

# Preface

## About SunFounder

SunFounder is a company focused on STEAM education with products like open source robots, development boards, STEAM kit, modules, tools and other smart devices distributed globally. In SunFounder, we strive to help elementary and middle school students as well as hobbyists, through STEAM education, strengthen their hands-on practices and problem-solving abilities. In this way, we hope to disseminate knowledge and provide skill training in a full-of-joy way, thus fostering your interest in programming and making, and exposing you to a fascinating world of science and engineering. To embrace the future of artificial intelligence, it is urgent and meaningful to learn abundant STEAM knowledge.

## About This kit

This Electronic kit applies to the Raspberry Pi B, model B+, Pi 2 model B , Pi 3 model B and Pi 3 model B+. It includes various components and chips that can help to create various interesting phenomena which you can get via some operation with the guidance of experiment instructions. In this process, you can learn some basic knowledge about programming. Also you can explore more application by yourself. Now go for it! This book displays how to operate the system in an illustrated form.

## Free Support



If you have any **TECHNICAL question**, add a topic under **FORUM** section on our website and we'll reply as soon as possible.



For **NON-TECH questions** like order and shipment issues, please **send an email to** [service@sunfounder.com](mailto:service@sunfounder.com). You're also welcomed to share your projects on FORUM.

# Warning

When you purchase or use this product, please note the following points:

- This product contains many small parts. Swallowing or improper operation can cause serious infections and death. Seek immediate medical attention when the accident happens.
- Strictly prohibit use this product and its parts near any AC electrical outlet or other circuits in case of the potential risk of electric shock.
- Strictly prohibit use this product near any liquid or fire.
- Keep conductive materials away from this product.
- Do not allow children under 3 years old to use this product without adult supervision. Please place this product in the position where children under 3 years old cannot reach.
- Do not store or use this product in any extreme environments such as extreme hot or cold, high humidity and etc.
- Remember to break the circuit when it is not needed.
- Some parts of this product may become warm to touch when used in certain circuit designs, which is normal.
- Improper operation may cause overheating.
- Using components not in accordance with the specification may cause damage to the product.

# Contents

Introduction.....	1
Component List.....	2
Required Components.....	10
Preparation.....	13
If You Have A Screen.....	13
Required Components.....	13
Procedures.....	14
If You Have No Screen.....	24
Required Components.....	24
Burn System.....	24
Connect the Raspberry Pi to the Internet.....	27
Start SSH.....	29
Get the IP Address.....	29
Use the SSH Remote Control.....	30
Remote Desktop.....	35
Libraries.....	45
RPi.GPIO.....	45
WiringPi.....	47
Download the Code.....	49
Lessons.....	50

Lesson 1 Blinking LED.....	50
Lesson 2 Flowing LED Lights.....	77
Lesson 3 Breathing LED.....	87
Lesson 4 RGB LED.....	98
Lesson 5 Controlling LED by Button.....	112
Lesson 6 Tilt Switch.....	124
Lesson 7 Slide Switch.....	136
Lesson 8 Relay.....	147
Lesson 9 4N35.....	161
Lesson 10 Active Buzzer.....	170
Lesson 11 Doorbell.....	179
Lesson 12 Passive Buzzer.....	189
Lesson 13 Button Piano.....	201
Lesson 14 Quiz Buzzer System.....	213
Lesson 15 NE555 Timer.....	229
Lesson 16 Servo.....	242
Lesson 17 LCD1602.....	255
Lesson 18 Driving LEDs by 74HC595.....	269
Lesson 19 7-segment.....	288
Lesson 20 Traffic Light.....	302

# Introduction

Electronic kit is a basic kit for beginners, having high rate of quantity and price. It contains 12 commonly used input and output components, three ICs, and a number of basic electronic components (such as resistor, capacitor). It is an economical choice for fast mastering programming.

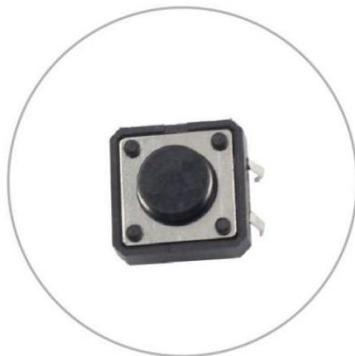
With this kit, you can quickly master the necessary knowledge to achieve complex projects with Raspberry Pi. In this matching document, you are provided with 20 courses for reference and learning. These courses are designed for beginners, even if you are firstly exposed to Raspberry Pi, or even firstly contact with programming, you can get started smoothly. Meanwhile, the practice difficulty of the course is gradually increasing. We hope that you can share your achievements or creations on our forum after you learn all courses.

# Component List



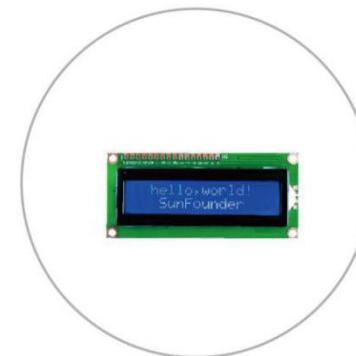
**7-Segment**

1 PCS



**Button**

10 PCS



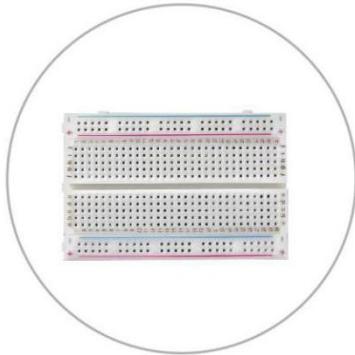
**LCD1602**

1 PCS



**Servo**

1 PCS



**Breadboard**

1 PCS

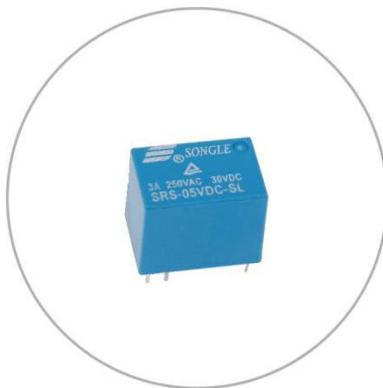


**1M USB cable**

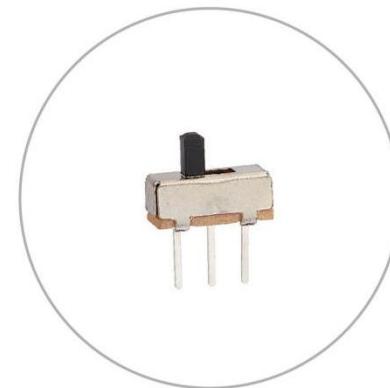
1 PCS



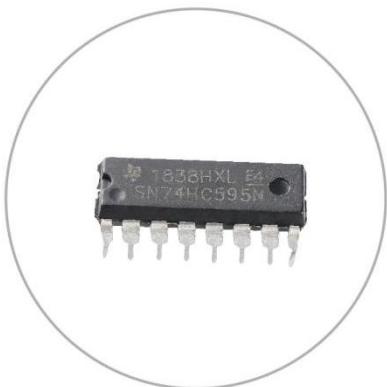
**Passive buzzer**  
1 PCS



**Relay**  
1 PCS



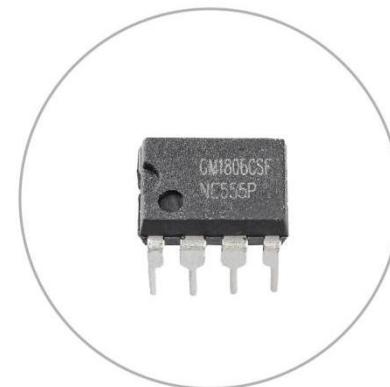
**Slide Switch**  
5 PCS



**74HC595**  
1 PCS



**4N35**  
1 PCS



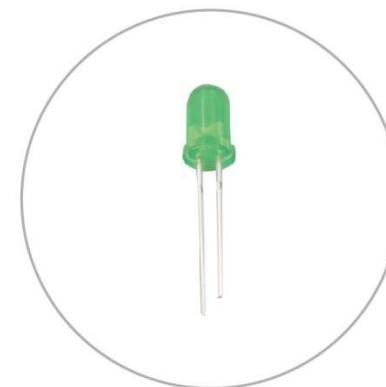
**555 Timer**  
1 PCS



**RGB LED**  
1 PCS



**LED (Red)**  
10 PCS



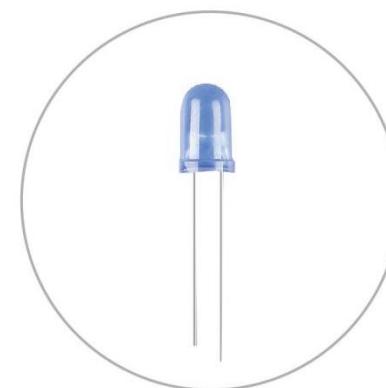
**LED (Green)**  
10 PCS



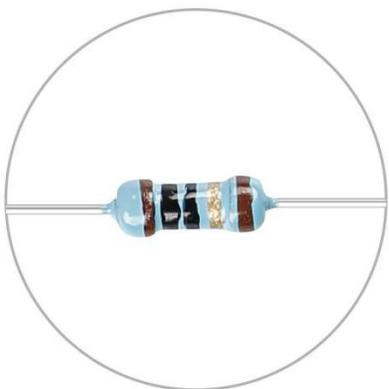
**LED (Yellow)**  
10 PCS



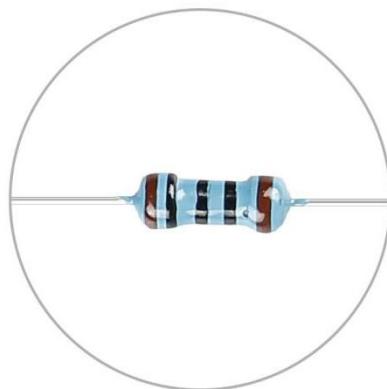
**LED (White)**  
10 PCS



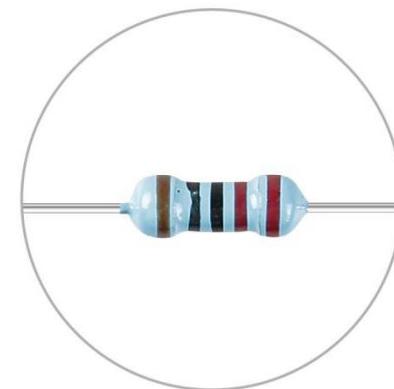
**LED(Blue)**  
10 PCS



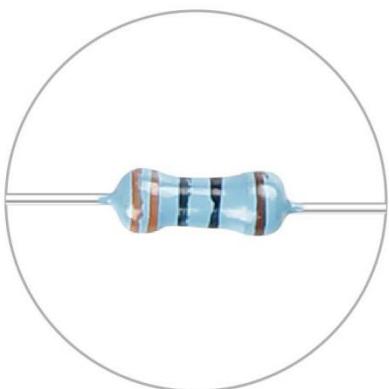
**Resistor(10Ω)**  
10 PCS



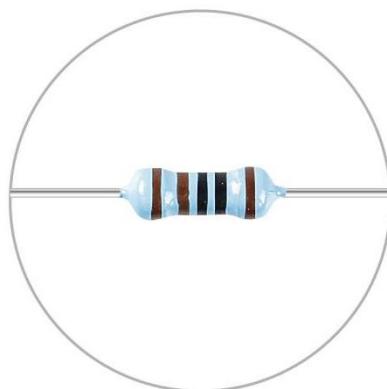
**Resistor(100Ω)**  
10 PCS



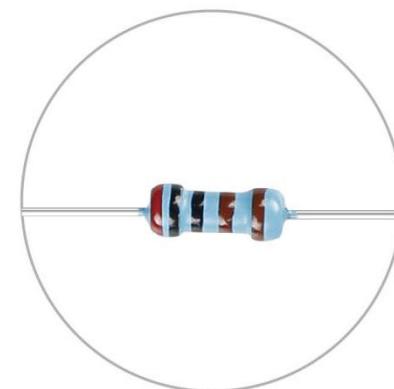
**Resistor(220Ω)**  
10 PCS



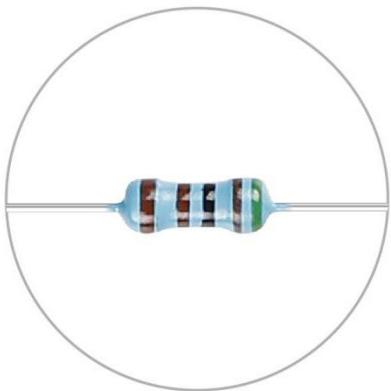
**Resistor(330Ω)**  
10 PCS



**Resistor(1KΩ)**  
10 PCS



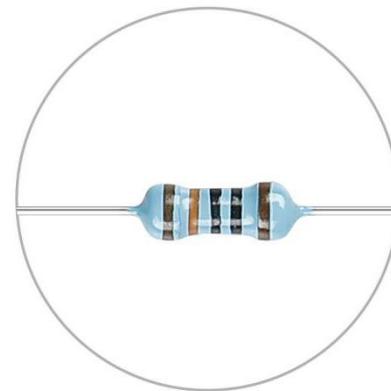
**Resistor(2KΩ)**  
10 PCS



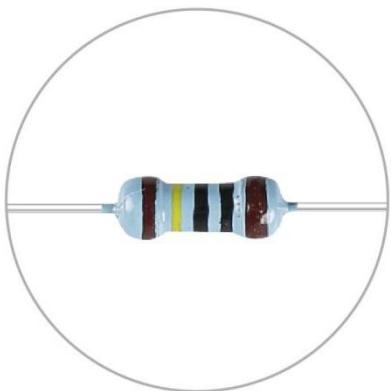
**Resistor(5.1KΩ)**  
10 PCS



**Resistor(10KΩ)**  
10 PCS



**Resistor(100KΩ)**  
10 PCS



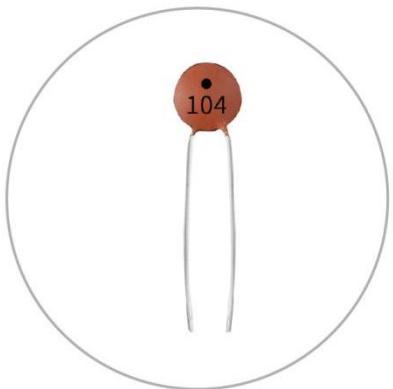
**Resistor(1MΩ)**  
10 PCS



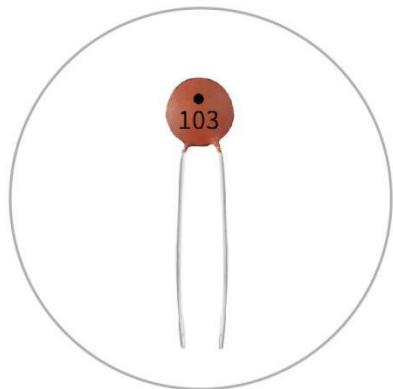
**1N4007**  
5 PCS



**Zener diode**  
1 PCS



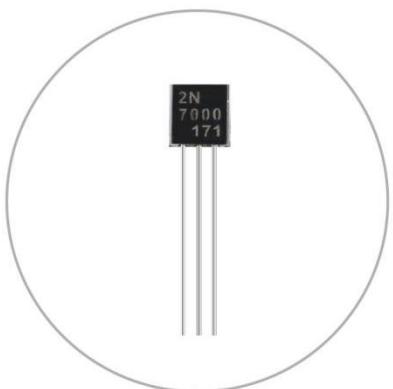
**104 Capacitor**  
10 PCS



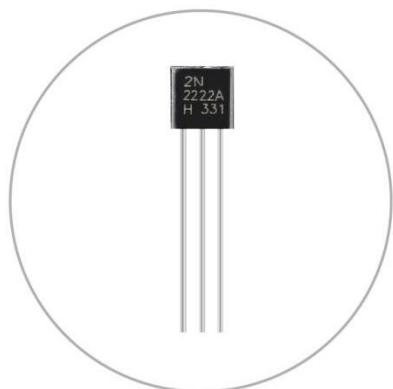
**103 Capacitor**  
10 PCS



**S8050 Transistor**  
2 PCS



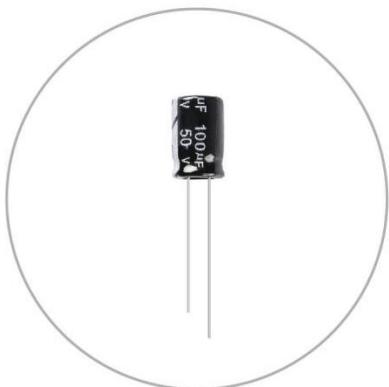
**2N7000 Transistor**  
1 PCS



**PN222A**  
5 PCS



**Capacitor (10UF)**  
5 PCS



**Capacitor (100UF)**  
5 PCS



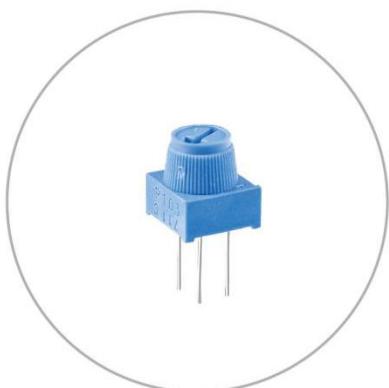
**Thermistor**  
1 PCS



**Tilt switch**  
1 PCS



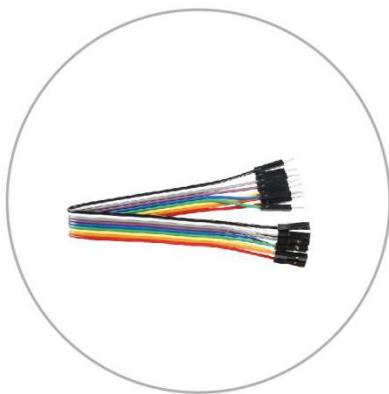
**Photoresistor**  
2 PCS



**Potentiometer**  
2 PCS



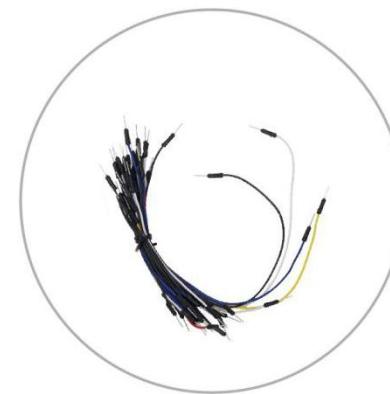
**Active buzzer**  
1 PCS



**DuPont wires**  
20 PCS



**Battery snap**  
1 PCS



**Jump wires**  
65 PCS

Note: After opening the package, please check whether the quantity of components is compliance with product description and whether all components are in good condition.

# What Do We Need?

## Required Components

### Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.

Our kit applies to the following versions of the products of Raspberry Pi :





## Power Adapter

To connect to a power socket, the Raspberry Pi has a micro USB port (the same found on many mobile phones). You will need a power supply which provides at least 2.5 amps.

## Micro SD Card

Your Raspberry Pi needs an SD card to store all its files and the Raspbian operating system. You will need a micro SD card with a capacity of at least 8 GB.

## Optional Components

### Screen

To view the desktop environment of Raspberry Pi, you need to use the screen that can be a TV screen or a computer monitor. If the screen has built-in speakers, the Pi plays sounds via them.

### Mouse & Keyboard

When you use a screen , a USB keyboard and a USB mouse are also needed.

## HDMI

The Raspberry Pi has a HDMI output port that is compatible with the HDMI ports of most modern TV and computer monitors. If your screen has only DVI or VGA ports, you will need to use the appropriate conversion line.

## Case

You can put the Raspberry Pi in a case; by this means, you can protect your device.

## Sound or Earphone

The Raspberry Pi is equipped with an audio port about 3.5 mm that can be used when your screen has no built-in speaker or when there is no screen operation.

# Preparation

Depending on the different devices you use, you can start up the Raspberry Pi in different methods. If you have a separate screen for Raspberry Pi, follow the instructions in this chapter. Otherwise, please refer to the corresponding steps in the following chapters.

## If You Have A Screen

If you have a screen, you can use the NOOBS (New Out Of Box System) to install the Raspbian system.

## Required Components

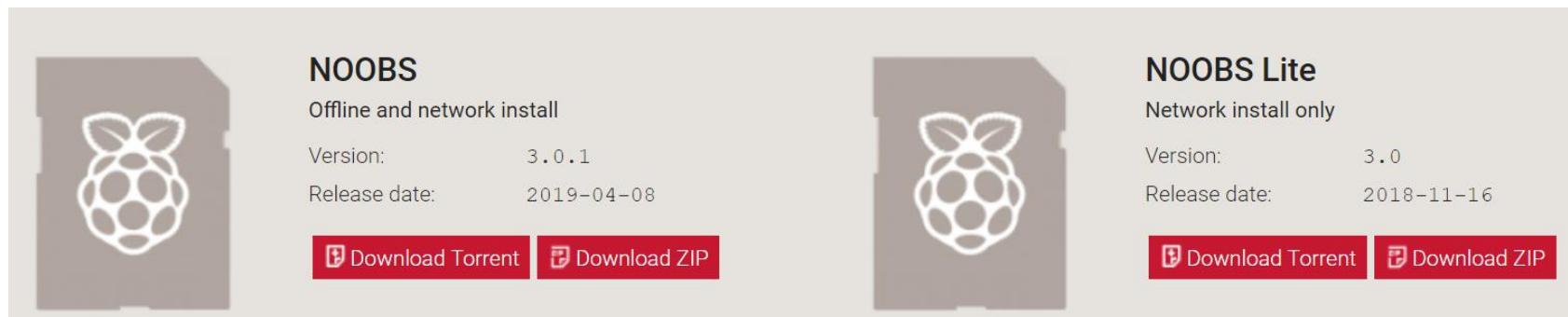
Any Raspberry Pi	1 * 2.5A Power Adapter
1 * Monitor	1 * Monitor Power Adapter
1 * HDMI cable	1 * Micro SD card
1 * Mouse	1 * Keyboard
1 * Personal Computer	

## Procedures

### Step 1

To download NOOBS from your PC, you can choose **NOOBS** or **NOOBS Lite** - the only difference is that there is a built-in offline Raspbian installer in **NOOBS**, while the **NOOBS Lite** can only be operated online. Here, you are suggested to use the former. Here is the download address of NOOBS:

<https://www.raspberrypi.org/downloads/noobs/>



### Step 2

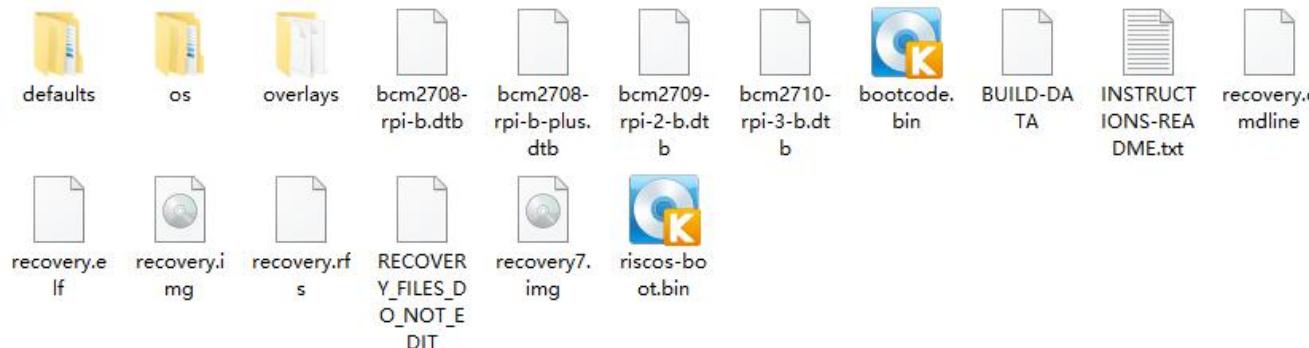
Format the SD card. If there are some important files in the SD card of Raspbian, please backup them first.

### Step 3

Next, you will need to extract the files from the NOOBS zip archive you downloaded from the Raspberry Pi website.

- Find the downloaded archive — by default, it should be in your Downloads folder.
- Double-click on it to extract the files, and keep the resulting Explorer/Finder window open.

Finally Select all the files in the NOOBS folder and copy them to the SD card.

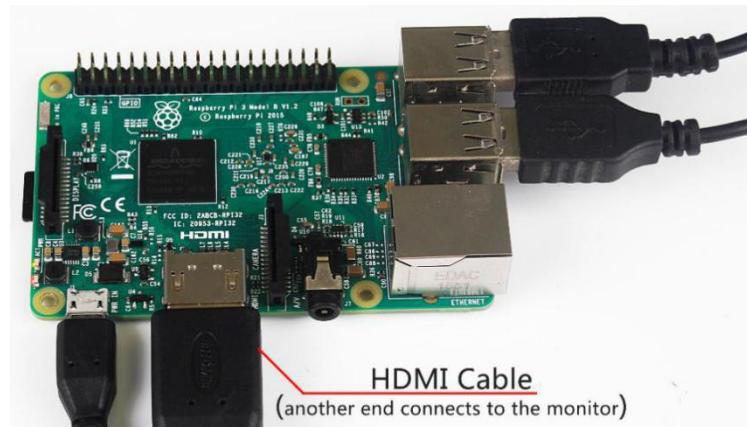


## Step 4

All the files transferred, the SD card pops up.

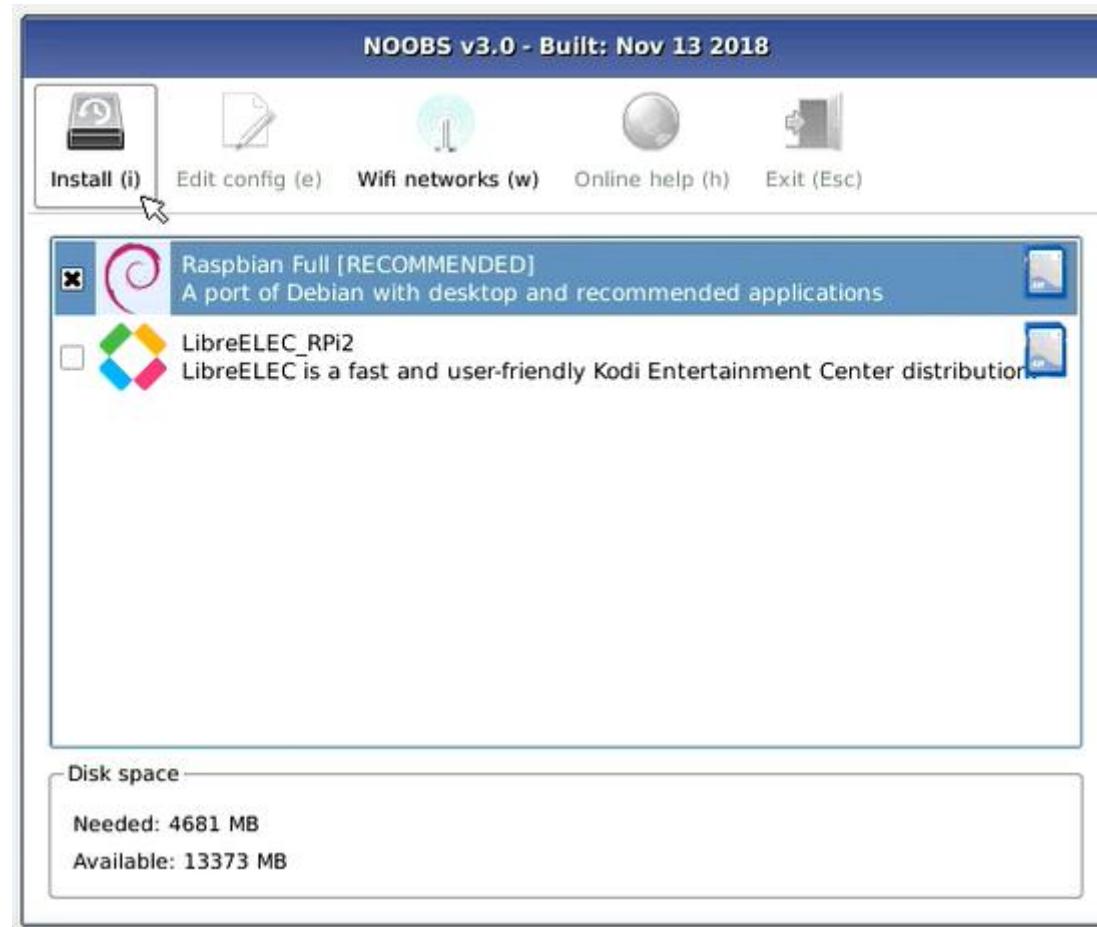
## Step 5

Insert the SD card into the Raspberry Pi. In addition, connect the screen, keyboard and mouse to it. Finally power up the Raspberry Pi with a 2.5A power adapter.



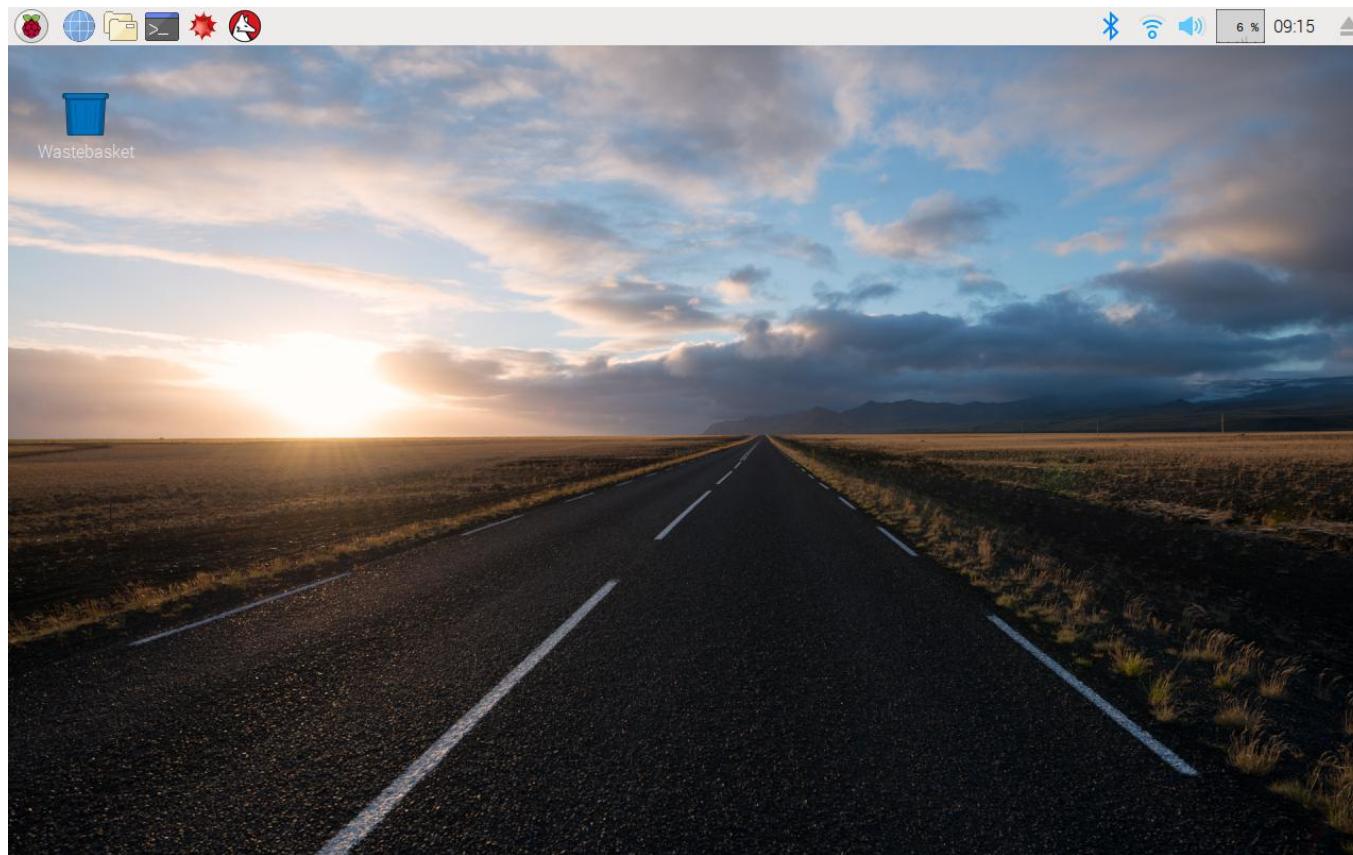
## Step 6

It will go to the NOOBS interface after starting up. If you use NOOBS LITE, you need to select Wi-Fi networks (w) first. Tick the checkbox of the Raspbian and click Install in the top left corner. The NOOBS will help to conduct the installation automatically. This process will take a few minutes.



## Step 7

When the installation is done, the system will restart automatically and the desktop of the system will appear.



## Step 8

If you run Raspberry Pi for the first time, the application of “Welcome to Raspberry Pi” pops up and guides you to perform the initial setup.



## Step 9

Set country/region, language and time zone, and then click “Next” again.



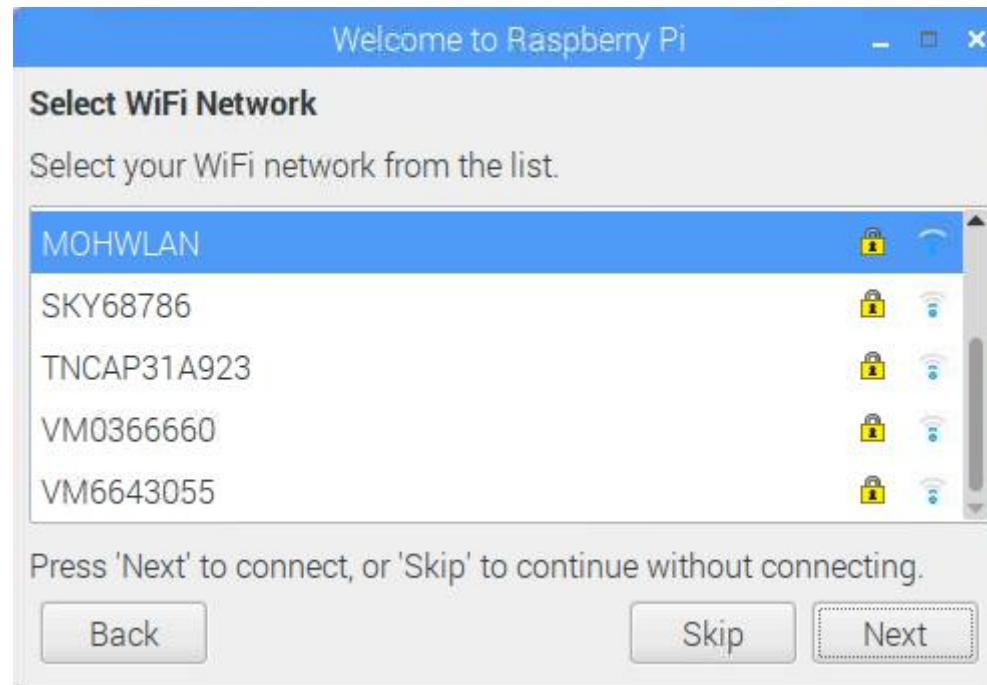
## Step 10

Input the new password of Raspberry Pi and click "Next".



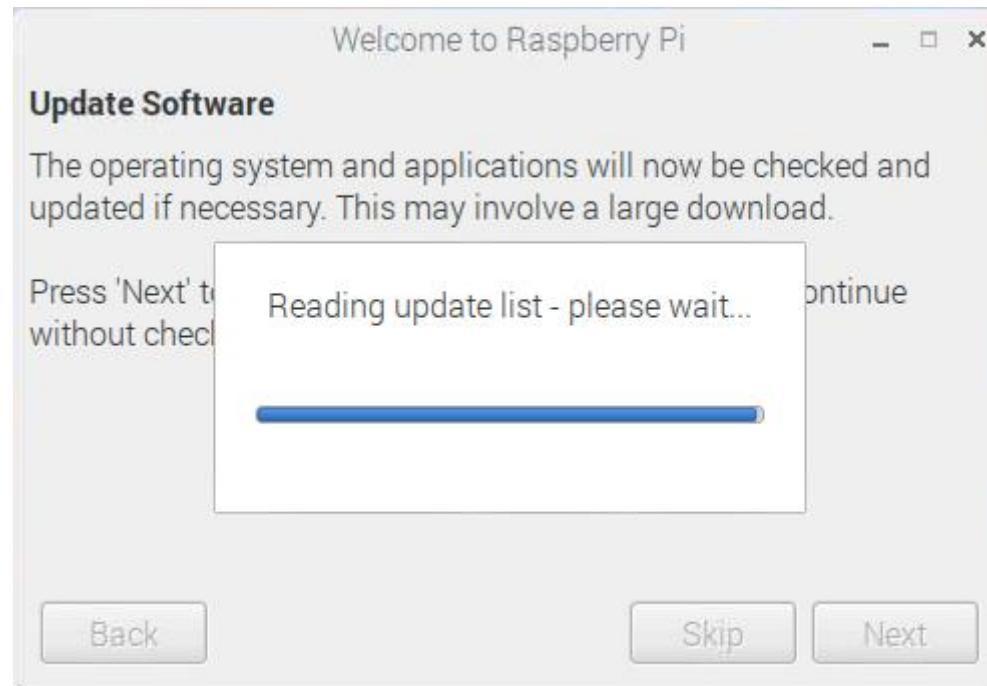
## Step 11

Connect the Raspberry Pi to Wi-Fi and click "Next".



## Step 12

Retrieve update.



## Step 13

Click "Done" to complete the Settings.



Now we can run the Raspberry Pi.

**Note:** You can check the complete tutorial of NOOBS on the official website of the Raspberry Pi: <https://www.raspberrypi.org/help/noobs-setup/>.

## If You Have No Screen

If we don't have a screen, we can directly write the raspbian system to the SD card and we can control the Raspberry Pi on PC remotely by directly modifying the configuration file of the network settings in the SD card.

### Required Components

Any Raspberry Pi	1 * 2.5A Power Adapter
1 * Micro SD card	1 * Personal computer

## Burn System

### Step 1

Prepare the tool of image burning. Here we use the **Etcher**. You can download the software from the link:  
<https://www.balena.io/etcher/>

### Step 2

Download the complete image on the official website by clicking this link:

<https://www.raspberrypi.org/downloads/raspbian/>. There are three different kinds of Raspbian Stretches available, among which the Raspbian Stretch with desktop will be the best choice if you have no other special requirements.



### Raspbian Stretch with desktop and recommended software

Image with desktop and recommended software based on Debian Stretch

Version: November 2018  
Release date: 2018-11-13  
Kernel version: 4.14  
Release notes: [Link](#)

[Download Torrent](#) | [Download ZIP](#)

SHA-256: 0ca644539fdaf4e19ec7cebf9e61c049b82ba45b1a21cdec91fa54bd59d660d2



### Raspbian Stretch with de...

Image with desktop based on Debian S...

Version: November 2018  
Release date: 2018-11-13  
Kernel version: 4.14  
Release notes: [Link](#)

[Download Torrent](#) | [Download](#)

SHA-256: a121652937ccde1c2583fe77d1caec407f2cd248327c49ac9bc97



### Raspbian Stretch Lite

Minimal image based on Debian Stretch

Version: November 2018  
Release date: 2018-11-13  
Kernel version: 4.14  
Release notes: [Link](#)

[Download Torrent](#) | [Download ZIP](#)

SHA-256: 47ef1b2501d0e5002675a50b6868074e693f78829822eef64f3878487953234d

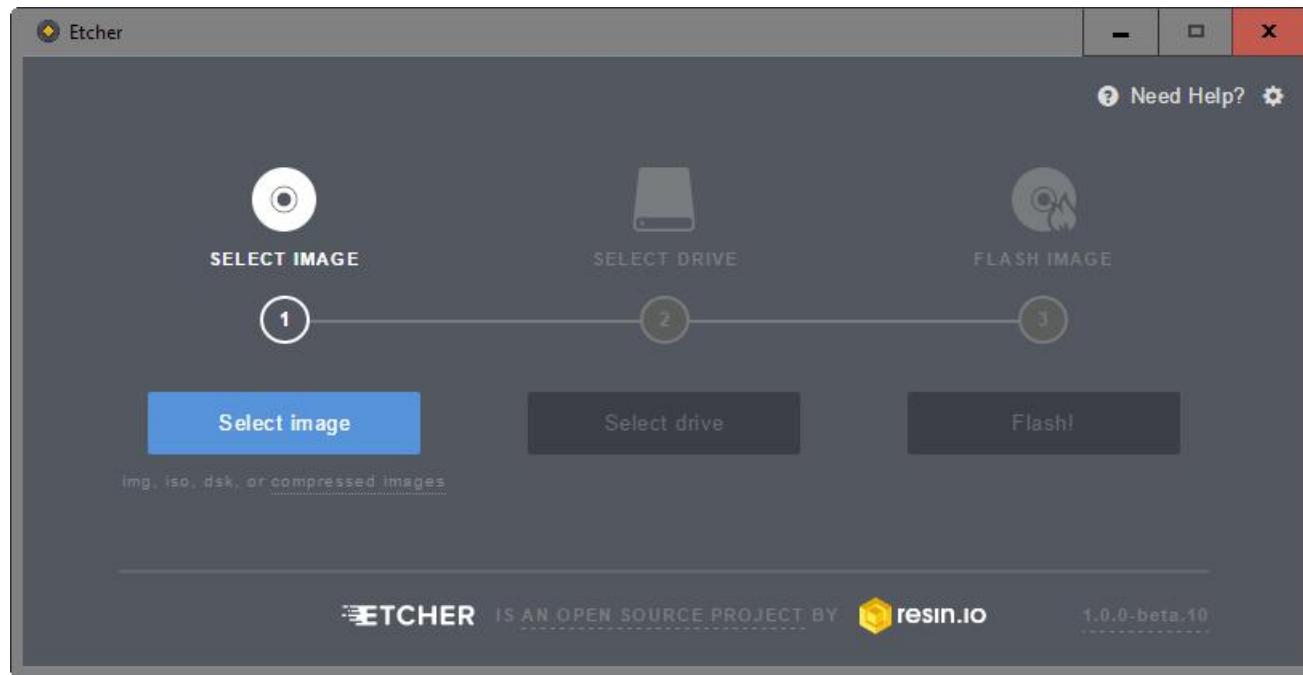
## Step 3

Unzip the package downloaded and you will see the *xxxx-xx-xx-raspbian-stretch.img* file inside.

