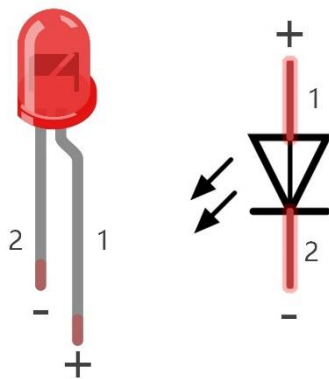


Component knowledge

LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-) output, which is also referred to as Ground (GND). This type of component is known as "Polar" (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burnt out.



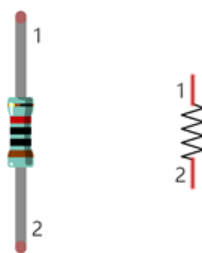
LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). $1M\Omega=1000k\Omega$, $1k\Omega=1000\Omega$.

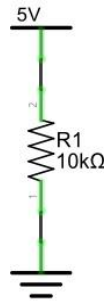
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: $I=V/R$ known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is: $I=U/R=5V/10k\Omega=0.0005A=0.5mA$.

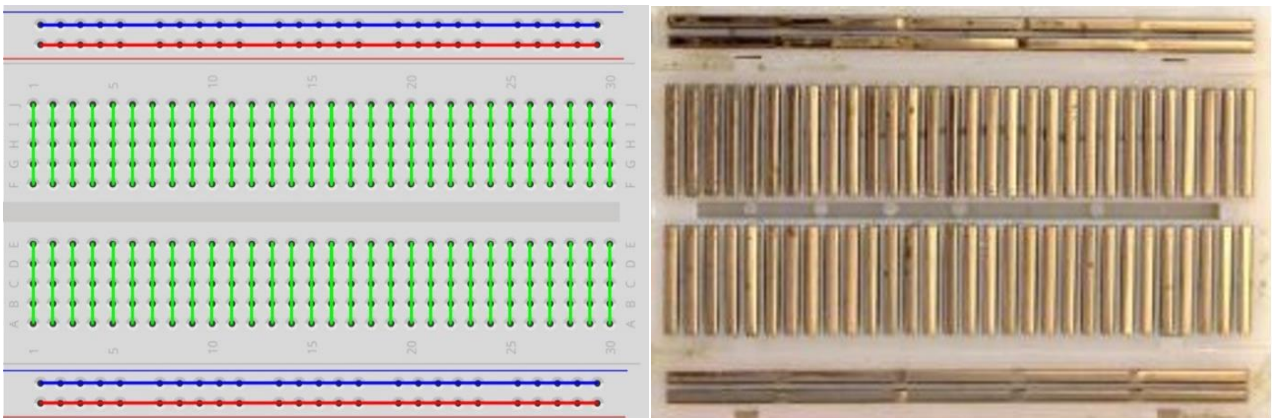


WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and are non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

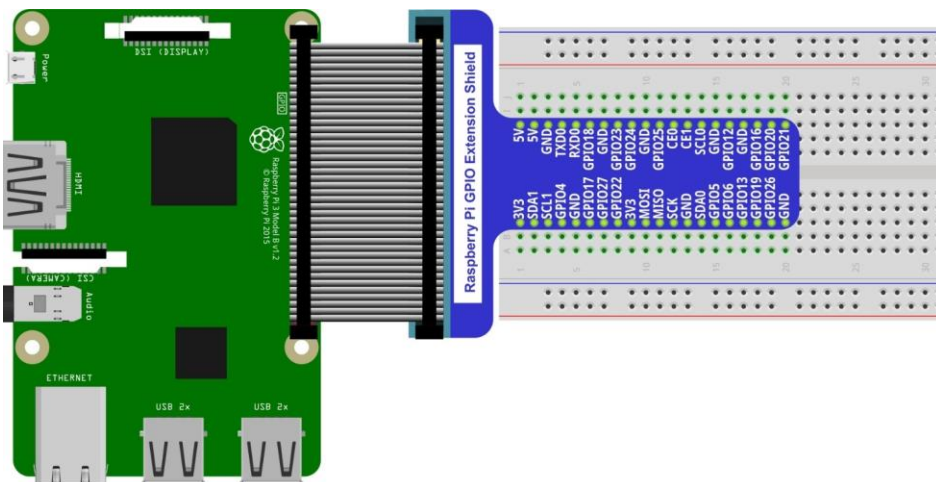
Breadboard

Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the ways the pins have shared electrical connection and the right picture shows the actual internal metal, which connects these rows electrically.



GPIO Extension Board

GPIO board is a convenient way to connect the RPi I/O ports to the breadboard directly. The GPIO pin sequence on Extension Board is identical to the GPIO pin sequence of RPi.



Code

According to the circuit, when the GPIO17 of RPi output level is high, the LED turns ON. Conversely, when the GPIO17 RPi output level is low, the LED turns OFF. Therefore, we can let GPIO17 cycle output high and output low level to make the LED blink. We will use both C code and Python code to achieve the target.

C Code 1.1.1 Blink

First, enter this command into the Terminal one line at a time. Then observe the results it brings on your project, and learn about the code in detail.

If you want to execute it with editor, please refer to section [Code Editor](#) to configure.

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

It is recommended that to execute the code via command line.

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_Kit/Code/C_Code/01.1.1_Blink
```

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

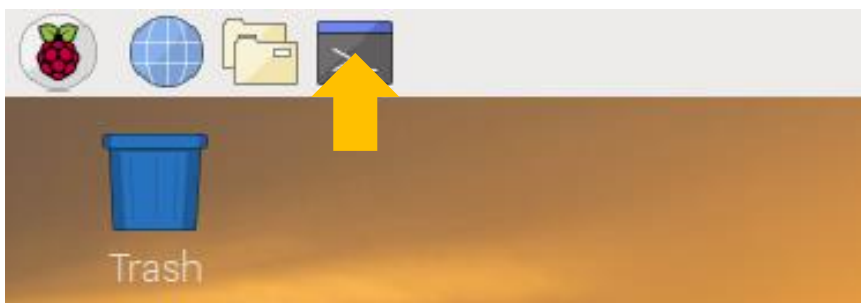
"l" of "lwiringPi" is low case of "L".

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

4. Then run the generated file "blink".

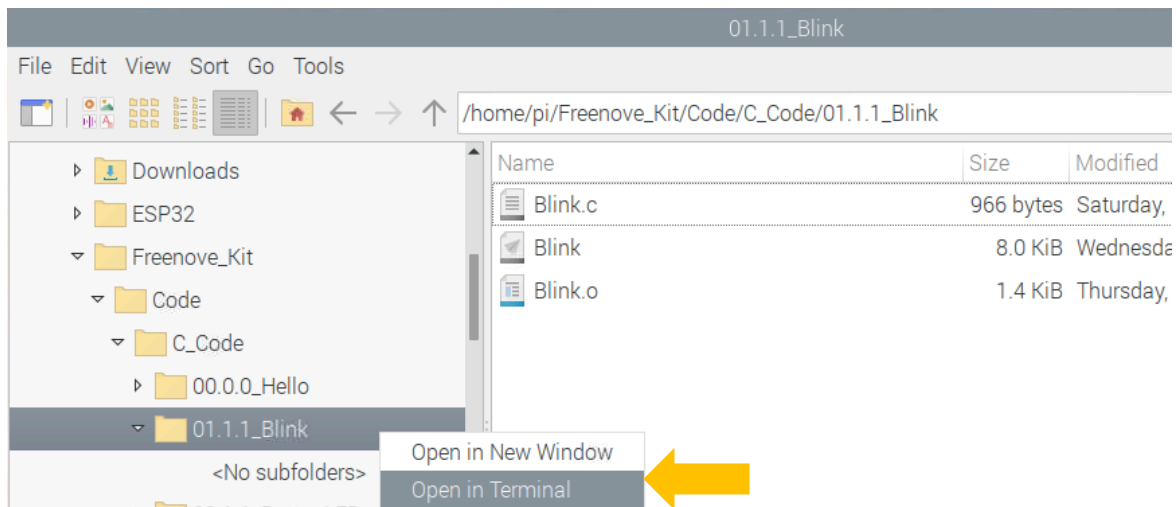
```
sudo ./Blink
```

Now your LED should start blinking! CONGRATUALTIONS! You have successfully completed your first RPi circuit!



```
pi@raspberrypi: ~/Freenove_Kit/Code/C_Code/01.1.1_Blink
File Edit Tabs Help
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Code/C_Code/01.1.1_Blink/
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/01.1.1_Blink $ gcc Blink.c -o Blink -lwiringPi
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/01.1.1_Blink $ sudo ./Blink
Program is starting ...
Using pin0
led turned on >>>
led turned off <<<
```

You can also use the file browser. On the left of folder tree, right-click the folder you want to enter, and click "Open in Terminal".



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    0 //define the led pin number
5
6  void main(void)
7  {
8      printf("Program is starting ... \n");
9
10     wiringPiSetup(); //Initialize wiringPi.
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.

For more related wiringpi functions, please refer to <http://wiringpi.com/reference/>

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

```
#define ledPin 0 //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.

```
wiringPiSetup(); //Initialize wiringPi.
```

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 1.1.1 Blink

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

First, observe the project result, and then learn about the code in detail.

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_Kit/Code/Python_Code/01.1.1_Blink
```

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

```
python Blink.py
```

The LED starts blinking.



