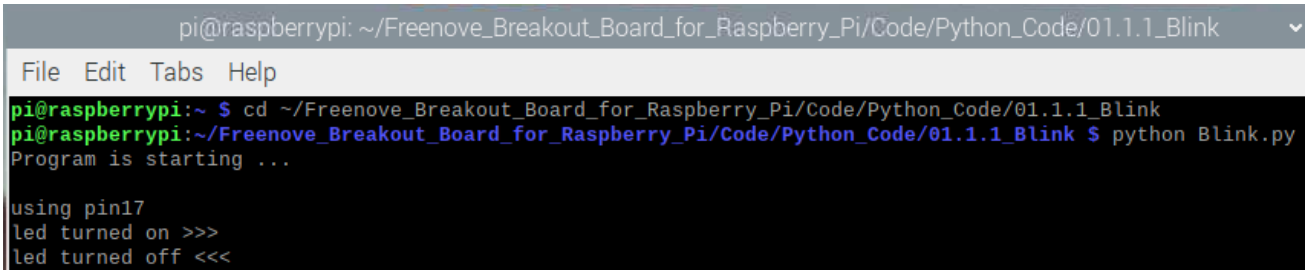
1. Use cd command to enter 01.1.1_Blink directory of Python code.

```
cd ~/Freenove_Breakout_Board_for_Raspberry_Pi/Code/Python_Code/01.1.1_Blink
```

2. Use python command to execute python code blink.py.

```
python Blink.py
```

The LED starts blinking.



You can press "**Ctrl+C**" to end the program. The following is the program code:

```
1  from gpiozero import LED
2  from time import sleep
3
4  led = LED(17)           # define LED pin according to BCM Numbering
5  '''
6  # pins numbering, the following lines are all equivalent
7  led = LED(17)           # BCM
8  led = LED("GPIO17")     # BCM
9  led = LED("BCM17")      # BCM
10 led = LED("BOARD11")    # BOARD
11 led = LED("WPI0")       # WiringPi
12 led = LED("J8:11")      # BOARD
13 '''
14
15 def loop():
16     while True:
17         led.on()    # turn on LED
18         print ('led turned on >>>')  # print message on terminal
19         sleep(1)    # wait 1 second
20         led.off()   # turn off LED
21         print ('led turned off <<<') # print message on terminal
22         sleep(1)    # wait 1 second
23
24 if __name__ == '__main__':    # Program entrance
25     print ('Program is starting ... \n')
26     try:
```

```
27          loop()
28      except KeyboardInterrupt:  # Press ctrl-c to end the program.
29          print("Ending program")
30      finally:
            led.close()
```

In Python, libraries and functions used in a script must be imported by name at the top of the file, with the exception of the functions built into Python by default.
For example, to use the LED interface from GPIO Zero, it should be explicitly imported:

```
from gpiozero import LED
```

Now LEDis available directly in your script:

```
led = LED(17)           # define LED pin according to BCM Numbering
#led = LED("J8:11")     # BOARD Numbering
```

Alternatively, the whole GPIO Zero library can be imported:

```
import gpiozero
```

In this case, all references to items within GPIO Zero must be prefixed:

```
led = gpiozero.LED(17)          # define LED pin according to BCM Numbering
#led = gpiozero.LED("J8:11")    # BOARD Numbering
```