**Note:** DO NOT extract the file.

## Step 4

With the application of Etcher, flash the image file, raspbian into the SD card.



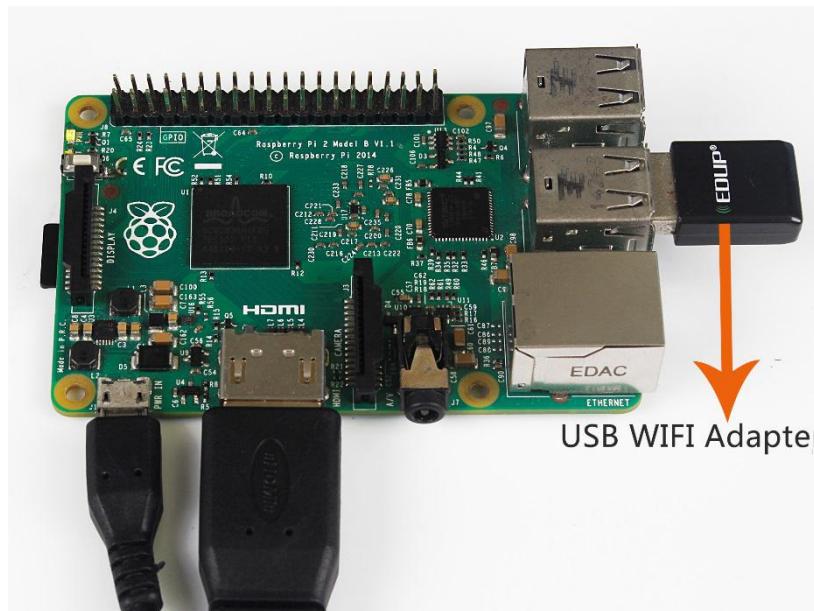
## Step 5

At this point, raspbian is installed; however, if you want to apply it ,what you need do next is to complete the settings accordingly.

## Connect the Raspberry Pi to the Internet

There are two methods to help get the Raspberry Pi connected to the network: the first one is using a network cable, the other way is using Wi-Fi. We will talk in detail about how to connect via Wi-Fi as below.

Since the 3B and above version of the product, Raspberry Pi has a built-in Wi-Fi function. If what you use is the early version of Raspberry Pi, a USB Wi-Fi Adapter is needed. Log in the website, [https://elinux.org/RPi\\_USB\\_Wi-Fi\\_Adapters](https://elinux.org/RPi_USB_Wi-Fi_Adapters) for more.



If you want to use the Wi-Fi function, you need to modify a Wi-Fi configuration file `wpa-supplicant.conf` in the SD card by your PC that is located in the directory `/etc/wpa-supplicant/`.

If your personal computer is working on a linux system, you can access the directory directly to modify the configuration file; however, if your PC use Windows system, then you can't access the directory and what you need next is to go to the directory, `/boot/` to create a new file with the same name, `wpa-supplicant.conf`.



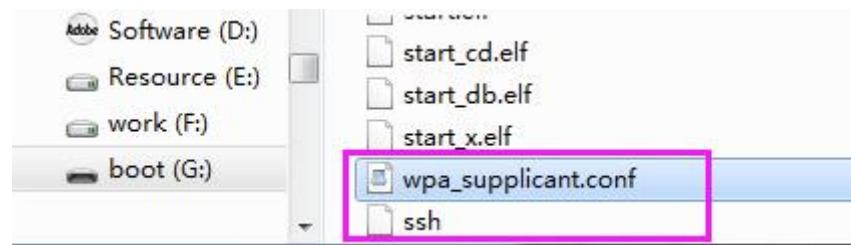
Input the following content in the file.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=GB
network={
    ssid="WiFi-A"
    psk="Sunfounder"
    key_mgmt=WPA-PSK
    priority=1
}
```

You need to replace “**WiFi-A**” with your custom name of Wi-Fi and “**Sunfounder**” with your password. By doing these, the Raspbian system will move this file to the target directory automatically to overwrite the original WIFI configuration file when it runs next time.

## Start SSH

To use the function of remote control of the Raspberry Pi, you need to start SSH firstly that is a more reliable protocol providing security for remote login sessions and other network services. Generally, SSH of Raspberry Pi is in a disabled state. Additionally, if you want to run it, you need to create a file named SSH under directory `/boot/`.



Now, the Raspbian system is configured. When the SD card is inserted into the Raspberry Pi, you can use it immediately.

## Get the IP Address

After the Raspberry Pi is connected to Wi-Fi, we need to get the IP address of it. There are many ways to know the IP address, and two of them are listed as follows.

### 1. Checking via the Router

If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the system, Raspbian is **raspberrypi**, and you need to find it. (If you are using ArchLinuxARM system, please find **alarmpi**.)

### 2. Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, [Advanced IP scanner](#) and so on.

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspbian system is **raspberrypi**, now you need to find the hostname.

## Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

### For Linux or/Mac OS X Users

#### Step 1

Go to **Applications->Utilities**, find the **Terminal**, and open it.

#### Step 2

Type in **ssh pi@ip\_address** . “pi”is your username and “ip\_address” is your IP address. For example:

```
ssh pi@192.168.18.197
```

#### Step 3

Input “yes”.



```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

## Step 4

Input the password and the default password is “**raspberry**”.

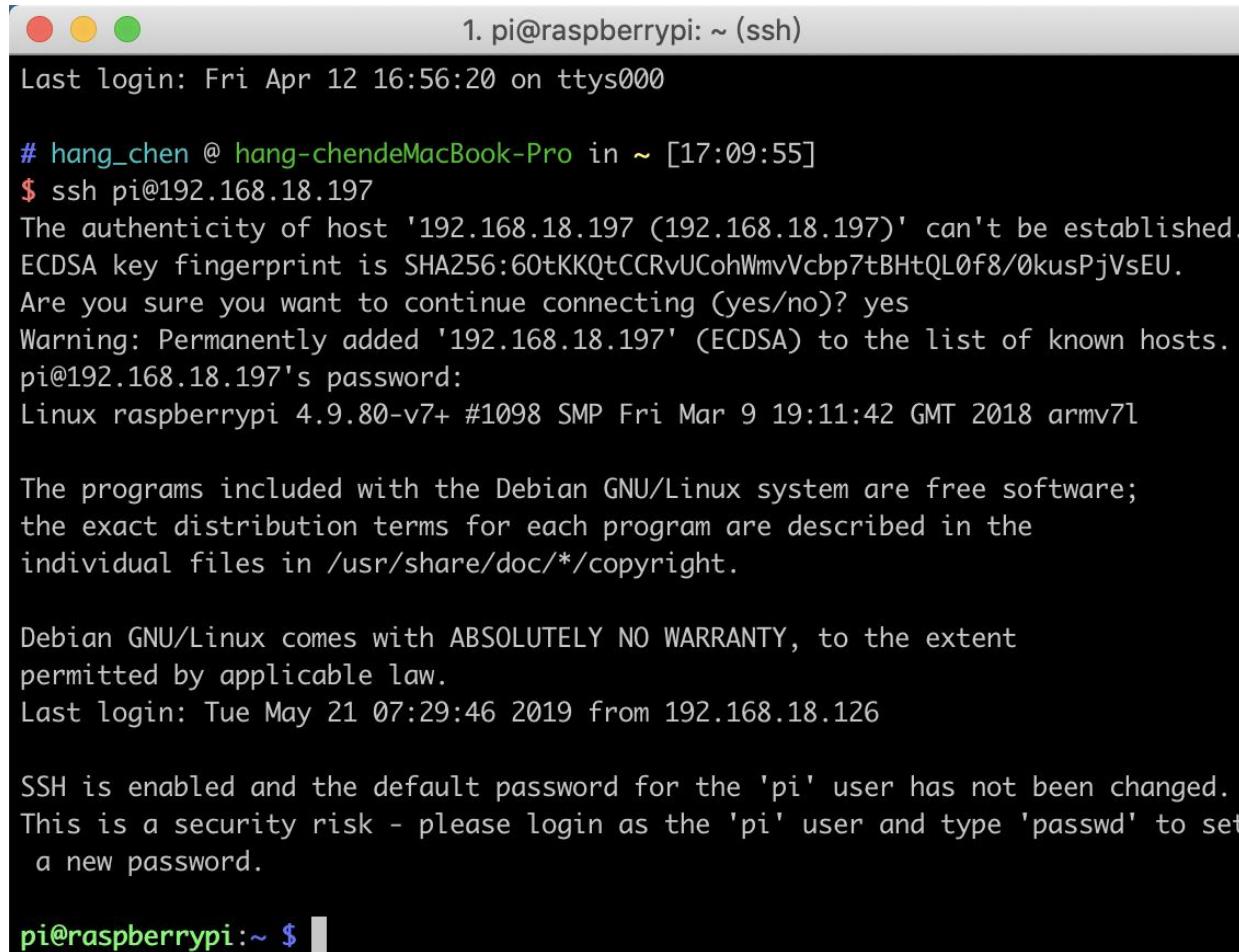


```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password: *
```

## Step 5

We now get the Raspberry Pi connected and are ready to go to the next step.



The screenshot shows a terminal window titled "1. pi@raspberrypi: ~ (ssh)". The session was last logged in on Friday, April 12, at 16:56:20 on ttys000. The user "hang\_chen" from a MacBook Pro connected via SSH at 17:09:55. The user "pi" is logged in from IP address 192.168.18.197. A warning message indicates that the host's authenticity cannot be established, and the ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU. The user responded "yes" to continue connecting. A warning message states that the host '192.168.18.197' (ECDSA) has been permanently added to the list of known hosts. The password for the user "pi" is requested. The system information shows it's running Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l. A copyright notice follows, stating that the programs are free software and the exact distribution terms are described in individual files in /usr/share/doc/\*/. A note about Debian's warranty follows, stating that it comes with ABSOLUTELY NO WARRANTY. The last login information is shown as "Last login: Tue May 21 07:29:46 2019 from 192.168.18.126". A final note about SSH security is present, stating that SSH is enabled and the default password for the 'pi' user has not been changed, which is a security risk. The user is advised to log in as the 'pi' user and type 'passwd' to set a new password. The prompt "pi@raspberrypi:~ \$" is visible at the bottom of the terminal window.

**Note:** When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

## For Windows Users

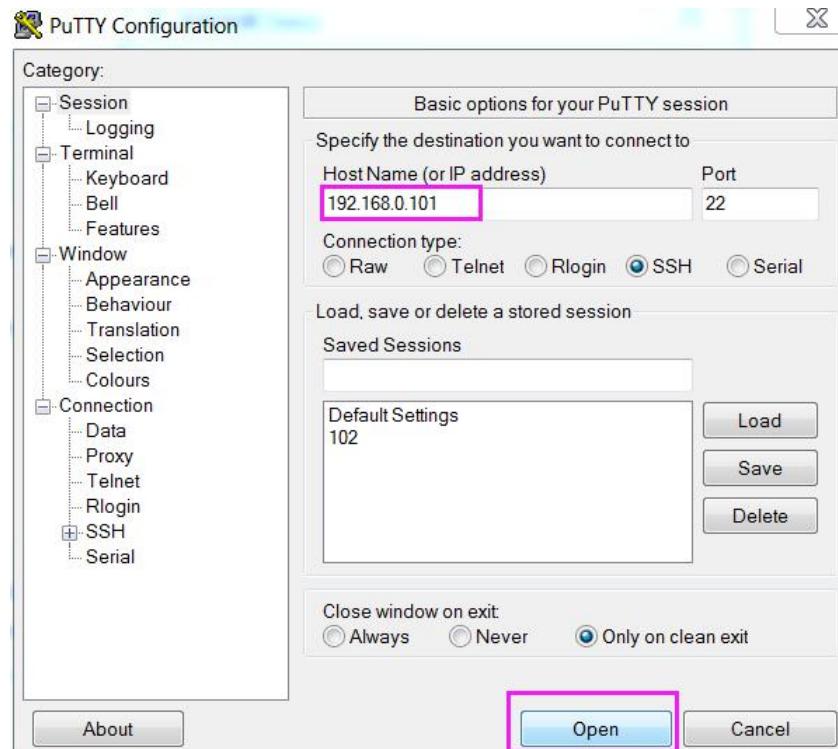
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

### Step 1

Download PuTTY.

### Step 2

Open PuTTY and click **Session** on the left tree-alike structure. Enter the IP address of the RPi in the text box under **Host Name (or IP address)** and **22** under **Port** (by default, it is 22).



## Step 3

Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, there prompts a security reminder. Just click **Yes**.

## Step 4

When the PuTTY window prompts “**login as:**”, type in “**pi**” (the user name of the RPi), and password: “**raspberry**” (the default one, if you haven't changed it).



## Step 5

Here, we get the Raspberry Pi connected and it is time to conduct the next step.

**Note:** When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

## Remote Desktop

If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily. There are two ways to control the desktop of the Raspberry Pi remotely : **VNC** and **XRDP**.

### VNC

You can use the function of remote desktop through VNC.

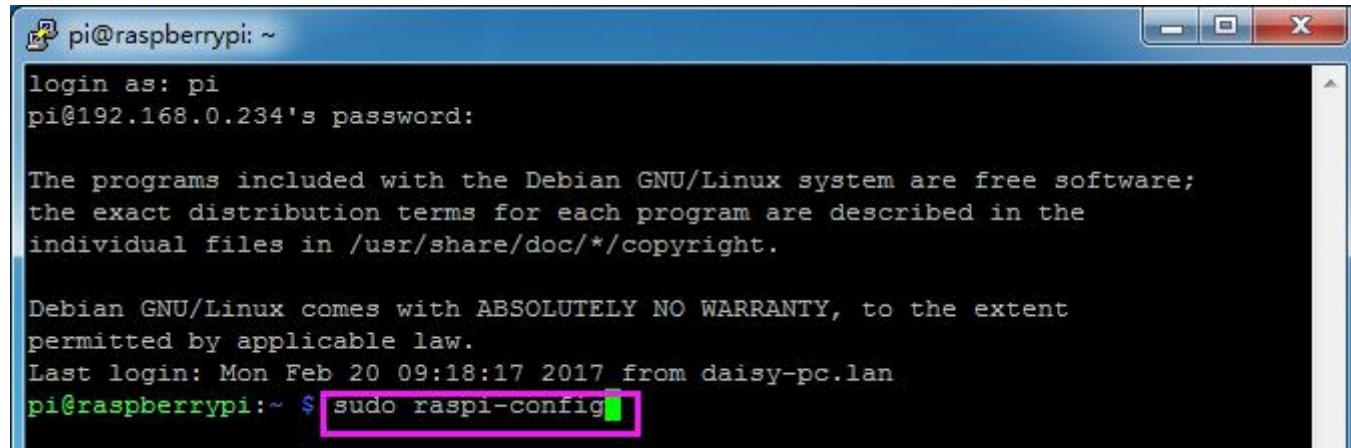
#### Enable VNC Service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

#### Step 1

Input the following command:

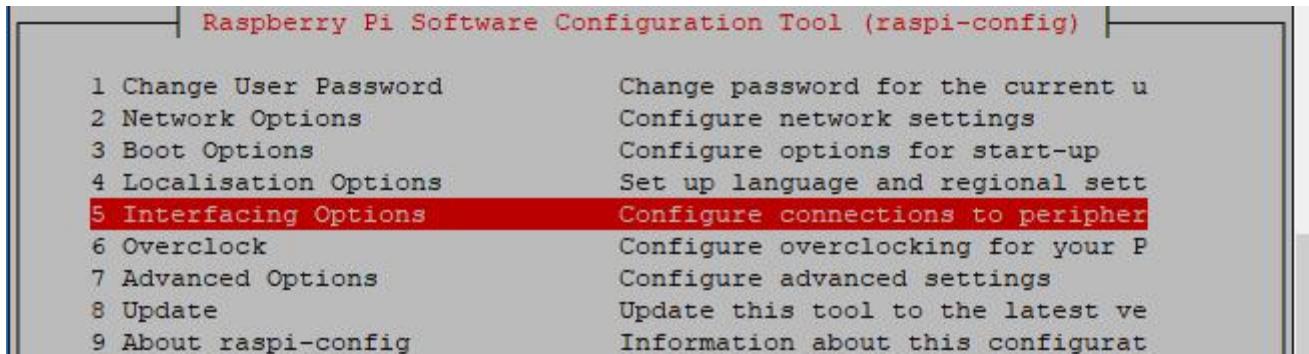
```
sudo raspi-config
```



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows a login prompt: "login as: pi" followed by "pi@192.168.0.234's password:". Below the password prompt is a standard Debian license notice. At the bottom of the window, the command "sudo raspi-config" is typed and highlighted with a pink rectangle. The terminal window has a blue header bar and a white body with black text.

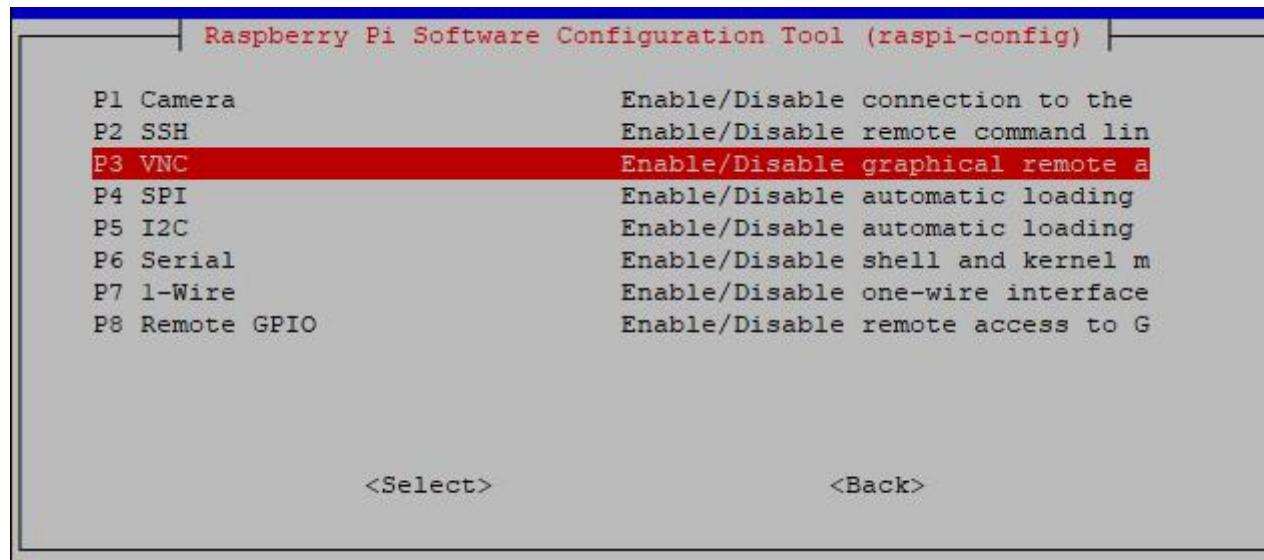
## Step 2

On the config interface, select “**Interfacing Options**” by the up, down, left and right keys on the keyboard.



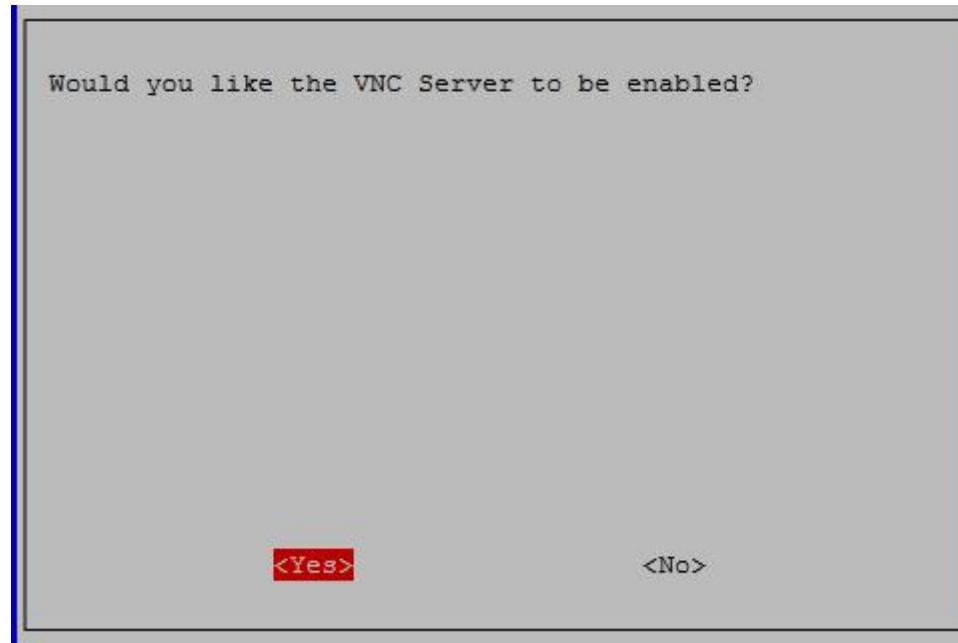
## Step 3

Select **VNC**.



## Step 4

Select **Yes** -> **OK** -> **Finish** to exit the configuration.



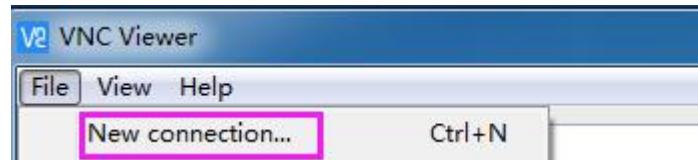
## Login to VNC

### Step 1

You need to install the **VNC Viewer** on personal computer. After the installation is done, open it.

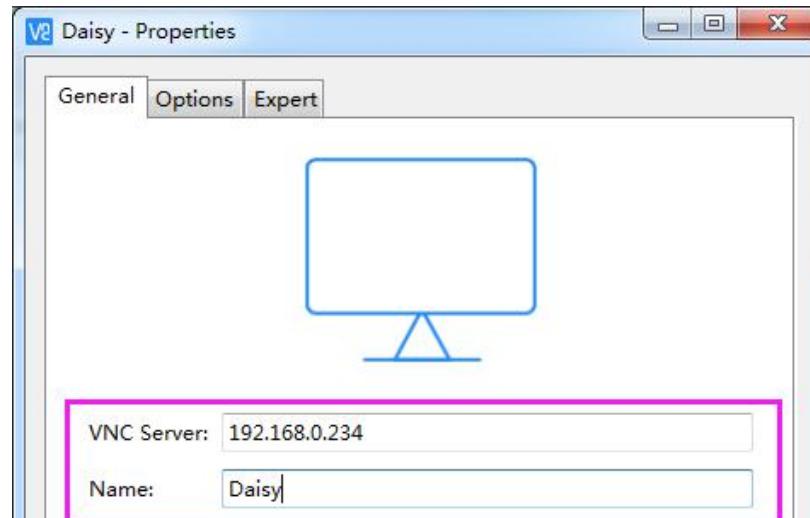
### Step 2

Then select “**New connection**”.



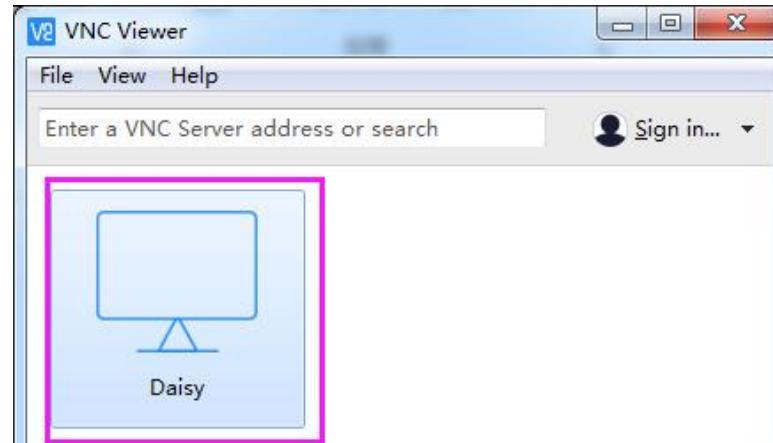
### Step 3

Input IP address of Raspberry Pi and any **Name**.



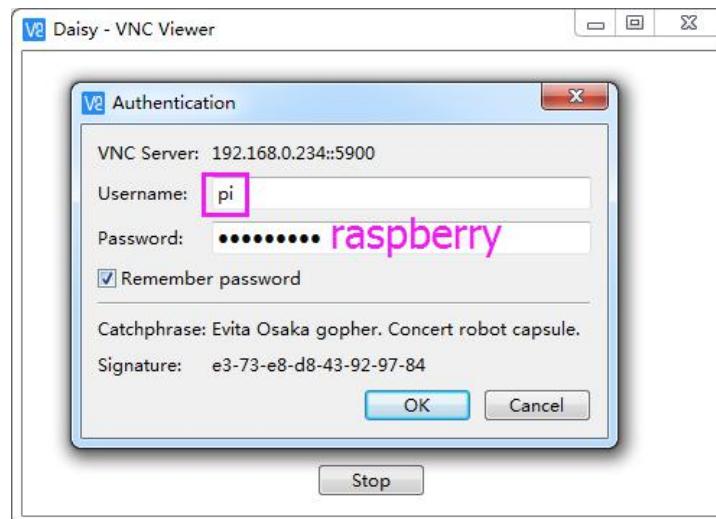
## Step 4

Double click the **connection** just created:



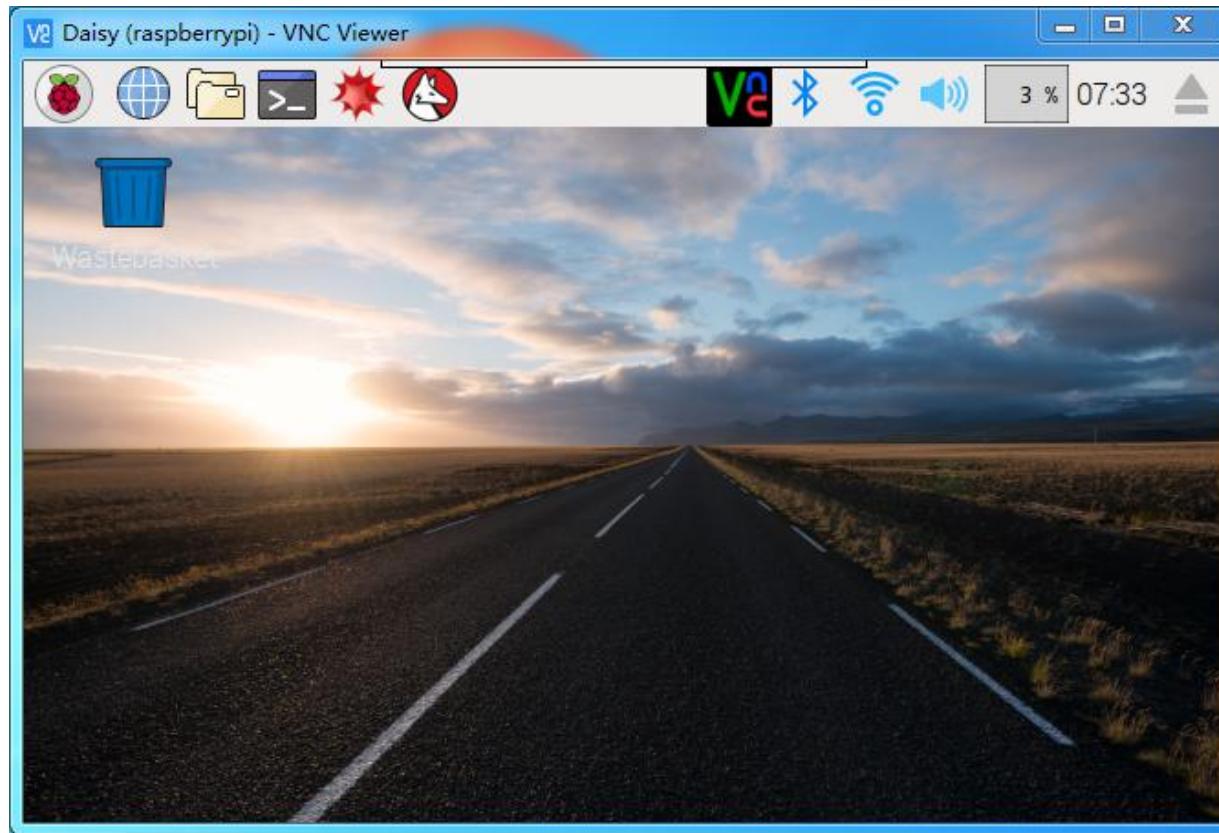
## Step 5

Enter Username ("pi") and Password ("raspberry" by default).



## Step 6

Now you can see the desktop of the Raspberry Pi:



## XRDP

XRDP provides a graphical login to remote machines using RDP (Microsoft Remote Desktop Protocol).

### Install XRDP

#### Step 1

Login to Raspberry Pi by using SSH.

#### Step 2

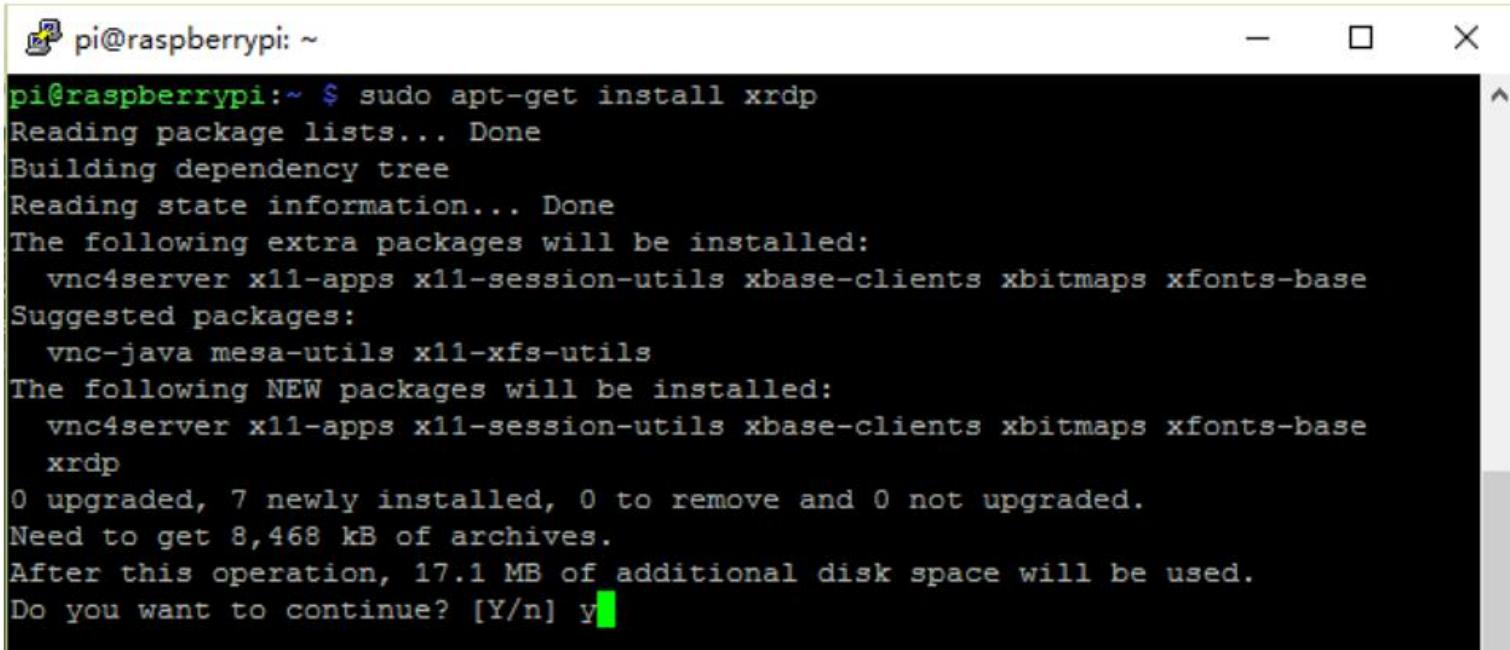
Input the following instructions to install XRDP.

```
sudo apt-get update  
sudo apt-get install xrdp
```

#### Step 3

Later, the installation starts.

Enter "Y", press key "Enter" to confirm.



pi@raspberrypi: ~ \$ sudo apt-get install xrdp  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following extra packages will be installed:  
vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base  
Suggested packages:  
vnc-java mesa-utils x11-xfs-utils  
The following NEW packages will be installed:  
vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base  
xrdp  
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.  
Need to get 8,468 kB of archives.  
After this operation, 17.1 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y

## Step 4

After the installation is completed, you can use Windows remote desktop applications to login to your RPi.

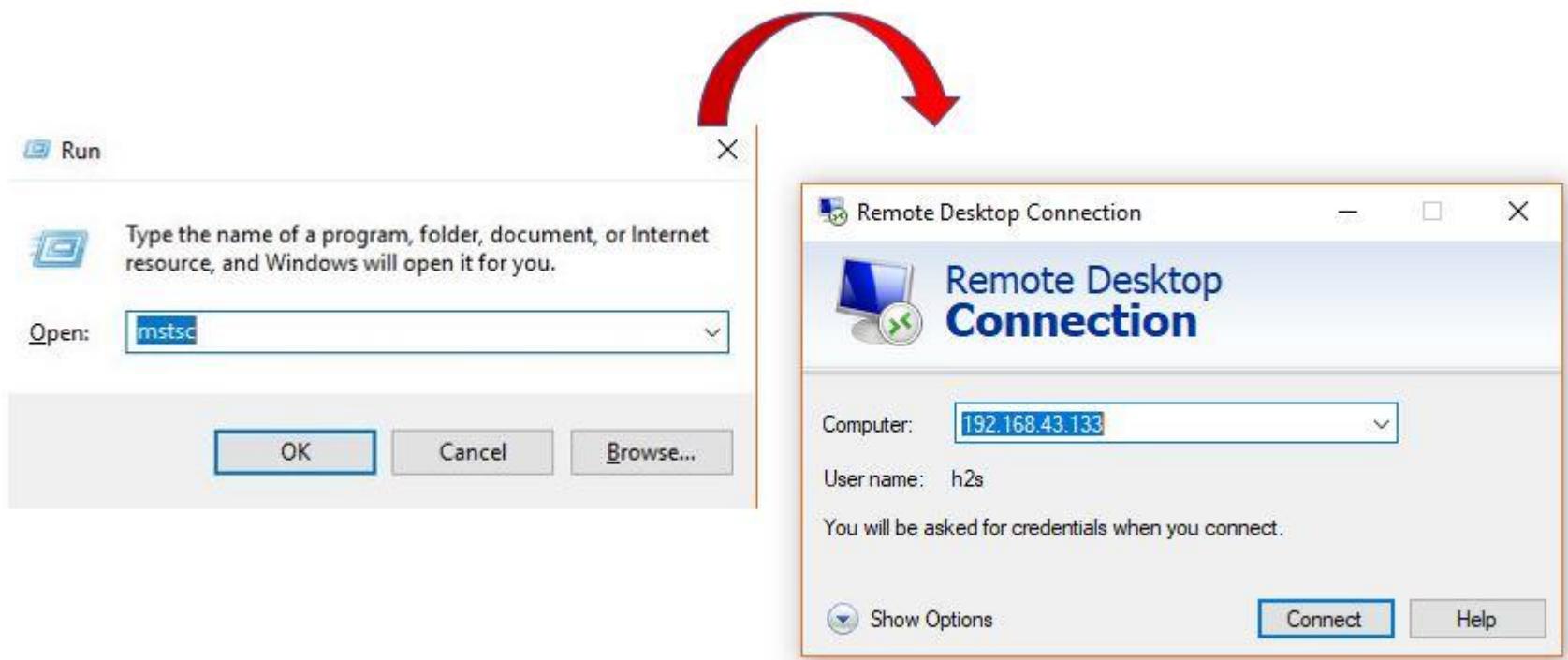
## Login to XRDP

### Step 1

If you are a Windows user, you can use the Remote Desktop feature that comes with Windows. If you are a Mac user, you can download and use Microsoft Remote Desktop from the APP Store, and there is not much difference between them. The next example is Windows remote desktop.

## Step 2

Type in “**mstsc**” in Run (WIN+R) to open the Remote Desktop Connection, and input the IP address of Raspberry Pi, then click on “Connect”.



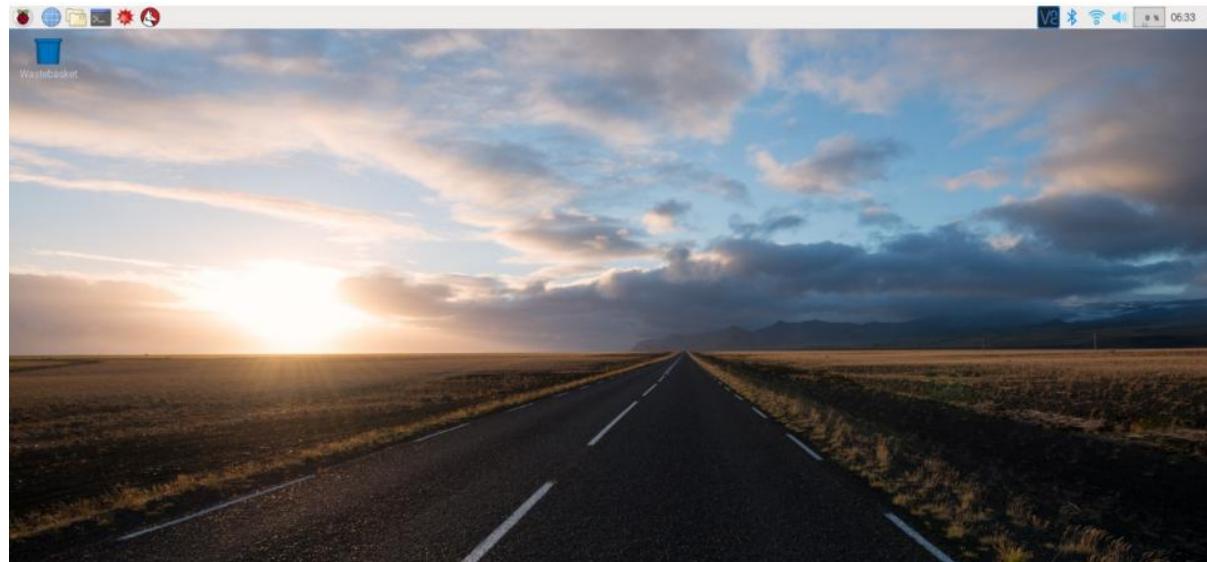
## Step 3

There will be xrdp login screen. Enter the user name and password of RPi and click OK. By default, the user name of Raspberry Pi is “**pi**” and the password is “**raspberry**”.



## Step 4

Here, you successfully login to RPi by using the remote desktop.



# Libraries

Two important libraries are used in programming with Raspberry Pi, and they are wiringPi and RPi.GPIO. The Raspbian OS image of Raspberry Pi installs them by default, so you can use them directly.

## RPi.GPIO

If you are a Python user, you can program GPIOs with API provided by RPi.GPIO.

RPi.GPIO is a module to control Raspberry Pi GPIO channels. This package provides a class to control the GPIO on a Raspberry Pi. For examples and documents, visit

<http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>

Test whether RPi.GPIO is installed or not, type in python:

```
python
```

```
pi@raspberrypi:~ $ python
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

In Python CLI, input “**import RPi.GPIO**”, if no error prompts, it means RPi.GPIO is installed.

```
import RPi.GPIO
```

```
pi@raspberrypi:~ $ python
Python 2.7.9 (default, Mar  8 2015, 00:52:26)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import RPi.GPIO
>>> █
```

If you want to quit python CLI, type in:

```
exit()
```

```
>>> exit()
pi@raspberrypi:~ $ █
```

## WiringPi

wiringPi is a C language GPIO library applied to the Raspberry Pi platform. It complies with GUN Lv3. The functions in wiringPi are similar to those in the wiring system of Arduino. They enable the users familiar with Arduino to use wiringPi more easily.

wiringPi includes lots of GPIO commands which enable you to control all kinds of interfaces on Raspberry Pi. You can test whether the wiringPi library is installed successfully or not by the following instructions.

```
gpio -v
```

```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.32
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty
```

If the message above appears, the wiringPi is installed successfully.

```
gpio readall
```

pi@raspberrypi:~ \$ gpio readall

Pi 3											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	ALT0	1	3	4		5V			
3	9	SCL.1	ALT0	1	5	6		0v			
4	7	GPIO. 7	IN	0	7	8	1	TxD	15	14	
		0v			9	10	1	RxD	16	15	
17	0	GPIO. 0	IN	0	11	12	0	GPIO. 1	1	18	
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	GPIO. 4	4	23	
		3.3v			17	18	0	GPIO. 5	5	24	
10	12	MOSI	ALT0	1	19	20		0v			
9	13	MISO	ALT0	1	21	22	0	GPIO. 6	6	25	
11	14	SCLK	ALT0	0	23	24	1	CE0	10	8	
		0v			25	26	1	CE1	11	7	
0	30	SDA.0	IN	1	27	28	1	GPIO. 26	26	12	
5	21	GPIO.21	IN	0	29	30		0v			
6	22	GPIO.22	IN	0	31	32	0	GPIO.27	27	16	
13	23	GPIO.23	IN	1	33	34		0v			
19	24	GPIO.24	IN	0	35	36	0	GPIO.28	28	20	
26	25	GPIO.25	IN	0	37	38	0	GPIO.29	29	21	
		0v			39	40	0				

For more details about wiringPi, you can refer to: <http://wiringpi.com/download-and-install/>

# Download the Code

Before you download the code, please note that the example code is **ONLY** test on Raspbian. We provide two methods for download:

## Method 1: Use Git Clone (Recommended)

Log into Raspberry Pi and then change directory to `/home/pi`.

```
cd /home/pi/
```

*Note:* `cd` to change to the intended directory from the current path. Informally, here is to go to the path `/home/pi/`.

Clone the repository from GitHub.