```
pi@raspberrypi: ~/Freenove_Kit/Code/Python_Code/01.1.1_Blink
File Edit Tabs Help
pi@raspberrypi:~ $ cd Freenove_Kit/Code/Python_Code/01.1.1_Blink/
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/01.1.1_Blink $ python Blink.py
Program is starting ...

using pin11
led turned on >>>
led turned off <<<
```

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

```
1 import RPi.GPIO as GPIO
2 import time
3
4 ledPin = 11    # define ledPin
5
6 def setup():
7     GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
8     GPIO.setup(ledPin, GPIO.OUT)   # set the ledPin to OUTPUT mode
9     GPIO.output(ledPin, GPIO.LOW)  # make ledPin output LOW level
10    print ('using pin%d'%ledPin)
11
12 def loop():
13     while True:
14         GPIO.output(ledPin, GPIO.HIGH) # make ledPin output HIGH level to turn on led
15         print ('led turned on >>>')    # print information on terminal
16         time.sleep(1)                   # Wait for 1 second
17         GPIO.output(ledPin, GPIO.LOW)   # make ledPin output LOW level to turn off led
18         print ('led turned off <<<')
19         time.sleep(1)                   # Wait for 1 second
20
21 def destroy():
22     GPIO.cleanup()                    # Release all GPIO
23
24 if __name__ == '__main__':           # Program entrance
```

```

25     print ('Program is starting ... \n')
26     setup()
27     try:
28         loop()
29     except KeyboardInterrupt:  # Press ctrl-c to end the program.
30         destroy()

```

About RPi.GPIO:

RPi.GPIO

This is a Python module to control the GPIO on a Raspberry Pi. It includes basic output function and input function of GPIO, and functions used to generate PWM.

GPIO.setmode(mode)

Sets the mode for pin serial number of GPIO.

mode=GPIO.BOARD, which represents the GPIO pin serial number based on physical location of RPi.

mode=GPIO.BCM, which represents the pin serial number based on CPU of BCM chip.

GPIO.setup(pin, mode)

Sets pin to input mode or output mode, "pin" for the GPIO pin, "mode" for INPUT or OUTPUT.

GPIO.output(pin, mode)

Sets pin to output mode, "pin" for the GPIO pin, "mode" for HIGH (high level) or LOW (low level).

For more functions related to RPi.GPIO, please refer to:

<https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/>

"import time" time is a module of python.

<https://docs.python.org/2/library/time.html?highlight=time%20time#module-time>

In subfunction setup(), GPIO.setmode (GPIO.BOARD) is used to set the serial number for GPIO based on physical location of the pin. GPIO17 uses pin 11 of the board, so define ledPin as 11 and set ledPin to output mode (output low level).

```

ledPin = 11    # define ledPin

def setup():
    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT) # set the ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW) # make ledPin output LOW level
    print ('using pin%d'%ledPin)

```


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 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
```

Finally, when the program is terminated, subfunction (a function within the file) will be executed, the LED will be turned off and then the IO port will be released. If you close the program Terminal directly, the program will also be terminated but the finish() function will not be executed. Therefore, the GPIO resources will not be released which may cause a warning message to appear the next time you use GPIO. Therefore, do not get into the habit of closing Terminal directly.

```
def finish():
    GPIO.cleanup() # Release all GPIO
```

Other Code Editors (Optional)

If you want to use other editor to edit and execute the code, you can learn them in this section.

vi

Here we will introduce three kinds of code editor: vi, nano and Geany. Among them, nano and vi are used to edit files directly in the terminal. And Geany is an independent editing software, which is recommended for beginner. We will use the three editors to open an example code "Hello.c" respectively. First we will show how to use vi and nano editor:

First, use `cd` command to enter the sample code folder.

```
cd ~  
cd ~/Freenove_Kit/Code/C_Code/00.0.0_Hello
```

Use the vi editor to open the file "Hello.c", then press ": q" and "Enter" to exit.

```
vi Hello.c
```

As is shown below:

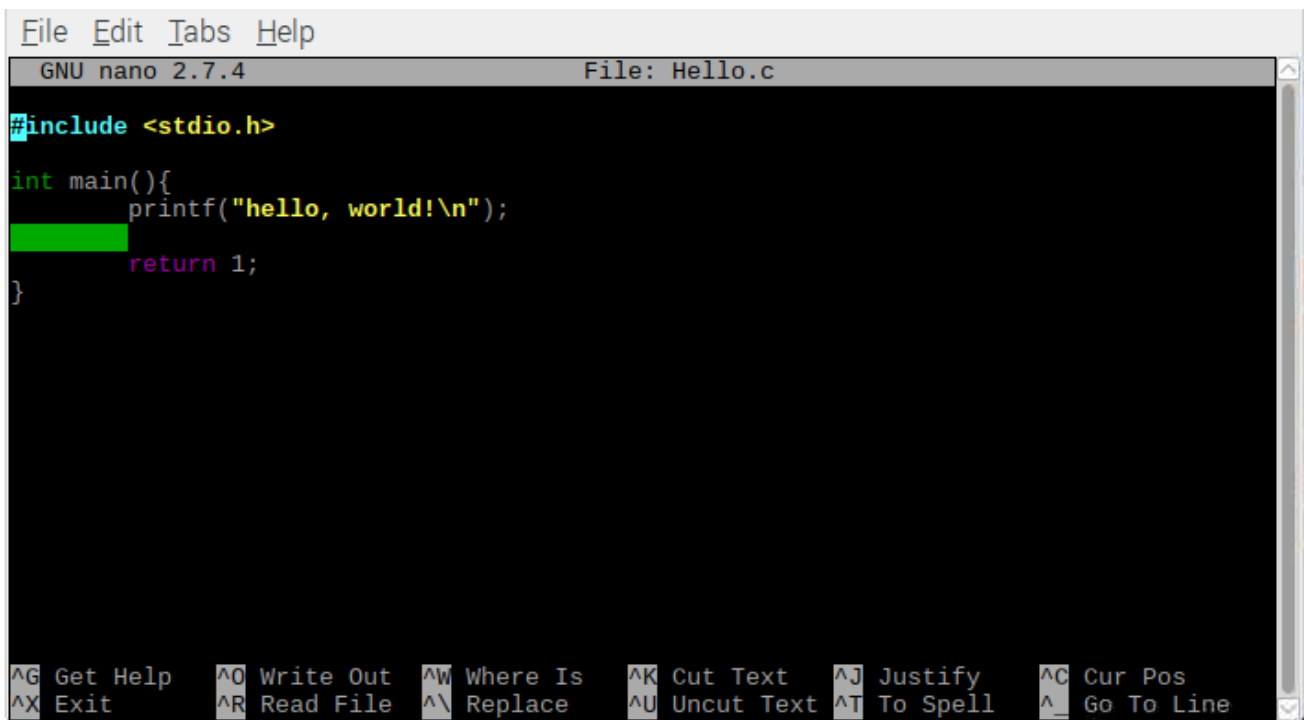
[illegible]

nano

Use the nano editor to open the file "Hello.c", then press " Ctrl+X " to exit.

```
nano Hello.c
```

As is shown below:



```
File Edit Tabs Help
GNU nano 2.7.4 File: Hello.c

#include <stdio.h>

int main(){
    printf("hello, world!\n");
    return 1;
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

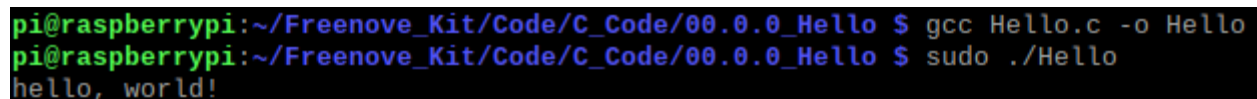
Use the following command to compile the code to generate the executable file "Hello".

```
gcc Hello.c -o Hello
```

Use the following command to run the executable file "Hello".