In loop(), there is a while loop, which is an endless loop (a while loop). That is, the program will always be executed in this loop, unless it is ended because of external factors. In this loop, set ledPin output high level, then the LED turns ON. After a period of time delay, set ledPin output low level, then the LED turns OFF, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
def loop():
    while True:
        GPIO.output(ledPin, GPIO.HIGH)  # make ledPin output HIGH level to turn on led
        print ('led turned on >>>')        # print information on terminal
        time.sleep(1)                   # Wait for 1 second
        GPIO.output(ledPin, GPIO.LOW)   # make ledPin output LOW level to turn off led
        print ('led turned off <<<')
        time.sleep(1)                   # Wait for 1 second
```

Reference

About GPIO Zero:

GPIO Zero

A simple interface to GPIO devices with Raspberry Pi, Using the GPIO Zero library makes it easy to get started with controlling GPIO devices with Python. The library is comprehensively documented at
https://gpiozero.readthedocs.io/en/stable/
https://github.com/gpiozero/gpiozero

For more information about the methods used by the LED class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#led
For more information about the methods used by the DigitalOutputDevice class in the GPIO Zero library,please refer to:
https://gpiozero.readthedocs.io/en/stable/api_output.html#digitaloutputdevice
"import time" time is a module of python.
https://docs.python.org/2/library/time.html?highlight=time%20time#module-time

GPIO Numbering Relationship

WingPi	BCM(Extension)	Physical		BCM(Extension)	WingPi
3.3V	**3.3V**	1	2	5V	**5V**
8	SDA1	3	4	**5V**	**5V**
9	SCL1	5	6	**GND**	**GND**
7	GPIO4	7	8	GPIO14/TXD0	15
GND	**GND**	9	10	GPIO15/RXD0	16
0	GPIO17	11	12	GPIO18	1
2	GPIO27	13	14	**GND**	**GND**
3	GPIO22	15	16	GPIO23	4
3.3V	**3.3V**	17	18	GPIO24	5
12	GPIO10/MOSI)	19	20	**GND**	**GND**
13	GPIO9/MOIS	21	22	GPIO25	6
14	GPIO11/SCLK	23	24	GPIO8 /CE0	10
GND	**GND**	25	26	GPIO7 CE1	11
30	GPIO0/SDA0	27	28	GPIO1 /SCL0	31
21	GPIO5	29	30	**GND**	**GND**
22	GPIO6	31	32	GPIO12	26
23	GPIO13	33	34	**GND**	**GND**
24	GPIO19	35	36	GPIO16	27
25	GPIO26	37	38	GPIO20	28
GND	**GND**	39	40	GPIO21	29

This library uses Broadcom (BCM) pin numbering for the GPIO pins, as opposed to physical (BOARD) numbering. Unlike in the RPi.GPIO library, this is not configurable. However, translation from other schemes can be used by providing prefixes to pin numbers (see below).
Any pin marked "GPIO" in the diagram below can be used as a pin number. For example, if an LED was attached to "GPIO17" you would specify the pin number as 17 rather than 11:

If you wish to use physical (BOARD) numbering you can specify the pin number as “BOARD11”. If you are familiar with the wiringPi pin numbers (another physical layout) you could use “WPI0” instead. Finally, you can specify pins as “header:number”, e.g. “J8:11” meaning physical pin 11 on header J8 (the GPIO header on modern Pis). Hence, the following lines are all equivalent:

```
    led = LED(17)
    led = LED("GPIO17")
    led = LED("BCM17")
    led = LED("BOARD11")
    led = LED("WPI0")
    led = LED("J8:11")
```

Note that these alternate schemes are merely translations. If you request the state of a device on the command line, the associated pin number will always be reported in the Broadcom (BCM) scheme:

```
    led = LED("BOARD11")
    led
<gpiozero.LED object on pin GPIO17, active_high=True, is_active=False>
```

In gpiozero, at the end of your script, cleanup is run automatically, restoring your GPIO pins to the state they were found.To explicitly close a connection to a pin, you can manually call the close() method on a device object:

```
    led = LED(17)
    led.on()
    led
<gpiozero.LED object on pin GPIO17, active_high=True, is_active=True>
    led.close()
    led
<gpiozero.LED object closed>
```

This means that you can reuse the pin for another device, and that despite turning the LED on (and hence, the pin high), after calling close() it is restored to its previous state (LED off, pin low).
In this tutorial, most projects have added an active run cleanup program to restore the GPIO pin to the found default state.

What's next?

Thanks for your reading.

This book is all over here. If you find any mistakes, missions or you have other ideas and questions about contents of this book or the kit and ect., please feel free to contact us, and we will check and correct it as soon as possible.

After completing the contents in this book, you can try to reform this smart car, such as purchasing and installing other Freenove electronic modules, or improving the code to achieve different functions. We will also try our best to add more new functions and update the code on our github (https://github.com/freenove).

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and orther interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

www.freenove.com

https://www.amazon.com/freenove

Thank you again for choosing Freenove products.