```
git clone https://github.com/sunfounder/electronic-kit-for-raspberry-pi.git
```

## Method 2: Download the Code.

Download the source code from GitHub: <https://github.com/sunfounder/electronic-kit-for-raspberry-pi.git>

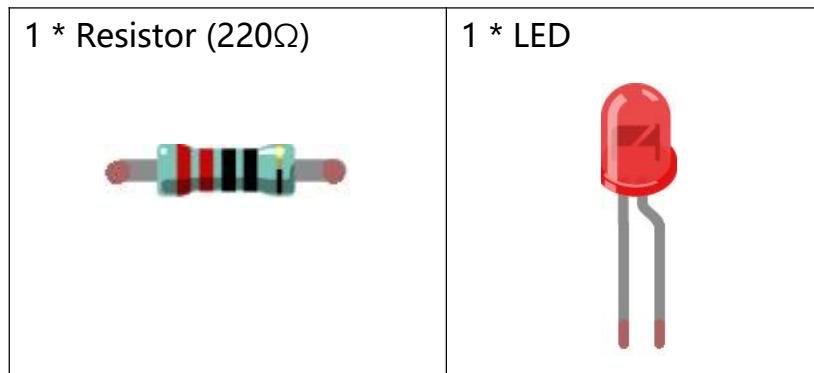
# Lessons

## Lesson 1 Blinking LED

### Introduction

In this lesson, with Raspberry Pi, we will learn how to make a blinking LED by programming. By the way, you can get many interesting phenomena by applying LED. Now get to start and you will enjoy the fun of DIY at once!

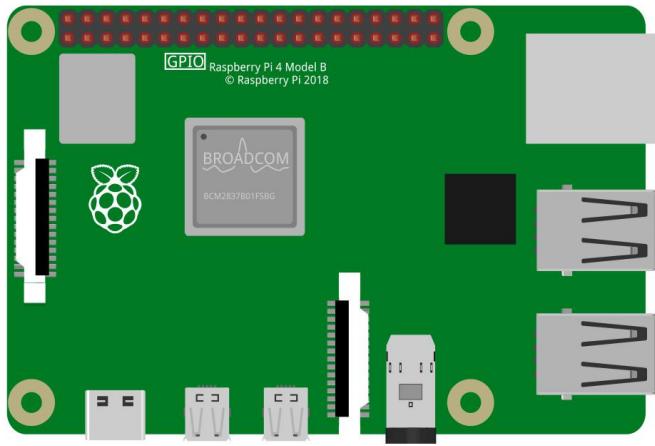
### Newly Added Components



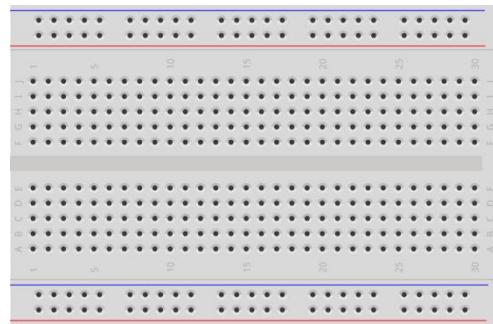
### Components

Note: This table gives the necessary product components of all lessons. In the following lessons, if there is no newly added component, the table will not appear again; instead, the list of newly added components will present for you.

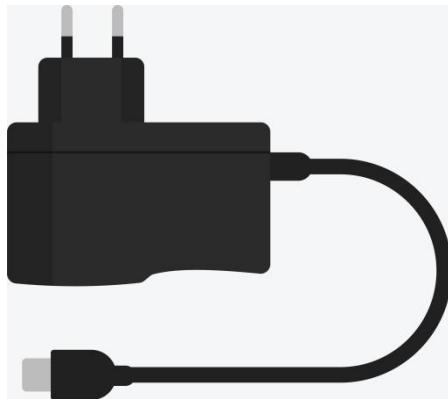
1 \* Raspberry Pi



1 \* Breadboard



1 \* 2.5A Power Adapter



Several Jumper Wires



1 \* TF card

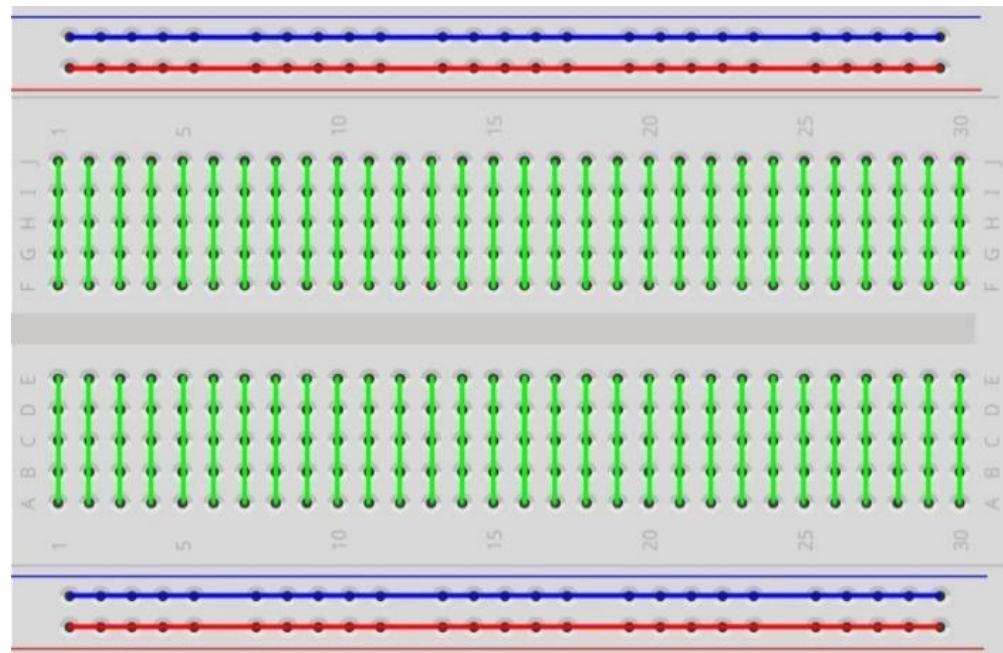


## Principle

### Breadboard

A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you to plug in and remove components easily.

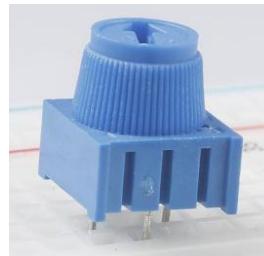
The picture shows the internal structure of a half+ breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



## Resistor

Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

Fixed resistor is applied in this kit. In the circuit, it is essential to protect the connected components. The following pictures show a real object,  $220\Omega$  resistor and two generally used circuit symbols of resistor.  $\Omega$  is the unit of resistance and the larger units include  $K\Omega$ ,  $M\Omega$ , etc. Their relationship can be shown as follows:  $1 M\Omega = 1000 K\Omega$ ,  $1 K\Omega = 1000 \Omega$ , which means  $1 M\Omega = 1000,000 \Omega = 10^6 \Omega$ .



Potentiometer



Fixed Resistor

Normally, the resistance can be marked directly, in color code, and by character. The resistors offered in this kit are marked by different colors. Namely, the bands on the resistor indicate the resistance.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster.

As shown in the card, each color stands for a number.

**6-Band**

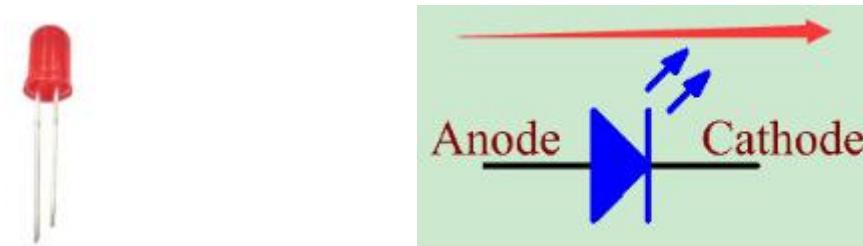
Color	1st Digit	2nd Digit	3rd Digit	Multiplier	Tolerance	Temperature Coefficient
Black	0	0	0	1 Ω	± 1%	250 ppm/K
Brown	1	1	1	10 Ω	± 2%	100 ppm/K
Red	2	2	2	100 Ω	± 0.5%	50 ppm/K
Orange	3	3	3	1k Ω	± 0.25%	15 ppm/K
Yellow	4	4	4	10k Ω	± 0.1%	25 ppm/K
Green	5	5	5	100k Ω	± 0.05%	20 ppm/K
Blue	6	6	6	1M Ω	± 0.025%	10 ppm/K
Violet	7	7	7		± 0.01%	5 ppm/K
Grey	8	8	8			1 ppm/K
White	9	9	9			
Gold				0.1 Ω	± 5%	
Silver				0.01 Ω	± 10%	

**4-Band**  $12 \times 10^5 \pm 5\%$  =  $1,200 \text{ k}\Omega \pm 5\%$

**5-Band**  $100 \times 10^2 \pm 1\%$  =  $10,000 \Omega \pm 1\%$

## LED

Semiconductor light-emitting diode is a type of component which can turn electric energy into light energy via PN junctions. In terms of wavelength, it can be categorized into laser diode, infrared light-emitting diode and visible light-emitting diode, known as light-emitting diode (LED).



Diode has unidirectional conductivity, so the current flow will be as the arrow indicates in figure circuit symbol. You can only provide the anode with a positive power and the cathode with a negative one. Thus the LED will light up.

An LED has two pins. **The longer one is anode**, and the **shorter one, cathode**. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$$R = (V_{\text{supply}} - V_D)/I$$

R stands for the resistance value of the current limiting resistor,  $V_{\text{supply}}$  for voltage supply,  $V_D$  for voltage drop and I for the working current of the LED.

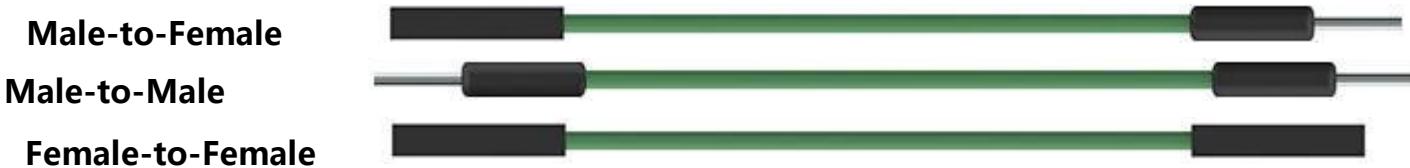
If we provide 5 Volt for the red LED, the minimum resistance of the current limiting resistor should be:  $(5V-1.8V)/20mA = 160 \Omega$ . Therefore, you need a  $160 \Omega$  or larger resistor to protect the LED. You are recommended to use the  $220\Omega$  resistor offered in the kit.

## Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Especially, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jump wires are fitted by inserting their "end connectors" into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The "end connectors" are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female.



More than one type of them may be used in a project. **The color of the jump wires is different but it doesn't mean their function is different accordingly; it's just designed as this way to better identify the connection between each circuit.**

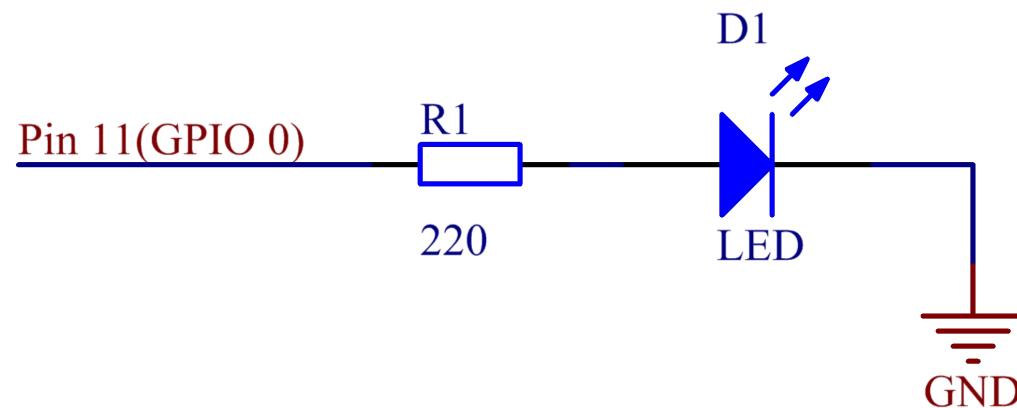
## Schematic Diagram

In this experiment, connect a  $220\Omega$  resistor to the anode (the long pin of the LED), then the resistor to Pin11 of Raspberry Pi, and connect the cathode (the short pin) of the LED to GND. **Therefore**, to turn on a LED, we need to make pin11 high level. We can get this phenomenon by programming.

Note: Pin11 refers to the 11th pin of the Raspberry Pi from left to right, and its corresponding wiringPi and BCM pin numbers are shown in the following table.

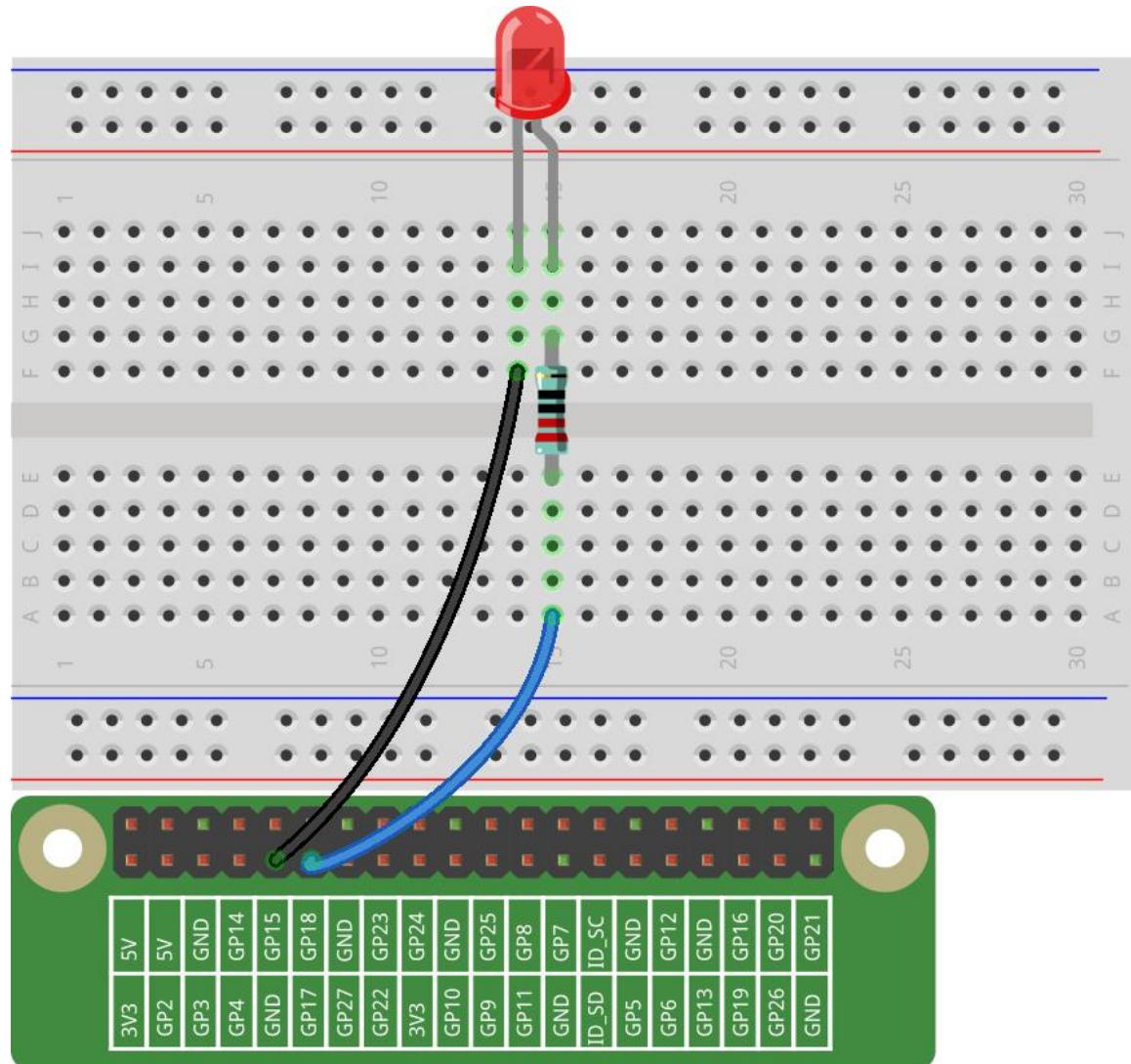
In the C language related content, we make GPIO 0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi Physical, Pin 11.

wiringPi	Physical	BCM
0	Pin11	17



## Build the Circuit

Note: the pin with a curve is the anode of the LED.



➤ For C Language Users:

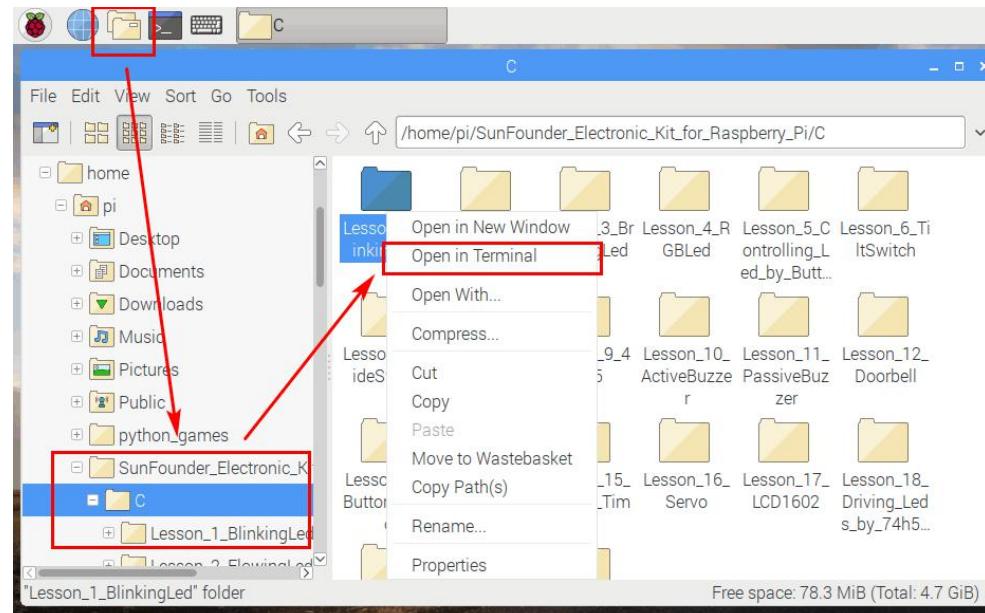
## Command

1. Go to the folder of the code.

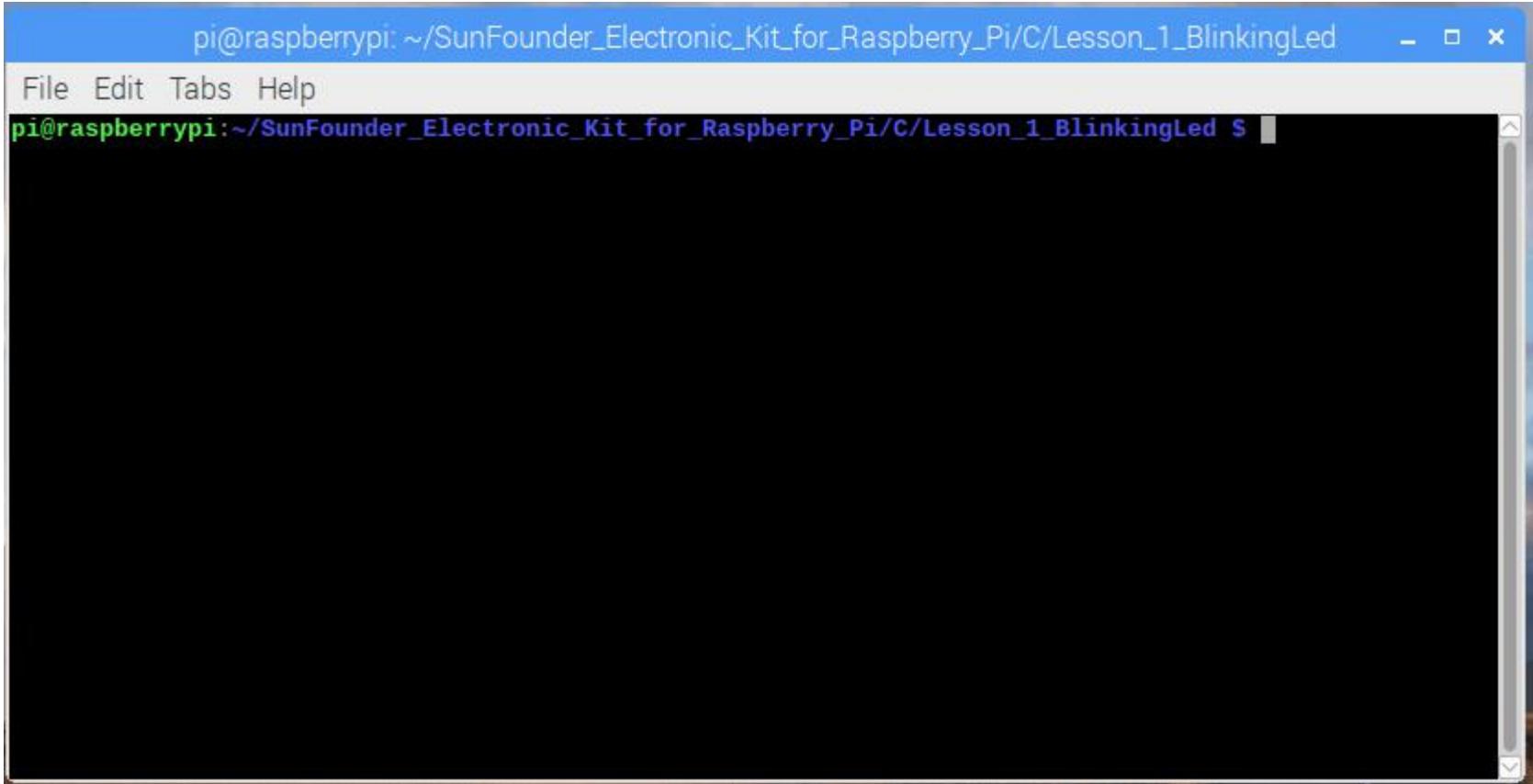
If you use a monitor, you're recommended to take the following steps.

Go to `/home/pi/` and find the folder **electronic-kit-for-raspberry-pi**.

Find **C** in the folder, right-click on it and select **Open in Terminal**.



Then a window will pop up as shown below. So now you've entered the path of the code **1\_BlinkingLed.c**

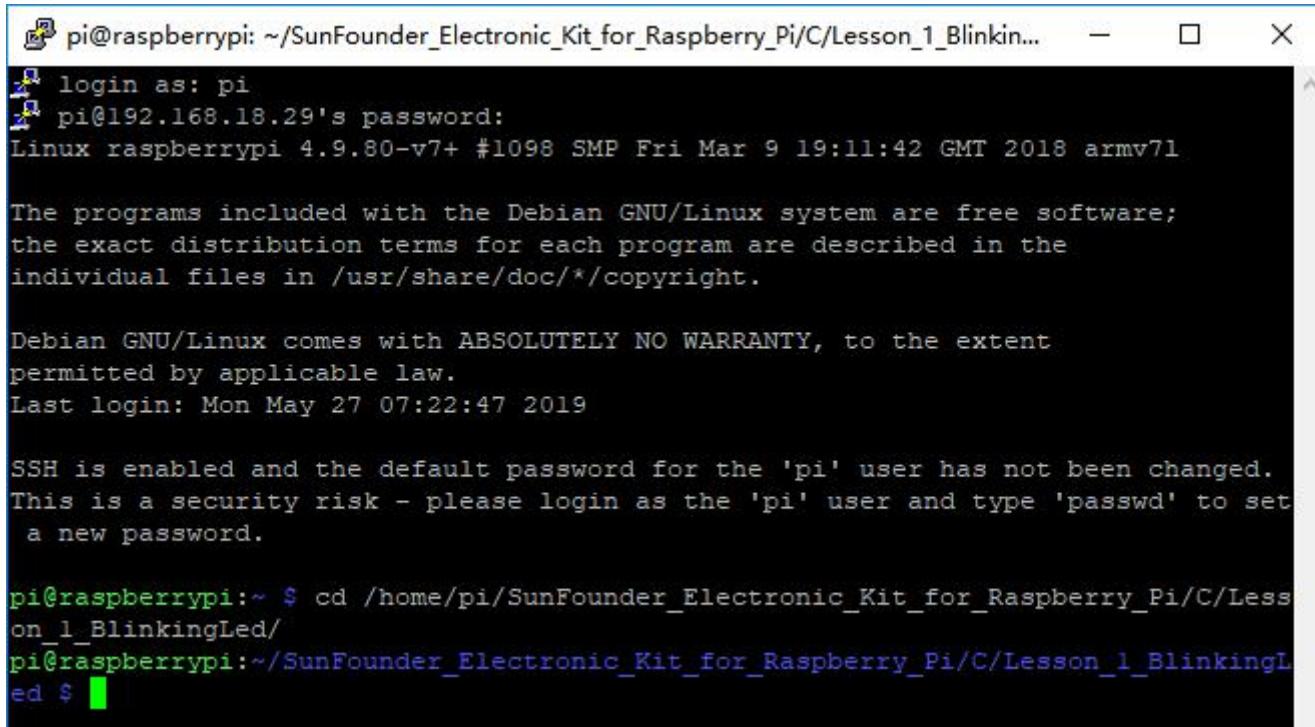


In the following lessons, we will use command to enter the code file instead of right-clicking. But you can choose the method you prefer.

**If you log into the Raspberry Pi remotely, use “cd” to change directory:**

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_1_BlinkingLed
```

**Note:** Change directory to the path of the code via **cd** in this experiment.



```
pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_Blinkin... - X
pi@raspberrypi: ~ login as: pi
pi@192.168.18.29's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 27 07:22:47 2019

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $ cd /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Less
on_1_BlinkingLed/
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL
ed $
```

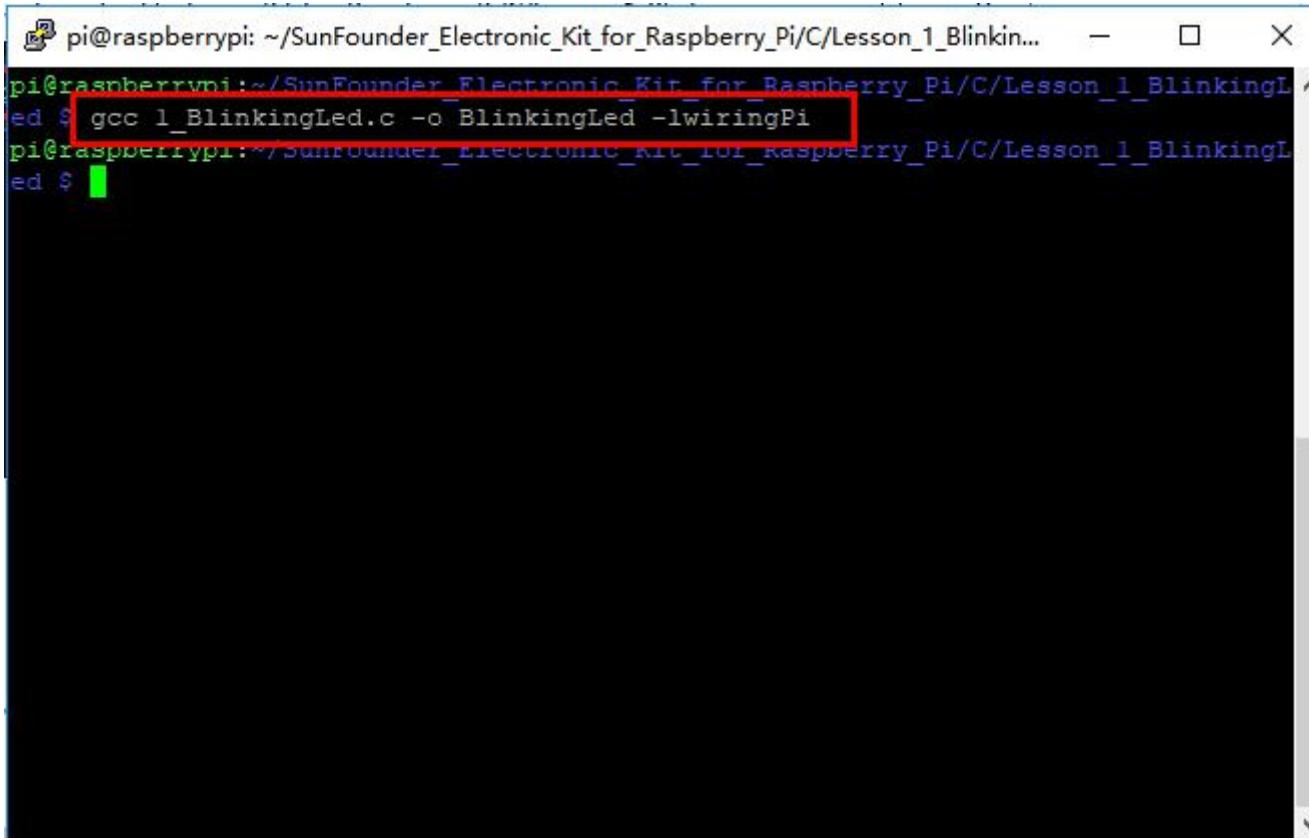
In either way, now you are in the folder *Lesson\_1\_BlinkingLed*. The subsequent procedures based on these two methods are the same. Let's move on.

## 2. Compile the code.

```
gcc 1_BlinkingLed.c -o BlinkingLed -lwiringPi
```

**Note:** **gcc** is **GNU Compiler Collection**. Here, its functions like compiling the C language file *1\_BlinkingLed.c* and outputting an executable file.

In the command, `-o` means outputting (the character immediately following `-o` is the filename output after compilation, and an executable named **BlinkingLed** will generate here) and `-lwiringPi` is to load the library `wiringPi` (`|` is the abbreviation of library).



A screenshot of a terminal window titled "pi@raspberrypi: ~/SunFounder\_Electronic\_Kit\_for\_Raspberry\_Pi/C/Lesson\_1\_Blinkin...". The window contains the following text:

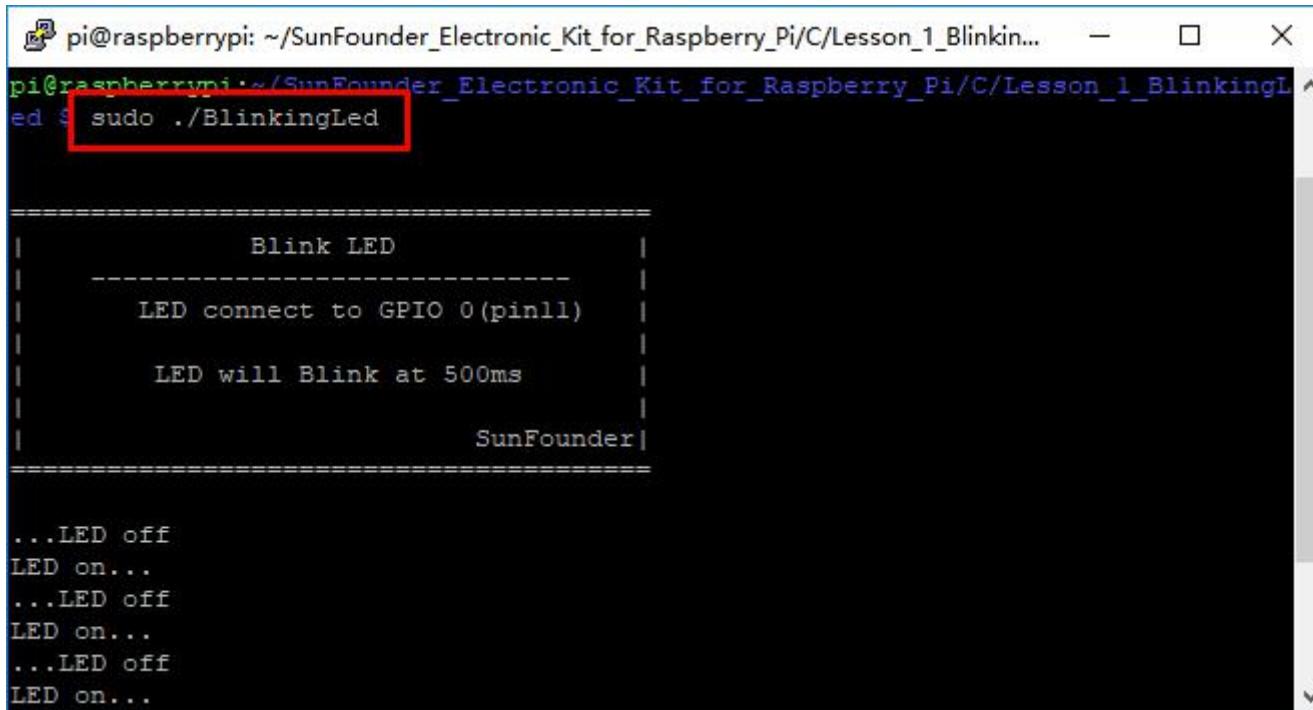
```
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL  
ed $ gcc l_BlinkingLed.c -o BlinkingLed -lwiringPi  
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL  
ed $
```

The command `gcc l_BlinkingLed.c -o BlinkingLed -lwiringPi` is highlighted with a red rectangle.

3. Run the executable file output in the previous step:

```
sudo ./BlinkingLed
```

**Note:** To control the GPIO, you need to run the program by the command, **sudo**(superuser do). The command "./" indicates the current directory. The whole command is to run the **BlinkingLed** in the current directory.



The screenshot shows a terminal window on a Raspberry Pi. The title bar reads "pi@raspberrypi: ~/SunFounder\_Electronic\_Kit\_for\_Raspberry\_Pi/C/Lesson\_1\_Blinkin...". The command "sudo ./BlinkingLed" is highlighted with a red box. The output of the program is displayed below, showing the LED configuration and the start of the blinking loop.

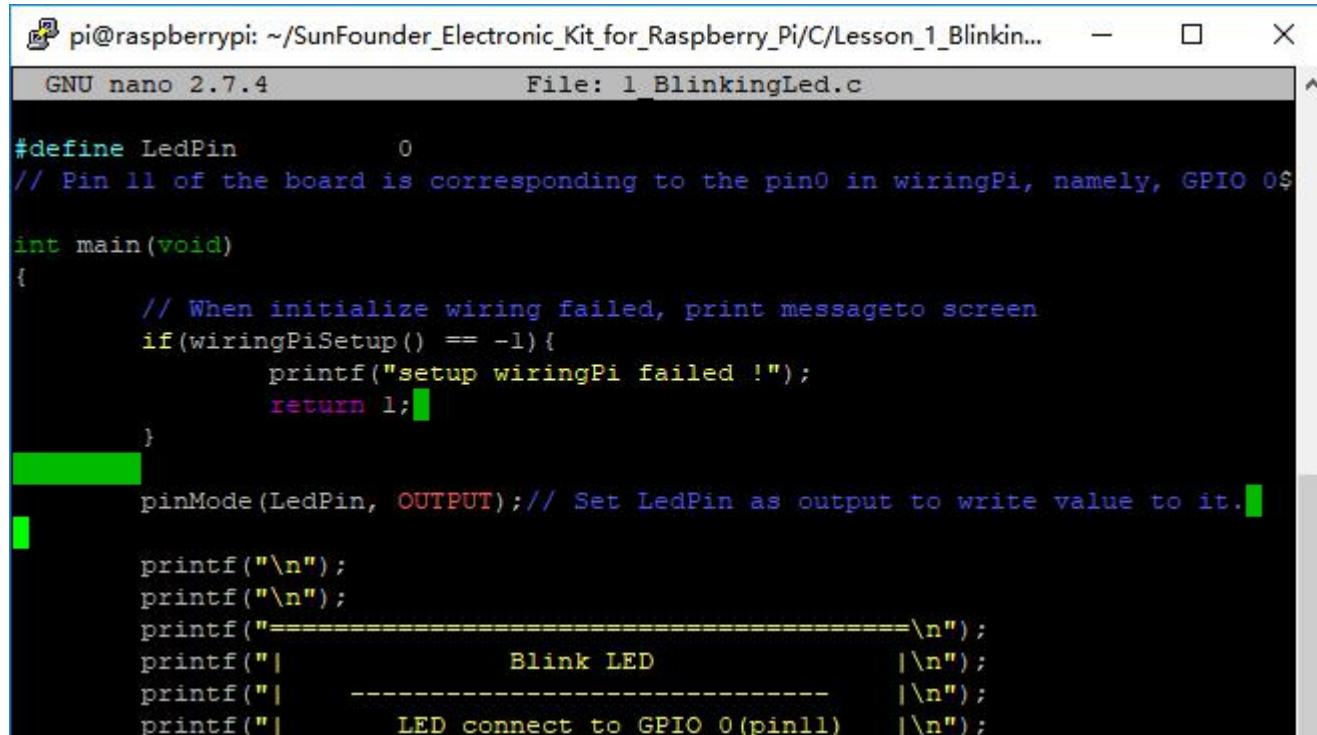
```
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingLed$ sudo ./BlinkingLed
=====
|           Blink LED
| -----
|   LED connect to GPIO 0(pin11)
|
|   LED will Blink at 500ms
|
|           SunFounder
=====
...
LED off
LED on...
LED off
LED on...
LED off
LED on...
```

As the code runs, you will see the LED blinking.

4. If you want to edit the code file *1\_BlinkingLed.c*, press **Ctrl + C** to stop running the current code. Then type the following command to open *1\_BlinkingLed.c*.

**nano 1\_BlinkingLed.c**

**Note:** nano is a text editor tool. The command is used to open the code file *1\_BlinkingLed.c* by this tool.



The screenshot shows a terminal window titled "pi@raspberrypi: ~/SunFounder\_Electronic\_Kit\_for\_Raspberry\_Pi/C/Lesson\_1\_Blinkin...". The window displays a C program named "1\_BlinkingLed.c" using the "GNU nano 2.7.4" editor. The code defines a pin as an output and initializes wiringPi. It then enters a loop where it toggles the LED connected to GPIO 0 (pin 11) every second. The terminal window has a light gray background with dark gray borders around the title bar and the code area.

```
#define LedPin      0
// Pin 11 of the board is corresponding to the pin0 in wiringPi, namely, GPIO 0

int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(LedPin, OUTPUT); // Set LedPin as output to write value to it.

    printf("\n");
    printf("\n");
    printf("=====|\n");
    printf("||     Blink LED      |\n");
    printf("||     -----      |\n");
    printf("||     LED connect to GPIO 0(pin11) |\n");
}
```

## Code

The program code is shown as follows:

1. #include <wiringPi.h>
2. #include <stdio.h>
3. #define LedPin 0
- 4.
5. int main(void)
6. {
7. // When initialize wiring failed, print message to screen

```
8.     if(wiringPiSetup() == -1){  
9.         printf("setup wiringPi failed !");  
10.        return 1;  
11.    }  
12.  
13.    pinMode(LedPin, OUTPUT);  
14.  
15.    while(1{  
16.        // LED off  
17.        digitalWrite(LedPin, LOW);  
18.        printf("...LED off\n");  
19.        delay(500);  
20.        // LED on  
21.        digitalWrite(LedPin, HIGH);  
22.        printf("LED on...\n");  
23.        delay(500);  
24.    }  
25.  
26.    return 0;  
27. }
```

## Code Explanation

1. `#include <wiringPi.h>`

The hardware drive library is designed for the C language of Raspberry Pi. Adding this library is conducive to the initialization of hardware, and the output of I/O ports, PWM, etc.

2. `#include <stdio.h>`

Standard I/O library. The **printf** function used for printing the data displayed on the screen is realized by this library. There are many other performance functions for you to explore.

3. `#define LedPin 0`

Assign GPIO **0** to **LedPin** that represents GPIO 0 in the code later.

```
8. if(wiringPiSetup() == -1){  
9.     printf("setup wiringPi failed !");  
10.    return 1;
```

This initializes wiringPi library and assumes that the calling program is going to be using the wiringPi pin numbering scheme. This function needs to be called with root privileges. When initialize wiring failed, print message to screen.

13. `pinMode(LedPin, OUTPUT);`

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 hardware PWM output, you can also set other pins to PWM output using the **softPWM** library. Only wiringPi pin 7 (BCM\_GPIO 4) supports CLOCK output modes. Here we set **LedPin** as **OUTPUT** mode to write value to it.

17.        digitalWrite(LedPin, LOW);

Writes the value **HIGH** or **LOW** (1 or 0) to the given pin which must have been previously set as OUTPUT. On Raspberry Pi, when the output voltage is less than 0.4V, by default, it is low level, **LOW**, and when the voltage is greater than 2.4V, it is high level, **HIGH**. Since the anode of LED is connected to GPIO 0, thus the LED will light up if GPIO 0 is set high. On the contrary, set GPIO 0 as low level, `digitalWrite (LedPin, LOW)`, LED will go out.

18.        printf("...LED off\n");

The **printf** function is a standard library function and its function prototype is in the header file "stdio.h". The general form of the call is: `printf("format control string", output table columns)`. The format control string is used to specify the output format, which is divided into format string and non-format string. The format string starts with "%" followed by format characters such as "%d" for decimal integer output. Non-format strings are printed as prototypes. What is used here is a non-format string, followed by "\n" that is a newline character, representing automatic line wrapping after printing a string.

19.        delay(500);

This is a function that suspends the program for a period of time. And the speed of the program is determined by our hardware. Here we turn on or off the LED. If there is no **delay** function, the program will run the whole program very fast and continuously loop and we can hardly observe the phenomenon. So we need the **delay** function to help us write and debug the program. **delay (500)** keeps the current HIGH or LOW state for 500ms(0.5s).