```
sudo ./Hello
```

After the execution, "Hello, World!" is printed out in terminal.



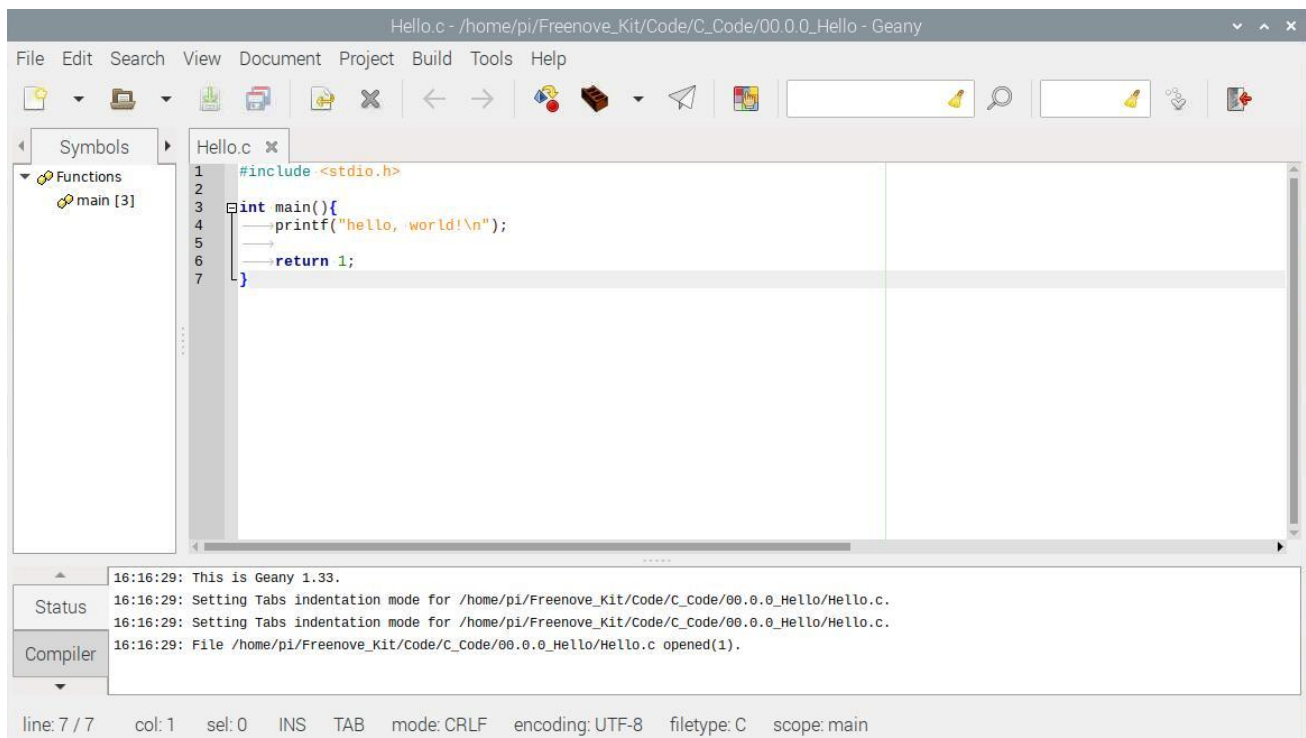
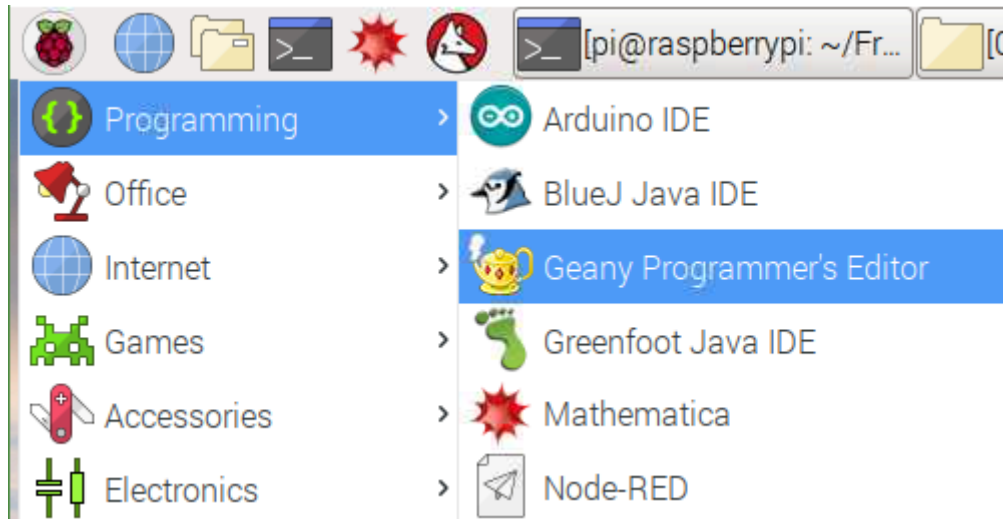
```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ gcc Hello.c -o Hello
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/00.0.0_Hello $ sudo ./Hello
hello, world!
```

geany

Next, learn to use the Geany editor. Use the following command to open the Geany in the sample file "Hello.c" file directory path.

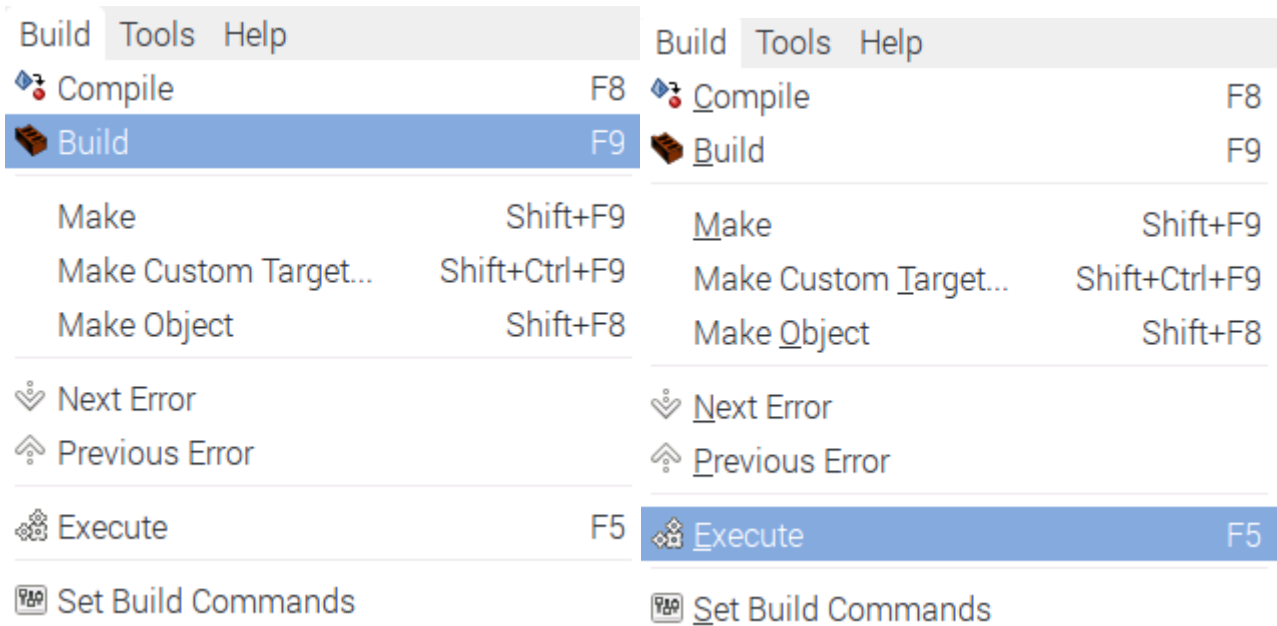
geany Hello.c

Or find and open Geany directly in the desktop main menu, and then click **File→Open** to open the "Hello.c", Or drag "Hello.c" to Geany directly.

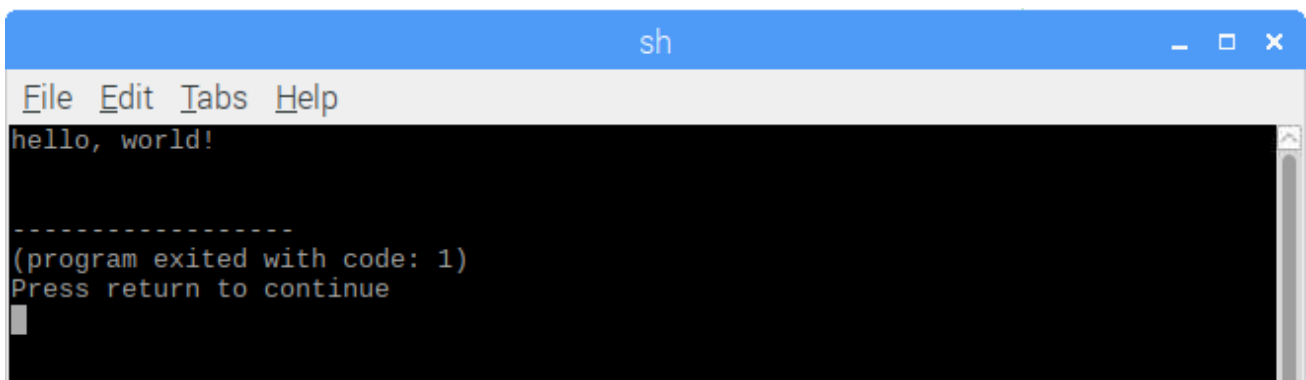


If you want to create a new code, click **File→New→File→Save as (name.c or name.py)**. Then write the code.

Generate an executable file by clicking menu bar Build->Build, then execute the generated file by clicking menu bar Build->Execute.



After the execution, a new terminal window will output the characters "Hello, World!", as shown below:



You can click Build->Set Build Commands to set compiler commands. In later projects, we will use various compiler command options. **If you choose to use Geany, you will need change the compiler command here.** As is shown below:

#	Label	Command	Working directory	Reset
C commands				
1.	Compile	gcc -Wall -c "%f" -lwiringPi		
2.	Build	gcc -Wall -o "%e" "%f" -lwiringPi		
3.	Lint	cppcheck --language=c --enable=v		
Error regular expression:				
Independent commands				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
Error regular expression:				
<i>Note: Item 2 opens a dialogue and appends the response to the command.</i>				
Execute commands				
1.	Execute	"/%e"		
2.				
<i>%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details.</i>				
				<input type="button" value="Cancel"/> <input type="button" value="OK"/>

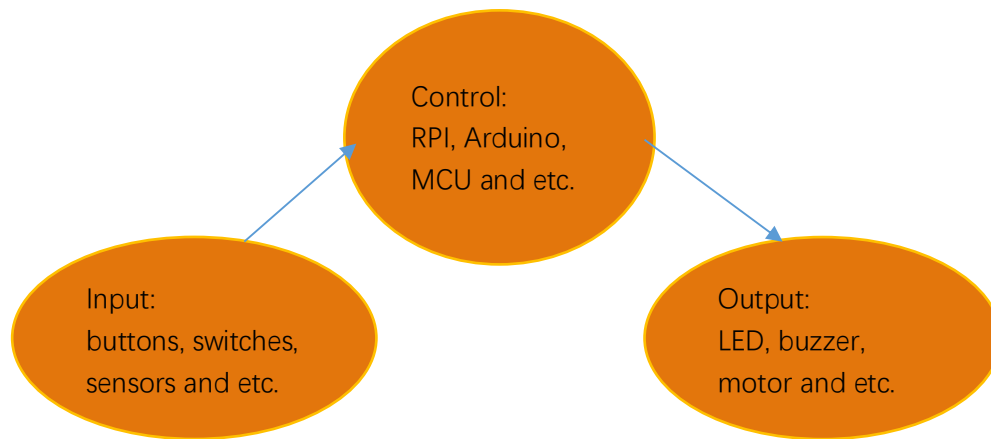
Here we have identified three code editors: vi, nano and Geany. There are also many other good code editors available to you, and you can choose whichever you prefer to use.

In later projects, we will only use terminal to execute the project code. This way will not modify the code by mistake.

Other Freenove Product

Chapter 2 Buttons & LEDs

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and RPI was the control part. In practical applications, we not only make LEDs flash, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as turn on LEDs, make a buzzer beep and so on.








Next, we will build a simple control system to control an LED through a push button switch.

Project 2.1 Push Button Switch & LED

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

Component List

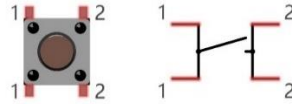
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	LED x1 	Resistor 220Ω x1 	Resistor 10kΩ x2 	Push Button Switch x1 
Jumper Wire 				

Please Note: In the code “button” represents switch action.

Component knowledge

Push Button Switch

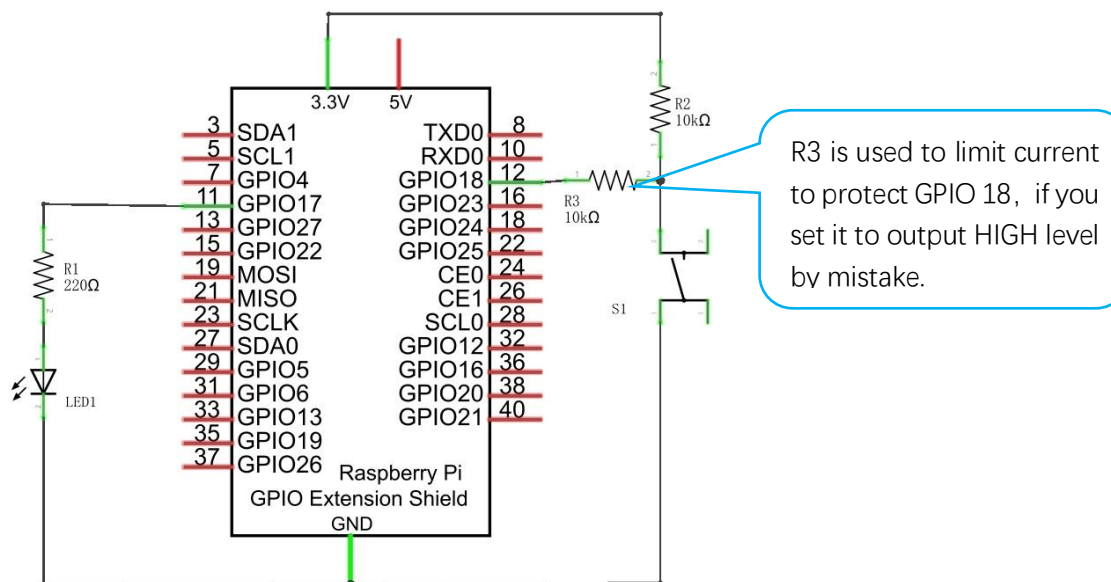
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

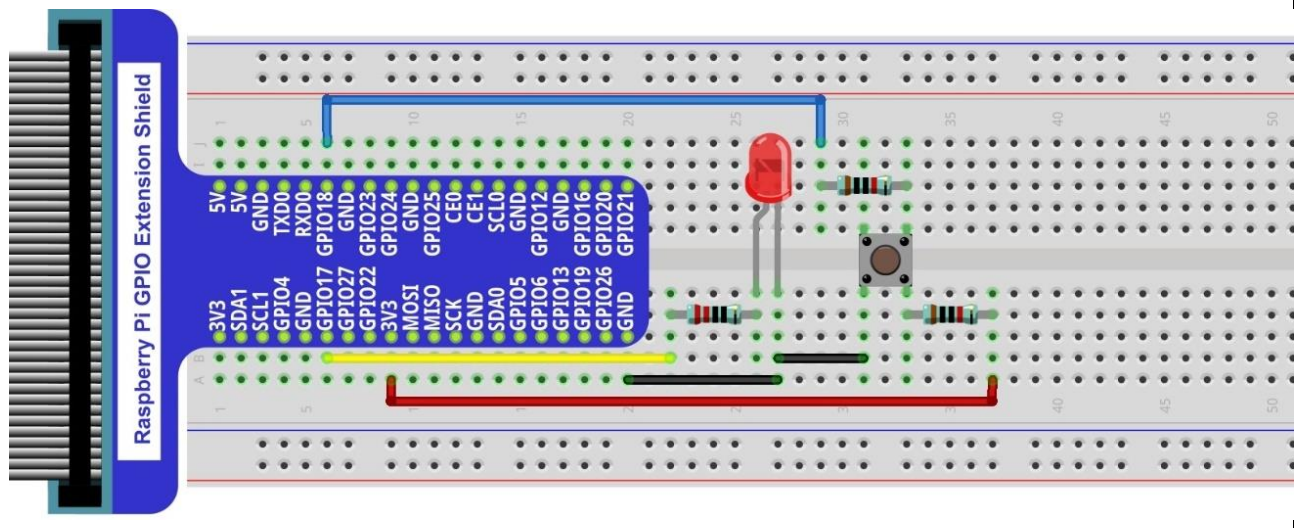
Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via:

support@freenove.com



There are two kinds of push button switch in this kit.

The smaller push button switches are contained in a plastic bag.

Code

This project is designed for learning how to use Push Button Switch to control an LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch.

C Code 2.1.1 ButtonLED

First, observe the project result, then learn about the code in detail.

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

1. Use cd command to enter 02.1.1_ButtonLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/02.1.1_ButtonLED
```

2. Use the following command to compile the code "ButtonLED.c" and generate executable file "ButtonLED"

```
gcc ButtonLED.c -o ButtonLED -lwiringPi
```

3. Then run the generated file "ButtonLED".

```
sudo ./ButtonLED
```

Later, the terminal window continues to print out the characters "led off...". Press the button, then LED is turned on and then terminal window prints out the "led on...". Release the button, then LED is turned off and then terminal window prints out the "led off...". You can press "Ctrl+C" to terminate the program.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define ledPin    0    //define the ledPin
5  #define buttonPin 1    //define the buttonPin
6
7  void main(void)
8  {
9      printf("Program is starting ... \n");
10
11     wiringPiSetup(); //Initialize wiringPi.
12
13     pinMode(ledPin, OUTPUT); //Set ledPin to output
14     pinMode(buttonPin, INPUT); //Set buttonPin to input
15
16     pullUpDnControl(buttonPin, PUD_UP); //pull up to HIGH level
17     while(1) {
18         if(digitalRead(buttonPin) == LOW) { //button is pressed
19             digitalWrite(ledPin, HIGH); //Make GPIO output HIGH level
20             printf("Button is pressed, led turned on >>>\n"); //Output information on
21             terminal
22         }
23         else { //button is released
```

```

24         digitalWrite(ledPin, LOW); //Make GPIO output LOW level
25         printf("Button is released, led turned off <<<\n"); //Output information on
26         terminal
27     }
28 }
29 }

```

In the circuit connection, LED and Button are connected with GPIO17 and GPIO18 respectively, which correspond to 0 and 1 respectively in wiringPi. So define ledPin and buttonPin as 0 and 1 respectively.

```

#define ledPin 0 //define the ledPin
#define buttonPin 1 //define the buttonPin

```

In the while loop of main function, use digitalRead(buttonPin) to determine the state of Button. When the button is pressed, the function returns low level, the result of "if" is true, and then turn on LED. Or, turn off LED.

```

if(digitalRead(buttonPin) == LOW) { //button has pressed down
    digitalWrite(ledPin, HIGH); //led on
    printf("led on...\n");
}
else { //button has released
    digitalWrite(ledPin, LOW); //led off
    printf("...led off\n");
}

```

Reference:

```
int digitalRead (int pin);
```

This function returns the value read at the given pin. It will be "HIGH" or "LOW"(1 or 0) depending on the logic level at the pin.

Python Code 2.1.1 ButtonLED

First, observe the project result, then learn about the code in detail. Remember in code “button” = switch function

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

1. Use cd command to enter 02.1.1_ButtonLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/02.1.1_ButtonLED
```

2. Use Python command to execute btnLED.py.

```
python ButtonLED.py
```

Then the Terminal window continues to show the characters “led off...”, press the switch button and the LED turns ON and then Terminal window shows “led on...”. Release the button, then LED turns OFF and then the terminal window text “led off...” appears. You can press “Ctrl+C” at any time to terminate the program.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2
3  ledPin = 11    # define ledPin
4  buttonPin = 12    # define buttonPin
5
6  def setup():
7
8      GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
9      GPIO.setup(ledPin, GPIO.OUT)    # set ledPin to OUTPUT mode
10     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
11     INPUT mode
12
13     def loop():
14         while True:
15             if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
16                 GPIO.output(ledPin,GPIO.HIGH)    # turn on led
17                 print ('led turned on >>>')      # print information on terminal
18             else : # if button is releassed
19                 GPIO.output(ledPin,GPIO.LOW) # turn off led
20                 print ('led turned off <<<')
21
22     def destroy():
23         GPIO.cleanup()                  # Release GPIO resource
24
25     if __name__ == '__main__':         # Program entrance
26         print ('Program is starting...')
27         setup()
28         try:
29             loop()
30         except KeyboardInterrupt:      # Press ctrl-c to end the program.
31             destroy()
```

In subfunction setup (), GPIO.setmode (GPIO.BOARD) is used to set the serial number of the GPIO, which is based on physical location of the pin. Therefore, GPIO17 and GPIO18 correspond to pin11 and pin12 respectively in the circuit. Then set ledPin to output mode, buttonPin to input mode with a pull resistor.

```
ledPin = 11    # define ledPin
buttonPin = 12 # define buttonPin

def setup():

    GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
    GPIO.setup(ledPin, GPIO.OUT) # set ledPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # set buttonPin to PULL UP
    INPUT mode
```

The loop continues endlessly to judge whether the key is pressed. When the button is pressed, the GPIO.input(buttonPin) will return low level, then the result of "if" is true, ledPin outputs high level, LED is turned on. Otherwise, LED will be turned off.

```
def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
            GPIO.output(ledPin,GPIO.HIGH)    # turn on led
            print ('led turned on >>>')      # print information on terminal
        else : # if button is released
            GPIO.output(ledPin,GPIO.LOW) # turn off led
            print ('led turned off <<<')
```

Execute the function destroy (), close the program and release the occupied GPIO pins.

About function GPIO.input ():

GPIO.input()

This function returns the value read at the given pin. It will be "**HIGH**" or "**LOW**"(1 or 0) depending on the logic level at the pin.

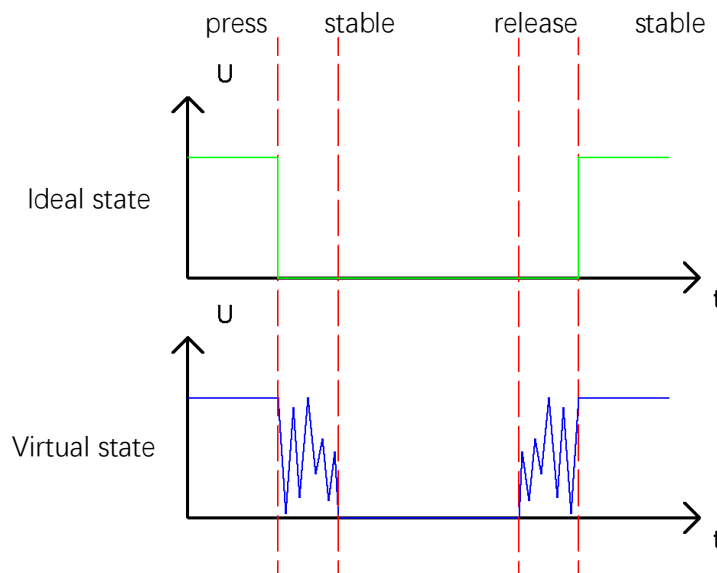
Project 2.2 MINI Table Lamp

We will also use a Push Button Switch, LED and RPi to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

First, let us learn something about the push button switch.

Debounce a Push Button Switch

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it stabilizes in a new state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.



Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).

This project needs the same components and circuits as we used in the previous section.

Code

In this project, we still detect the state of Push Button Switch to control an LED. Here we need to define a variable to define the state of LED. When the button switch is pressed once, the state of LED will be changed once. This will allow the circuit to act as a virtual table lamp.

C Code 2.2.1 Tablelamp

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 02.2.1_Tablelamp directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/02.2.1_Tablelamp
```

2. Use the following command to compile "Tablelamp.c" and generate executable file "Tablelamp".

```
gcc Tablelamp.c -o Tablelamp -lwiringPi
```