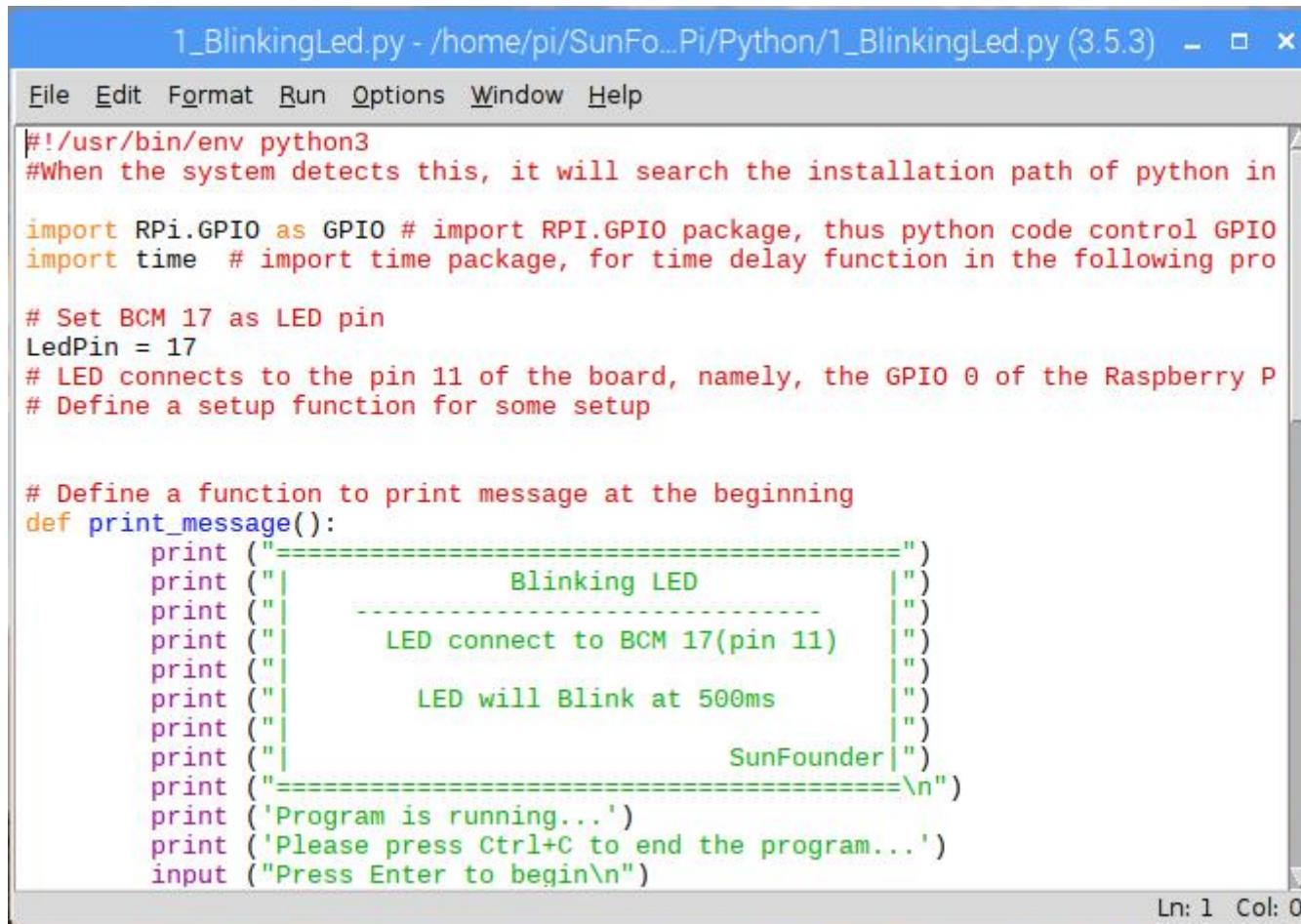
26.        **return 0;**

Usually, it is placed in the last position of the **main** function, indicating that the function returns 0 after executing the function.

## ➤ For Python Language Users

If you use a monitor, you're recommended to take the following steps.

Find **1\_BlinkingLed.py** and double click it to open the file.



The screenshot shows a terminal window titled "1\_BlinkingLed.py - /home/pi/SunFo...Pi/Python/1\_BlinkingLed.py (3.5.3)". The window contains the following Python code:

```
#!/usr/bin/env python3
#When the system detects this, it will search the installation path of python in

import RPi.GPIO as GPIO # import RPI.GPIO package, thus python code control GPIO
import time # import time package, for time delay function in the following pro

# Set BCM 17 as LED pin
LedPin = 17
# LED connects to the pin 11 of the board, namely, the GPIO 0 of the Raspberry P
# Define a setup function for some setup

# Define a function to print message at the beginning
def print_message():
    print ("====")
    print ("|          Blinking LED      |")
    print ("|-----|")
    print ("|      LED connect to BCM 17(pin 11) |")
    print ("|-----|")
    print ("|      LED will Blink at 500ms     |")
    print ("|-----|")
    print ("|          SunFounder           |")
    print ("====\n")
    print ('Program is running...')
    print ('Please press Ctrl+C to end the program...')
    input ("Press Enter to begin\n")

Ln: 1 Col: 0
```

Click **Run ->Run Module** in the window and the following contents will appear.

The screenshot shows a terminal window titled '\*Python 3.5.3 Shell\*' with the following content:

```
File Edit Shell Debug Options Window Help
[GCC 5.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/1_BlinkingLed.py

Warning (from warnings module):
  File "/home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/1_BlinkingLed.py", line 34
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.
=====
|           Blinking LED
|   -----
|   LED connect to BCM 17(pin 11)
|   -----
|   LED will Blink at 500ms
|   -----
|           SunFounder
=====

Program is running...
Please press Ctrl+C to end the program...
Press Enter to begin

...LED ON
LED OFF...
...LED ON
LED OFF...
|
```

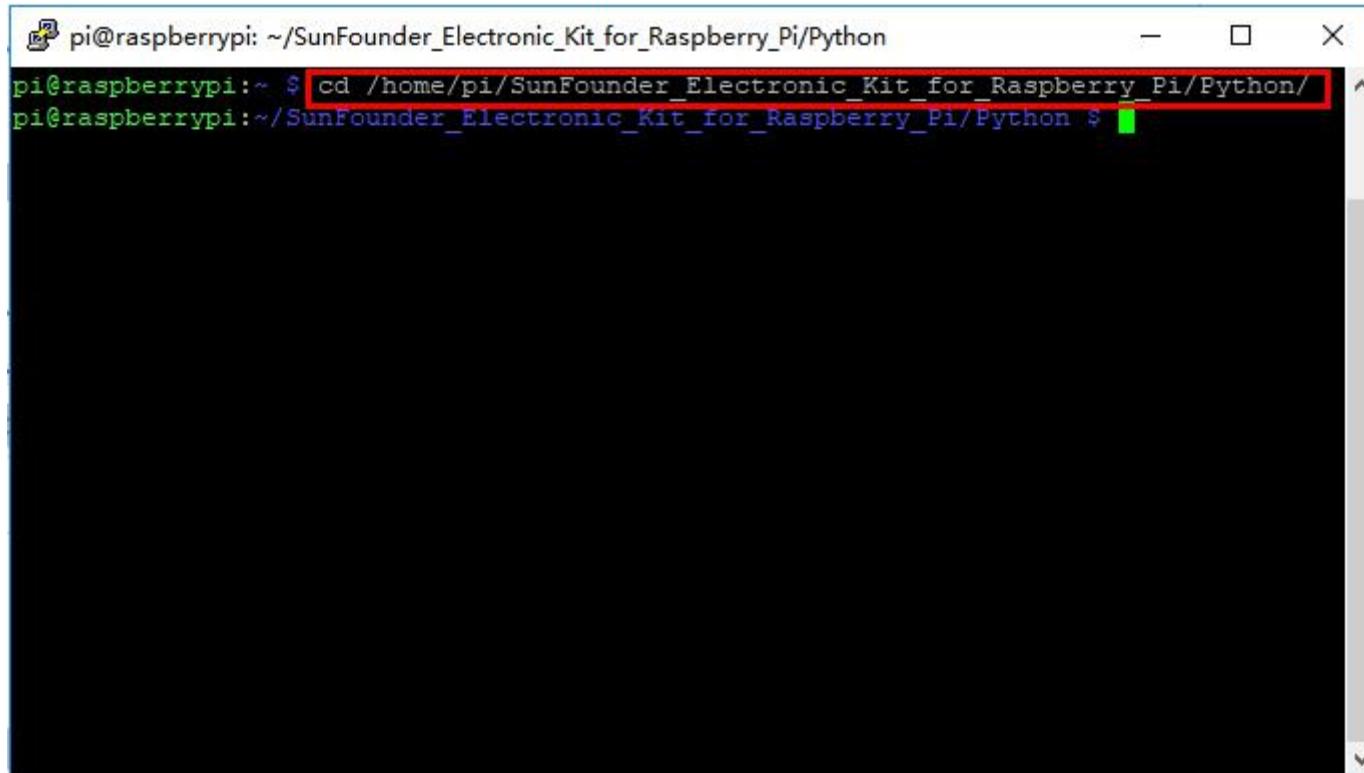
Ln: 28 Col: 0

To stop it from running, just click the **X** button on the top right corner to close it and then you'll back to the code. If you modify the code, before clicking Run Module (**F5**) you need to save it first. Then you can see the results.

If you log into the Raspberry Pi remotely, type in the command:

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

**Note:** Change directory to the path of the code via **cd** in this experiment.

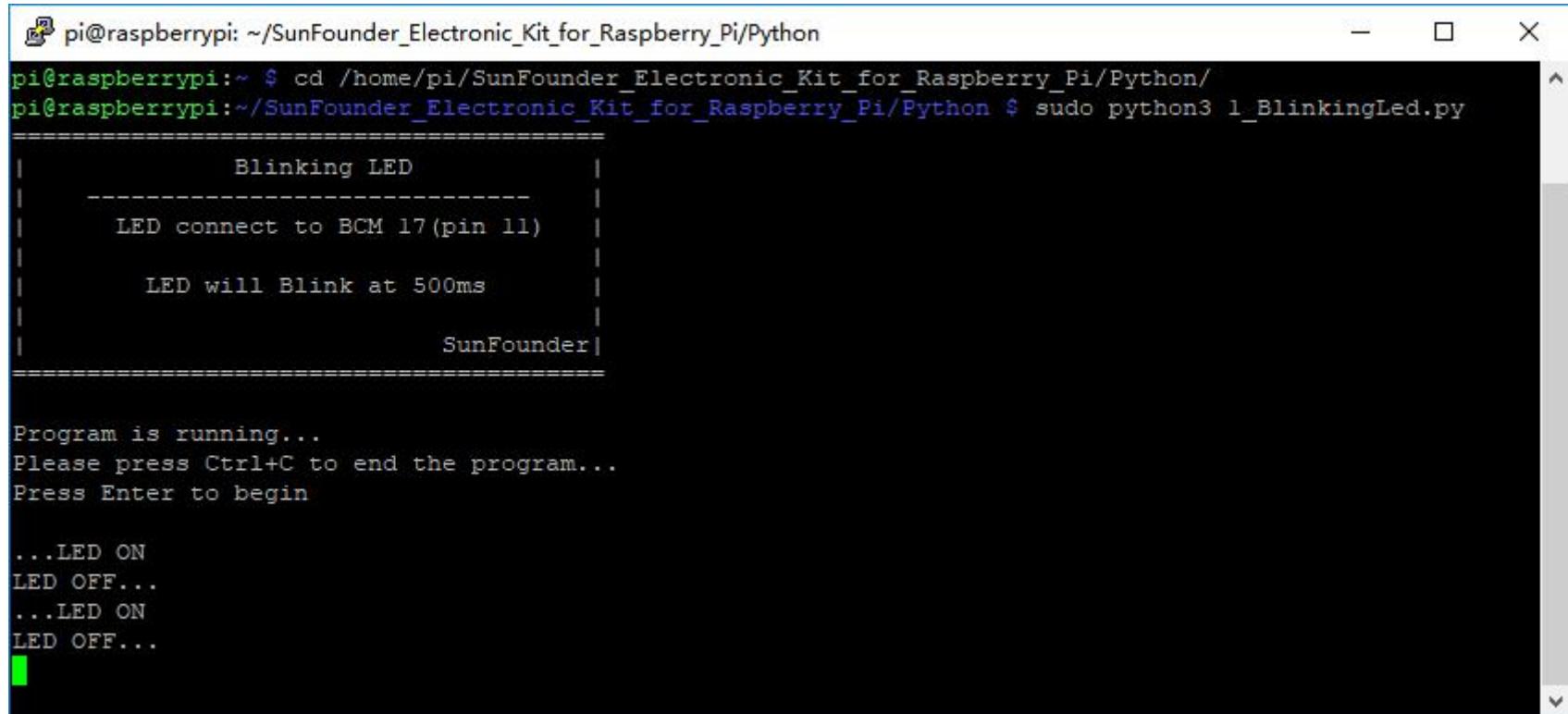


```
pi@raspberrypi: ~$ cd /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/  
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python $
```

## 2. Run the code.

```
sudo python 1_BlinkingLed.py
```

**Note:** Here, **sudo** means superuser do, and the command **python** means to run the file by the programming language, Python. If you have both python2 and python3, you need to change the python in the command to **python3**.



The screenshot shows a terminal window titled "pi@raspberrypi: ~/SunFounder\_Electronic\_Kit\_for\_Raspberry\_Pi/Python". The user runs the command "sudo python3 1\_BlinkingLed.py". The program outputs the following text:

```
Blinking LED
-----
LED connect to BCM 17(pin 11)
LED will Blink at 500ms
SunFounder

Program is running...
Please press Ctrl+C to end the program...
Press Enter to begin

...LED ON
LED OFF...
...LED ON
LED OFF...
```

A small green vertical bar is visible on the left side of the terminal window.

As the code runs, you will see the LED blinking.

3. If you want to edit the code file **1\_BlinkingLed.c**, press **Ctrl + C** to stop running the current code. Then type the following command to open **1\_BlinkingLed.c**

```
nano 1_BlinkingLed.py
```

**Note:** nano is a text editor tool. The command is used to open the code file **1\_BlinkingLed.c** by this tool.

pi@raspberrypi: ~/SunFounder\_Electronic\_Kit\_for\_Raspberry\_Pi/Python

GNU nano 2.7.4 File: 1\_BlinkingLed.py

```
# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,
    # and initial level to LOW(0v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
    # Print messages
    print_message()
    while True:
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)
        # delay 0.5 second, which is equals to the delay in C language, using second as the unit

        print ('LED OFF...')
        # Turn off LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)

^G Get Help ^C Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page
^X Exit      ^R Read File ^\ Replace ^U Uncut Text ^T To Linter ^ Go To Line ^V Next Page
```

## Code

The following is the program code:

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. # Set BCM 17 as LED pin
5. LedPin = 17
6.
7. # Define a setup function for some setup
8. def setup():
9.     GPIO.setmode(GPIO.BCM)
10.    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
11.
12. # Define a main function for main process
13. def main():
14.     while True:
15.         print ('...LED ON')
16.         # Turn on LED
17.         GPIO.output(LedPin, GPIO.HIGH)
18.         time.sleep(0.5)
19.         print ('LED OFF...')
20.         # Turn off LED
21.         GPIO.output(LedPin, GPIO.LOW)
```

```
22.     time.sleep(0.5)
23.
24. # Define a destroy function for clean up everything after the script finished
25. def destroy():
26.     # Turn off LED
27.     GPIO.output(LedPin, GPIO.LOW)
28.     # Release resource
29.     GPIO.cleanup()
30.
31. # If run this script directly, do:
32. if __name__ == '__main__':
33.     setup()
34.     try:
35.         main()
36.     # When 'Ctrl+C' is pressed, the child program
37.     # destroy() will be executed.
38.     except KeyboardInterrupt:
39.         destroy()
```

## Code Explanation

1. import RPi.GPIO as GPIO

In this way, import the RPi.GPIO library, then define a variable, **GPIO** to replace **RPI.GPIO** in the following code.

2. import time

Import **time** library to help use **delay** function in the following program.

5. `LedPin = 17`

LED connects to the pin 11 of the board, namely, the **BCM 17** of the Raspberry Pi.

9. `GPIO.setmode(GPIO.BCM)`

There are two ways of numbering the I/O pins on a Raspberry Pi within RPi.GPIO: BOARD numbers and BCM numbers. In our lessons, what we use is **BCM** numbering method.

10. `GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)`

You need to set up every channel you use as input mode or output mode. Here we set the mode of LedPin to **GPIO.OUT**, and initial level to **LOW( 0v )**.

17. `GPIO.output(LedPin, GPIO.HIGH)`

Set LedPin to output high level to light up LED.

18. `time.sleep(0.5)`

Delay for 0.5 second. Here, the statement is similar to delay function in C language, the unit is second.

32. `if __name__ == '__main__':`

33.  `setup()`

34.  `try:`

35.  `main()`

36.  `# When 'Ctrl+C' is pressed, the program`

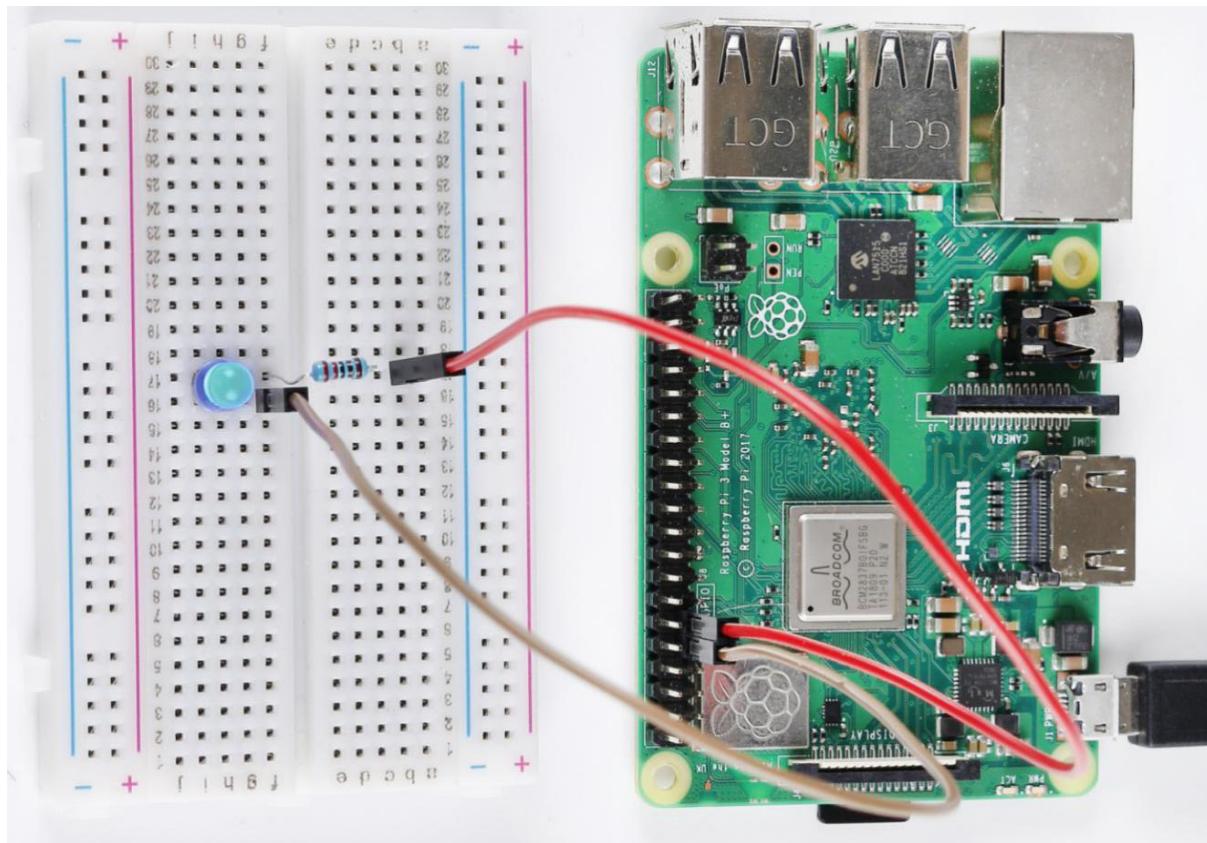
37.  `# destroy() will be executed.`

38. `except KeyboardInterrupt:`

39. `destroy()`

This is the general running structure of the code. When the program starts to run, it initializes the pin by running the `setup()`, and then runs the code in the `main()` function to set the pin to high and low levels. When 'Ctrl+C' is pressed, the program, `destroy()` will be executed.

## Phenomenon Picture

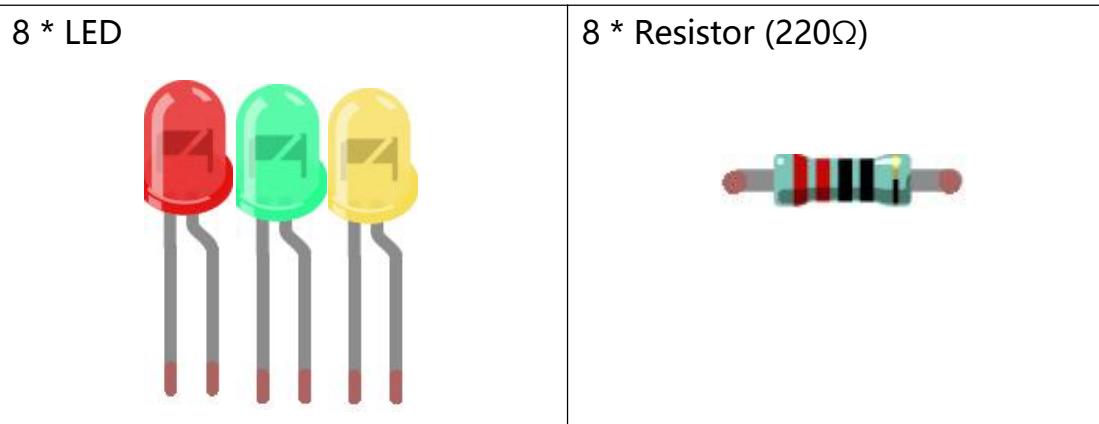


## Lesson 2 Flowing LED Lights

### Introduction

In this lesson, we will learn how to make eight LEDs blink as flowing water based on Raspberry Pi.

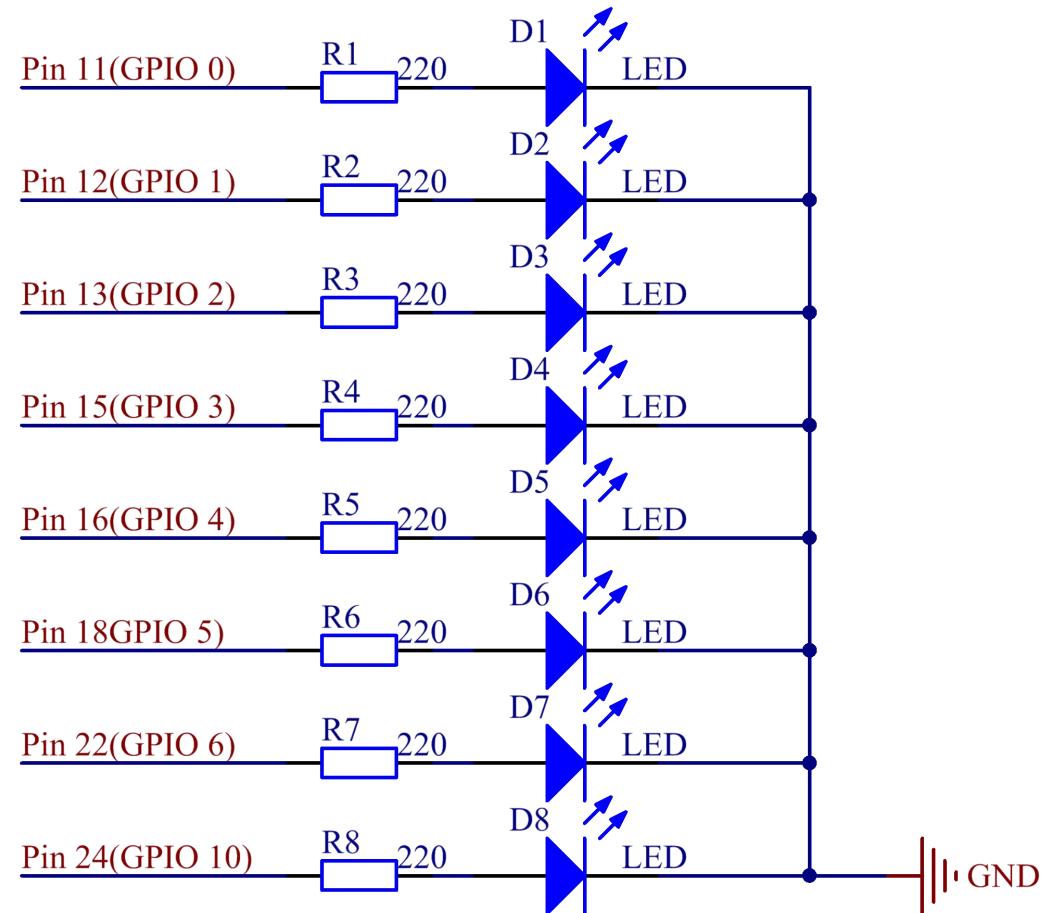
### Newly Added Components



### Schematic Diagram

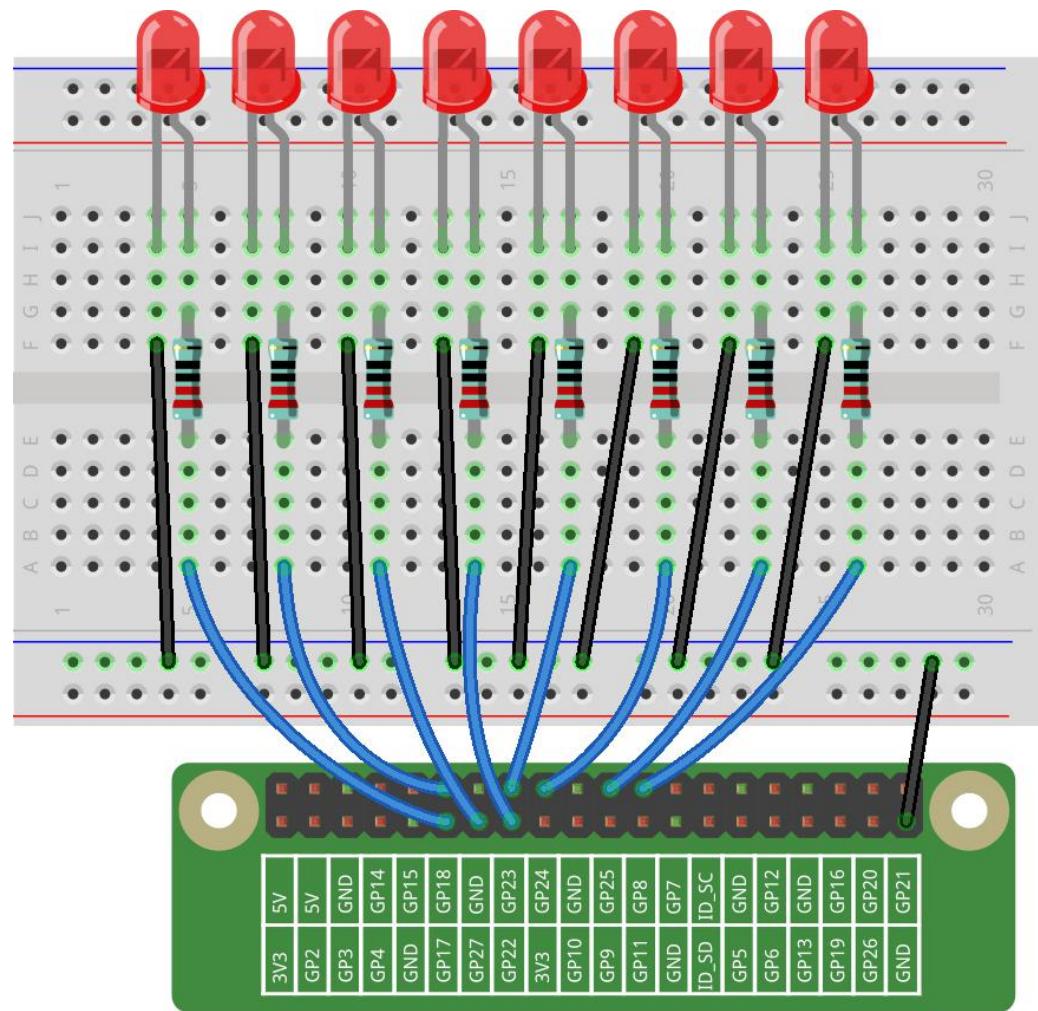
In this experiment, connect **220Ω** resistors to the anode (the longer pin of the LED) respectively, then the resistors to Pin **11, 12, 13, 15, 16, 18, 22** and **24** of Raspberry Pi, and connect the cathode (the short pin) of the LEDs to **GND**. We can see from the schematic diagram that the anode of LED connect to a current-limiting resistor and then to Raspberry Pi. Therefore, to turn on an LED, we need to make pins high level. This process can be realized by programming.

wiringPi	Physical	BCM
0	Pin11	17
1	Pin12	18
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24
6	Pin22	25
10	Pin24	8



## Build the Circuit

LEDs	Raspberry Pi
LED1	Pin 11
LED2	Pin 12
LED3	Pin 13
LED4	Pin 15
LED5	Pin 16
LED6	Pin 18
LED7	Pin 22
LED8	Pin 24
GND	GND



## ➤ For C Language Users:

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_2_FlowingLedLights
```

2. Compile the code.

```
gcc 2_FlowingLedLights.c -lwiringPi
```

**Note:** When using the **gcc** command, if you do not use **-o**, it will automatically output as **a.out**.

3. Run the executable file.

```
sudo ./a.out
```

Now, you will see these 8 LEDs are lit one by one from left to right, and then one by one from right to left.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. const int LedPin[]={0,1,2,3,4,5,6,10}; //Define 8 LED pin
5.
6. int main(void)
7. {
8.     // When initialize wiring failed, print message to screen
9.     if(wiringPiSetup() == -1){
```

```
10.     printf("setup wiringPi failed !");
11.     return 1;
12. }
13.
14. for(int j=0;j<8;j++)
15. {
16.     pinMode(LedPin[j], OUTPUT); // Set LedPin as output to write value to it.
17.     digitalWrite(LedPin[j], LOW);
18. }
19.
20. while(1){
21.     for(int i=0;i<8;i++)
22.     {
23.         // LED on
24.         digitalWrite(LedPin[i], HIGH);
25.         delay(100);
26.     }
27.     for(int i=7;i>-1;i--)
28.     {
29.         // LED off
30.         digitalWrite(LedPin[i], LOW);
31.         delay(100);
32.     }
33. }
```

34.

```
35.     return 0;  
36. }
```

## Code Explanation

```
4. const int LedPin[]={0,1,2,3,4,5,6,10};
```

Create an array, **LedPin** to define the eight LEDs then connect them to **GPIO0~GPIO6, GPIO10** respectively.

```
14.     for(int j=0;j<8;j++)  
15.     {  
16.         pinMode(LedPin[j], OUTPUT);  
17.         digitalWrite(LedPin[j], LOW);  
18.     }
```

Use a **for** loop to set all 8 pins connected to LEDs to **OUTPUT** mode and **LOW** level.

```
21.     for(int i=0;i<8;i++)  
22.     {  
23.         // LED on  
24.         digitalWrite(LedPin[i], HIGH);  
25.         delay(100);  
26.     }
```

Light up the LEDs in **GPIO0~6** and **GPIO10** successively. **i** increases progressively from **0** to **7**, LED0 to LED7 changes accordingly, making it like a flowing LED light from left to right.

```
27.     for(int i=7;i>-1;i--)
```

```
28.      {
29.          // LED off
30.          digitalWrite(LedPin[i], LOW);
31.          delay(100);
32.      }
```

Close the LEDs in GPIO0~6 and GPIO10 successively. **i** increases progressively from **7** to **0**, LED0 to LED7 changes accordingly, making it like a flowing LED light from right to left.

## ➤ For Python Language Users:

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 2_FlowingLed.py
```

Now, you will see these 8 LEDs are lit one by one from left to right, and then one by one from right to left.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. pins = [17,18,27,22,23,24,25,8]
5.
```

```
6. # Define a setup function for some setup
7. def setup():
8.     GPIO.setmode(GPIO.BCM)
9.     for i in range(0, 8, 1):
10.         GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)
11.
12. # Define a main function for main process
13. def main():
14.     while True:
15.         # print ('...LED ON')
16.         # Turn on LED
17.         for i in range(0, 8, 1):
18.             GPIO.output(pins[i], GPIO.HIGH)
19.             time.sleep(0.1)
20.
21.         # print ('LED OFF...')
22.         # Turn off LED
23.         for i in range(7, -1, -1):
24.             GPIO.output(pins[i], GPIO.LOW)
25.             time.sleep(0.1)
26.
27. # Define a destroy function for clean up everything after the script finished
28. def destroy():
29.     # Turn off LED
```

```
30.     for i in range(0, 8, 1):
31.         GPIO.output(pins[i], GPIO.LOW)
32.     # Release resource
33.     GPIO.cleanup()
34.
35.# If run this script directly, do:
36.if __name__ == '__main__':
37.    setup()
38.    try:
39.        main()
40.    # When 'Ctrl+C' is pressed, the child program
41.    # destroy() will be executed.
42.    except KeyboardInterrupt:
43.        destroy()
```

## Code Explanation

```
9.     for i in range(0, 8, 1):
10.         GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)
```

Use a **for** loop to set all 8 pins connected to LEDs to output mode and LOW level.

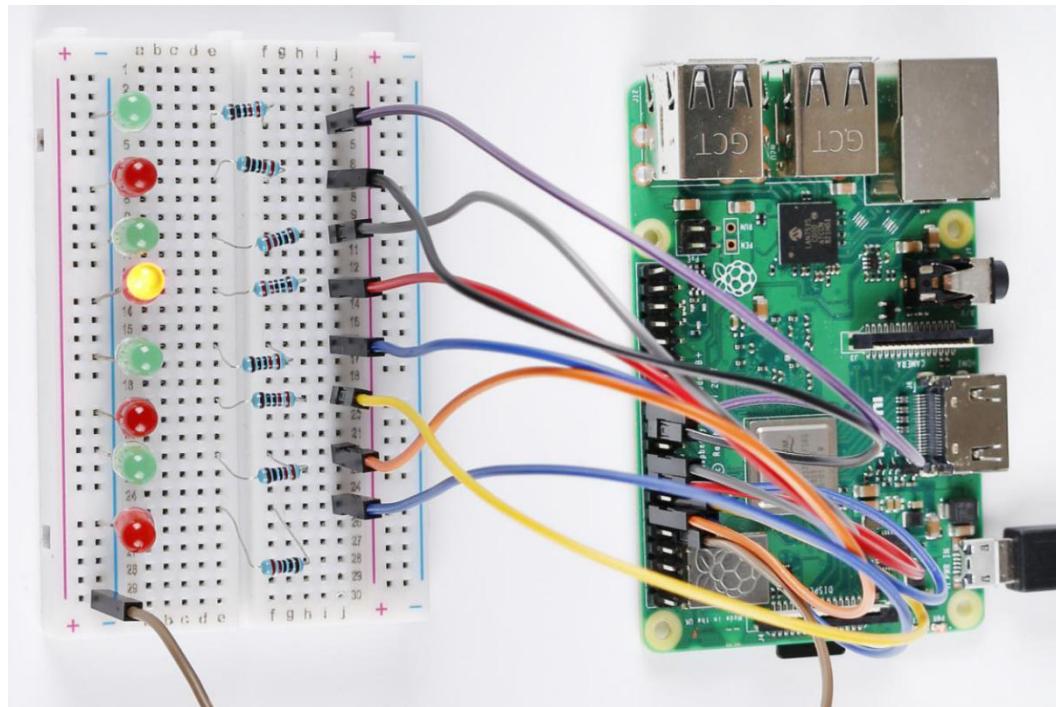
```
17.     for i in range(0, 8, 1):
18.         GPIO.output(pins[i], GPIO.HIGH)
19.         time.sleep(0.1)
```

Variable **i** increases progressively from **0** to **8**, increasing by 1 every time. Accordingly, set the pins in the array **pins[i]** to **HIGH** respectively to light up the LEDs and the lighting time is **0.1s**. Then, you will see 8 LEDs light up one by one.

```
23.     for i in range(7, -1, -1):  
24.         GPIO.output(pins[i], GPIO.LOW)  
25.         time.sleep(0.1)
```

Variable **i** decreases progressively from **7** to **-1**, decreasing by 1 every time. Then LED0~LED7 change accordingly, making it like a flowing LED light from right to left.

## Phenomenon Picture

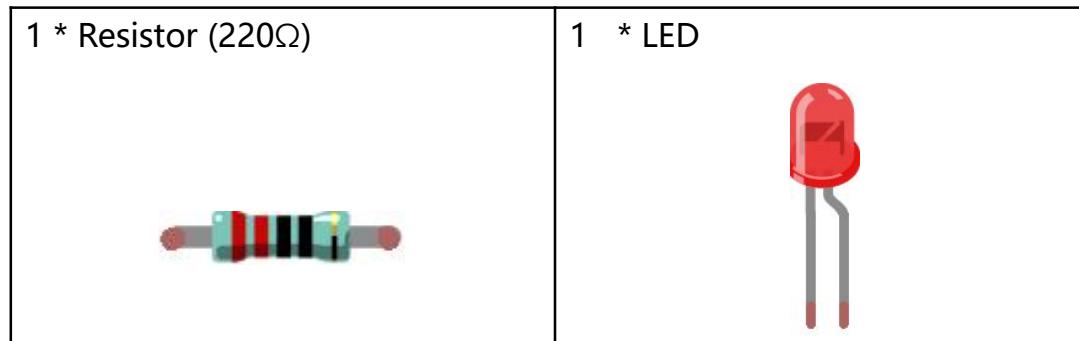


## Lesson 3 Breathing LED

### Introduction

In this lesson, we will try something interesting - gradually increase or decrease the luminance of an LED with PWM, just like breathing. So we give it a magical name - Breathing LED.

### Newly Added Components



### Principle

#### PWM

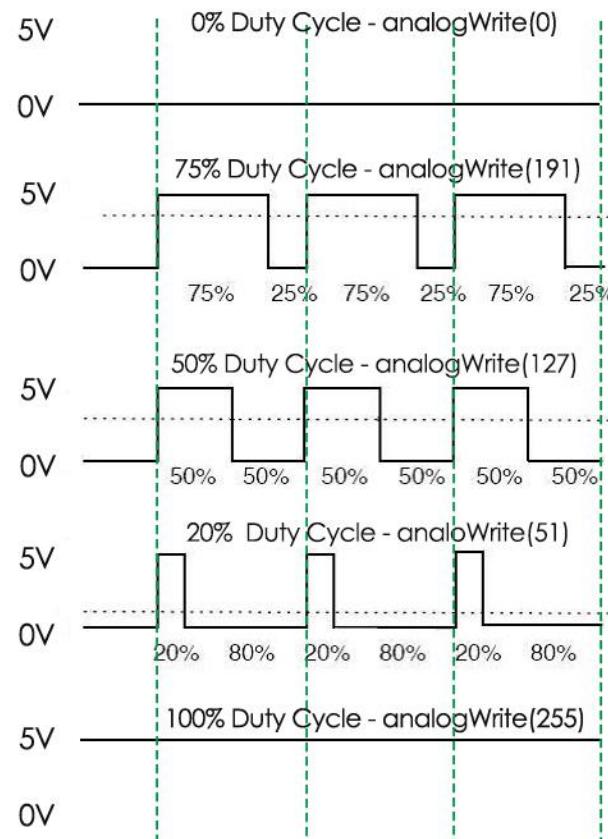
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you can change or modulate this width. If you repeat this on-off pattern fast enough with some device, an LED for example, the result would be like this: the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

## Duty Cycle

A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

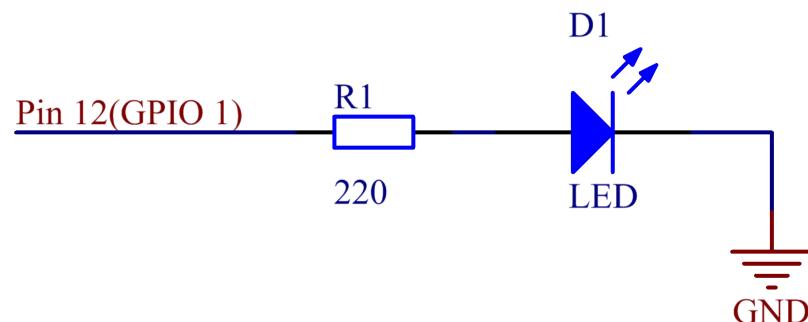
$$D=T/P \times 100\%$$

Where  $D$  is the duty cycle,  $T$  is the time the signal is active, and  $P$  is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.

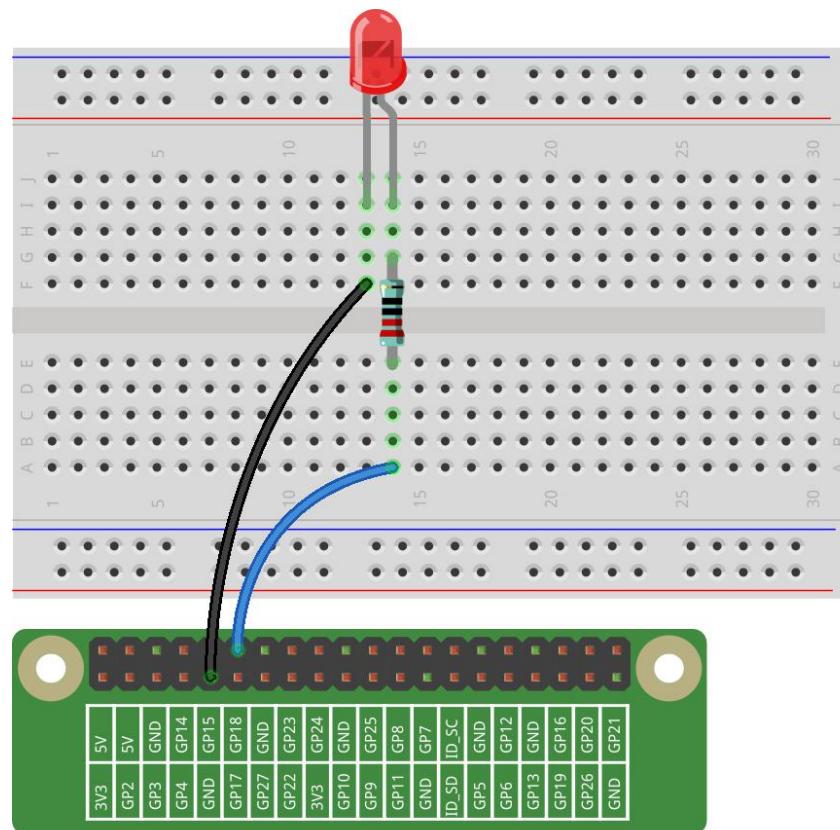


## Schematic Diagram

wiringPi	Physical	BCM
1	Pin12	18



## Build the Circuit



## ➤ For C Language Users:

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_3_BreathingLed
```

2. Compile the code.

```
gcc 3_BreathingLed.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

As the code runs, you can see that the brightness of the LED becomes stronger or weaker.

### Code

```
1. #include <stdio.h>
2. #include <wiringPi.h>
3. #include <softPwm.h>
4.
5. #define LedPin 1
6.
7. int main (void)
8. {
9. // When initialize wiring failed, print message to screen
10. if(wiringPiSetup() == -1){
11.     printf("setup wiringPi failed !");
```

```
12.         return 1;
13.     }
14.
15.     softPwmCreate(LedPin, 0, 100);
16.
17.     int i;
18.
19.     while(1) // loop forever
20.     {
21.         for(i=0;i<100;i++){ // i,as the value of pwm, increases progressively during 0-1024.
22.             softPwmWrite(LedPin, i);
23.             delay(10);
24.         }
25.
26.         for(i=100;i>=0;i--){
27.             softPwmWrite(LedPin, i);
28.             delay(10);
29.         }
30.     }
31.     return 0 ;
32. }
```

## Code Explanation

```
3. #include <softPwm.h>
```

WiringPi includes a software-driven **PWM** library of outputting a PWM signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is  $100\mu\text{S}$ . That combined with the default suggested range of 100 gives a PWM frequency of 100Hz. Within these limitations, control of a light/LED or a motor is very achievable.