3. Tablelamp: Then run the generated file "Tablelamp".

```
sudo ./Tablelamp
```

When the program is executed, press the Button Switch once, the LED turns ON. Pressing the Button Switch again turns the LED OFF.

```

1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define ledPin    0    //define the ledPin
5  #define buttonPin 1    //define the buttonPin
6  int ledState=LOW;    //store the State of led
7  int buttonState=HIGH; //store the State of button
8  int lastbuttonState=HIGH; //store the lastState of button
9  long lastChangeTime; //store the change time of button state
10 long captureTime=50; //set the stable time for button state
11 int reading;
12 int main(void)
13 {
14     printf("Program is starting...\n");
15
16     wiringPiSetup(); //Initialize wiringPi.
17
18     pinMode(ledPin, OUTPUT); //Set ledPin to output
19     pinMode(buttonPin, INPUT); //Set buttonPin to input
20
21     pullUpDnControl(buttonPin, PUD_UP); //pull up to high level
22     while(1) {
23         reading = digitalRead(buttonPin); //read the current state of button
24         if( reading != lastbuttonState) { //if the button state has changed, record the time
25             point
26                 lastChangeTime = millis();
27         }

```

```

28      //if changing-state of the button last beyond the time we set, we consider that
29      //the current button state is an effective change rather than a buffeting
30      if(millis() - lastChangeTime > captureTime){
31          //if button state is changed, update the data.
32          if(reading != buttonState){
33              buttonState = reading;
34              //if the state is low, it means the action is pressing
35              if(buttonState == LOW){
36                  printf("Button is pressed!\n");
37                  ledState = !ledState; //Reverse the LED state
38                  if(ledState){
39                      printf("turn on LED ... \n");
40                  }
41                  else {
42                      printf("turn off LED ... \n");
43                  }
44              }
45              //if the state is high, it means the action is releasing
46              else {
47                  printf("Button is released!\n");
48              }
49          }
50      }
51      digitalWrite(ledPin, ledState);
52      lastbuttonState = reading;
53  }
54
55  return 0;
56  }

```

This code focuses on eliminating the buffeting (bounce) of the button switch. We define several variables to define the state of LED and button switch. Then read the button switch state constantly in while () to determine whether the state has changed. If it has, then this time point is recorded.

```

reading = digitalRead(buttonPin); //read the current state of button
if( reading != lastbuttonState){
    lastChangeTime = millis();
}

```

millis()

This returns a number representing the number of milliseconds since your program called one of the wiringPiSetup functions. It returns to an unsigned 32-bit number value after 49 days because it “wraps” around and restarts to value 0.

Then according to the recorded time point, evaluate the duration of the button switch state change. If the duration exceeds captureTime (buffeting time) we have set, it indicates that the state of the button switch has changed. During that time, the while () is still detecting the state of the button switch, so if there is a change, the time point of change will be updated. Then the duration will be evaluated again until the duration is determined to be a stable state because it exceeds the time value we set.

```
if(millis() - lastChangeTime > captureTime) {  
    //if button state is changed, update the data.  
    if(reading != buttonState) {  
        buttonState = reading;
```

Finally, we need to judge the state of Button Switch. If it is low level, the changing state indicates that the button Switch has been pressed, if the state is high level, then the button has been released. Here, we change the status of the LED variable, and then update the state of the LED.

```
if(buttonState == LOW) {  
    printf("Button is pressed!\n");  
    ledState = !ledState; //Reverse the LED state  
    if(ledState) {  
        printf("turn on LED ... \n");  
    }  
    else {  
        printf("turn off LED ... \n");  
    }  
}  
//if the state is high, it means the action is releasing  
else {  
    printf("Button is released!\n");  
}
```

Python Code 2.2.1 Tablelamp

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 02.2.1_Tablelamp directory of Python code

```
cd ~/Freenove_Kit/Code/Python_Code/02.2.1_Tablelamp
```

2. Use python command to execute python code "Tablelamp.py".

```
python Tablelamp.py
```

When the program is executed, pressing the Button Switch once turns the LED ON. Pressing the Button Switch again turns the LED OFF.

```
1  import RPi.GPIO as GPIO
2
3  ledPin = 11          # define ledPin
4  buttonPin = 12       # define buttonPin
5  ledState = False
6
7  def setup():
8      GPIO.setmode(GPIO.BOARD)          # use PHYSICAL GPIO Numbering
9      GPIO.setup(ledPin, GPIO.OUT)       # set ledPin to OUTPUT mode
10     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # set buttonPin to PULL UP
11     INPUT mode
12
13     def buttonEvent(channel): # When button is pressed, this function will be executed
14         global ledState
15         print ('buttonEvent GPIO%d' %channel)
16         ledState = not ledState
17         if ledState :
18             print ('Led turned on >>>')
19         else :
20             print ('Led turned off <<<')
21         GPIO.output(ledPin, ledState)
22
23     def loop():
24         #Button detect
25         GPIO.add_event_detect(buttonPin, GPIO.FALLING, callback = buttonEvent, bouncetime=300)
26         while True:
27             pass
28
29     def destroy():
30         GPIO.cleanup()                  # Release GPIO resource
31
32     if __name__ == '__main__':         # Program entrance
33         print ('Program is starting...')
34         setup()
35         try:
```

```
36     loop()
37 except KeyboardInterrupt: # Press ctrl-c to end the program.
38     destroy()
```

RPi.GPIO provides us with a simple but effective function to eliminate “jitter”, that is GPIO.add_event_detect(). It uses the callback function. Once it detects that the buttonPin has a specified action FALLING, it executes a specified function buttonEvent(). In the function buttonEvent, each time the ledState is reversed, the state of the LED will be updated.

```
def buttonEvent(channel): # When button is pressed, this function will be executed
    global ledState
    print ('buttonEvent GPIO%d' %channel)
    ledState = not ledState
    if ledState :
        print ('Led turned on >>>')
    else :
        print ('Led turned off <<<')
    GPIO.output(ledPin, ledState)

def loop():
    #Button detect
    GPIO.add_event_detect(buttonPin, GPIO.FALLING, callback = buttonEvent, bouncetime=300)
    while True:
        pass
```

Of course, you can also use the same programming idea in C code above to achieve this target.

GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)

This is an event detection function. The first parameter specifies the IO port to be detected. The second parameter specifies the action to be detected. The third parameter specifies a function name; the function will be executed when the specified action is detected. The fourth parameter is used to set the jitter time.




Chapter 3 LED Bar Graph

We have learned how to control one LED to blink. Next, we will learn how to control a number of LEDs.

Project 3.1 Flowing Water Light

In this project, we use a number of LEDs to make a flowing water light.

Component List

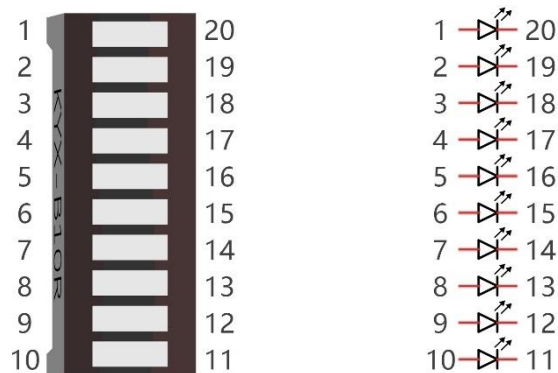
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Bar Graph LED x1 	Resistor 220Ω x10 
Jumper Wire x 1 		

Component knowledge

Let us learn about the basic features of these components to use and understand them better.

Bar Graph LED

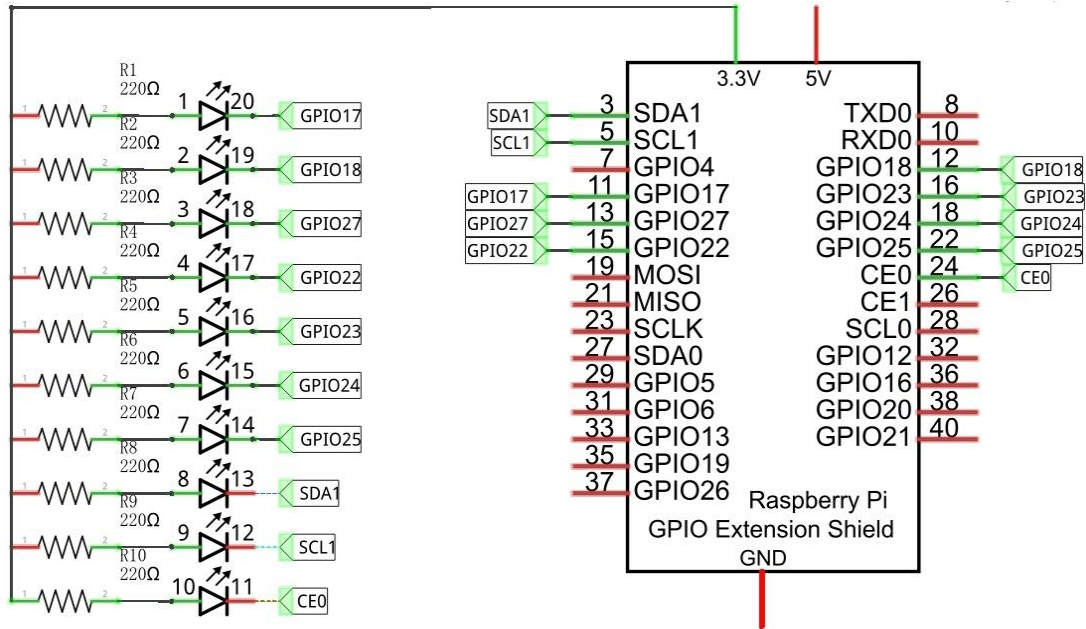
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



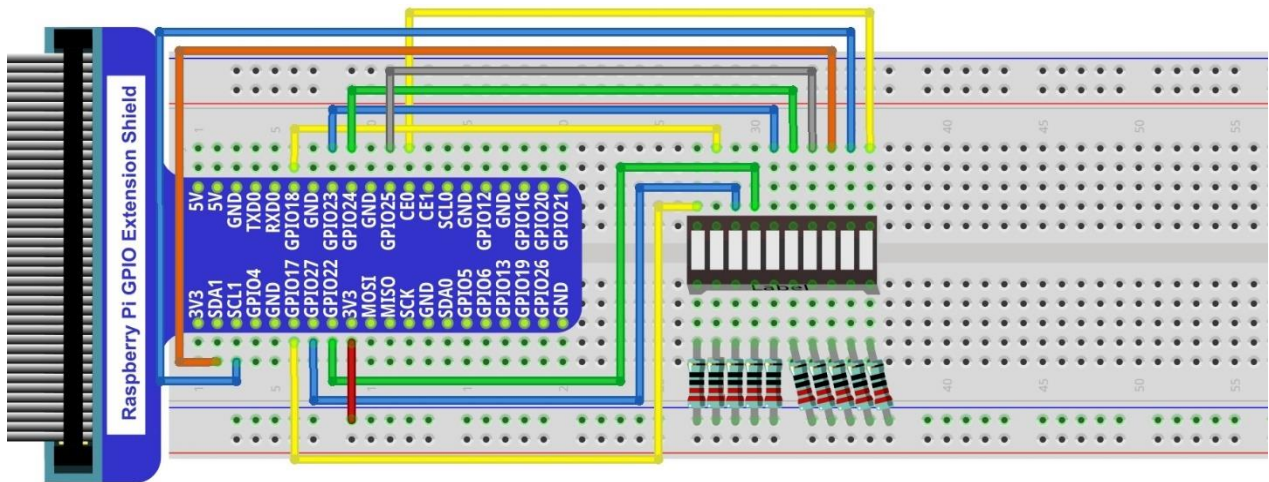
Circuit

A reference system of labels is used in the circuit diagram below. Pins with the same network label are connected together.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



If LEDbar doesn't work, rotate LEDbar 180° to try. The label is random.

In this circuit, the cathodes of the LEDs are connected to the GPIO, which is different from the previous circuit. The LEDs turn ON when the GPIO output is low level in the program.

Code

This project is designed to make a flowing water lamp, which are these actions: First turn LED #1 ON, then

turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the “movements” of flowing water.

C Code 3.1.1 LightWater

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 03.1.1_LightWater directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/03.1.1_LightWater
```

2. Use the following command to compile “LightWater.c” and generate executable file “LightWater”.

```
gcc LightWater.c -o LightWater -lwiringPi
```

3. Then run the generated file “LightWater”.

```
sudo ./LightWater
```

After the program is executed, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.

The following is the program code:

```

1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define ledCounts 10
5  int pins[ledCounts] = {0, 1, 2, 3, 4, 5, 6, 8, 9, 10};
6
7  void main(void)
8  {
9      int i;
10     printf("Program is starting ... \n");
11
12     wiringPiSetup(); //Initialize wiringPi.
13
14     for(i=0;i<ledCounts;i++){ //Set pinMode for all led pins to output
15         pinMode(pins[i], OUTPUT);
16     }
17     while(1) {
18         for(i=0;i<ledCounts;i++){ // move led(on) from left to right
19             digitalWrite(pins[i], LOW);
20             delay(100);
21             digitalWrite(pins[i], HIGH);
22         }
23         for(i=ledCounts-1;i>=0;i--){ // move led(on) from right to left
24             digitalWrite(pins[i], LOW);
25             delay(100);
26             digitalWrite(pins[i], HIGH);
27         }
28     }
29 }
```

In the program, configure the GPIO0-GPIO9 to output mode. Then, in the endless “while” loop of main function, use two “for” loop to realize flowing water light from left to right and from right to left.

```
while(1) {  
    for(i=0;i<ledCounts;i++){ // move led(on) from left to right  
        digitalWrite(pins[i],LOW);  
        delay(100);  
        digitalWrite(pins[i],HIGH);  
    }  
    for(i=ledCounts-1;i>-1;i--){ // move led(on) from right to left  
        digitalWrite(pins[i],LOW);  
        delay(100);  
        digitalWrite(pins[i],HIGH);  
    }  
}
```

Python Code 3.1.1 LightWater

First observe the project result, and then view the code.

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

1. Use cd command to enter 03.1.1_LightWater directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/03.1.1_LightWater
```

2. Use Python command to execute Python code "LightWater.py".

```
python LightWater.py
```

After the program is executed, you will see that LED Bar Graph starts with the flowing water way to be turned on from left to right, and then from right to left.

The following is the program code:

```

1  import RPi.GPIO as GPIO
2  import time
3
4  ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]
5
6  def setup():
7      GPIO.setmode(GPIO.BOARD)          # use Physical GPIO Numbering
8      GPIO.setup(ledPins, GPIO.OUT)      # set all ledPins to OUTPUT mode
9      GPIO.output(ledPins, GPIO.HIGH)    # make all ledPins output HIGH level, turn off all led
10
11  def loop():
12      while True:
13          for pin in ledPins:            # make led(on) move from left to right
14              GPIO.output(pin, GPIO.LOW)
15              time.sleep(0.1)
16              GPIO.output(pin, GPIO.HIGH)
17          for pin in ledPins[::-1]:      # make led(on) move from right to left
18              GPIO.output(pin, GPIO.LOW)
19              time.sleep(0.1)
20              GPIO.output(pin, GPIO.HIGH)
21
22  def destroy():
23      GPIO.cleanup()                    # Release all GPIO
24
25  if __name__ == '__main__':            # Program entrance
26      print('Program is starting...')
27      setup()
28      try:
29          loop()
30      except KeyboardInterrupt:          # Press ctrl-c to end the program.
31          destroy()

```

In the program, first define 10 pins connected to LED, and set them to output mode in subfunction setup(). Then in the loop() function, use two "for" loops to realize flowing water light from right to left and from left to right. ledPins[::-1] is used to get elements of ledPins in reverse order.


```
def loop():
    while True:
        for pin in ledPins:      #make led on from left to right
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)
        for pin in ledPins[::-1]:  #make led on from right to left
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)
```

Chapter 4 Analog & PWM




In previous chapters, we learned that a Push Button Switch has two states: Pressed (ON) and Released (OFF), and an LED has a Light ON and OFF state. Is there a middle or intermediated state? We will next learn how to create an intermediate output state to achieve a partially bright (dim) LED.

First, let us learn how to control the brightness of an LED.

Project 4.1 Breathing LED

We describe this project as a Breathing Light. This means that an LED that is OFF will then turn ON gradually and then gradually turn OFF like "breathing". Okay, so how do we control the brightness of an LED to create a Breathing Light? We will use PWM to achieve this goal.

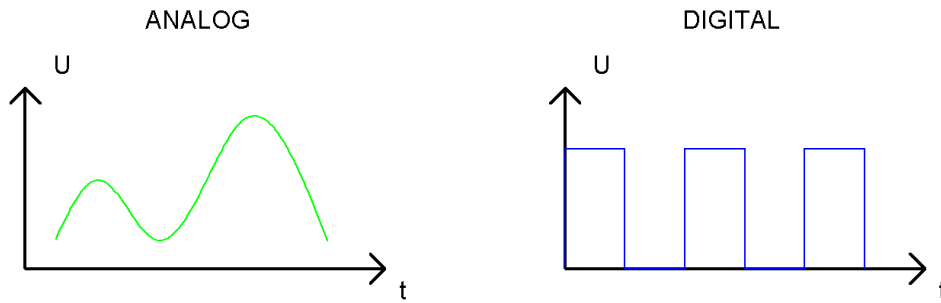
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1 	Resistor 220Ω x1 
Jumper Wire 		

Component Knowledge

Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or **discrete-time signal is a time series consisting of a sequence of quantities**. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0°C to 10°C. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



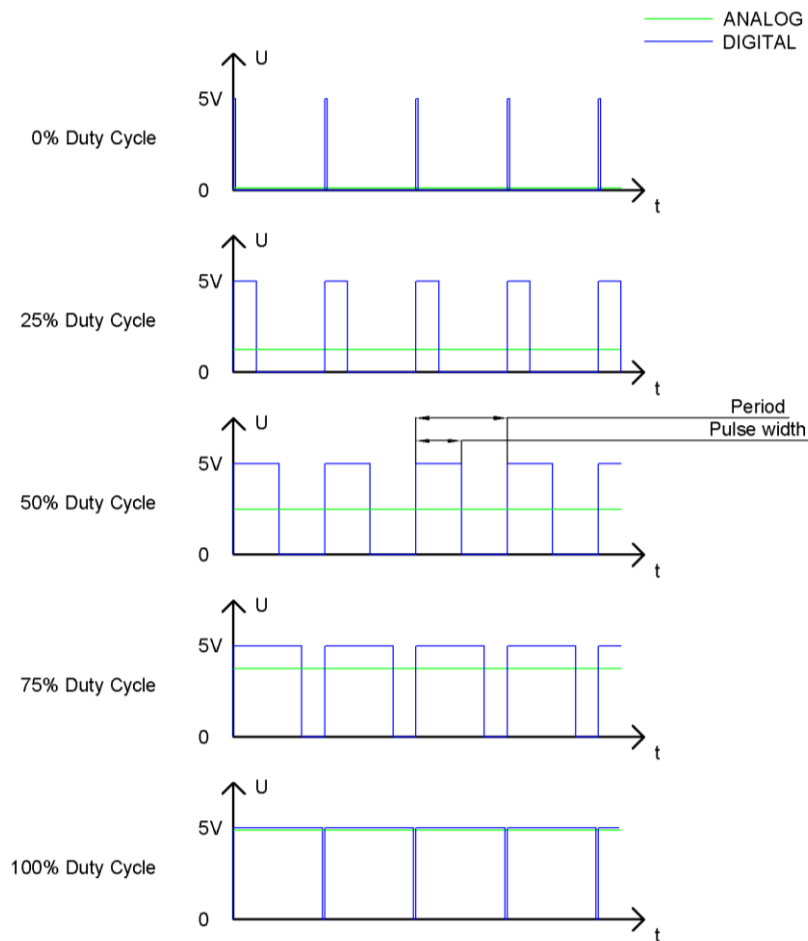
Note that the Analog signals are curved waves and the Digital signals are “Square Waves”.

In practical applications, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Digital processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals).

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called “pulse width”, and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform. The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

It is evident, from the above, that PWM is not actually analog but the effective value of voltage is equivalent to the corresponding analog value. Therefore, by using PWM, we can control the output power of to an LED and control other devices and modules to achieve multiple effects and actions.

In RPi, GPIO18 pin has the ability to output to hardware via PWM with a 10-bit accuracy. This means that 100% of the pulse width can be divided into $2^{10}=1024$ equal parts.

The wiringPi library of C provides both a hardware PWM and a software PWM method, while the wiringPi library of Python does not provide a hardware PWM method. There is only a software PWM option for Python.

The hardware PWM only needs to be configured, does not require CPU resources and is more precise in time control. The software PWM requires the CPU to work continuously by using code to output high level and low level. This part of the code is carried out by multi-threading, and the accuracy is relatively not high enough.

In order to keep the results running consistently, we will use PWM.

Circuit

Schematic diagram

Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

Code

This project uses the PWM output from the GPIO18 pin to make the pulse width gradually increase from 0% to 100% and then gradually decrease from 100% to 0% to make the LED glow brighter then dimmer.