```
15.     softPwmCreate(LedPin, 0, 100);
```

The function is to use software library to create a PWM pin, set its period between  $0x100\text{us}-100\times100\text{us}$ .

The prototype of the function `softPwmCreate(LedPinRed, 0, 100)` is as follows:

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

**pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.

**initialValue:** The initial pulse width is that initialValue times  $100\text{us}$ .

**pwmRange:** The period of PWM is that pwmRange times  $100\text{us}$ .

```
22.     softPwmWrite(LedPin, i);
```

The function is used to write the PWM value **i** to the **LedPin**.

The prototype of the function `softPwmWrite(LedPinBlue, b_val)` is as follows:

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

**pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.

**Value:** The pulse width of PWM is value times 100us. Note that value can only be less than pwmRange defined previously, if it is larger than pwmRange, the value will be given a fixed value, pwmRange.

```
23.         delay(10);
```

Wait for 10ms, interval time between the changes indicates the speed of breathing.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 3_BreathingLed.py
```

As the code runs, you can see that the brightness of the LED becomes stronger or weaker.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. LedPin = 18
5.
6. def setup():
7.     global pLed
8.     GPIO.setmode(GPIO.BCM)
```

```
9.     GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
10.    pLed = GPIO.PWM(LedPin, 1000)
11.    pLed.start(0)
12.
13. def main():
14.     # Set increase/decrease step
15.     step =2
16.     delay = 0.05
17.     while True:
18.         # Increase duty cycle from 0 to 100
19.         for dc in range(0, 101, step):
20.             pLed.ChangeDutyCycle(dc)
21.             print (' ++ Duty cycle: %s'%dc)
22.             time.sleep(delay)
23.             time.sleep(1)
24.
25.         # decrease duty cycle from 100 to 0
26.         for dc in range(100, -1, -step):
27.             # Change duty cycle to dc
28.             pLed.ChangeDutyCycle(dc)
29.             print (' -- Duty cycle: %s'%dc)
30.             time.sleep(delay)
31.             time.sleep(1)
32.
```

```
33. def destroy():
34.     # Stop pLed
35.     pLed.stop()
36.     # Turn off LED
37.     GPIO.output(LedPin, GPIO.LOW)
38.     # Release resource
39.     GPIO.cleanup()
40.
41. # If run this script directly, do:
42. if __name__ == '__main__':
43.     setup()
44.     try:
45.         main()
46.     # When 'Ctrl+C' is pressed, the child program
47.     # destroy() will be executed.
48.     except KeyboardInterrupt:
49.         destroy()
```

## Code Explanation

```
10.     pLed = GPIO.PWM(LedPin, 1000)
```

To create a PWM instance. Set **pLed** as pwm output and frequency to **1KHz**.

```
11.     pLed.start(0)
```

Set **pLed** begin with value **0**.

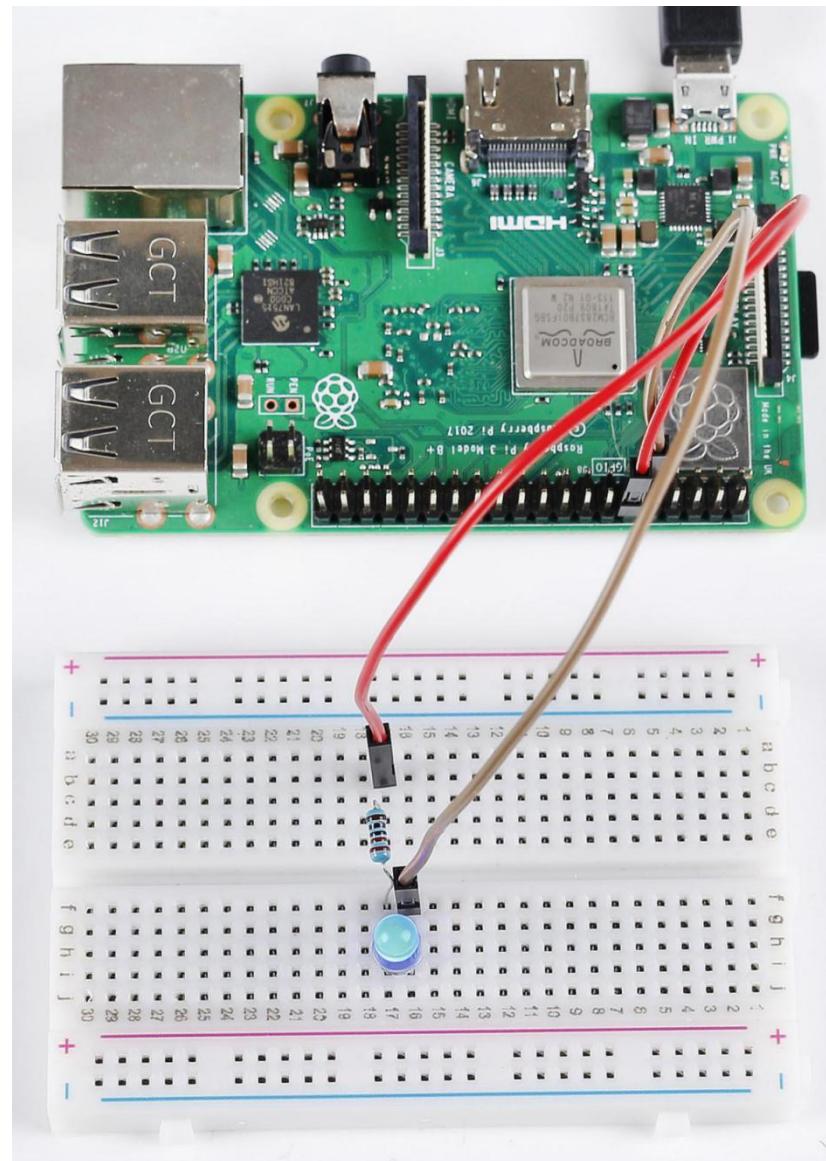
```
19.     for dc in range(0, 101, step):  
20.         # Change duty cycle to dc  
21.         pLed.ChangeDutyCycle(dc)  
22.         print (' ++ Duty cycle: %s'%dc)  
23.         time.sleep(delay)
```

Increase the duty cycle by 2 at a time, from **0** to **101**, and you'll see the LED getting brighter and brighter.

```
26.     for dc in range(100, -1, -step):  
27.         pLed.ChangeDutyCycle(dc)  
28.         print (' -- Duty cycle: %s'%dc)  
29.         time.sleep(delay)
```

Similarly, when the duty cycle is reduced by 2 from **100** to **-1**, the LED brightness will be dimmer and dimmer.

## Phenomenon Picture

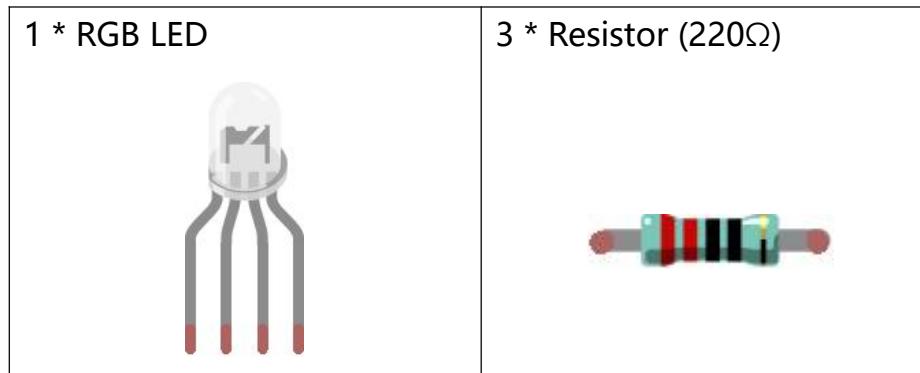


## Lesson 4 RGB LED

### Introduction

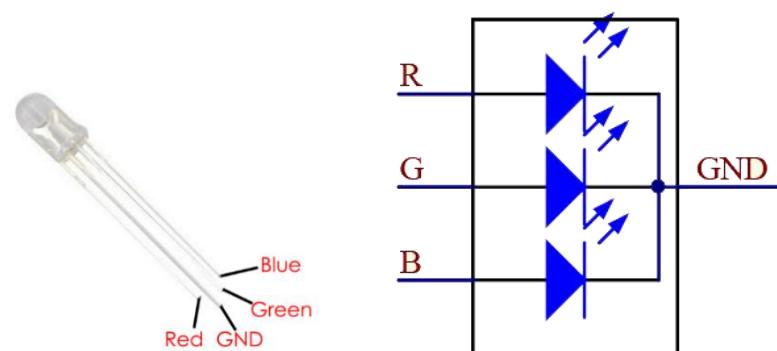
Previously we've used the PWM technology to control an LED's brightness. In this lesson, we will use it to control an RGB LED to flash various kinds of colors.

### Newly Added Components



### Principle

#### RGB LED

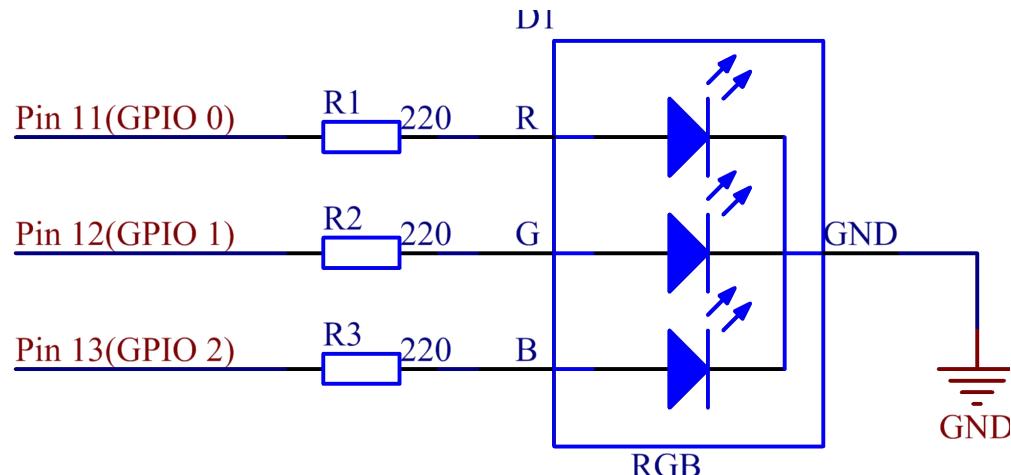


The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with the hardware PWM of Raspberry Pi. Fortunately, the **softPwm** library simulates PWM (softPwm) by programming. You only need to include the header file **softPwm.h** (for C language users), and then call the API it provides to easily control the RGB LED by multi-channel PWM output, so as to display all kinds of color.

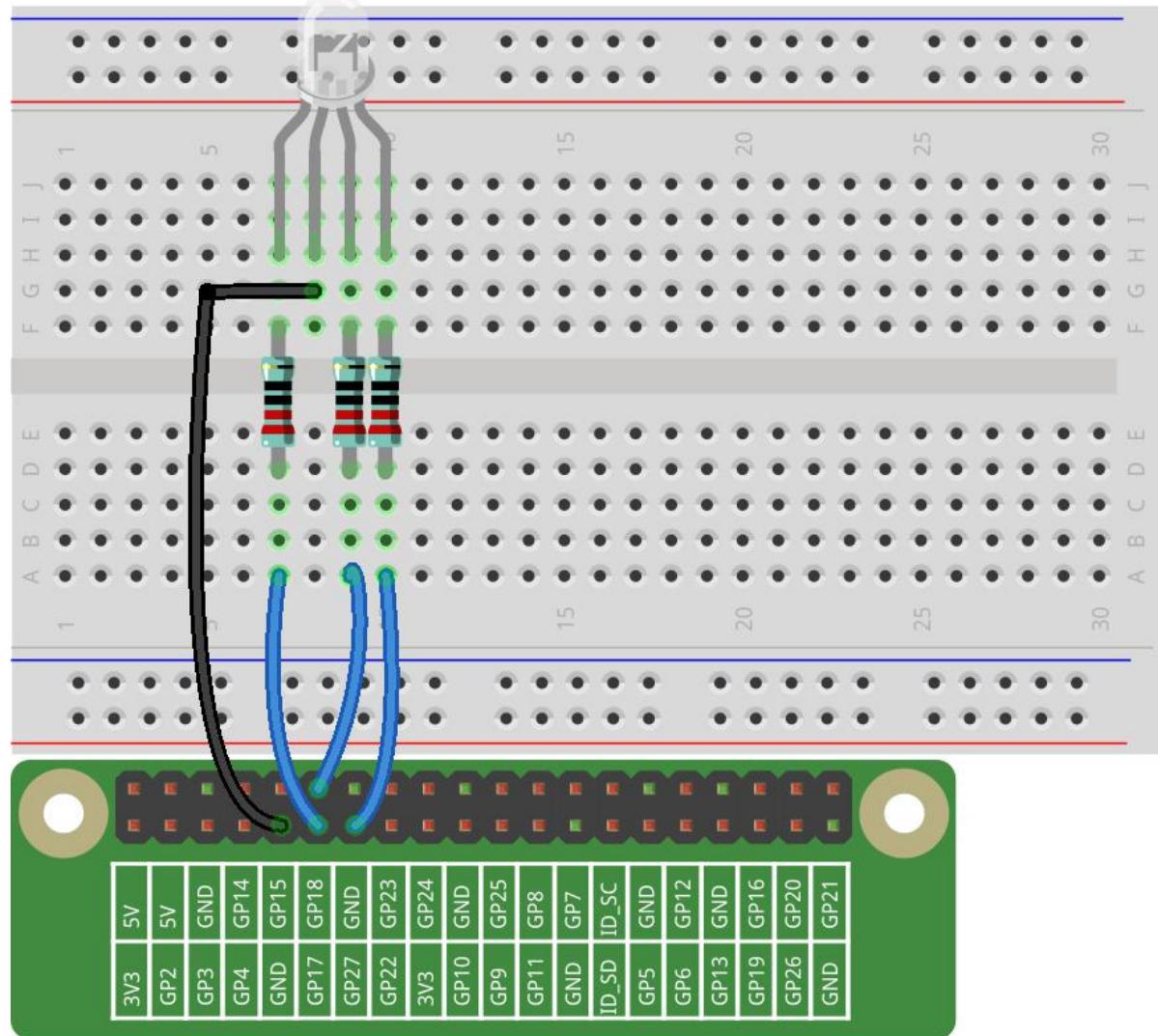
## Schematic Diagram

After connecting the pins of R, G, and B to a current limiting resistor, connect them to the pin 11, pin 12, and pin 13 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.

wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_4_RGBLed
```

2. Compile the code.

```
gcc 4_rgbLed.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

After the code runs, you will see that RGB displays red, green, blue, yellow, pink, and cyan.

### Code

```
1. #include <wiringPi.h>
2. #include <softPwm.h>
3. #include <stdio.h>
4.
5. #define uchar unsigned char
6.
7. #define LedPinRed    0
8. #define LedPinGreen   1
9. #define LedPinBlue   2
10.
```

```
11.// define function used for initializing I/O port to output for pwm.  
12.void ledInit(void) {  
13.    softPwmCreate(LedPinRed, 0, 100);  
14.    softPwmCreate(LedPinGreen, 0, 100);  
15.    softPwmCreate(LedPinBlue, 0, 100);  
16.}  
17.  
18.void ledColorSet(uchar r_val, uchar g_val, uchar b_val) {  
19.    softPwmWrite(LedPinRed, r_val);  
20.    softPwmWrite(LedPinGreen, g_val);  
21.    softPwmWrite(LedPinBlue, b_val);  
22.}  
23.  
24.int main(void) {  
25.    if(wiringPiSetup() == -1){ //when initialize wiring failed, printf message to screen  
26.        printf("setup wiringPi failed !");  
27.        return 1;  
28.    }  
29.  
30.    ledInit();  
31.  
32.    while(1){  
33.        printf("Red\n");  
34.        ledColorSet(0xff,0x00,0x00); //red
```

```
35.     delay(500);
36.     printf("Green\n");
37.     ledColorSet(0x00,0xff,0x00); //green
38.     delay(500);
39.     printf("Blue\n");
40.     ledColorSet(0x00,0x00,0xff); //blue
41.     delay(500);
42.     printf("Yellow\n");
43.     ledColorSet(0xff,0xff,0x00); //yellow
44.     delay(500);
45.     printf("Purple\n");
46.     ledColorSet(0xff,0x00,0xff); //purple
47.     delay(500);
48.     printf("Cyan\n");
49.     ledColorSet(0xc0,0xff,0x3e); //cyan
50.     delay(500);
51. }
52.
53. return 0;
54. }
```

## Code Explanation

```
12. void ledInit(void){  
13.     softPwmCreate(LedPinRed, 0, 100);  
14.     softPwmCreate(LedPinGreen,0, 100);  
15.     softPwmCreate(LedPinBlue, 0, 100);  
16. }
```

Create a function to set the **LedPinRed**, **LedPinGreen** and **LedPinBlue** as PWM pins, then set their period between 0x100us-100x100us.

The prototype of the function `softPwmCreate(LedPinRed, 0, 100)` is as follows:

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

**pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.

**initialValue:** The initial pulse width is that initialValue times100us.

**pwmRange:** the period of PWM is that pwmRange times100us.

```
18. void ledColorSet(uchar r_val, uchar g_val, uchar b_val){  
19.     softPwmWrite(LedPinRed, r_val);  
20.     softPwmWrite(LedPinGreen, g_val);  
21.     softPwmWrite(LedPinBlue, b_val);  
22. }
```

This function is to set the colors of the LED. Using RGB, the formal parameter **r\_val** represents the luminance of the red one, **g\_val** of the green one, **b\_val** of the blue one.

The prototype of the function `softPwmWrite(LedPinBlue, b_val)` is as follows:

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

**pin:** Any GPIO pin of Raspberry Pi can be set as a PWM pin.

**Value:** The pulse width of PWM is value times 100us. Note that **value** can only be less than **pwmRange** defined previously, if it is larger than pwmRange, the **value** will be given a fixed value, pwmRange.

```
30.    ledInit();
```

Call the **ledInit()** function in the **main** function to initialize the LED.

```
34.    ledColorSet(0xff,0x00,0x00); //red
```

Call the function defined before. Write **0xff** into LedPinRed and **0x00** into LedPinGreen and LedPinBlue. Only the Red LED lights up after running this code. If you want to light up LEDs in other colors, just modify the parameters.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 4_rgbLed.py
```

After the code runs, you will see that RGB displays red, green, blue, yellow, pink, and cyan.

### Code

```
1. import RPi.GPIO as GPIO
```

```
2. import time
3.
4. COLOR = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]
5. pins = {'Red':17, 'Green':18, 'Blue':27}
6.
7. def setup():
8.     global p_R, p_G, p_B
9.     GPIO.setmode(GPIO.BCM)
10.    for i in pins:
11.        GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)
12.
13.    # Set all led as pwm channel and frequece to 2KHz
14.    p_R = GPIO.PWM(pins['Red'], 2000)
15.    p_G = GPIO.PWM(pins['Green'], 2000)
16.    p_B = GPIO.PWM(pins['Blue'], 2000)
17.
18.    # Set all begin with value 0
19.    p_R.start(0)
20.    p_G.start(0)
21.    p_B.start(0)
22.
23. def MAP(x, in_min, in_max, out_min, out_max):
24.     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
25.
```

```
26. def setColor(color):  
27.     # Devide colors from 'color' veriable  
28.     R_val = (color & 0xFF0000) >> 16  
29.     G_val = (color & 0x00FF00) >> 8  
30.     B_val = (color & 0x0000FF) >> 0  
31.  
32.     # Map color value from 0~255 to 0~100  
33.     R_val = MAP(R_val, 0, 255, 0, 100)  
34.     G_val = MAP(G_val, 0, 255, 0, 100)  
35.     B_val = MAP(B_val, 0, 255, 0, 100)  
36.  
37.     # Change the colors  
38.     p_R.ChangeDutyCycle(R_val)  
39.     p_G.ChangeDutyCycle(G_val)  
40.     p_B.ChangeDutyCycle(B_val)  
41.  
42.     print ("color_msg: R_val = %s,  G_val = %s, B_val = %s"%(R_val, G_val, B_val))  
43.  
44. def main():  
45.     while True:  
46.         for color in COLOR:  
47.             setColor(color)  
48.             time.sleep(0.5)  
49.
```

```
50. def destroy():
51.     # Stop all pwm channel
52.     p_R.stop()
53.     p_G.stop()
54.     p_B.stop()
55.     # Turn off all LEDs
56.     GPIO.output(pins, GPIO.LOW)
57.     # Release resource
58.     GPIO.cleanup()
59.
60. # If run this script directly, do:
61. if __name__ == '__main__':
62.     setup()
63.     try:
64.         main()
65.     # When 'Ctrl+C' is pressed, the child program
66.     # destroy() will be executed.
67.     except KeyboardInterrupt:
68.         destroy()
```

## Code Explanation

```
14.     p_R = GPIO.PWM(pins['Red'], 2000)
```

This statement is used to set the pin to a specific PWM frequency, in this case **2000Hz**.

```
23. def MAP(x, in_min, in_max, out_min, out_max):
```

```
24.     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
```

Define a **map** function for mapping values. For instance,  $x=50$ ,  $in\_min=0$ ,  $in\_max=255$ ,  $out\_min=0$ ,  $out\_max=100$ . After the map function mapping, it returns  $(50-0) * (100-0)/(255-0) +0=19.6$ , meaning that 50 in 0-255 equals 19.6 in 0-100.

```
26. def setColor(color):
```

```
27.     R_val = (color & 0xFF0000) >> 16
```

```
28.     G_val = (color & 0x00FF00) >> 8
```

```
29.     B_val = (color & 0x0000FF) >> 0
```

Create a **setColor()** function to assign different value to the three variables: R\_val, G\_val, B\_val. Input color should be hexadecimal with red value, blue value, green value. Assign the first two values of the hexadecimal to R\_val, the middle two to G\_val, assign the last two values to B\_val, please refer to the shift operation of the hexadecimal for details. For example, color=0xFF00FF, then R\_val=(0xFF00FF & 0xFF0000)>>16 = 0xFF, G\_val = 0x00, B\_val=0xFF.

```
32.     # Map color value from 0~255 to 0~100
```

```
33.     R_val = MAP(R_val, 0, 255, 0, 100)
```

```
34.     G_val = MAP(G_val, 0, 255, 0, 100)
```

```
35.     B_val = MAP(B_val, 0, 255, 0, 100)
```

```
36.
```

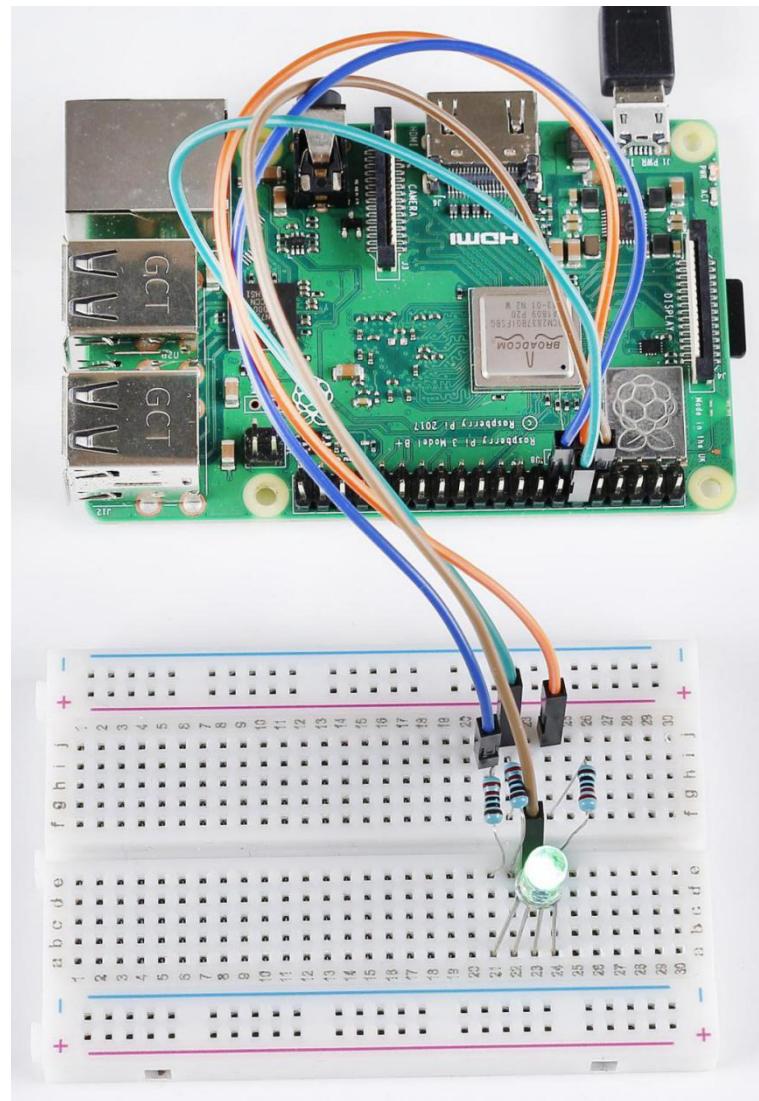
```
37. # Change the colors  
38. p_R.ChangeDutyCycle(R_val)  
39. p_G.ChangeDutyCycle(G_val)  
40. p_B.ChangeDutyCycle(B_val)
```

Map the RGB value from **0-255** to **0-100**. After that, get a value. Then set it to be the duty cycle of R\_val, G\_val and B\_val, and the RGB LED displays corresponding colors.

```
46. for color in COLOR:  
47.     setColor(color)  
48.     time.sleep(0.5)
```

Assign every item in the **COLOR** list to the color respectively and change the color of the RGB LED via the **setColor()** function.

## Phenomenon Picture

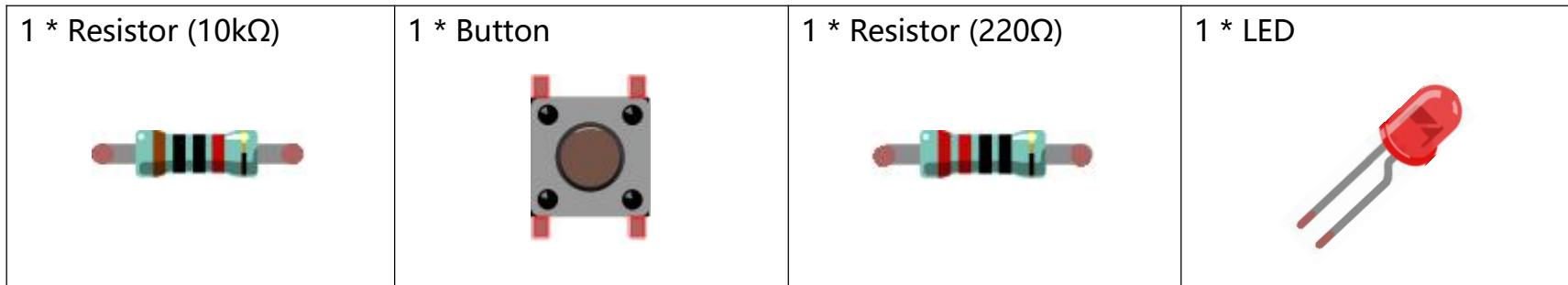


# Lesson 5 Controlling LED by Button

## Introduction

In this lesson, we will learn how to turn an LED on or off by a button.

## Newly Added Components

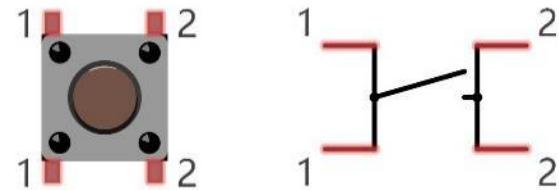


## Principle

### Button

Button is a common component used to control electronic devices. It is usually used as switch to connect or break circuits. Although buttons come in a variety of sizes and shapes, the one used here is a 6mm mini-button as shown in the following pictures.

Two pins on the same side are connected, which is shown below:



This following symbol represents a button in circuit.

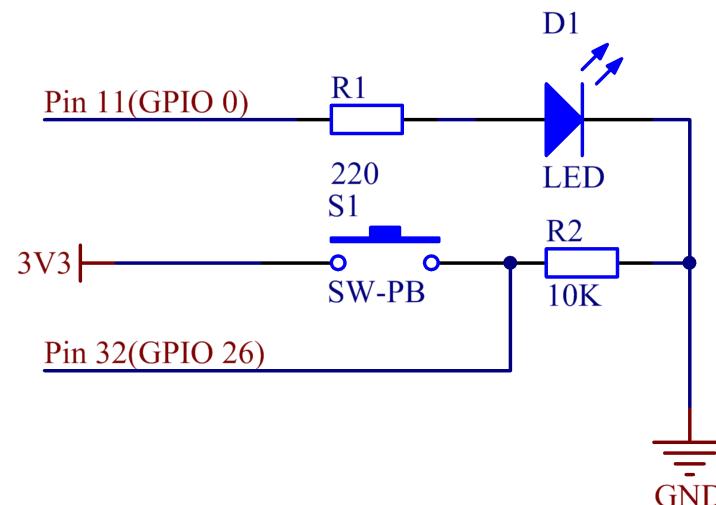


When the button is pressed, the 4 pins are connected, thus closing the circuit.

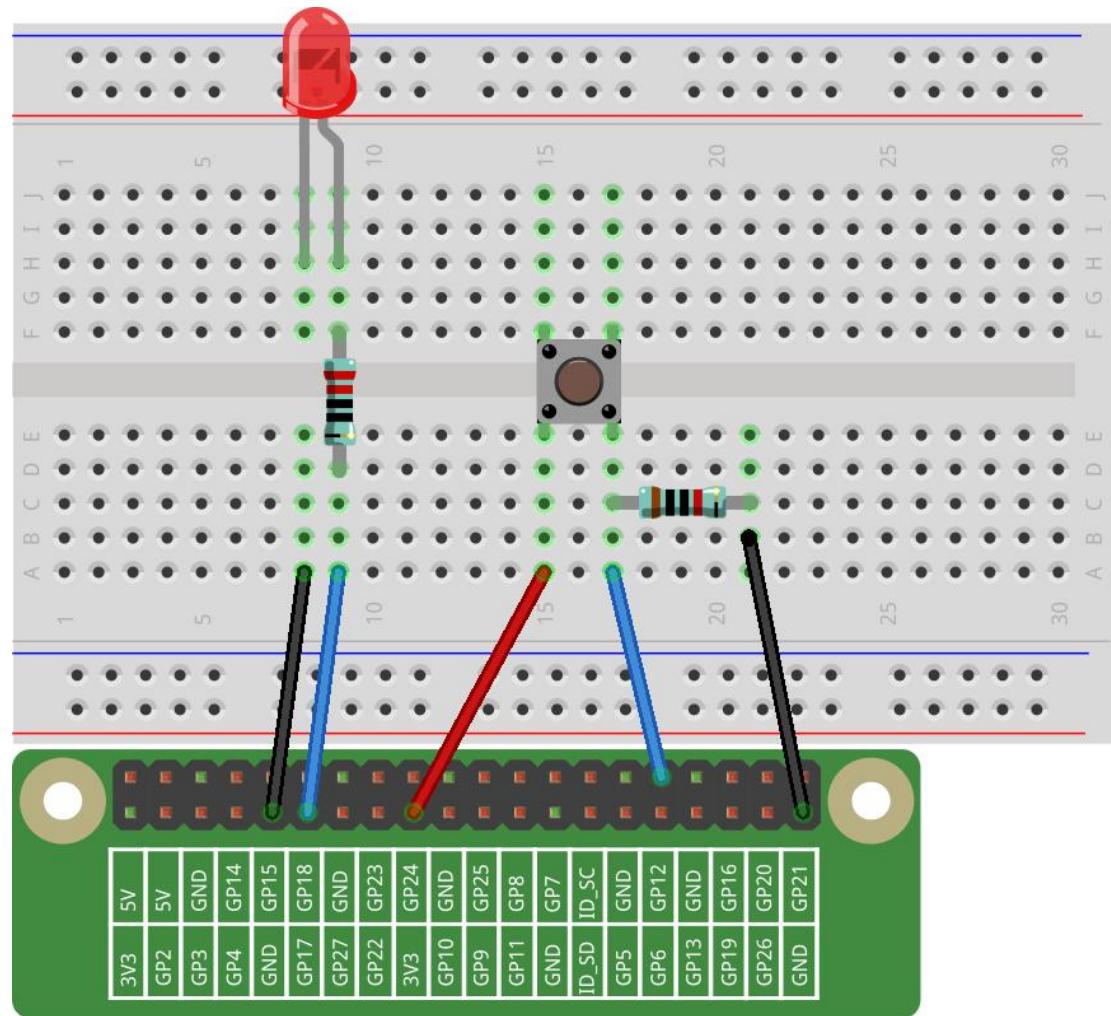
## Schematic Diagram

When the button is pressed once, pin 32 is 3.3V (HIGH). Set the pin 11(integrated with an LED) as high level by programming at the same time. Then press the button again and set pin 11 to Low. So we will see the LED light on and off alternately as the button is pressed many times.

wiringPi	Physical	BCM
0	Pin 11	17
26	Pin 32	12



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_5_Controling_Led_by_Button
```

2. Compile the code.

```
gcc 5_Button.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

When you press the button for the first time, the LED lights up. When the button is pressed again, the LED lights off.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define LedPin      0
5. #define ButtonPin   26
6. int state = 0;
7.
8. int main(void){
9.     // When initialize wiring failed, print message to screen
10.    if(wiringPiSetup() == -1){
```

```
11.     printf("setup wiringPi failed !");
12.     return 1;
13. }
14.
15. pinMode(LedPin, OUTPUT);
16. pinMode(ButtonPin, INPUT);
17. pullUpDnControl(ButtonPin, PUD_DOWN);
18.
19. while(1){
20.     // Indicate that button has pressed down
21.     if(digitalRead(ButtonPin) == 1)
22.     {
23.         delay(10);
24.         if(digitalRead(ButtonPin) == 1)
25.         {
26.             state++;
27.             if(state%2 == 1)
28.             {
29.                 digitalWrite(LedPin,HIGH);
30.                 delay(100);
31.             }
32.             if(state%2 == 0)
33.             {
34.                 digitalWrite(LedPin,LOW);
```

```
35.             delay(100);
36.         }
37.     }
38. }
39. }
40. return 0;
41. }
```

## Code Explanation

```
6. int state = 0;
```

Define a variable **state** to record the number of times it is pressed and the initial number of times is **0**.

```
15. pinMode(LedPin, OUTPUT);
16. pinMode(ButtonPin, INPUT);
```

Set the LedPin to **OUTPUT** mode, ButtonPin to **INPUT** mode.

```
17. pullUpDnControl(ButtonPin, PUD_DOWN);
```

When the button is not pressed, ButtonPin is in suspension at which time the read value is changing. To enable ButtonPin to output a stable low level, **PUD\_DOWN** is added to the code, keeping ButtonPin at the forced pull-down state till the button is pressed.

```
21. if(digitalRead(ButtonPin) == 1)
22. {
23.     delay(10);
24.     if(digitalRead(ButtonPin) == 1)
```

25.           {

Usually the buttons we use are mechanical buttons, so in the process of pressing down and releasing, there will be no direct change from 0 to 1, but will be more than 10ms of level jitter. In order to ensure that the program only responds to the button once when it is closed or broken, the jitter elimination of the button must be carried out. An **if** function is used to detect whether the button is pressed. When the signal of the button is pressed is detected, a delay of 10ms is used to eliminate the possibility of false judgment, and another **if** function is used to detect again. If both **if** conditions are met, confirm that it is a button press, and then execute the program in the **if**.

26.                 state ++;

If the button is pressed, the number of times it is pressed is increased by one. (**state ++** is the same as **state = state+1**).

```
27.                 if(state%2 == 1)  
28.                     {  
29.                         digitalWrite(LedPin,HIGH);  
30.                         delay(100);  
31.                 }
```

% is a modulo operator in C language; state%2 is that state is divided by 2 to return the remainder. If state=17, then state%2 =1. Here, determine whether state%2 is equal to 1. If it is, it means that the number of times of pressing the button is a singular number, and then turn on the LED.

```
32.                 if(state%2 == 0)  
33.                     {  
34.                         digitalWrite(LedPin,LOW);
```

```
35.           delay(100);  
36.       }
```

Here, judge whether state%2 is equal to 0. If so, it means that the number of times the button is pressed is an even number, and then turn off the LED.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 5_Button.py
```

When you press the button for the first time, the LED lights up. When the button is pressed again, the LED lights off.

### Code

```
1. import RPi.GPIO as GPIO  
2. import time  
3.  
4. LedPin = 17  
5. BtnPin = 12  
6. Led_status = False  
7.  
8. # Define a setup function for some setup
```

```
9. def setup():
10.     GPIO.setmode(GPIO.BCM)
11.     GPIO.setup(BtnPin, GPIO.IN)
12.     GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
13.     GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
14.
15. # Define a callback function for button callback
16. def swLed(ev=None):
17.     global Led_status
18.     Led_status = not Led_status
19.     GPIO.output(LedPin, Led_status)
20.
21. # Define a main function for main process
22. def main():
23.     while True:
24.         # Don't do anything.
25.         time.sleep(1)
26.
27. # Define a destroy function for clean up everything after
28. # the script finished
29. def destroy():
30.     # Turn off LED
31.     GPIO.output(LedPin, GPIO.LOW)
32.     # Release resource
```

```
33.     GPIO.cleanup()
34.
35. # If run this script directly, do:
36. if __name__ == '__main__':
37.     setup()
38.     try:
39.         main()
40.     # When 'Ctrl+C' is pressed, the child program
41.     # destroy() will be executed.
42.     except KeyboardInterrupt:
43.         destroy()
```

## Code Explanation

```
6. Led_status = False
```

Set a variable **Led\_status** to record the current status of the LED; when **Led\_status** is **True**, it indicates that the current lamp is in bright state; when **Led\_status** is **False**, it means that the light is off.

```
11.     GPIO.setup(BtnPin, GPIO.IN)
```

Set **BtnPin** as input mode to read the state of the button to determine whether to execute the corresponding program. Note that when **GPIO.setup** sets the pin to input mode, then there is no need to set the initial value.

```
12.     GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
```

Specify an initial value for your output channel. Here the LED is the output component, so we set **LedPin** to **GPIO.OUT** mode. Then initialize the state of LED to **GPIO.LOW** which means that the light is off.

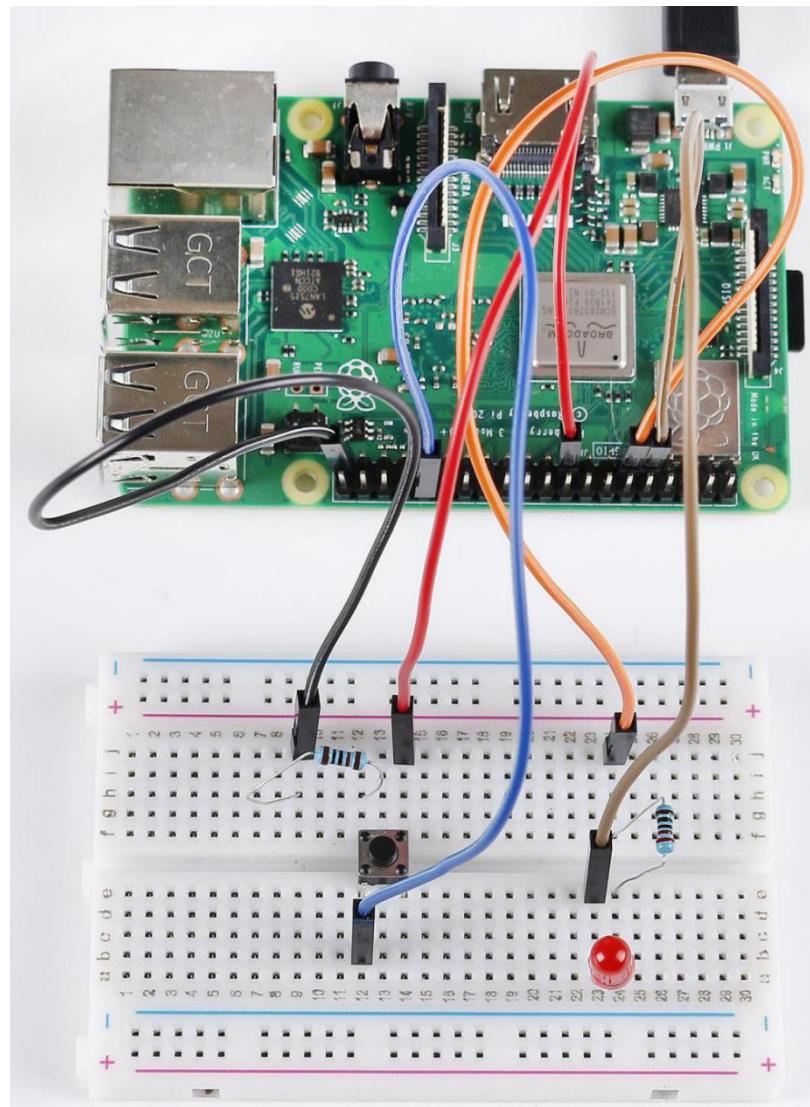
```
13. GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

The **event\_detected()** function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy working on other things. Set up a falling detect on BtnPin, when the BtnPin pin is detected to change from high level to low level, **swLed** function is called.

```
13. def swLed(ev=None):  
14.     global Led_status  
15.     Led_status = not Led_status  
16.     GPIO.output(LedPin, Led_status)
```

RPi.GPIO runs a second thread for callback functions. This means that callback functions can be run at the same time as your main program, in immediate response to an edge. Define a callback function for button callback, execute the function after the callback of the interrupt. When this function is executed, the state of the LED is firstly reversed(If **True**, make it **False**, and vice versa). Then input the function to LedPin. And "**ev = None**" means that if no parameter is passed when calling **swLed**, take **None** as the default value of **ev**.

## Phenomenon Picture

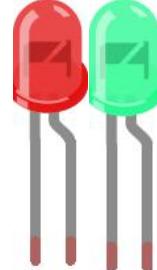


## Lesson 6 Tilt Switch

### Introduction

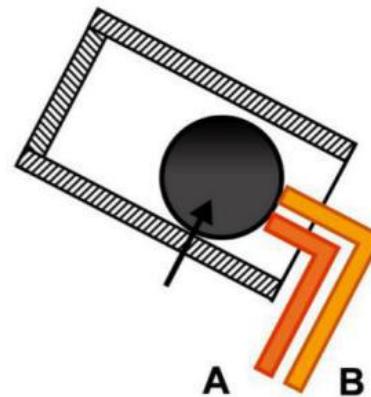
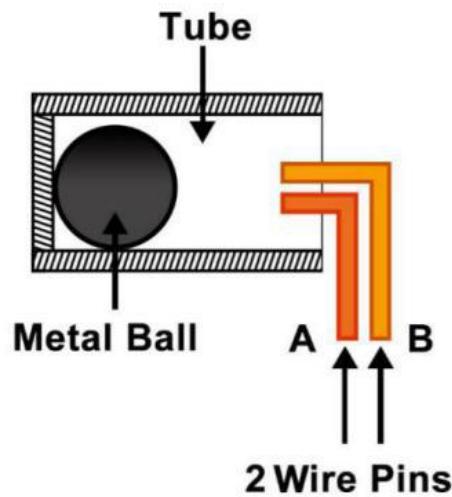
In this lesson, we'll learn a new switch component, tilt switch. Here we apply two LEDs to indicate the current state of tilt switch. You can also use this kind of switch to make a sense light with the clamshell box.

### Newly Added Components

1 * Tilt Switch	2 * Resistor (220Ω)	2 * LED
		

## Principle

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.



**Ball Slides Down to Connect Both Contacts**



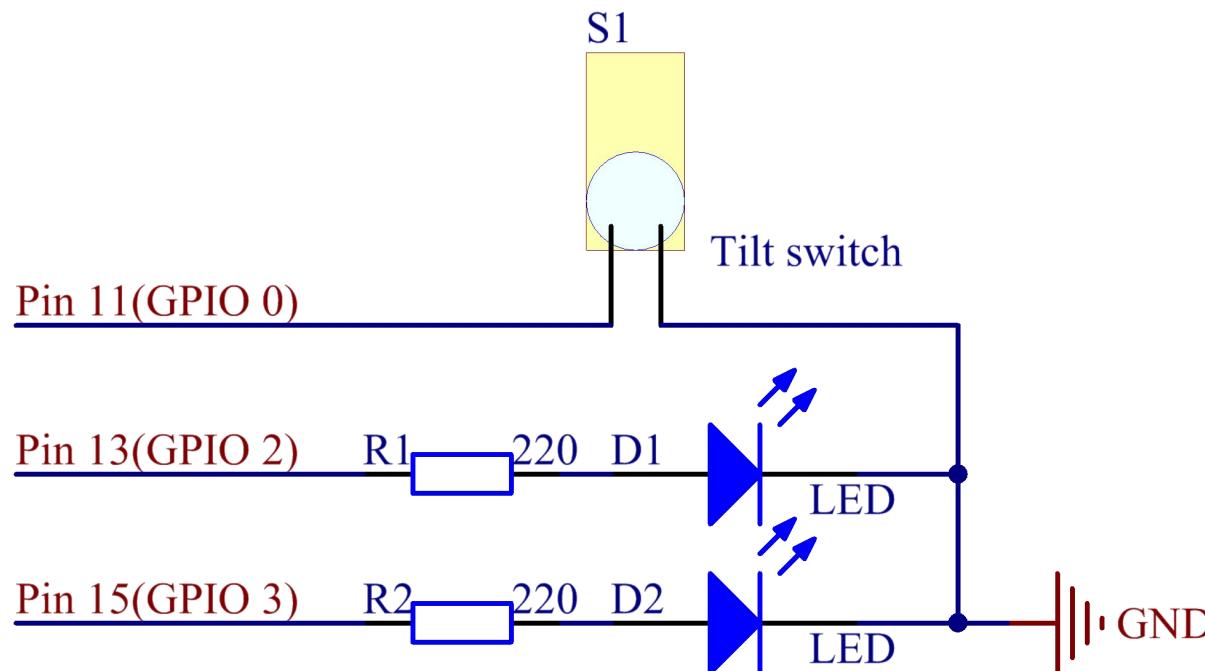
**Schematic Equivalent**



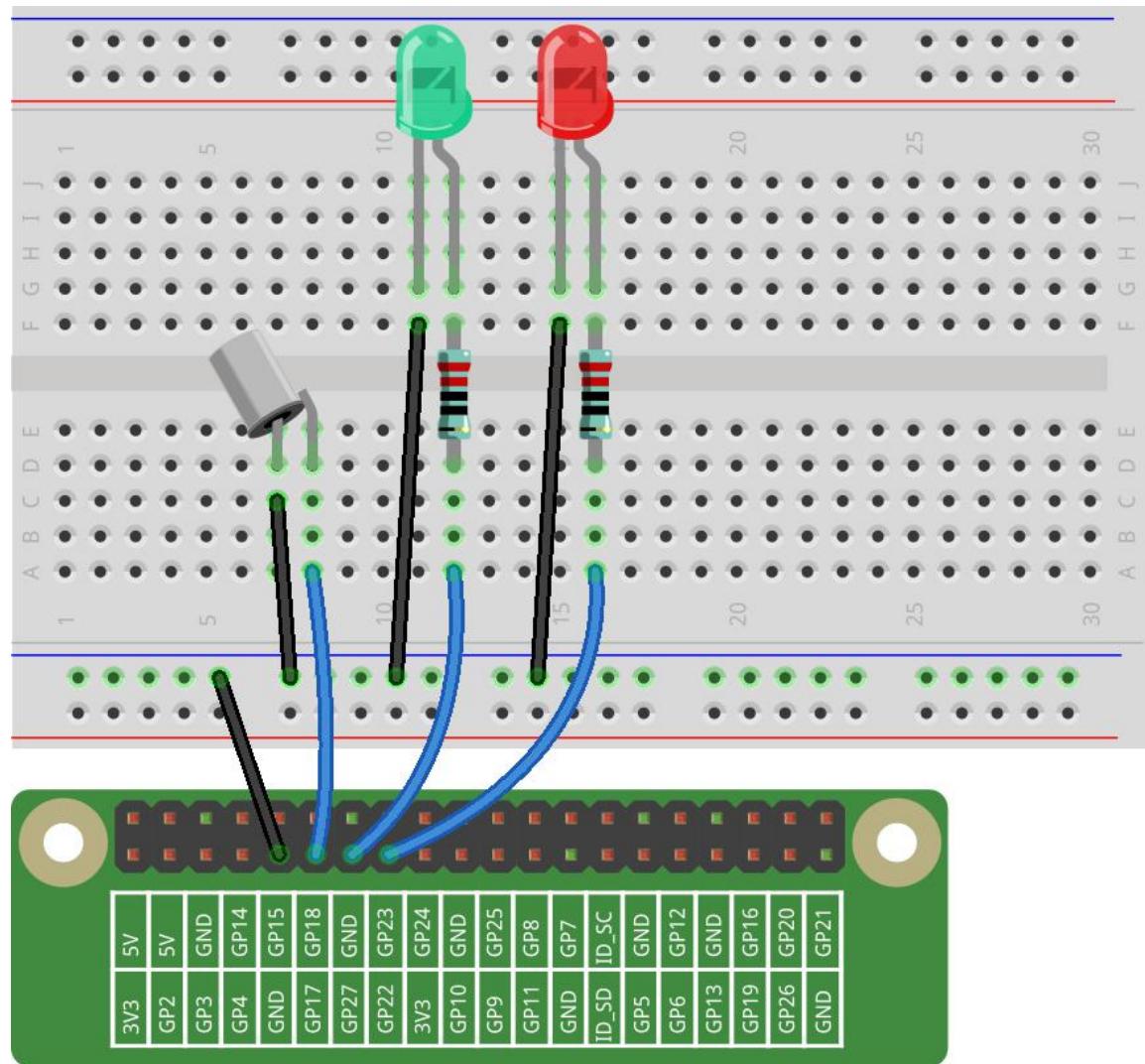
**Schematic Equivalent**

## Schematic Diagram

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_6_TiltSwitch
```

2. Compile the code.

```
gcc 6_Tilt.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

When the tilt switch is level, the green LED turns on. If you tilt the switch, the red LED will turn on.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define TiltPin      0
5. #define Gpin         2
6. #define Rpin         3
7.
8. int main(void)
9. {
10.    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
```

```
11.     printf("setup wiringPi failed !");
12.     return 1;
13. }
14.
15. pinMode(TiltPin, INPUT);
16. pinMode(Gpin, OUTPUT, );
17. pinMode(Rpin, OUTPUT);
18.
19. while(1){
20.
21.     if(1 == digitalRead(TiltPin)){
22.         delay(10);
23.         if(1 == digitalRead(TiltPin)){
24.             digitalWrite(Rpin, HIGH);
25.             digitalWrite(Gpin, LOW);
26.             printf("RED\n");
27.         }
28.     }
29.     else if(0 == digitalRead(TiltPin)){
30.         delay(10);
31.         if(0 == digitalRead(TiltPin)){
32.             while(!digitalRead(TiltPin));
33.             digitalWrite(Rpin, LOW);
34.             digitalWrite(Gpin, HIGH);
```

```
35.             printf("GREEN\n");
36.         }
37.     }
38. }
39. return 0;
40. }
```

## Code Explanation

```
15. pinMode(TiltPin, INPUT);
16. pinMode(Gpin, OUTPUT);
17. pinMode(Rpin, OUTPUT);
```

Initialize pins, then set the pin of tilt switch to **INPUT** mode, and LEDs to **OUTPUT** mode.

```
21. if(1 == digitalRead(TiltPin)){
```

It is used to judge whether the tilt switch is tilted or not. The value of **TiltPin** is firstly read, if it is equal to **1**, the codes inside the **if()** statement run; otherwise, the codes of **if** are skipped.

```
21. if(1 == digitalRead(TiltPin)){
22.     delay(10);
23.     if(1 == digitalRead(TiltPin)){
24.         digitalWrite(Rpin, HIGH);
25.         digitalWrite(Gpin, LOW);
26.         printf("RED\n");
27.     }
28. }
```

When the tilt is tilted, the tilt switch is on; the Raspberry Pi reads a high level at the tilt pin, so the red LED is on and green LED off.

```
29.     else if(0 == digitalRead(TiltPin)){
30.         delay(10);
31.         if(0 == digitalRead(TiltPin)){
32.             while(!digitalRead(TiltPin));
33.             digitalWrite(Rpin, LOW);
34.             digitalWrite(Gpin, HIGH);
35.             printf("GREEN\n");
36.         }
37.     }
```

When the tilt is level, the tilt switch is off; the Raspberry Pi reads a low level at the tilt pin, so the red LED is off and green LED on.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 6_Tilt.py
```

When the tilt switch is level, the green LED turns on. If you tilt the switch, the red LED will turn on.

### Code

```
1. import RPi.GPIO as GPIO
2.
3. TiltPin = 17
4. Gpin    = 27
5. Rpin    = 22
6.
7. def setup():
8.     GPIO.setmode(GPIO.BCM)
9.     GPIO.setup(Gpin, GPIO.OUT,initial=GPIO.HIGH)
10.    GPIO.setup(Rpin, GPIO.OUT,initial=GPIO.LOW)
11.    GPIO.setup(TiltPin, GPIO.IN)
12.    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)
13.
```

```
14. def Led(x):  
15.     if x == 0:  
16.         GPIO.output(Rpin, 1)  
17.         GPIO.output(Gpin, 0)  
18.     if x == 1:  
19.         GPIO.output(Rpin, 0)  
20.         GPIO.output(Gpin, 1)  
21.  
22. def Print(x):  
23.     if x == 0:  
24.         print ('      *****')  
25.         print (' * Tilt! *')  
26.         print ('      *****')  
27.  
28. def detect(chn):  
29.     Led(GPIO.input(TiltPin))  
30.     Print(GPIO.input(TiltPin))  
31.  
32. def loop():  
33.     while True:  
34.         pass  
35.  
36. def destroy():  
37.     GPIO.output(Gpin, GPIO.LOW)      # Green LED off
```

```
38.     GPIO.output(Rpin, GPIO.LOW)      # Red LED off
39.     GPIO.cleanup()                  # Release resource
40.
41. if __name__ == '__main__':      # Program start from here
42.     setup()
43.     try:
44.         loop()
45.     # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
46.     except KeyboardInterrupt:
47.         destroy()
```

## Code Explanation

```
12.     GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)
```

Set up a falling detect on **TiltPin**, and callback function to detect. Here **bouncetime** is to add rise threshold detection on the channel and ignore edge operations less than **200ms** caused by switch jitter.

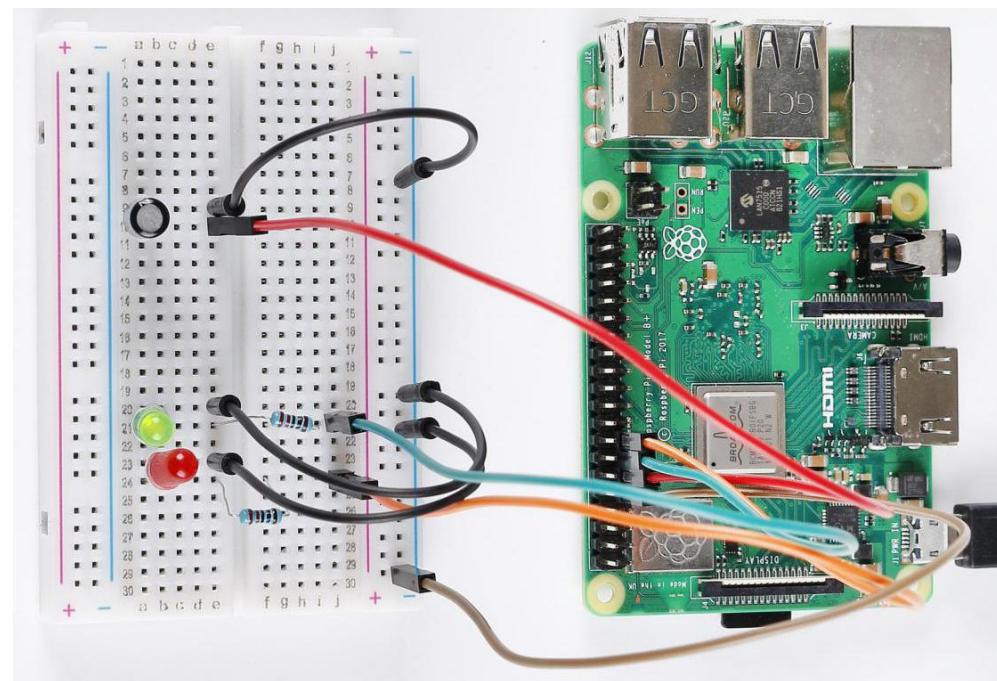
```
13. def Led(x):
14.     if x == 0:
15.         GPIO.output(Rpin, 1)
16.         GPIO.output(Gpin, 0)
17.     if x == 1:
18.         GPIO.output(Rpin, 0)
19.         GPIO.output(Gpin, 1)
```

Define a **Led()** function to set the mode of the two LEDs. When  $x = 0$ , the red LED goes on and the green light goes off; when  $x = 1$ , the red LED goes off, the green LED goes on. When the function is called, the mode of the LED can be set directly with the statement **Led(1)** or **Led(0)**.

```
28. def detect(chn):  
29.     Led(GPIO.input(TiltPin))  
30.     Print(GPIO.input(TiltPin))
```

This is a callback function that executes when a trigger is detected. Assign the current **TiltPin** state (0 or 1) to the **Led** function, that is, pass parameters to the **Led** function. The **Led** function then performs the corresponding operation on the LEDs.

## Phenomenon Picture

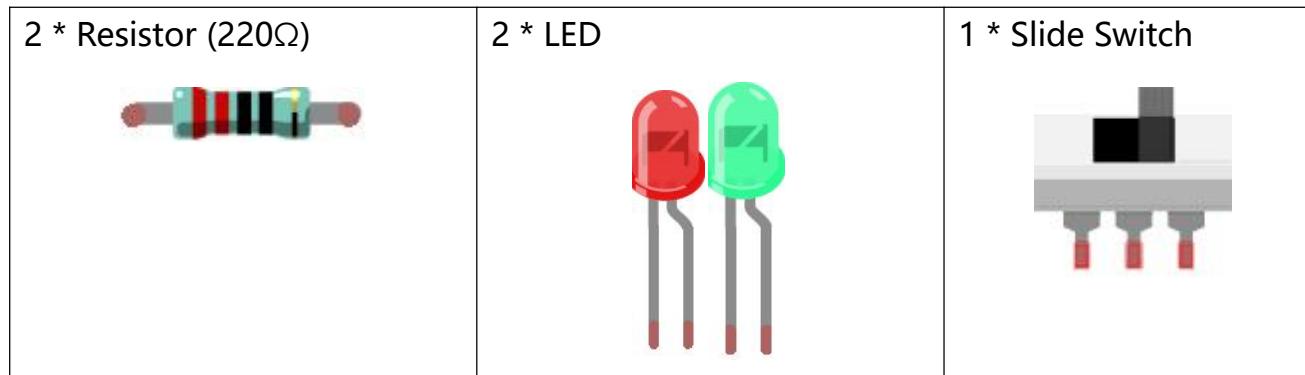


## Lesson 7 Slide Switch

### Introduction

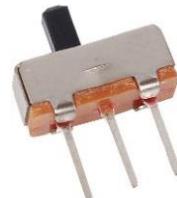
In this lesson, we are going to use a slide switch to turn on the 2 LEDs. The slide switch is a device to connect or disconnect the circuit by sliding its handle. They are quite common in our surroundings. Now let's see how it works.

### Newly Added Components



### Principle

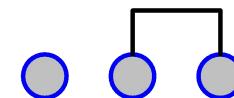
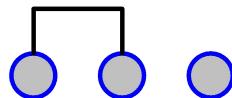
#### Slide Switch



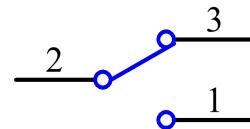
Just as its name suggests, slide switch is to connect or disconnect the circuit by sliding its switch handle so as to switch the circuit. The common types of slide switch include single pole double throw, single pole triple throw,

double pole double throw, and double pole triple throw and so on. Generally, it is used in circuits with a low voltage and features flexibility and stabilization. Slide switches are commonly used in all kinds of instruments/meters equipment, electronic toys and other fields related.

How it works: The middle pin is fixed. When the handle is pushed to the left, the left two pins are connected; when push it to the right, the two pins on the right connect, thus switching circuits.



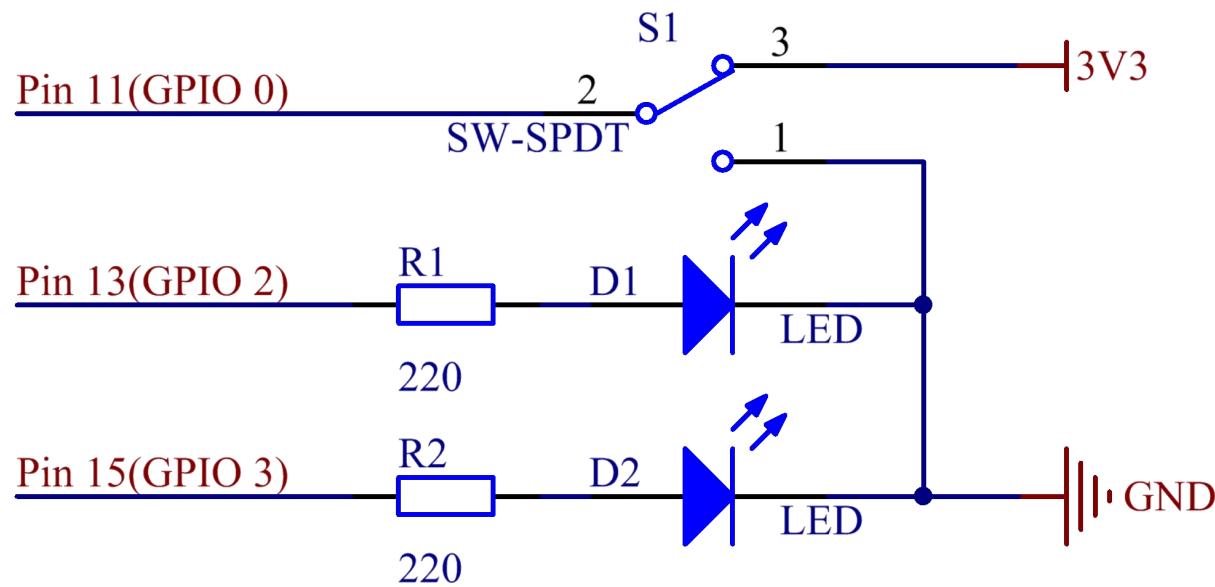
See the circuit symbol of slide switch and 2 is the middle pin.



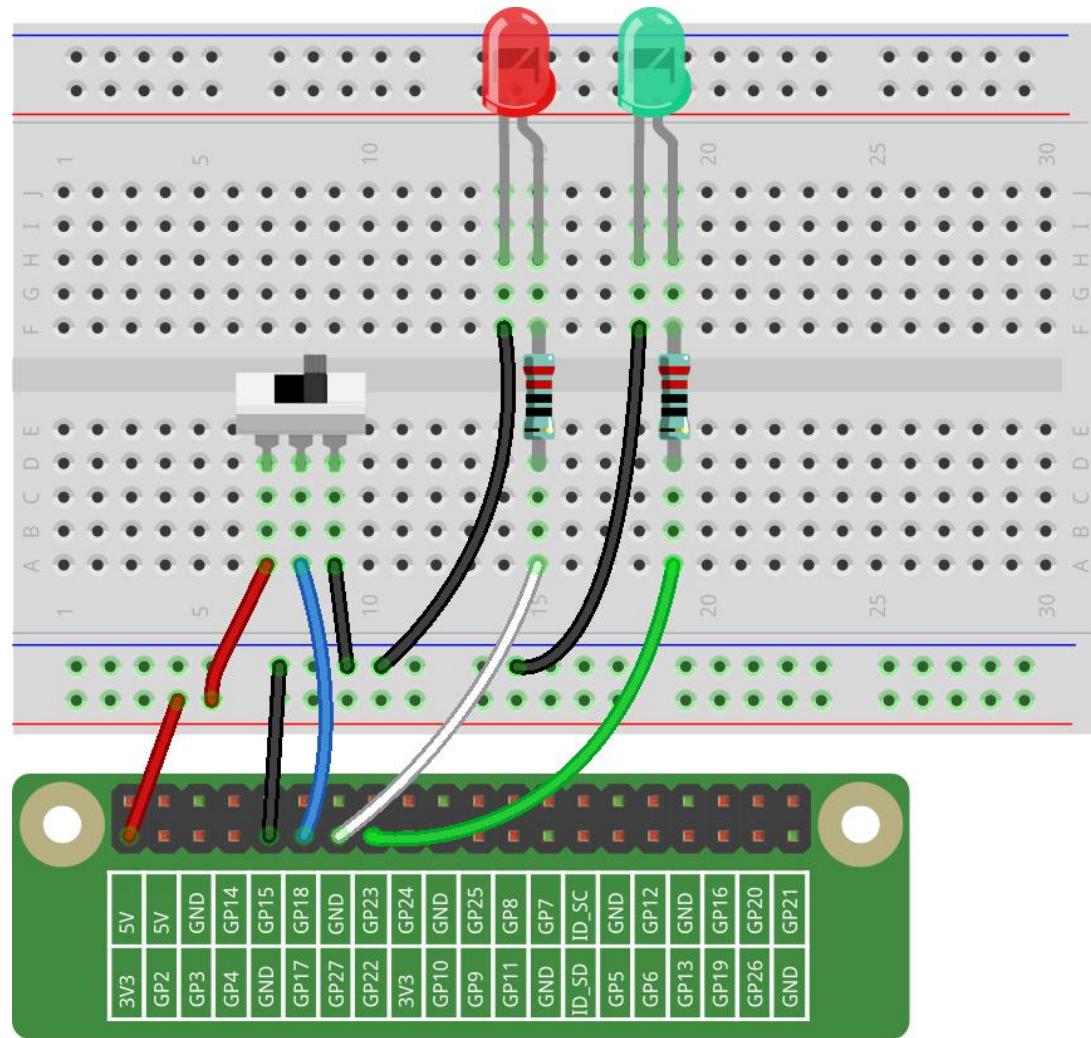
## Schematic Diagram

Here we use a slide switch to turn the LED on/off, which is simple. Connect the middle pin of the switch to pin 11. Connect the left pin of the switch to **GND**, the right to **3.3V**. Attach the anode pins (the longer pins) of the two LEDs to pin **13** and pin **15** respectively after getting them connected with two **220Ω** resistors. In addition, insert the cathodes of the two LEDs into **GND**. Get the slide switch connected to the left, the signal read on pin 11 is 0 (a low level), so the LED 1 lights up; to the right, the signal read on pin 11 is 1 (a high level), then the LED 2 turns on.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_7_SlideSwitch
```

2. Compile the code.

```
gcc 7_Slider.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

When the slide is pulled to the right, the LED2 is on and LED1 off. If the slide is pulled to the left, the LED1 is on and LED2 off.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define slidePin      0
5. #define led1          2
6. #define led2          3
7.
8. int main(void)
9. {
10.    // When initialize wiring failed, print message to screen
```

```
11.     if(wiringPiSetup() == -1){  
12.         printf("setup wiringPi failed !");  
13.         return 1;  
14.     }  
15.  
16.     pinMode(slidePin, INPUT);  
17.     pinMode(led1, OUTPUT);  
18.     pinMode(led2, OUTPUT);  
19.  
20.     while(1){  
21.         // slide switch high, led1 on  
22.         if(digitalRead(slidePin) == 1){  
23.             digitalWrite(led1, HIGH);  
24.             digitalWrite(led2, LOW);  
25.             printf("LED1 on\n");  
26.             delay(100);  
27.         }  
28.         // slide switch low, led2 on  
29.         if(digitalRead(slidePin) == 0){  
30.             digitalWrite(led2, HIGH);  
31.             digitalWrite(led1, LOW);  
32.             printf(".....LED2 on\n");  
33.             delay(100);  
34.     }
```

```
35.    }
36.    return 0;
37.}
```

## Code Explanation

```
16.    pinMode(slidePin, INPUT);
17.    pinMode(led1, OUTPUT);
18.    pinMode(led2, OUTPUT);
```

Initialize the pins connected to slide switch to the **INPUT** mode, and initialize the LED lights to the **OUTPUT** mode.

```
22.    if(digitalRead(slidePin) == 1){
23.        digitalWrite(led1, HIGH);
24.        digitalWrite(led2, LOW);
25.        printf("LED1 on\n");
26.        delay(100);
27.    }
```

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off.

```
28.    if(digitalRead(slidePin) == 0){
29.        digitalWrite(led2, HIGH);
30.        digitalWrite(led1, LOW);
31.        printf(".....LED2 on\n");
32.        delay(100);
```

```
33. }
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code

```
sudo python 7_Slider.py
```

When the slide is pulled to the right, the LED2 is on and LED1 off. If the slide is pulled to the left, the LED1 is on and LED2 off.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. slidePin = 17
5. led1Pin = 27
6. led2Pin = 22
7.
8. # Define a setup function for some setup
```

```
9. def setup():
10.     GPIO.setmode(GPIO.BCM)
11.     GPIO.setup(slidePin, GPIO.IN)
12.     GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.LOW)
13.     GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.LOW)
14.
15. def main():
16.     while True:
17.         # slide switch high, led1 on
18.         if GPIO.input(slidePin) == 1:
19.             print ('LED1 ON ')
20.             GPIO.output(led2Pin, GPIO.LOW)
21.             GPIO.output(led1Pin, GPIO.HIGH)
22.
23.         # slide switch low, led2 on
24.         if GPIO.input(slidePin) == 0:
25.             print ('    LED2 ON ')
26.             GPIO.output(led1Pin, GPIO.LOW)
27.             GPIO.output(led2Pin, GPIO.HIGH)
28.             time.sleep(0.5)
29.
30. def destroy():
31.     # Turn off LED
32.     GPIO.output(led1Pin, GPIO.LOW)
```

```
33.     GPIO.output(led2Pin, GPIO.LOW)
34.     # Release resource
35.     GPIO.cleanup()
36.
37. # If run this script directly, do:
38. if __name__ == '__main__':
39.     setup()
40.     try:
41.         main()
42.     # When 'Ctrl+C' is pressed, the child program
43.     # destroy() will be executed.
44.     except KeyboardInterrupt:
45.         destroy()
```

## Code Explanation

```
11.     GPIO.setup(slidePin, GPIO.IN)
12.     GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.LOW)
13.     GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.LOW)
```

Initialize the pin, then set the pin connected to slide switch to the input mode and LEDs to the output mode.

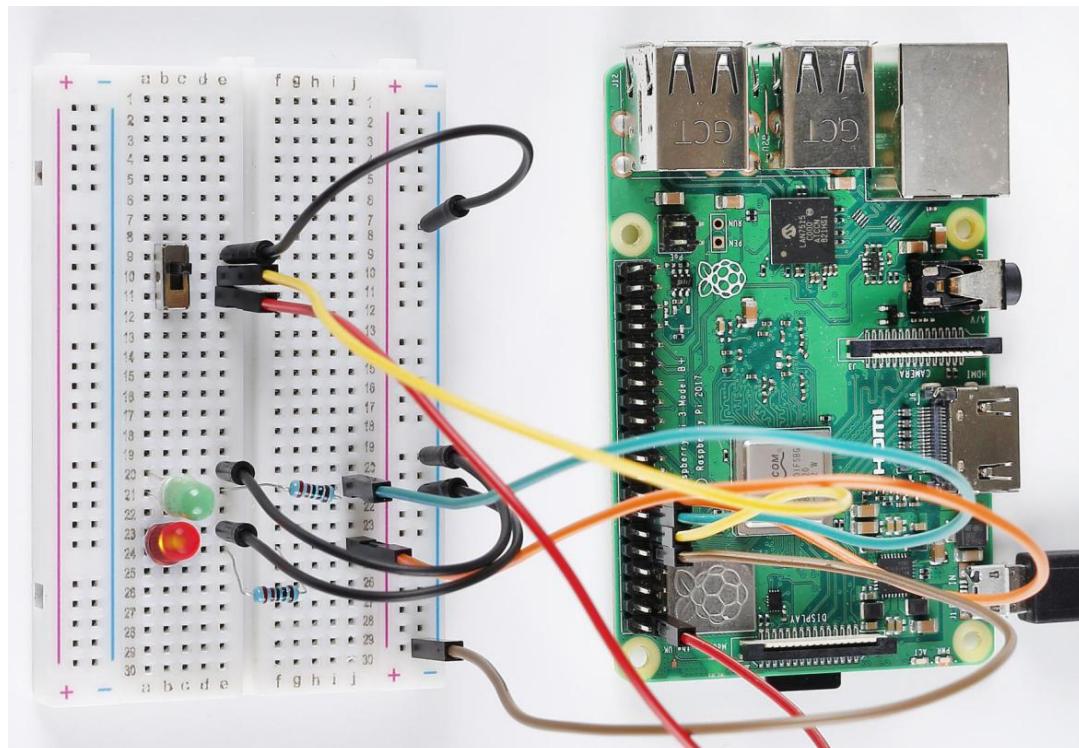
```
18. if GPIO.input(slidePin) == 1:
19.     GPIO.output(led2Pin, GPIO.LOW)
20.     GPIO.output(led1Pin, GPIO.HIGH)
```

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off.

```
24.     if GPIO.input(slidePin) == 0:  
25.         GPIO.output(led1Pin, GPIO.LOW)  
26.         GPIO.output(led2Pin, GPIO.HIGH)
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off.

## Phenomenon Picture



## Lesson 8 Relay

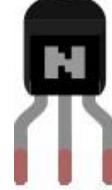
### Introduction

In this lesson, we will learn to use a relay. It is one of the commonly used components in automatic control system. When the voltage, current, temperature, pressure, etc., reaches, exceeds or is lower than the predetermined value, the relay will connect or interrupt the circuit, to control and protect the equipment.

### Newly Added Components

1 * Resistor(1kΩ)	1 * Resistor (220Ω)	1 * Diode1N4007 (Rectifier)
		

1 * LED	1 * NPN S8050	1 * Relay
		

## Principle

### Relay

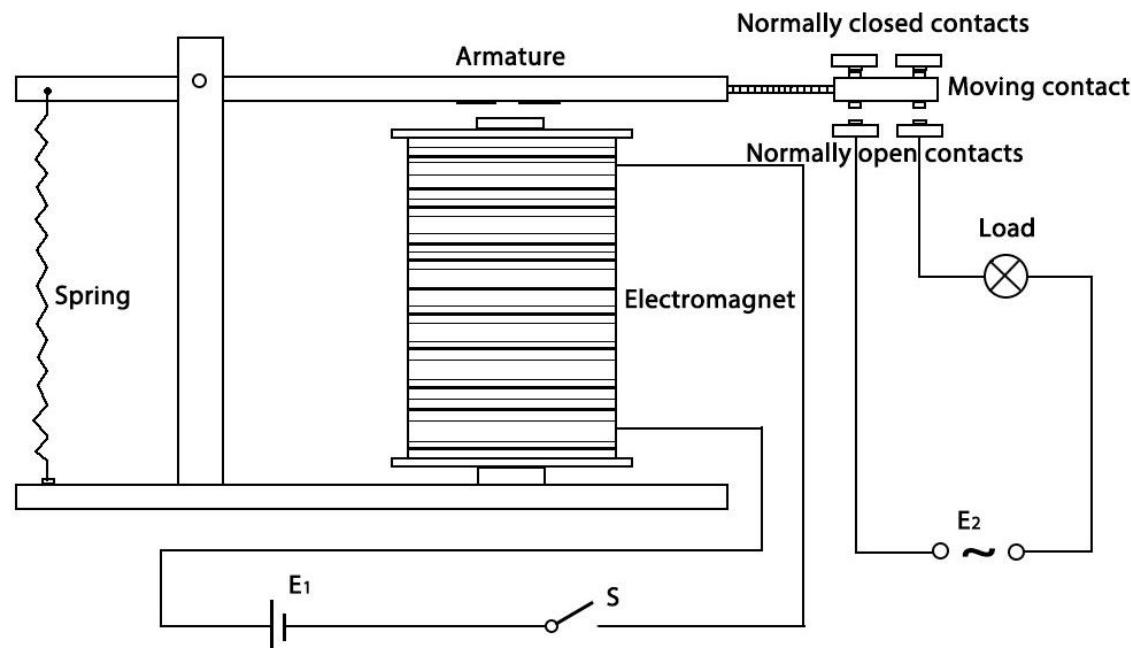
As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:

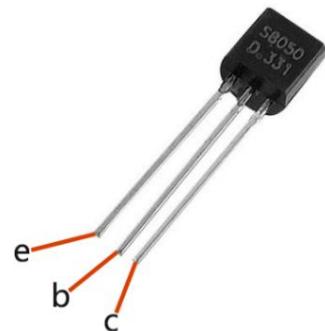
1. **Electromagnet** - It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.
2. **Armature** - The movable magnetic strip is known as armature. When current flows through them, the coil is energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).
3. **Spring** - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.
4. Set of electrical **contacts** - There are two contact points:
  - . **Normally open** - connected when the relay is activated, and disconnected when it is inactive.
  - . **Normally close** - not connected when the relay is activated, and connected when it is inactive.
5. Molded frame - Relays are covered with plastic for protection.

## Working of Relay

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

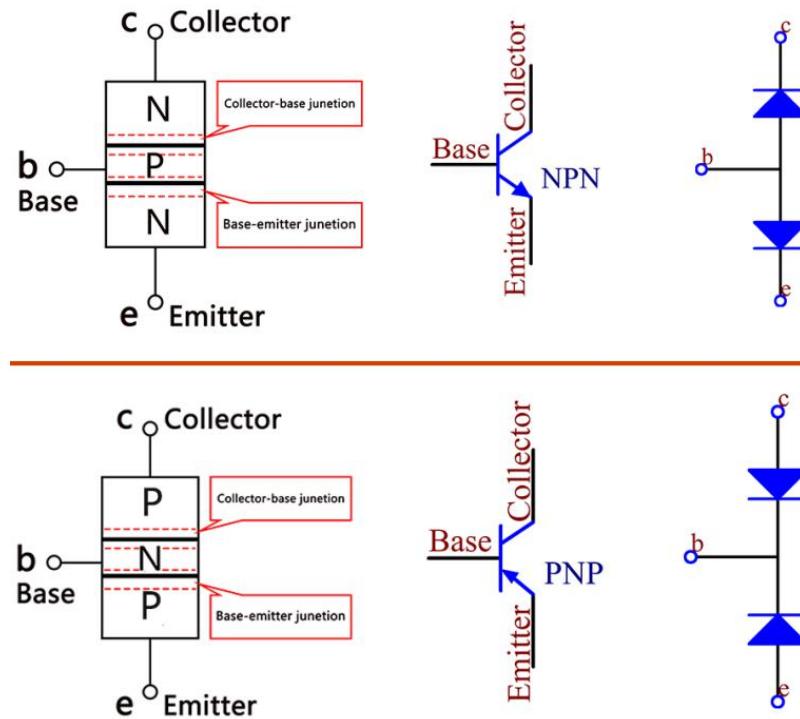


## Transistor



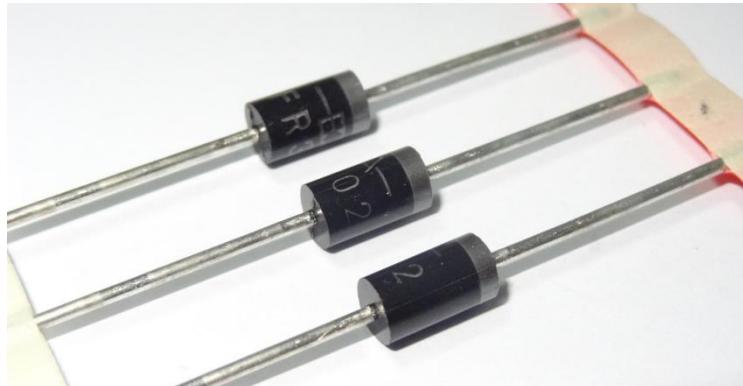
Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch. A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, while the other one is the collector region. This composition enables the transistor to be an amplifier.

From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction. Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.



When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

## Diode1N4007



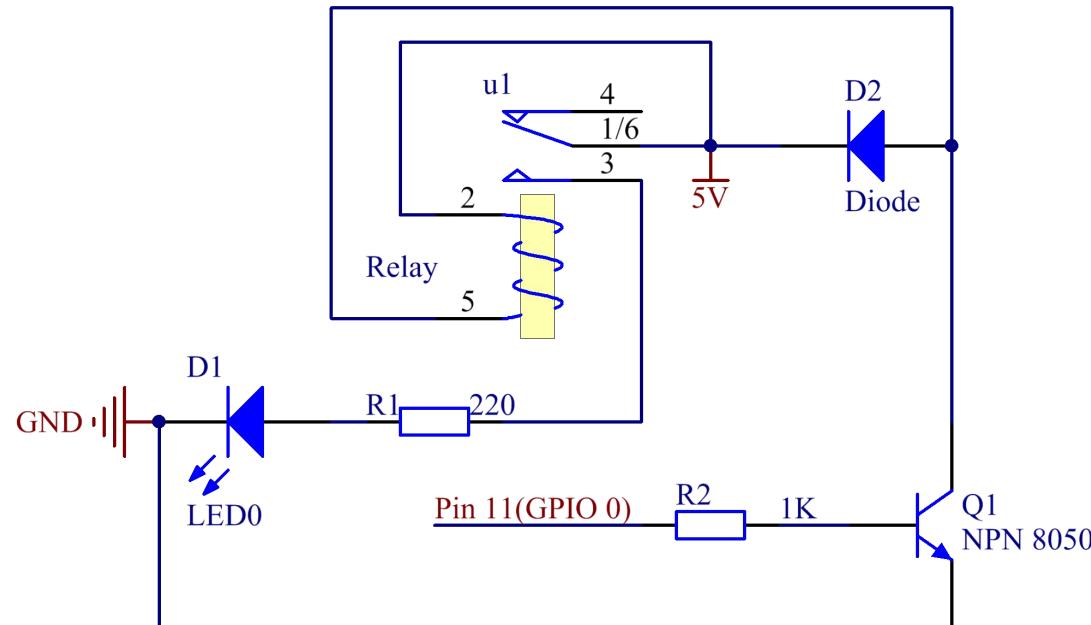
1N4007 is a semiconductor device for converting alternating current into direct current. By using the one-way conductivity of the diode, alternating current with alternating directions can be converted into a single-direction pulse direct current.

With a positive large current, 1N4007 has a low voltage drop (representative value 0.7 V) called as forward conduction state. If the opposite voltage is applied, the potential barrier is increased to withstand a high reverse voltage or to flow through a very small reverse current (called reverse leakage current) called as a reverse blocking state. Thus, the rectifier diode has a significant one-way conductivity. In this lesson, we apply this characteristic of diode.

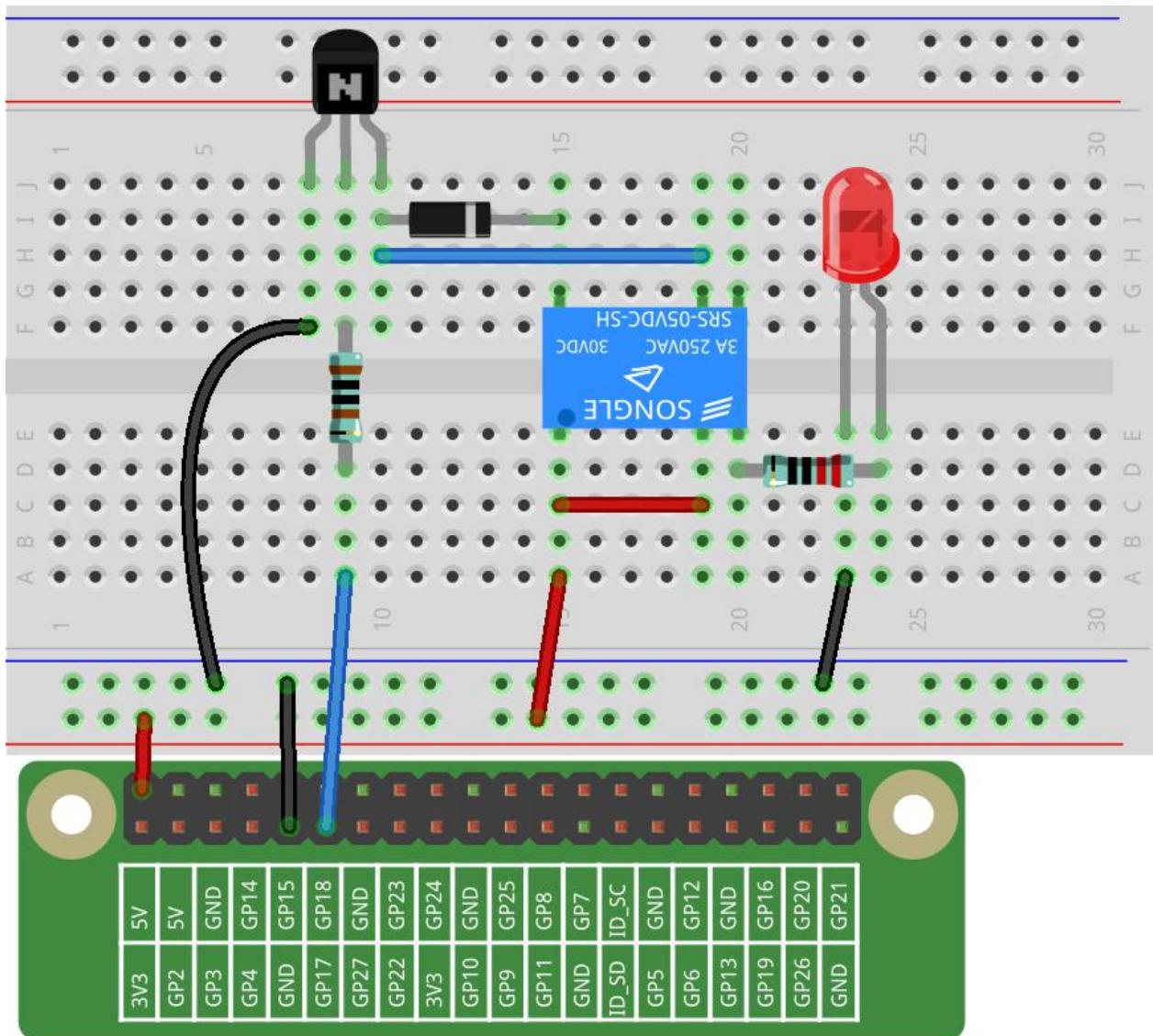
## Schematic Diagram

When a high level signal is given to Pin 11, the transistor is energized, thus making the coil of the relay conductive. Then its normally open contact is closed, and the LED will light up. When Pin 11 is given a Low level, the LED will stay dim. In this experiment, we apply Freewheeling Diode that connects to both ends of the relay coil in parallel to prevent relay from breakdown or burnout caused by induced voltage.

wiringPi	Physical	BCM
0	Pin11	17



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_8_Relay
```

2. Compile the code.

```
gcc 8_Relay.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

Now, the LED will blink, you can hear a tick-tock caused by breaking the normally close contact and closing the normally open one.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define RelayPin 0
5.
6. int main(void){
7.     if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
8.         printf("setup wiringPi failed !");
9.     }
10. }
```

```
11.  
12.    pinMode(RelayPin, OUTPUT);  
13.  
14.    while(1){  
15.        // Tick  
16.        printf(".....Relay Open \n");  
17.        digitalWrite(RelayPin, LOW);  
18.        delay(1000);  
19.        // Tock  
20.        printf("Relay Close.....\n");  
21.        digitalWrite(RelayPin, HIGH);  
22.        delay(1000);  
23.    }  
24.    return 0;  
25. }
```

## Code Explanation

17.     digitalWrite(RelayPin, LOW);

Set the I/O port **RelayPin** as **LOW** (0V), so the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens and the LED remains off.