C Code 4.1.1 BreathingLED

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 04.1.1_BreathingLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/04.1.1_BreathingLED
```

2. Use following command to compile "BreathingLED.c" and generate executable file "BreathingLED".

```
gcc BreathingLED.c -o BreathingLED -lwiringPi
```

3. Then run the generated file "BreathingLED"

```
sudo ./BreathingLED
```

After the program is executed, you'll see that LED is turned from on to off and then from off to on gradually like breathing.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <softPwm.h>
4  #define ledPin 1
5  void main(void)
6  {
7      int i;
```

```

8
9     printf("Program is starting ... \n");
10
11     wiringPiSetup(); //Initialize wiringPi.
12
13     softPwmCreate(ledPin, 0, 100); //Creat SoftPWM pin
14
15     while(1) {
16         for(i=0; i<100; i++) { //make the led brighter
17             softPwmWrite(ledPin, i);
18             delay(20);
19         }
20         delay(300);
21         for(i=100; i>=0; i--) { //make the led darker
22             softPwmWrite(ledPin, i);
23             delay(20);
24         }
25         delay(300);
26     }
27 }

```

First, create a software PWM pin.

```
softPwmCreate(ledPin, 0, 100); //Creat SoftPWM pin
```

There are two “for” loops in the next endless “while” loop. The first loop outputs a power signal to the ledPin PWM from 0% to 100% and the second loop outputs a power signal to the ledPin PWM from 100% to 0%.

```

while(1) {
    for(i=0; i<100; i++) {
        softPwmWrite(ledPin, i);
        delay(20);
    }
    delay(300);
    for(i=100; i>=0; i--) {
        softPwmWrite(ledPin, i);
        delay(20);
    }
    delay(300);
}

```

You can also adjust the rate of the state change of LED by changing the parameter of the delay() function in the “for” loop.

```
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
```

This creates a software controlled PWM pin.

```
void softPwmWrite (int pin, int value) ;
```

This updates the PWM value on the given pin.

For more details, please refer <http://wiringpi.com/reference/software-pwm-library/>

Python Code 4.1.1 BreathingLED

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 04.1.1_BreathingLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/04.1.1_BreathingLED
```

2. Use the Python command to execute Python code "BreathingLED.py".

```
python BreathingLED.py
```

After the program is executed, you will see that the LED gradually turns ON and then gradually turns OFF similar to "breathing".

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  LedPin = 12      # define the LedPin
5
6  def setup():
7      global p
8      GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
9      GPIO.setup(LedPin, GPIO.OUT)   # set LedPin to OUTPUT mode
10     GPIO.output(LedPin, GPIO.LOW)  # make ledPin output LOW level to turn off LED
11
12     p = GPIO.PWM(LedPin, 500)       # set PWM Frequency to 500Hz
13     p.start(0)                      # set initial Duty Cycle to 0
14
15  def loop():
16     while True:
17         for dc in range(0, 101, 1): # make the led brighter
18             p.ChangeDutyCycle(dc)   # set dc value as the duty cycle
19             time.sleep(0.01)
20         time.sleep(1)
21         for dc in range(100, -1, -1): # make the led darker
22             p.ChangeDutyCycle(dc)   # set dc value as the duty cycle
23             time.sleep(0.01)
24         time.sleep(1)
25
26  def destroy():
27     p.stop() # stop PWM
28     GPIO.cleanup() # Release all GPIO
29
30  if __name__ == '__main__':        # Program entrance
31     print ('Program is starting ... ')
32     setup()
33     try:
34         loop()
```

```

35     except KeyboardInterrupt: # Press ctrl-c to end the program.
36         destroy()

```

The LED is connected to the IO port called GPIO18. The LedPin is defined as pin 12 and set to output mode according to the corresponding chart for pin designations. Then create a PWM instance and set the PWM frequency to 1000HZ and the initial duty cycle to 0%.

```

LedPin = 12          # define the LedPin

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)      # use PHYSICAL GPIO Numbering
    GPIO.setup(LedPin, GPIO.OUT)   # set LedPin to OUTPUT mode
    GPIO.output(LedPin, GPIO.LOW)  # make ledPin output LOW level to turn off LED

    p = GPIO.PWM(LedPin, 500)      # set PWM Frequency to 500Hz
    p.start(0)                     # set initial Duty Cycle to 0

```

There are two “for” loops used to control the breathing LED in the next endless “while” loop. The first loop outputs a power signal to the ledPin PWM from 0% to 100% and the second loop outputs a power signal to the ledPin PWM from 100% to 0%.

```

def loop():
    while True:
        for dc in range(0, 101, 1): # make the led brighter
            p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
            time.sleep(0.01)
        time.sleep(1)
        for dc in range(100, -1, -1): # make the led darker
            p.ChangeDutyCycle(dc)    # set dc value as the duty cycle
            time.sleep(0.01)
        time.sleep(1)

```

The related functions of PWM are described as follows:

p = GPIO.PWM(channel, frequency)

To create a PWM instance:

p.start(dc)

To start PWM, where dc is the duty cycle (0.0 <= dc <= 100.0)

p.ChangeFrequency(freq)

To change the frequency, where freq is the new frequency in Hz

p.ChangeDutyCycle(dc)

To change the duty cycle where 0.0 <= dc <= 100.0

p.stop()

To stop PWM.

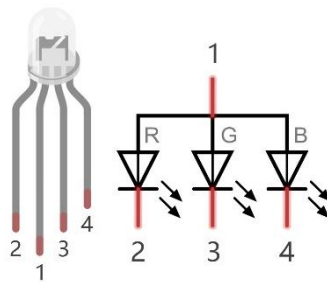
For more details regarding methods for using PWM with RPi.GPIO, please refer to:

<https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/>

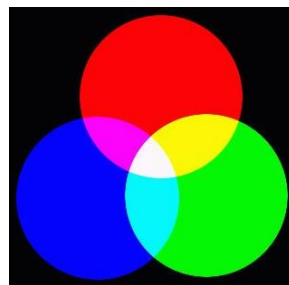
Chapter 5 RGB LED

In this chapter, we will learn how to control a RGB LED.

An RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Anode (+) or positive lead, the other 3 are the Cathodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Cathodes (2, 3 & 4) of the RGB LED



Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon.



RGB

If we use a three 8 bit PWM to control the RGB LED, in theory, we can create $2^8 * 2^8 * 2^8 = 16777216$ (16 million) colors through different combinations of RGB light brightness.

Next, we will use RGB LED to make a multicolored LED.

Project 5.1 Multicolored LED

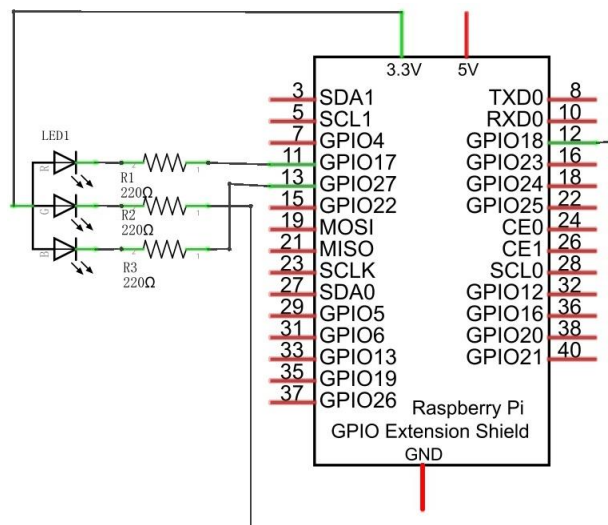
In this project, we will make a multicolored LED, which we can program the RGB LED to automatically change colors.

Component List

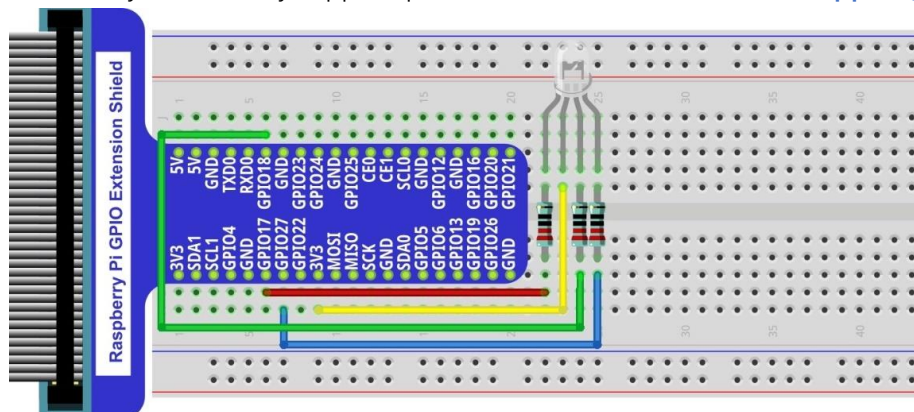
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Wire x1 Breadboard x1	RGB LED x1 	Resistor 220Ω x3 
Jumper Wire 		

Circuit

Schematic diagram



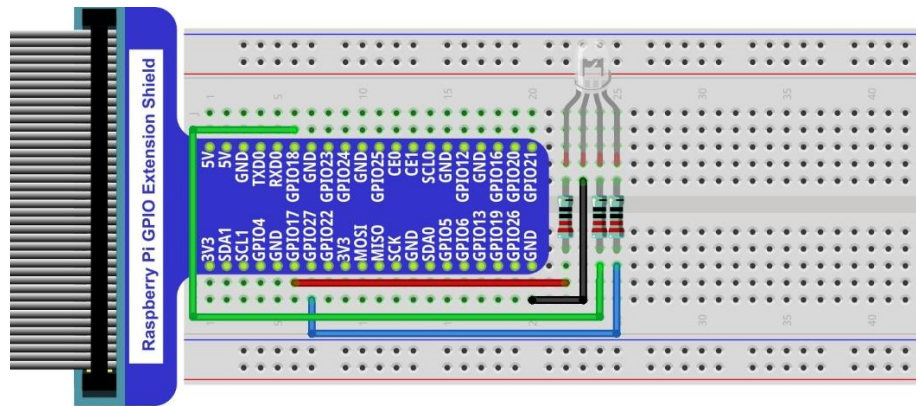
Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



In this kit, the RGB led is **Common anode**. The **voltage difference** between LED will make it work. There is no visible GND. The GPIO ports can also receive current while in output mode.

If circuit above doesn't work, the RGB LED may be common cathode. Please try following wiring.

There is no need to modify code for random color.