21.     digitalWrite(RelayPin, HIGH);

Set the I/O port as **HIGH** (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes. Then you can see the LED is lit.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 8_Relay.py
```

Now, the LED is blinking, you can hear a tick-tock caused by breaking the normally closed contact and closing the normally open one.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. relayPin = 17
5.
6. # Define a setup function for some setup
7. def setup():
8.     GPIO.setmode(GPIO.BCM)
9.     GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.LOW)
10.
11.# Define a main function for main process
12.def main():
```

```
13.     while True:  
14.         print ('...Relay open')  
15.         # Tick  
16.         GPIO.output(relayPin, GPIO.LOW)  
17.         time.sleep(1)  
18.         print ('Relay close...')  
19.         # Tock  
20.         GPIO.output(relayPin, GPIO.HIGH)  
21.         time.sleep(1)  
22.  
23. def destroy():  
24.     # Turn off LED  
25.     GPIO.output(relayPin, GPIO.LOW)  
26.     # Release resource  
27.     GPIO.cleanup()  
28.  
29. # If run this script directly, do:  
30. if __name__ == '__main__':  
31.     setup()  
32.     try:  
33.         main()  
34.     # When 'Ctrl+C' is pressed, the child program  
35.     # destroy() will be executed.  
36.     except KeyboardInterrupt:
```

37.       destroy()

## Code Explanation

9.       GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.LOW)

Initialize pins. And the output pin of relay is set to output mode and default low level.

17.      time.sleep(1)

Wait for **1** second. Change the switching frequency of the relay by changing this parameter.

Note: Relay is a kind of metal dome formed in mechanical structure. So its lifespan will be shortened under high-frequency using.

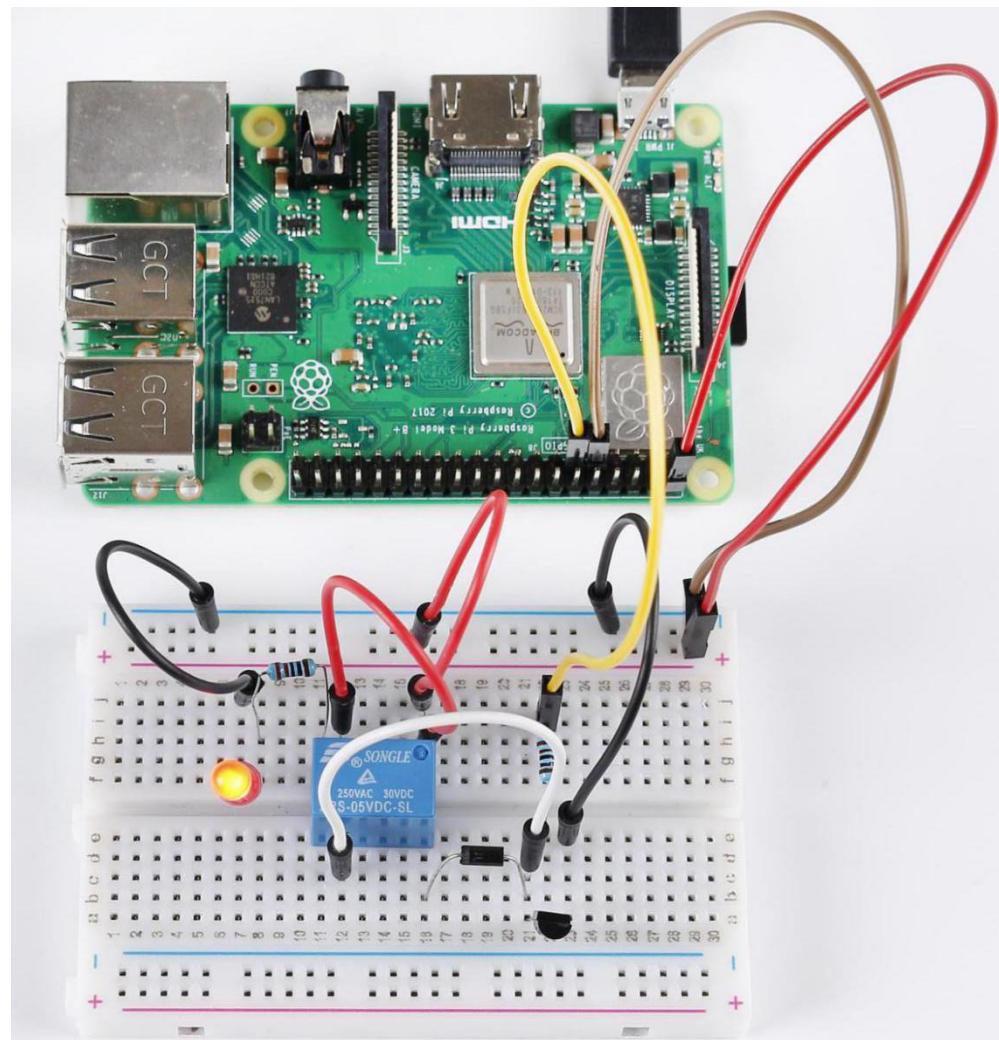
16.      GPIO.output(relayPin, GPIO.LOW)

Set the I/O port as low level (0V), thus the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens and the LED remains off.

20.      GPIO.output(relayPin, GPIO.HIGH)

Set the I/O port as high level (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes. Then you can see the LED is lit.

## Phenomenon Picture



## Lesson 9 4N35

### Introduction

In this lesson, let's learn the operational principle of 4N35 chip. We analyze the principle in this way: using 4N35 chip to drive a LED, and then explaining the phenomenon of LED and the internal structure of the chip.

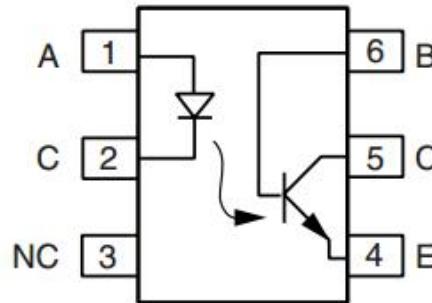
### Newly Added Components

1*4N35	1 * Resistor (220Ω)	1 * LED	1*Resistor(1kΩ)
			

### Principle

#### 4N35

4N35 is a general-purpose optocoupler. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor. What an optocoupler does is to break the connection between signal source and signal receiver, so as to stop electrical interference. 4N35 can be used in AV conversion audio circuits that is widely used in electrical isolation of a general optocoupler.

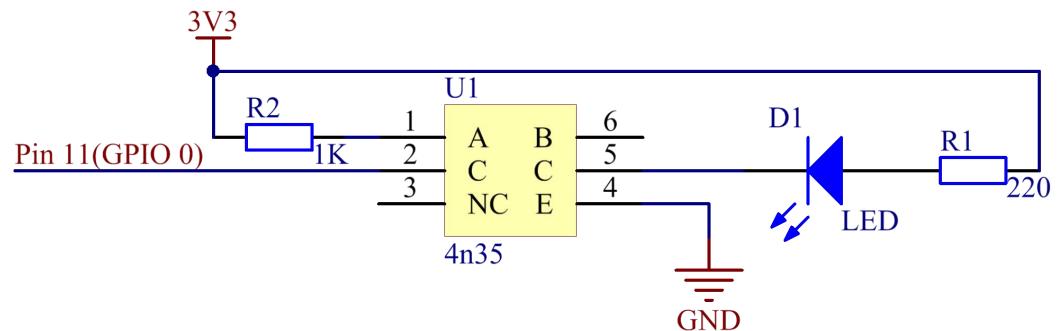


See the internal structure of the 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays, and then it can control the load connected to the phototransistor. Even when the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

## Schematic Diagram

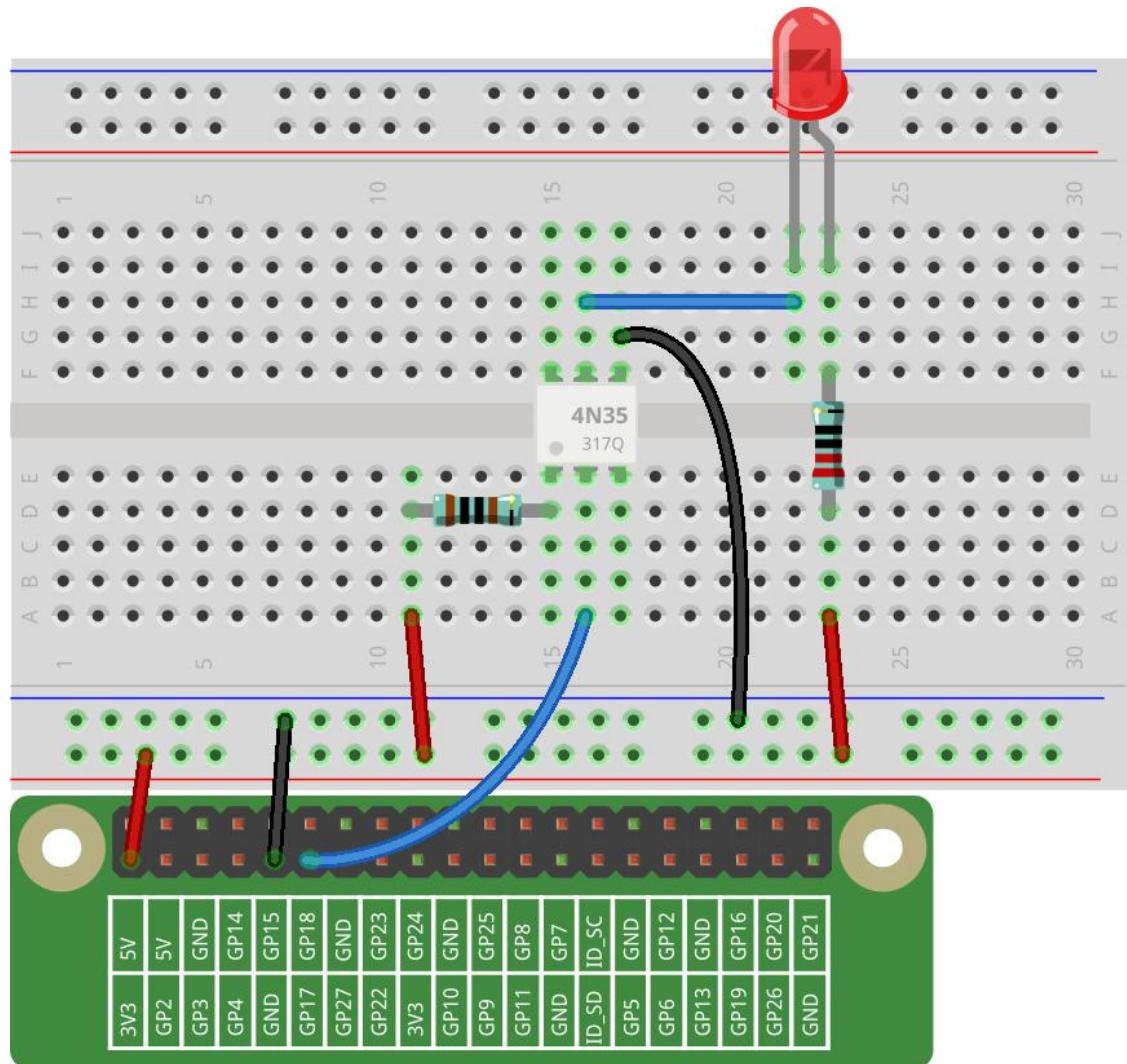
In this experiment, use an LED as the load connected to the NPN phototransistor. In program, a LOW level is given to **Pin 11**, then the infrared LED will emit infrared rays. After that, the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus turning on the LED.

wiringPi	Physical	BCM
0	Pin11	17



## Build the Circuit

Note: pay attention to the direction of the chip by the concave on it.



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_9_4N35
```

2. Compile the code.

```
gcc 9_4N35.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

You will see the LED blinking.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define OptoPin  0
5.
6. int main(void)
7. {
8.     // When initialize wiring failed, print message to screen
9.     if(wiringPiSetup() == -1){
10.         printf("setup wiringPi failed !");
```

```
11.     return 1;
12. }
13.
14. pinMode(OptoPin,OUTPUT);
15.
16. while(1){
17.     // Turn LED off
18.     digitalWrite(OptoPin, HIGH);
19.     delay(500);
20.     // Turn LED on
21.     digitalWrite(OptoPin, LOW);
22.     delay(500);
23. }
24. return 0;
25. }
```

## Code Explanation

```
14. pinMode(OptoPin,OUTPUT);
```

Initialize pins. Set the output pin of 4N35, Optopin to **OUTPUT** mode.

```
18. digitalWrite(OptoPin, LOW);
```

Set **OptoPin** as **LOW** (0V), thus the optocoupler is energized, and the pin connected to LED conduct to low level. Then the LED will light up.

```
21.     digitalWrite(OptoPin, HIGH);
```

Set **OptoPin** as **HIGH** (3.3V), thus the optocoupler is not energized, and the pin connected to LED cannot conduct to low level. Then the LED goes out.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 9_4N35.py
```

You will see the LED blinking.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. Pin_4N35 = 17
5.
6. # Define a setup function for some setup
7. def setup():
8.     GPIO.setmode(GPIO.BCM)
9.     GPIO.setup(Pin_4N35, GPIO.OUT, initial=GPIO.LOW)
```

```
10.  
11. # Define a main function for main process  
12. def main():  
13.     while True:  
14.         # Turn off LED  
15.         GPIO.output(Pin_4N35, GPIO.HIGH)  
16.         time.sleep(0.5)  
17.         # Turn on LED  
18.         GPIO.output(Pin_4N35, GPIO.LOW)  
19.         time.sleep(0.5)  
20.  
21. def destroy():  
22.     # Turn off LED  
23.     GPIO.output(Pin_4N35, GPIO.HIGH)  
24.     # Release resource  
25.     GPIO.cleanup()  
26.  
27. # If run this script directly, do:  
28. if __name__ == '__main__':  
29.     setup()  
30.     try:  
31.         main()  
32.     # When 'Ctrl+C' is pressed, the child program  
33.     # destroy() will be executed.
```

```
34.    except KeyboardInterrupt:  
35.        destroy()
```

## Code Explanation

```
15.    GPIO.output(Pin_4N35, GPIO.HIGH)
```

Set **OptoPin** as **high** level (3.3V), thus the optocoupler is not energized, and the pin connected to LED cannot conduct to low level. Then the LED goes out.

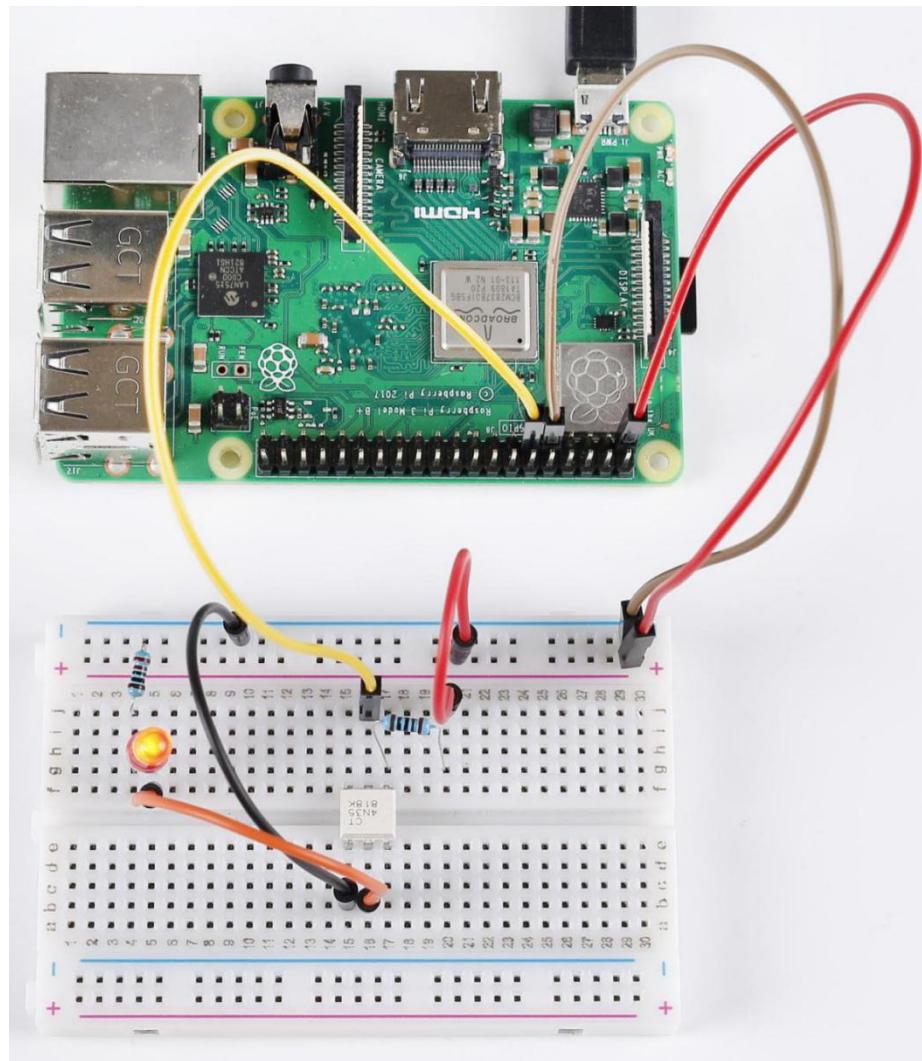
```
16.    time.sleep(0.5)
```

Wait for **0.5** second. The on-off frequency of the optocoupler can be changed by modifying this parameter.

```
18.    GPIO.output(Pin_4N35, GPIO.LOW)
```

Set **OptoPin** as low level (0V), thus the optocoupler is energized, and the pin connected to LED conduct to low level. Then the LED will light up.

## Phenomenon Picture

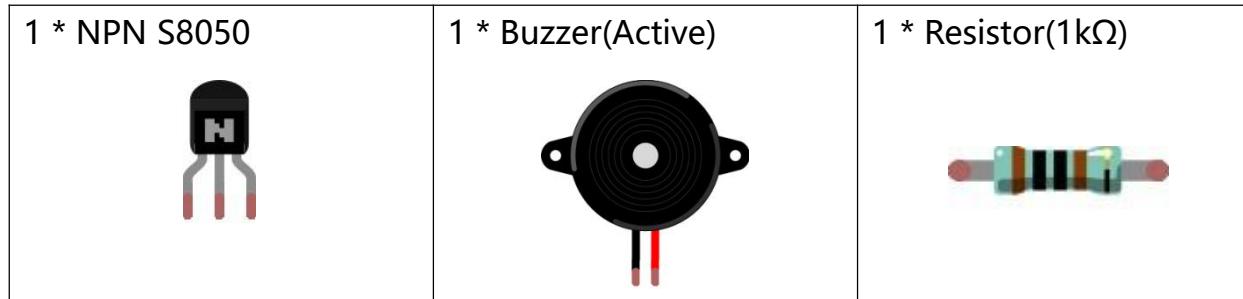


## Lesson 10 Active Buzzer

### Introduction

A buzzer is a great tool in your experiments whenever you want to make sounds.

### Newly Added Components



### Principle

As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices. Buzzers can be categorized as active and passive ones (see the following picture). Turn the buzzer so that its pins are facing up, and the buzzers with a green circuit board is a passive buzzer, while the one enclosed with a black tape is an active one.

The difference between an active buzzer and a passive buzzer:



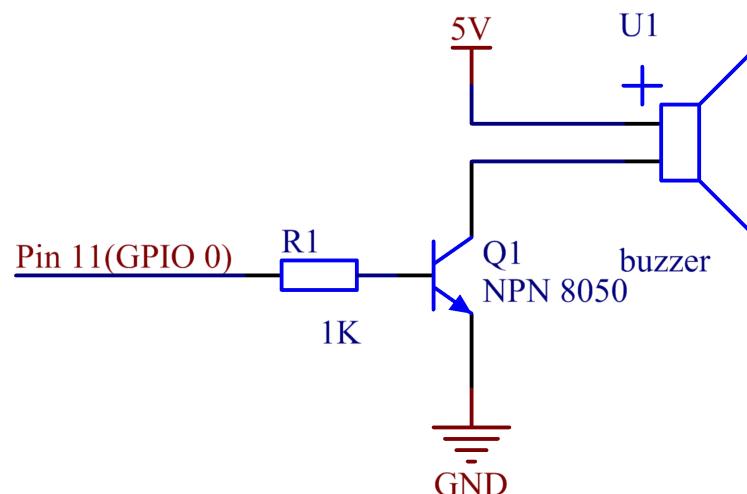
An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not tweet if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

In this experiment, we use an active buzzer.

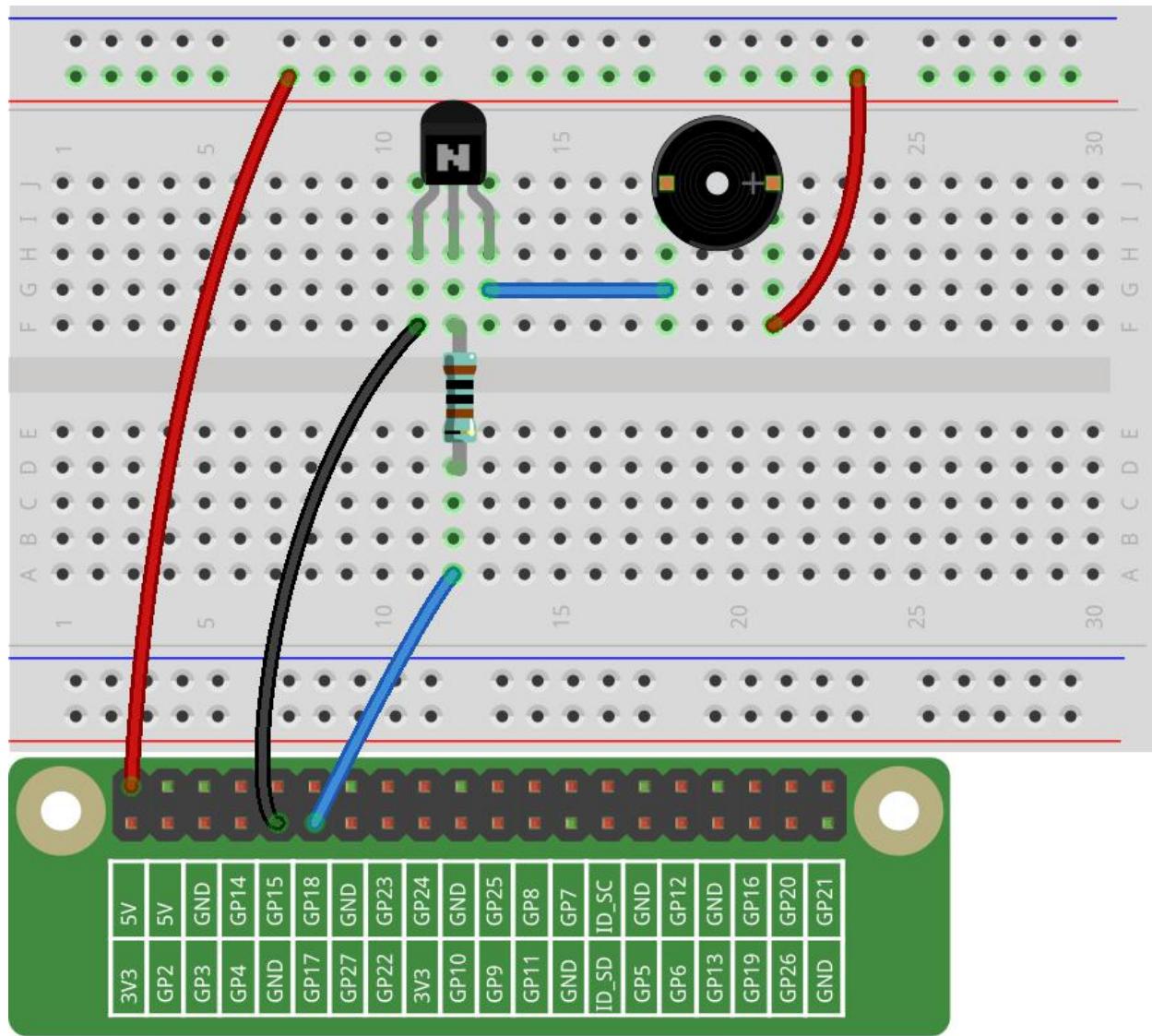
## Schematic Diagram

When **Pin11** is input into high voltage, the transistor will be switched on, and the collector will output low level. When there is a level difference between the two pins of the buzzer, the buzzer is ringing. When **Pin11** inputs low power level, the transistor is cut off and the collector is at high level, here, both ends of the buzzer are at high level, the buzzer will be silent.

wiringPi	Physical	BCM
0	Pin11	17



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_10_ActiveBuzzer
```

2. Compile the code.

```
gcc 10_ActiveBuzzer.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

Now, you may hear the buzzer beep.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define BeepPin 0
5.
6. int main(void){
7.     if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
8.         printf("setup wiringPi failed !");
9.         return 1;
10.    }
```

```
11.  
12.  pinMode(BeepPin, OUTPUT);  
13.  
14.  while(1){  
15.      //beep on  
16.      digitalWrite(BeepPin, HIGH);  
17.      delay(100);  
18.      //beep off  
19.      digitalWrite(BeepPin, LOW);  
20.      delay(100);  
21.  }  
22.  return 0;  
23. }
```

## Code Explanation

```
12.  pinMode(BeepPin, OUTPUT);
```

Set the pin connected to the buzzer to **OUTPUT** mode.

```
16.      digitalWrite(BeepPin, HIGH);
```

When BeepPin is at high level, the base pin(b pin) of the connected transistor inputs high level and the collector pin(c pin) output low level. That is, when the cathode of the buzzer is at low level and the anode of the buzzer is connected to a 5V high level, the buzzer sounds.

```
19.      digitalWrite(BeepPin, LOW);
```

The **BeepPin** is connected to the transistor and then to the cathode of the buzzer. When BeepPin is low level, the base pin (b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer is silent.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 10_ActiveBuzzer.py
```

Now, you should hear the buzzer beep.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. BeepPin = 17
5.
6. def setup():
7.     GPIO.setmode(GPIO.BCM)
8.     GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
9.
10. def main():
```

```
11. while True:  
12.     # Buzzer on (Beep)  
13.     GPIO.output(BeepPin, GPIO.HIGH)  
14.     time.sleep(0.1)  
15.     # Buzzer off  
16.     GPIO.output(BeepPin, GPIO.LOW)  
17.     time.sleep(0.1)  
18.  
19. def destroy():  
20.     # Turn off buzzer  
21.     GPIO.output(BeepPin, GPIO.LOW)  
22.     # Release resource  
23.     GPIO.cleanup()  
24.  
25.# If run this script directly, do:  
26.if __name__ == '__main__':  
27.    setup()  
28.    try:  
29.        main()  
30.        # When 'Ctrl+C' is pressed, the child program  
31.        # destroy() will be executed.  
32.    except KeyboardInterrupt:  
33.        destroy()
```

## Code Explanation

8.     GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)

Initialize the pin connected to the buzzer to output mode and set it to the default low level.

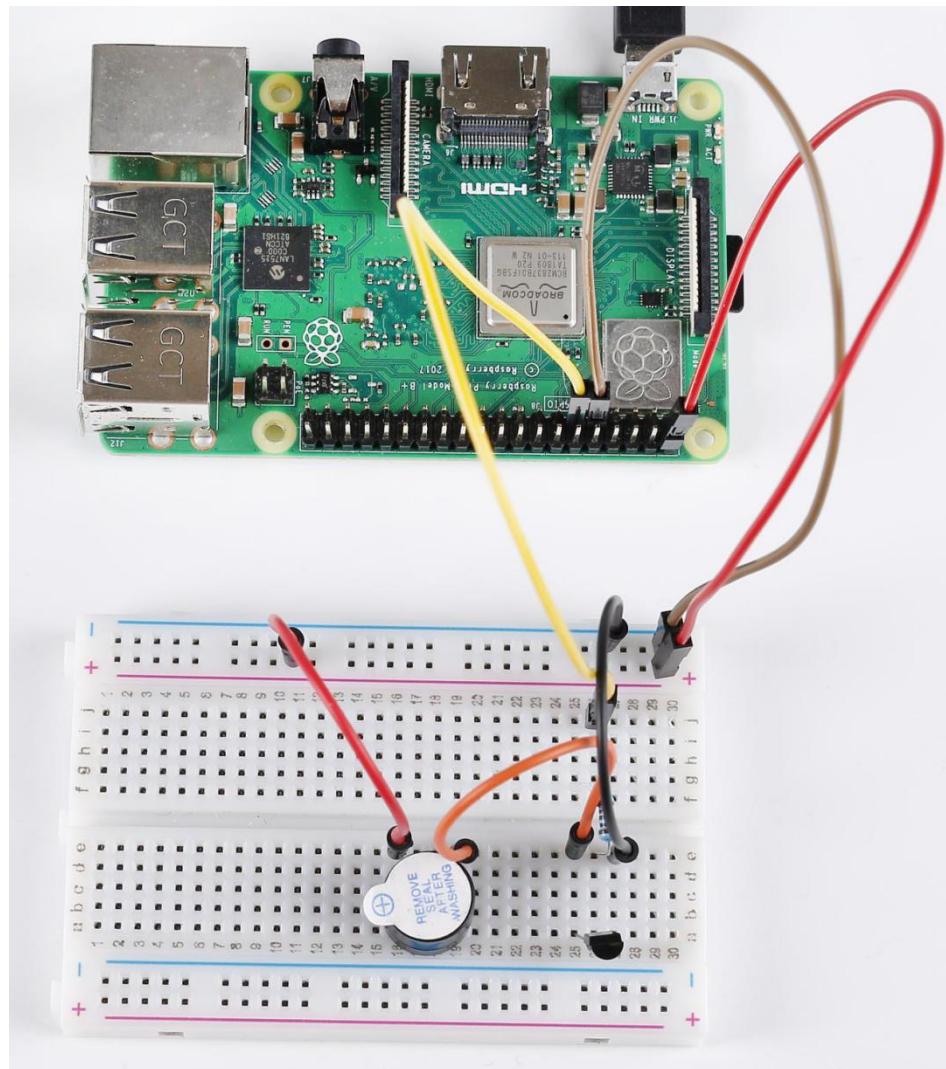
13.    GPIO.output(BeepPin, GPIO.HIGH)

The **BeepPin** is connected to the transistor and then to the cathode of the buzzer. When BeepPin is at high level, the base pin(b pin) of the connected transistor inputs high level, then the collector pin(c pin) outputs low level; that is, when the cathode of the buzzer is at low level and the anode of the buzzer is connected to a 5V high level, the buzzer sounds.

16.    GPIO.output(BeepPin, GPIO.LOW)

When **BeepPin** is at low level, the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer is silent.

## Phenomenon Picture

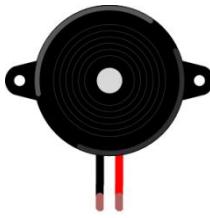
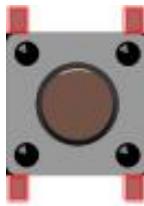


# Lesson 11 Doorbell

## Introduction

In this lesson, we will learn how to drive an active buzzer to build a simple doorbell.

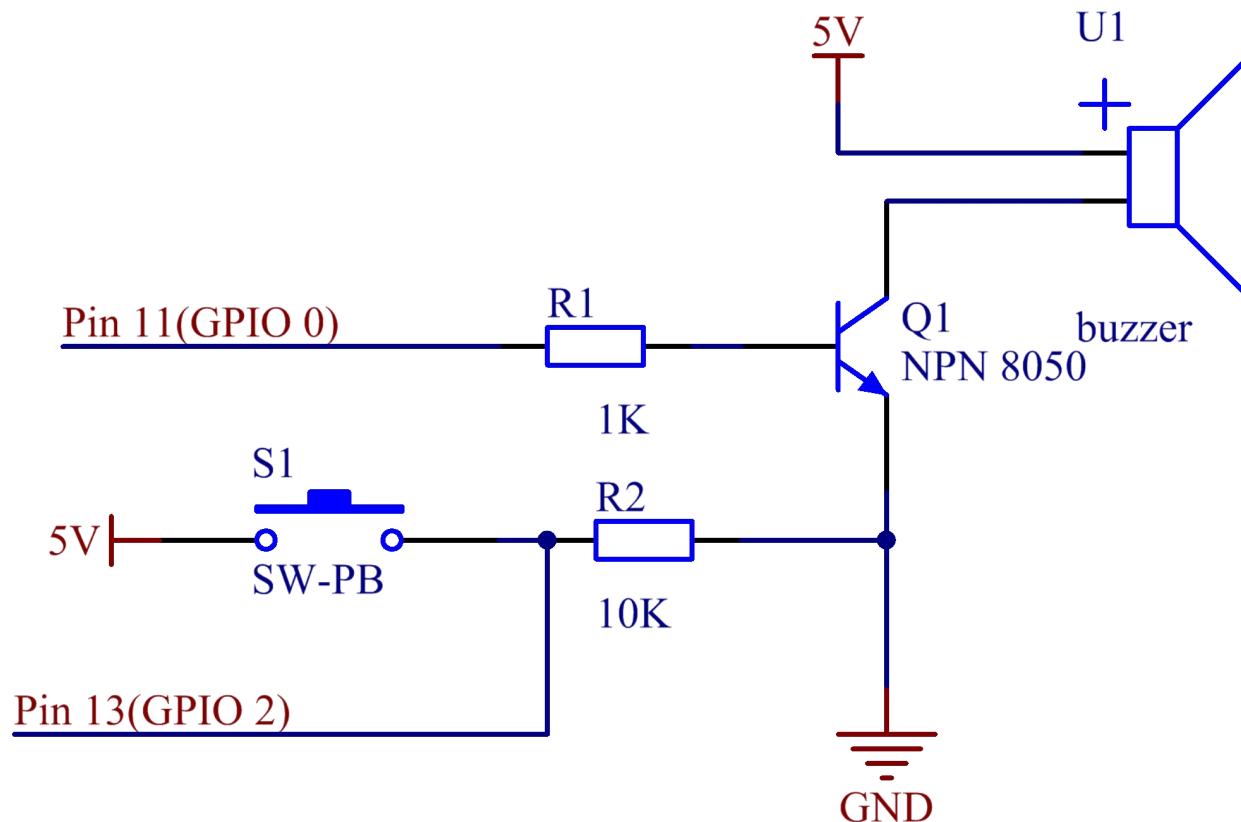
## Newly Added Components

1 * NPN S8050	1 * Buzzer(Active)	1 * Resistor(1kΩ) 	1 * Resistor (10kΩ) 
			

## Schematic Diagram

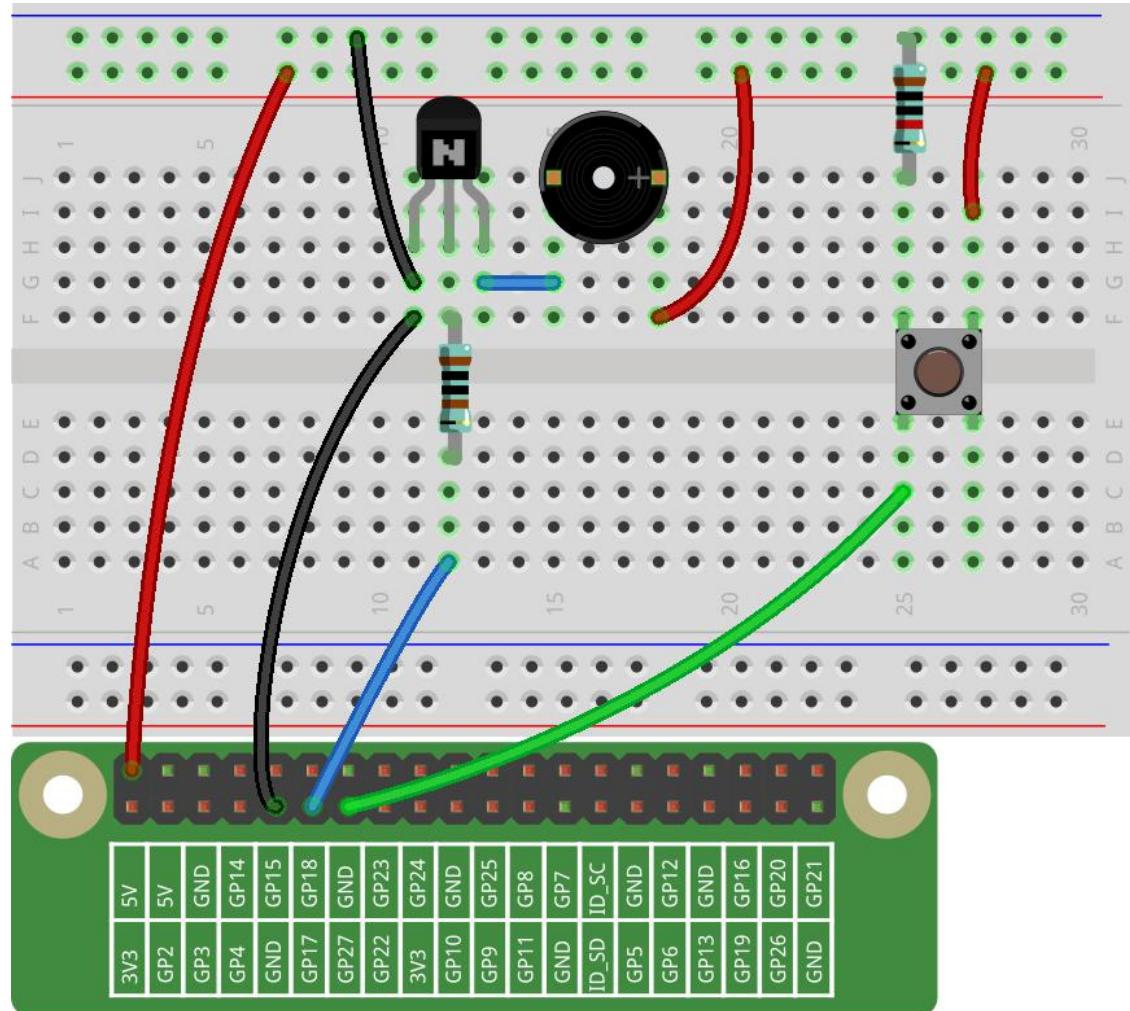
When the button is pressed, **Pin13** is connected to the **5V** power supply and reads out high level; therefore, the program responds to make **Pin11** output the high level so as to energize the transistor, and the collector will output low level that means the buzzer rings. When pin11 outputs at low level the buzzer will be silent.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27



## Build the Circuit

Note: Long pins of buzzer is the Anode and the short pin is Cathode.