Code

We need to use the software to make the ordinary GPIO output PWM, since this project requires 3 PWM and in RPi only one GPIO has the hardware capability to output PWM,

C Code 5.1.1 Colorful LED

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 05.1.1_ColorfulLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/05.1.1_ColorfulLED
```

2. Use following command to compile "ColorfulLED.c" and generate executable file "ColorfulLED".

Note: in this project, the software PWM uses a multi-threading mechanism. So "-lpthread" option need to be add to the compiler.

```
gcc ColorfulLED.c -o ColorfulLED -lwiringPi -lpthread
```

3. And then run the generated file "ColorfulLED".

```
sudo ./ColorfulLED
```

After the program is executed, you will see that the RGB LED shows lights of different colors randomly.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <softPwm.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define ledPinRed    0
7  #define ledPinGreen  1
8  #define ledPinBlue   2
9
10 void setupLedPin(void)
```

```

11 {
12     softPwmCreate(ledPinRed, 0, 100); //Creat SoftPWM pin for red
13     softPwmCreate(ledPinGreen, 0, 100); //Creat SoftPWM pin for green
14     softPwmCreate(ledPinBlue, 0, 100); //Creat SoftPWM pin for blue
15 }
16
17 void setLedColor(int r, int g, int b)
18 {
19     softPwmWrite(ledPinRed, r); //Set the duty cycle
20     softPwmWrite(ledPinGreen, g); //Set the duty cycle
21     softPwmWrite(ledPinBlue, b); //Set the duty cycle
22 }
23
24 int main(void)
25 {
26     int r, g, b;
27
28     printf("Program is starting ... \n");
29
30     wiringPiSetup(); //Initialize wiringPi.
31
32     setupLedPin();
33     while(1) {
34         r=random()%100; //get a random in (0,100)
35         g=random()%100; //get a random in (0,100)
36         b=random()%100; //get a random in (0,100)
37         setLedColor(r, g, b); //set random as the duty cycle value
38         printf("r=%d, g=%d, b=%d \n", r, g, b);
39         delay(1000);
40     }
41     return 0;
42 }

```

First, in subfunction of ledInit(), create the software PWM control pins used to control the R, G, B pin respectively.

```

void setupLedPin(void)
{
    softPwmCreate(ledPinRed, 0, 100); //Creat SoftPWM pin for red
    softPwmCreate(ledPinGreen, 0, 100); //Creat SoftPWM pin for green
    softPwmCreate(ledPinBlue, 0, 100); //Creat SoftPWM pin for blue
}

```

Then create subfunction, and set the PWM of three pins.

```

void setLedColor(int r, int g, int b)
{

```

```
    softPwmWrite(ledPinRed, r); //Set the duty cycle
    softPwmWrite(ledPinGreen, g); //Set the duty cycle
    softPwmWrite(ledPinBlue, b); //Set the duty cycle
}
```

Finally, in the “while” loop of main function, get three random numbers and specify them as the PWM duty cycle, which will be assigned to the corresponding pins. So RGB LED can switch the color randomly all the time.

```
while(1) {
    r=random()%100; //get a random in (0,100)
    g=random()%100; //get a random in (0,100)
    b=random()%100; //get a random in (0,100)
    setLedColor(r,g,b); //set random as the duty cycle value
    printf("r=%d, g=%d, b=%d \n", r,g,b);
    delay(1000);
}
```

The related function of PWM Software can be described as follows:

long random();

This function will return a random number.

For more details about Software PWM, please refer to: <http://wiringpi.com/reference/software-pwm-library/>

Python Code 5.1.1 ColorfulLED

First, observe the project result, and then learn about the code in detail.

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

1. Use cd command to enter 05.1.1_ColorfulLED directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/05.1.1_ColorfulLED
```

2. Use python command to execute python code "ColorfulLED.py".

```
python ColorfulLED.py
```

After the program is executed, you will see that the RGB LED randomly lights up different colors.

The following is the program code:

```

1  import RPi.GPIO as GPIO
2  import time
3  import random
4
5  pins = [11, 12, 13]          # define the pins for R:11,G:12,B:13
6
7  def setup():
8      global pwmRed, pwmGreen, pwmBlue
9      GPIO.setmode(GPIO.BOARD)    # use PHYSICAL GPIO Numbering
10     GPIO.setup(pins, GPIO.OUT)    # set RGBLED pins to OUTPUT mode
11     GPIO.output(pins, GPIO.HIGH)  # make RGBLED pins output HIGH level
12     pwmRed = GPIO.PWM(pins[0], 2000) # set PWM Frequency to 2kHz
13     pwmGreen = GPIO.PWM(pins[1], 2000) # set PWM Frequency to 2kHz
14     pwmBlue = GPIO.PWM(pins[2], 2000) # set PWM Frequency to 2kHz
15     pwmRed.start(0)              # set initial Duty Cycle to 0
16     pwmGreen.start(0)
17     pwmBlue.start(0)
18
19     def setColor(r_val, g_val, b_val):    # change duty cycle for three pins to r_val, g_val, b_val
20         pwmRed.ChangeDutyCycle(r_val)    # change pwmRed duty cycle to r_val
21         pwmGreen.ChangeDutyCycle(g_val)
22         pwmBlue.ChangeDutyCycle(b_val)
23
24     def loop():
25         while True :
26             r=random.randint(0,100) #get a random in (0,100)
27             g=random.randint(0,100)
28             b=random.randint(0,100)
29             setColor(r, g, b)          #set random as a duty cycle value
30             print ( ' r=%d, g=%d, b=%d ' %(r , g, b))
31             time.sleep(1)
32
33     def destroy():
34         pwmRed.stop()
35         pwmGreen.stop()

```

```
36     pwmBlue.stop()
37     GPIO.cleanup()
38
39     if __name__ == '__main__':      # Program entrance
40         print ('Program is starting ... ')
41         setup()
42         try:
43             loop()
44         except KeyboardInterrupt:    # Press ctrl-c to end the program.
45             destroy()
```

In last chapter, we learned how to use Python language to make a pin output PWM. In this project, we output to three pins via PWM and the method is exactly the same as we used in the last chapter. In the “while” loop of “loop” function, we first generate three random numbers, and then specify these three random numbers as the PWM values for the three pins, which will make the RGB LED produce multiple colors randomly.

```
def loop():
    while True :
        r=random.randint(0,100)  #get a random in (0,100)
        g=random.randint(0,100)
        b=random.randint(0,100)
        setColor(r, g, b)         #set random as a duty cycle value
        print (' r=%d, g=%d, b=%d ' %(r ,g, b))
        time.sleep(1)
```

About the randint() function :

random.randint(a, b)

This function can return a random integer (a whole number value) within the specified range (a, b).







Chapter 6 Buzzer

In this chapter, we will learn about buzzers and the sounds they make. And in our next project, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

Project 6.1 Doorbell

We will make a doorbell with this functionality: when the Push Button Switch is pressed the buzzer sounds and when the button is released, the buzzer stops. This is a momentary switch function.

Component List

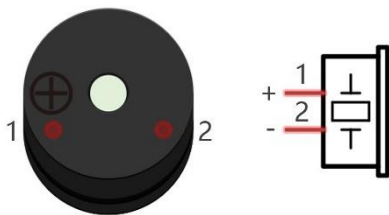
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire 		
NPN transistorx1 (S8050) 	Active buzzer x1 	Push Button Switch x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component knowledge

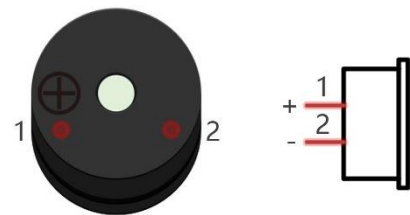
Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.

Active buzzer



Passive buzzer



Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

How to identify active and passive buzzer?

1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes, view of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model).



Active buzzer bottom



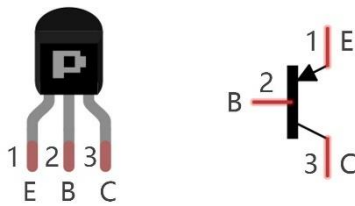
Passive buzzer bottom

Transistors

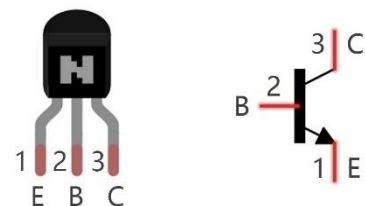
A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to amplify the current.

Transistors, full name: semiconductor transistor, is a semiconductor device that controls current (think of a transistor as an electronic “amplifying or switching device”. Transistors can be used to amplify weak signals, or to work as a switch. Transistors have three electrodes (PINs): base (b), collector (c) and emitter (e). When there is current passing between “be” then “ce” will have a several-fold current increase (transistor magnification), in this configuration the transistor acts as an amplifier. When current produced by “be” exceeds a certain value, “ce” will limit the current output. at this point the transistor is working in its saturation region and acts like a switch. Transistors are available as two types as shown below: PNP and NPN,

PNP transistor



NPN transistor



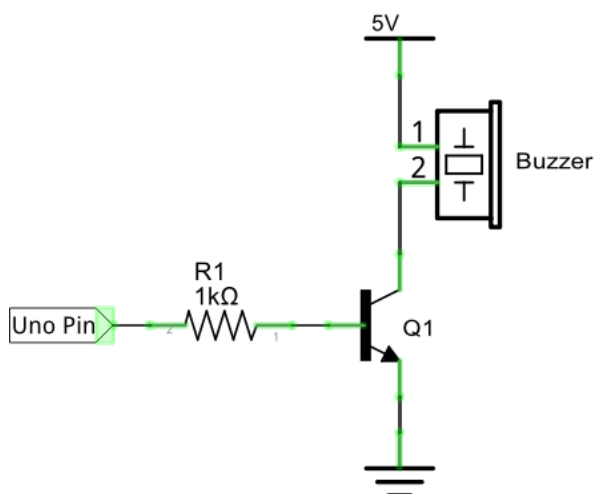
In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Thanks to the transistor's characteristics, they are often used as switches in digital circuits. As micro-controllers output current capacity is very weak, we will use a transistor to amplify its current in order to drive components requiring higher current.

When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not conduct current and buzzer will remain silent (no sounds).

When we use a PNP transistor to drive a buzzer, we often use the following method. If GPIO outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If GPIO outputs high level, no current flows through R1, the transistor will not conduct current and buzzer will remain silent (no sounds). Below are the circuit schematics for both a NPN and PNP transistor to power a buzzer.

NPN transistor to drive buzzer



PNP transistor to drive buzzer