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_11_Doorbell
```

2. Compile the code.

```
gcc 11_Doorbell.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

When the button is pressed, the buzzer makes a sound to simulate a doorbell.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define BeepPin 0
5. #define ButtonPin 2
6.
7. int main(void){
8.     if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
9.         printf("setup wiringPi failed !");
10.    return 1;
```

```
11.    }
12.
13.    pinMode(BeepPin, OUTPUT);
14.    pinMode(ButtonPin, INPUT);
15.    pullUpDnControl(ButtonPin, PUD_DOWN);
16.
17.    while(1{
18.        // Indicate that button has pressed down
19.        if(digitalRead(ButtonPin) == 1){
20.            delay(10);
21.            if(digitalRead(ButtonPin) == 1){
22.                //beep on
23.                printf("Buzzer on\n");
24.                digitalWrite(BeepPin, HIGH);
25.                delay(100);
26.            }
27.        }
28.        else{
29.            printf("Buzzer off\n");
30.            //beep off
31.            digitalWrite(BeepPin, LOW);
32.            delay(100);
33.        }
34.    }
```

```
35.     return 0;  
36. }
```

## Code Explanation

```
20.         delay(10);
```

Software removes button jitter. When the signal that the button is pressed is detected, a delay of 10ms is used to eliminate the possibility of false judgment. If both **if** conditions are met, confirm that the button is pressed, and then execute the program in if.

```
21.     if(digitalRead(ButtonPin) == 1){  
22.         //beep on  
23.         printf("Buzzer on\n");  
24.         digitalWrite(BeepPin, HIGH);  
25.         delay(100);  
26.     }
```

If the button is recognized to be pressed, the **BeepPin** is at high level. The base pin(b pin) of the connected transistor inputs high level, while the collector pin(c pin) outputs low level. That is, the cathode of buzzer is at low level, and the anode is connected with a high level 5V. Then the buzzer rings.

```
28. else{  
29.     printf("Buzzer off\n");  
30.     //beep off  
31.     digitalWrite(BeepPin, LOW);  
32.     delay(100);  
33. }
```

Otherwise, **BeepPin** is at low level, and the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, the level at both ends of the buzzer is high, and the buzzer does not ring.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 11_DoorBell.py
```

When the button is pressed, the buzzer makes a sound to simulate a doorbell.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. BeepPin = 17
5. BtnPin = 27
6.
7. def setup():
8.     GPIO.setmode(GPIO.BCM)
9.     GPIO.setup(BtnPin, GPIO.IN)
10.    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
```

```
11.  
12. def main():  
13.     while True:  
14.         if GPIO.input(BtnPin) == 0:  
15.             #Buzzer off  
16.             print ('Buzzer Off')  
17.             GPIO.output(BeepPin, GPIO.LOW)  
18.             time.sleep(0.1)  
19.         if GPIO.input(BtnPin) == 1:  
20.             #Buzzer on  
21.             print ('Buzzer On')  
22.             GPIO.output(BeepPin, GPIO.HIGH)  
23.             time.sleep(0.1)  
24.  
25. def destroy():  
26.     # Turn off buzzer  
27.     GPIO.output(BeepPin, GPIO.LOW)  
28.     # Release resource  
29.     GPIO.cleanup()  
30.  
31. # If run this script directly, do:  
32. if __name__ == '__main__':  
33.     setup()  
34.     try:
```

```
35.     main()
36.     # When 'Ctrl+C' is pressed, the child program
37.     # destroy() will be executed.
38.     except KeyboardInterrupt:
39.         destroy()
```

## Code Explanation

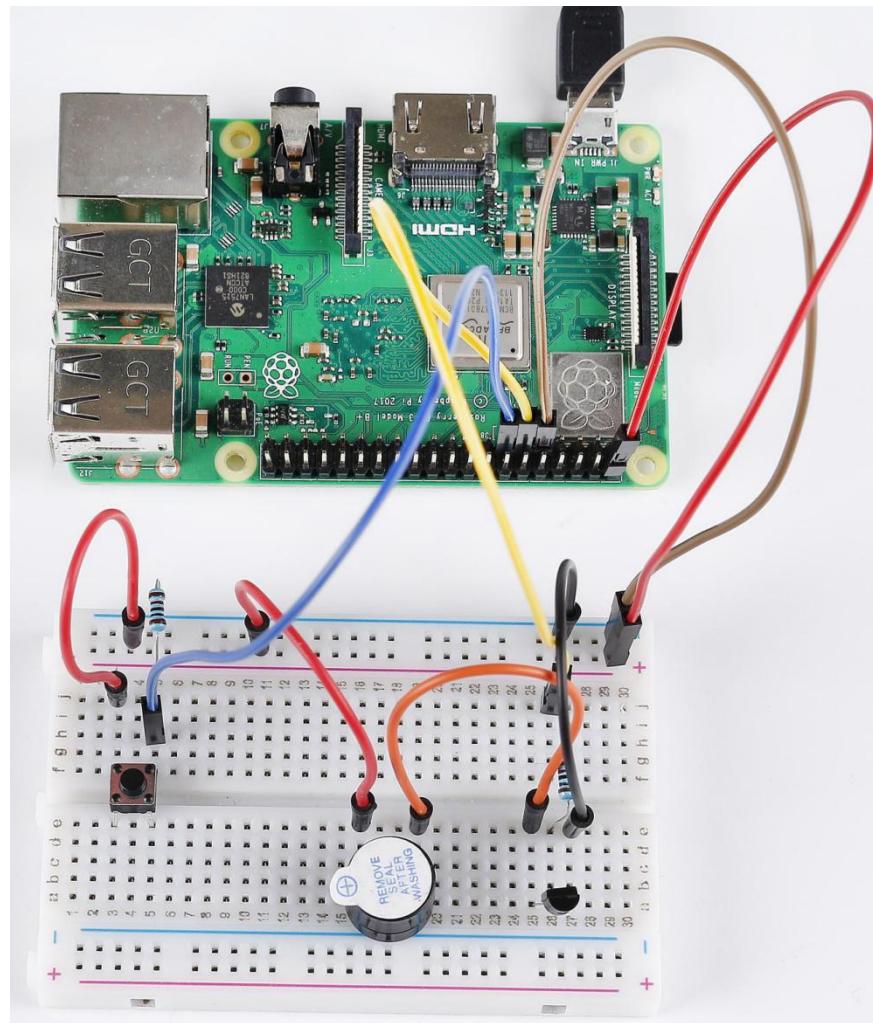
```
14.     if GPIO.input(BtnPin) == 0:
15.         #Buzzer off
16.         print ('Buzzer Off')
17.         GPIO.output(BeepPin, GPIO.LOW)
18.         time.sleep(0.1)
```

If it is judged that the button is not pressed, BeepPin is at low level, and the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer does not ring.

```
19.     if GPIO.input(BtnPin) == 1:
20.         #Buzzer off
21.         print ('Buzzer On')
22.         GPIO.output(BeepPin, GPIO.HIGH)
23.         time.sleep(0.1)
```

If the button is recognized to be pressed, the BeepPin is at high level. The base pin(b pin) of the connected transistor inputs high level, while the collector pin(c pin) outputs low level. That is, the cathode of buzzer is at low level, and the anode is connected with a high level 5V. Then the buzzer rings.

## Phenomenon Picture

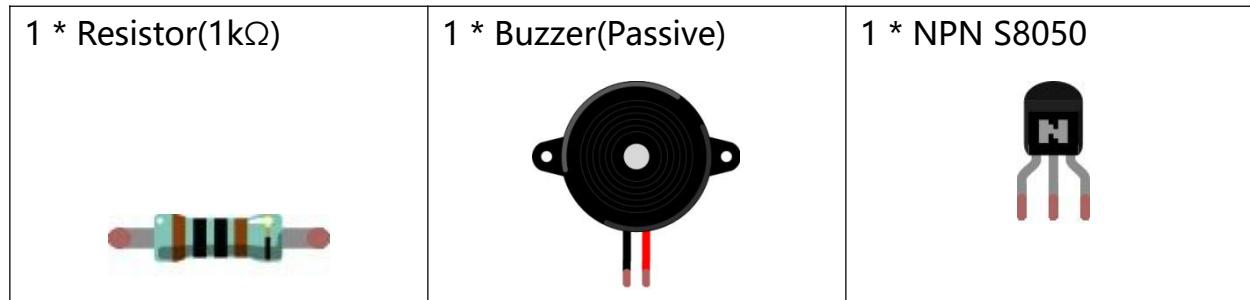


# Lesson 12 Passive Buzzer

## Introduction

In this lesson, we will learn how to make a passive buzzer to play music.

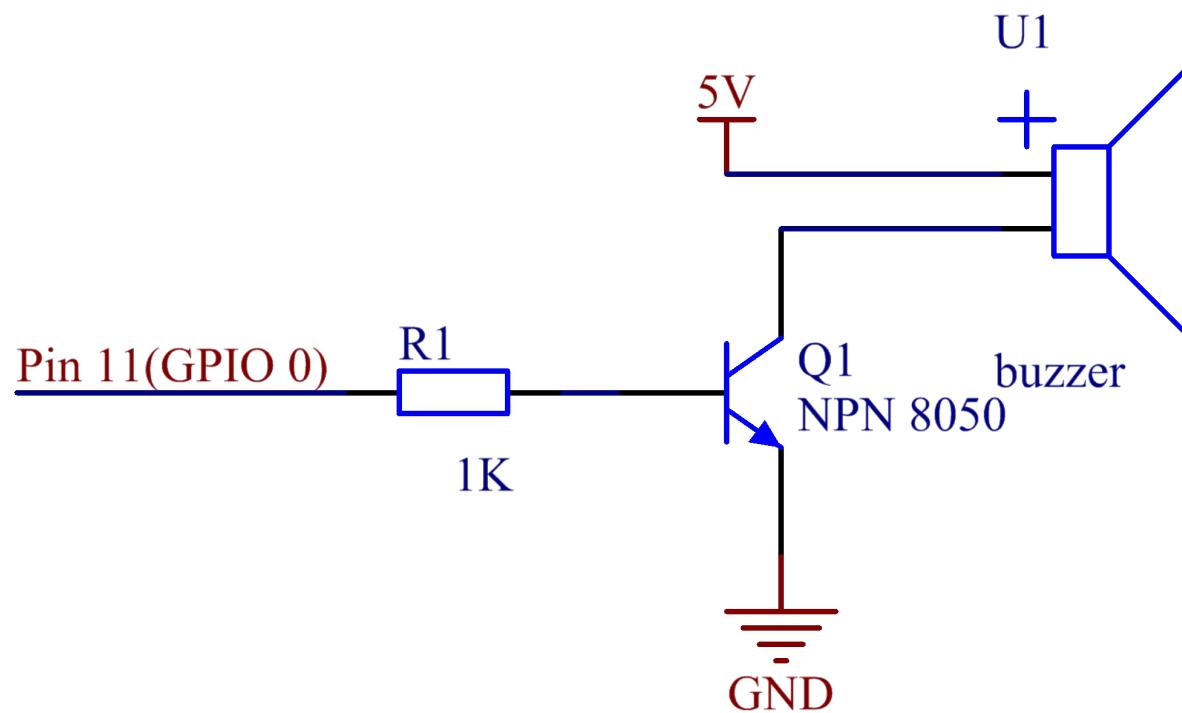
## Newly Added Components



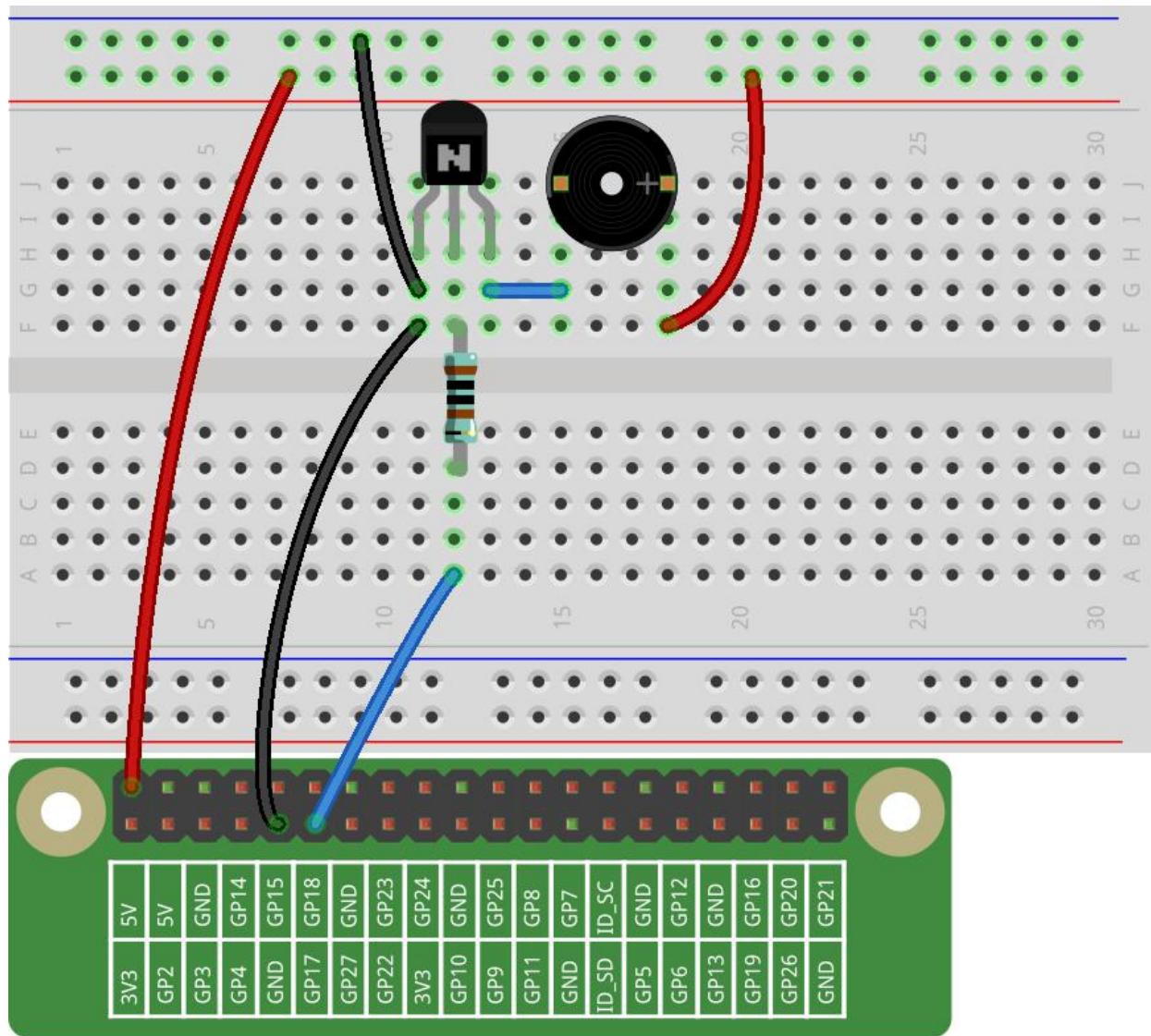
## Schematic Diagram

The base pin(b pin) of the transistor is connected to pin11, the collector pin(c pin) to the cathode pin of the buzzer, and the emitter pin(e pin) to GND. The anode of the buzzer is connected to 5 v power supply. When pin11 inputs high voltage, the transistor will be switched on, and the collector will output low level. When there is a level difference between the two pins of the buzzer, the buzzer rings. When pin11 inputs low power level, the transistor is cut off, and the collector is at high level, and both ends of the buzzer are at high level, so the buzzer is silent.

wiringPi	Physical	BCM
0	Pin11	17



## Build the Circuit



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_12_PassiveBuzzer
```

2. Compile the code.

```
gcc 12_PassiveBuzzer.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

Now, the buzzer automatically plays music on a loop.

### Code

```
1. #include <wiringPi.h>
2. #include <softTone.h>
3. #include <stdio.h>
4.
5. #define BuzPin    0
6.
7. #define  CM1    262
8. #define  CM2    294
9. #define  CM3    330
10. #define   CM4   350
```

```
11. #define CM5 393
12. #define CM6 441
13. #define CM7 495
14.
15. #define CH1 525
16. #define CH2 589
17. #define CH3 661
18. #define CH4 700
19. #define CH5 786
20. #define CH6 882
21. #define CH7 990
22.
23. int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};
24. int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};
25.
26. int main(void)
27. {
28.     int i, j;
29.
30.     if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
31.         printf("setup wiringPi failed !");
32.         return 1;
33.     }
34.
```

```
35.     if(softToneCreate(BuzPin) == -1){  
36.         printf("setup softTone failed !");  
37.         return 1;  
38.     }  
39.  
40.     while(1){  
41.         printf("music is being played...\n");  
42.         for(int i=0;i<sizeof(song)/4;i++){  
43.             softToneWrite(BuzPin, song[i]);  
44.             delay(beat[i] * 250);  
45.         }  
46.     }  
47.     return 0;  
48. }
```

## Code Explanation

```
2. #include <softTone.h>
```

WiringPi includes a software-driven sound handler capable of outputting a simple tone/square wave signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is  $100\mu\text{s}$ . That gives a maximum frequency of  $1/0.0002 = 5000\text{Hz}$ . Within these limitations, simple tones on a high impedance speaker or piezo sounder is possible.

```
3. #define CM1 262  
4. #define CM2 294  
5. #define CM3 330
```

```
6. #define CM4 350  
7. #define CM5 393  
8. #define CM6 441  
9. #define CM7 495
```

These frequencies of each note are as shown. CM refers to middle note, CH high note, 1-7 correspond to the notes C, D, E, F, G, A, B.

```
23. int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};  
24. int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};
```

Define a section of music and the corresponding beat. The number in **beat[]** refers to the beat of each note in the **song**(0.5s for each beat).

```
35.     if(softToneCreate(BuzPin) == -1){
```

**softToneCreate()** creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the **wiringPiSetup()** function you used. The return value is 0 for success. This is used to determine whether it is successful for the software to control tone pin; if it fails, it will not execute the program.

```
42.     for(int i=0;i<sizeof(song)/4;i++){  
43.         softToneWrite(BuzPin, song[i]);  
44.         delay(beat[i] * 250);  
45.     }
```

Employ a for statement to play song\_1.

In the judgment condition, `i < sizeof(song_1) / 4`, "devide by 4" is used because the array `song_1[]` is an array of the data type of integer, and each element takes up four bytes.

The number of elements in `song` (the number of musical notes) is gotten by deviding `sizeof(song)` by 4.

To enable each note to play for `beat * 500ms`, the function `delay(beat_1[i] * 500)` is called.

The prototype of `softToneWrite(BuzPin, song_1[i])`:

```
void softToneWrite (int pin, int freq);
```

This updates the tone frequency value on the given pin. The tone does not stop playing until you set the frequency to 0.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 12_PassiveBuzzer.py
```

Now, the buzzer automatically plays music on a loop.

### Code

```
1. import RPi.GPIO as GPIO  
2. import time  
3.
```

```
4. Buzzer = 17
5.
6. CL = [0, 131, 147, 165, 175, 196, 211, 248]      # Frequency of Low C notes
7. CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Middle C notes
8. CH = [1, 525, 589, 661, 700, 786, 882, 990]      # Frequency of High C notes
9.
10. song = [     CH[5], CH[2], CM[6], CH[2], CH[3], CH[6],CH[0], CH[3], # Notes of song
11.           CH[5], CH[3], CM[6], CH[2],CH[0]]
12.
13. beat = [     1,1,1,1,1,1,2,1,1,1,1,1,3      ]
14.
15. def setup():
16.     GPIO.setmode(GPIO.BCM)
17.     GPIO.setup(Buzzer, GPIO.OUT)
18.     global Buzz
19.     Buzz = GPIO.PWM(Buzzer, 440)
20.     Buzz.start(50)
21.
22. def loop():
23.     while True:
24.         print ('\n      Playing song... ')
25.         for i in range(1, len(song)):
26.             if song[i] == 1 :
27.                 time.sleep(beat[i] *0.25)
```

```
28.         else:
29.             Buzz = GPIO.PWM(Buzzer, song[i])
30.             Buzz.start(50)
31.             time.sleep(beat[i] * 0.25)
32.             Buzz.stop()
33.             time.sleep(1)                      # Wait a second for next song.
34.
35. def destory():
36.     Buzz.stop()
37.     GPIO.output(Buzzer, LOW)
38.     GPIO.cleanup()
39.
40. if __name__ == '__main__':      # Program start from here
41.     setup()
42.     try:
43.         loop()
44.     except KeyboardInterrupt:      # When 'Ctrl+C' is pressed, the child program destroy() will be
        executed.
45.     destory()
```

## Code Explanation

6. CL = [0, 131, 147, 165, 175, 196, 211, 248]
7. CM = [0, 262, 294, 330, 350, 393, 441, 495]
8. CH = [1, 525, 589, 661, 700, 786, 882, 990]

These are the frequencies of each note. The first 0 is to skip **CL[0]** so that the number **CL[1]-CL[7]** corresponds to the CDEFGAB of the note.

```
10. int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};  
13. int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};
```

Define a section of music and the corresponding beats. The number in **beat[]** refers to the beat of each note in the **song**(0.5s for each beat).

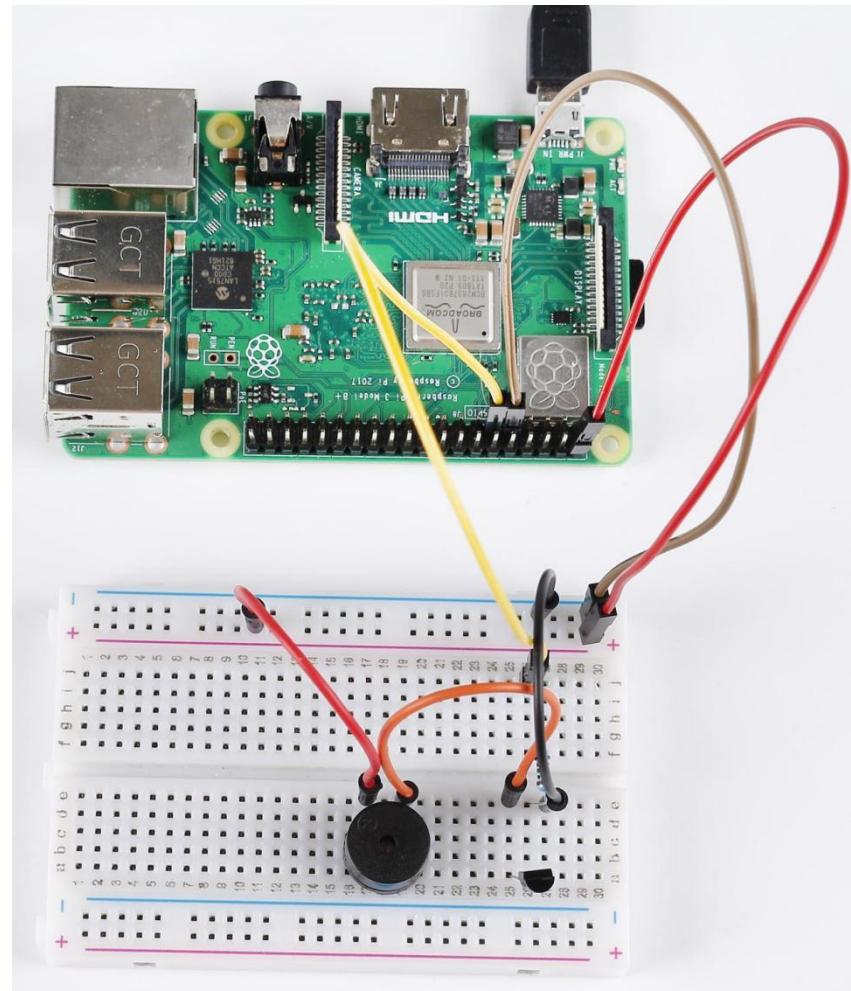
```
19. Buzz = GPIO.PWM(Buzzer, 440)  
20. Buzz.start(50)
```

Define pin Buzzer as PWM pin, then set its frequency to 440 and **Buzz.start(50)** is used to run PWM. What's more, set the duty cycle to 50%.

```
22. def loop():  
23.     while True:  
24.         print ('\n      Playing song...')  
25.         for i in range(1, len(song)):  
26.             if song[i] == 1 :  
27.                 time.sleep(beat[i] *0.25)  
28.             else:  
29.                 Buzz = GPIO.PWM(Buzzer, song[i])  
30.                 Buzz.start(50)  
31.                 time.sleep(beat[i] * 0.25)  
32.                 Buzz.stop()  
33.             time.sleep(1)
```

Play music in the while loop. As i increases gradually, the buzzer plays following the note in song[].

## Phenomenon Picture

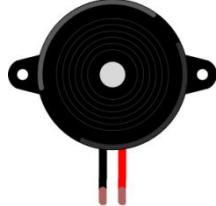
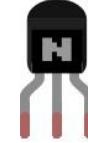


## Lesson 13 Button Piano

### Introduction

In our past lesson, we learned how to use PWM waves to drive a passive buzzer to ring. In this lesson, we make a simple keyboard by applying a passive buzzer. Let's get started!

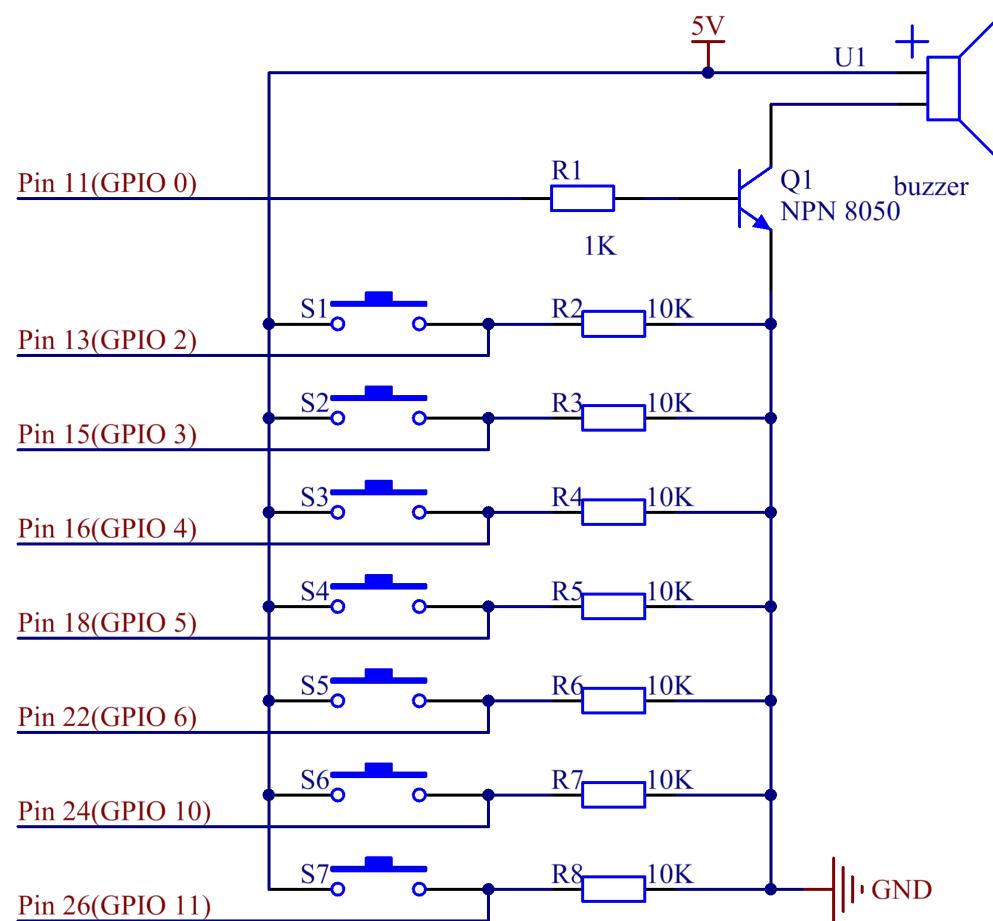
### Newly Added Components

8 * Resistor (10kΩ)	7 * Button	1 * Buzzer(Passive)	1 * NPN S8050
			

### Schematic Diagram

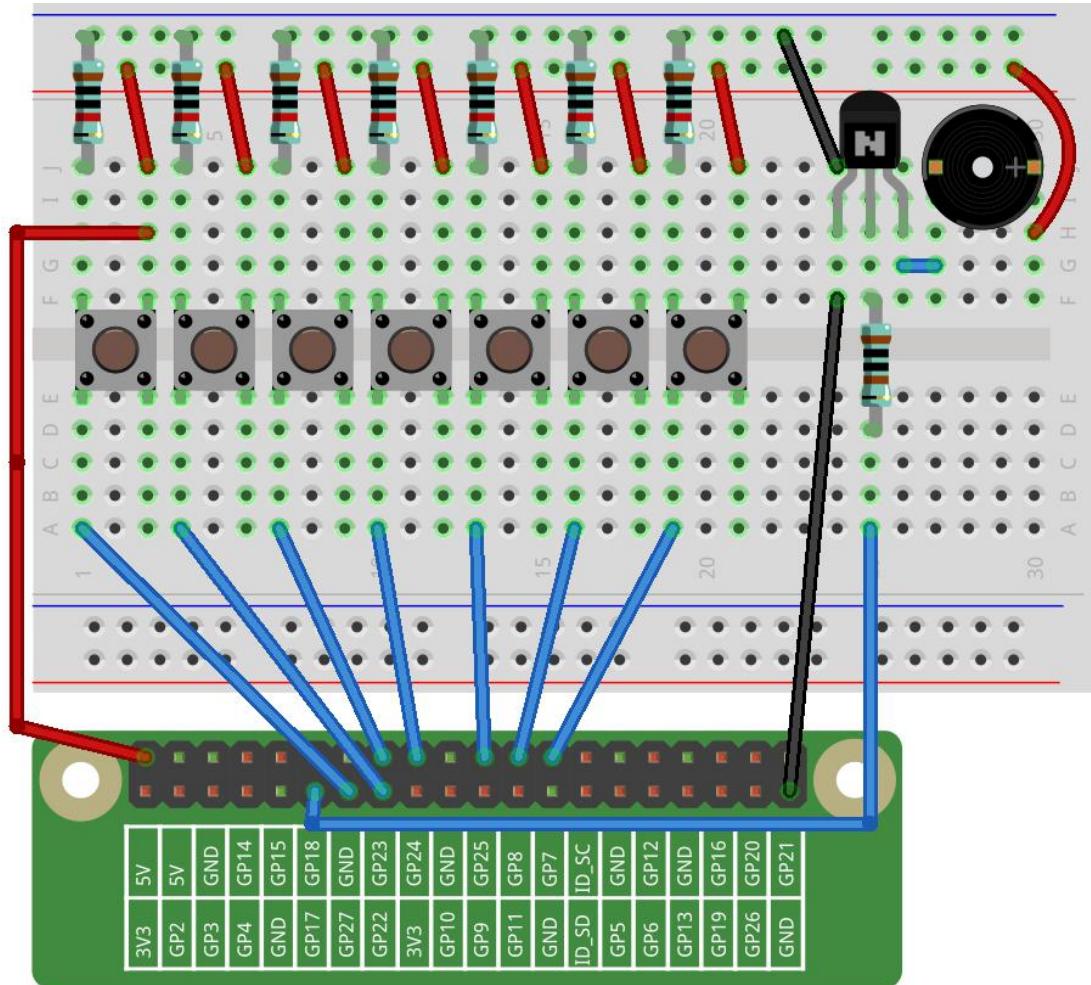
wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24

6	Pin22	25
10	Pin24	8
11	Pin26	7



## Build the Circuit

Components	Raspberry Pi
Botton1	Pin 13
Botton2	Pin 15
Botton3	Pin 16
Botton4	Pin 18
Botton5	Pin 22
Botton6	Pin 24
Botton7	Pin 26
NPN	Pin 11
GND	GND
Buzzer+	5V



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_13_Button_Piano
```

2. Compile the code.

```
gcc 13_ButtonPiano.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

Now press the seven buttons, and the buzzer will emit the notes: DO, RE, MI, FA, SO, LA, TI. You can play a song with these seven buttons.

### Code

```
1. #include <wiringPi.h>
2. #include <softTone.h>
3. #include <stdio.h>
4.
5. #define BuzPin    0
6.
7. const int Tone[] = {262,294,330,350,393,441,495};//define DO, RE, MI, FA, SO, LA, TI
8. int beat[] = {1,1,1,1,1,1,1};
9. const int Btn[] = {2,3,4,5,6,10,11}//define 7 buttons
```

```
10.  
11. int main(void)  
12. {  
13.     int i, j;  
14.  
15.     if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen  
16.         printf("setup wiringPi failed !");  
17.         return 1;  
18.     }  
19.  
20.     if(softToneCreate(BuzPin) == -1){  
21.         printf("setup softTone failed !");  
22.         return 1;  
23.     }  
24.  
25.     //set the buttons mode  
26.     for(int j=0;j<7;j++)  
27.     {  
28.         pinMode.Btn[j], INPUT);  
29.     }  
30.  
31.     while(1){  
32.         //printf("Please press button to play the piano\n");  
33.         // Indicate that button has pressed down
```

```
34.     for(i=0;i<7;i++)
35.     {
36.         if(digitalRead(Btn[i])==1)
37.         {
38.             delay(10); //Prevent the button's vibration
39.             if(digitalRead(Btn[i])==1)
40.             {
41.                 softToneWrite(BuzPin, Tone[i]);
42.                 delay(beat[i]*250);
43.                 printf("1");
44.             }
45.         }
46.         else
47.             softToneWrite(BuzPin, 0);
48.         if(i==7)
49.             i=0;
50.     }
51. }
52. return 0;
53. }
```

## Code Explanation

```
7. const int Tone[] = {262,294,330,350,393,441,495};
8. int beat[] = {1,1,1,1,1,1,1};
```

In the array **Tone[]**, define the frequencies of DO, RE, MI, FA, SO, LA, TI and the number in **beat[]** refers to the beat of each note in this song(0.5s for each beat).

```
26.    for(int j=0;j<7;j++)  
27.    {  
28.        pinMode(Btn[j], INPUT);  
29.    }
```

Set the mode of all buttons to input mode in the for loop.

```
34.    for(i=0;i<7;i++)  
35.    {  
36.        if(digitalRead(Btn[i])==1)  
37.        {  
38.            delay(10);//Prevent the button's vibration  
39.            if(digitalRead(Btn[i])==1)  
40.            {  
41.                softToneWrite(BuzPin, Tone[i]);  
42.                delay(beat[i]*250);  
43.                printf("1");  
44.            }  
45.        }
```

Use a **for** loop to check all the buttons. When one button in array **Btn[i]** is detected to be pressed, the buzzer will respond to the corresponding note in array **Tone[i]**.

```
46.    else
```

```
47.         softToneWrite(BuzPin, 0);  
48.         if(i==7)  
49.             i=0;  
50.     }
```

If no button is pressed, turn off the buzzer.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 13_ButtonPiano.py
```

Now press the seven buttons, and the buzzer will emit the notes: DO, RE, MI, FA, SO, LA, TI. You can play a song with these seven buttons.

### Code

```
1. import RPi.GPIO as GPIO  
2. import time  
3.  
4. Buzzer = 17  
5. BtnPin = [18,27,22,23,24,25,8,7]  
6.
```

```
7. CL = [0, 131, 147, 165, 175, 196, 211, 248]      # Frequency of Low C notes
8. CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Middle C notes
9. CH = [1, 525, 589, 661, 700, 786, 882, 990]      # Frequency of High C notes
10.
11. song = [      0,CM[1],CM[2],CM[3],CM[4],CM[5],CM[6],CM[7]      ]
12. beat = [      1,1, 1, 1, 1, 1, 1, 1]
13.
14. def setup():
15.     GPIO.setmode(GPIO.BCM)
16.     for i in range(1, len(BtnPin)):
17.         GPIO.setup(BtnPin[i],GPIO.IN)
18.     GPIO.setup(Buzzer, GPIO.OUT)
19.
20. def loop():
21.     global Buzz
22.     while True:
23.         #print ('\n      Please playing piano... ')
24.         for i in range(1, len(BtnPin)):
25.             if GPIO.input(BtnPin[i]) == 1:
26.                 Buzz = GPIO.PWM(Buzzer, song[i])
27.                 Buzz.start(50)
28.                 time.sleep(beat[i] * 0.25)
29.                 Buzz.stop()
30.
```

```

31. def destroy():
32.     Buzz.stop()
33.     GPIO.output(Buzzer, 0)
34.     GPIO.cleanup()
35.
36. if __name__ == '__main__':          # Program start from here
37.     setup()
38.     try:
39.         loop()
40.     except KeyboardInterrupt:      # When 'Ctrl+C' is pressed, the child program destroy() will be
        executed.
41.     destroy()

```

## Code Explanation

```

7. CL = [0, 131, 147, 165, 175, 196, 211, 248]      # Frequency of Low C notes
8. CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Middle C notes
9. CH = [1, 525, 589, 661, 700, 786, 882, 990]      # Frequency of High C notes

```

These are the frequencies of each note. The first 0 is to skip **CL[0]** so that the number **CL[1]-CL[7]** corresponds to the CDEFGAB of the note.

```

10. song = [      0,CM[1],CM[2],CM[3],CM[4],CM[5],CM[6],CM[7]      ]
11. beat = [      1,1, 1, 1, 1, 1, 1, 1]

```

Define a section of music and the corresponding beats. The number in **beat[]** refers to the beat of each note in the song(0.5s for each beat).

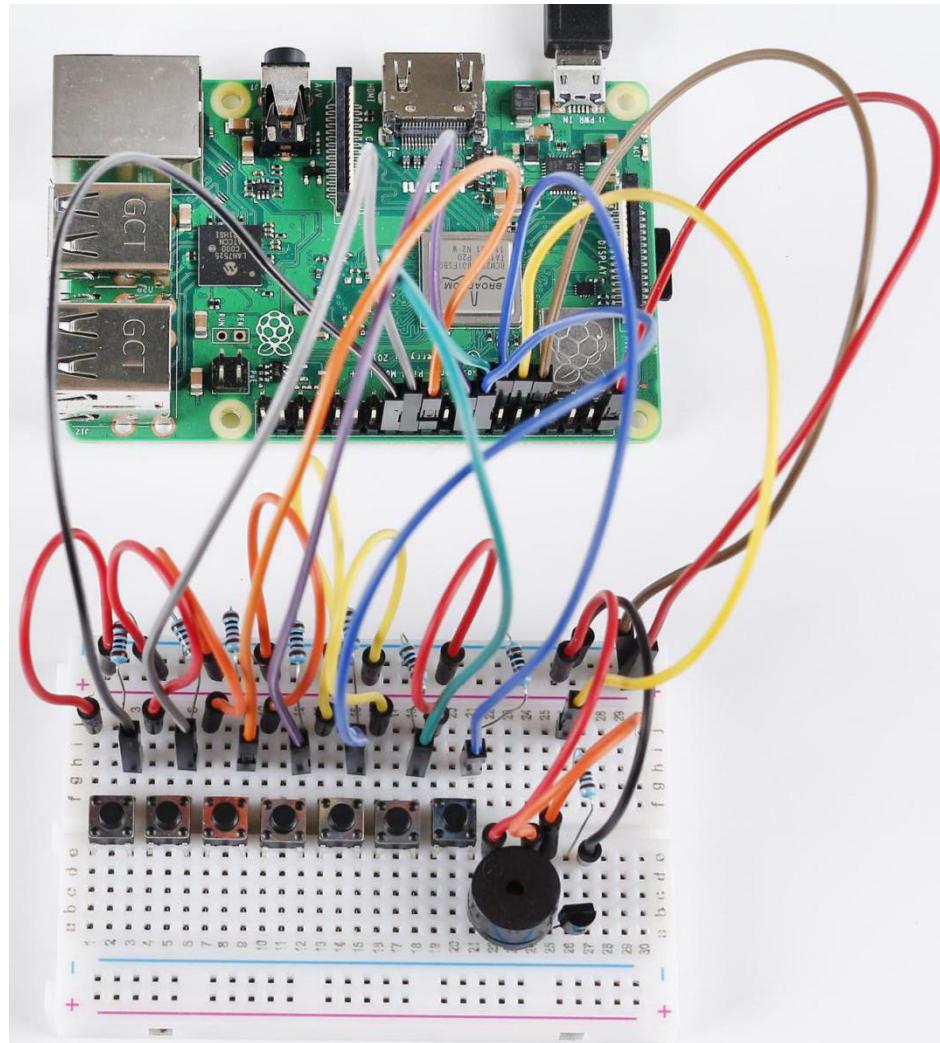
```
16. for i in range(1, len(BtnPin)):  
17.     GPIO.setup(BtnPin[i],GPIO.IN)
```

Set the mode of all buttons to input mode in the for loop.

```
24.     for i in range(1, len(BtnPin)):  
25.         if GPIO.input(BtnPin[i]) == 1:  
26.             Buzz = GPIO.PWM(Buzzer, song[i])  
27.             Buzz.start(50)  
28.             time.sleep(beat[i] * 0.25)  
29.             Buzz.stop()
```

Use a for loop to check all the buttons. When one button in array **button[i]** is detected to be pressed, the buzzer will respond to the corresponding note in array **song[i]**.

## Phenomenon Picture

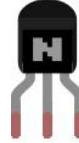
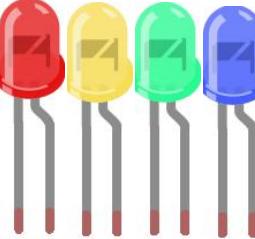
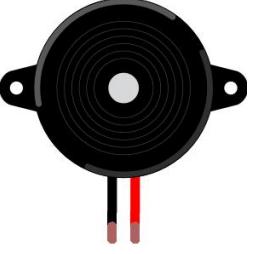


# Lesson 14 Quiz Buzzer System

## Introduction

In quiz shows, especially entertainment activities (e.g. competitive answering activities), organizers often apply a quiz buzzer system in order to accurately, fairly and visually determine the seat number of a responder. In this lesson, we will use some buttons, buzzers, and LEDs to make a quiz buzzer system.

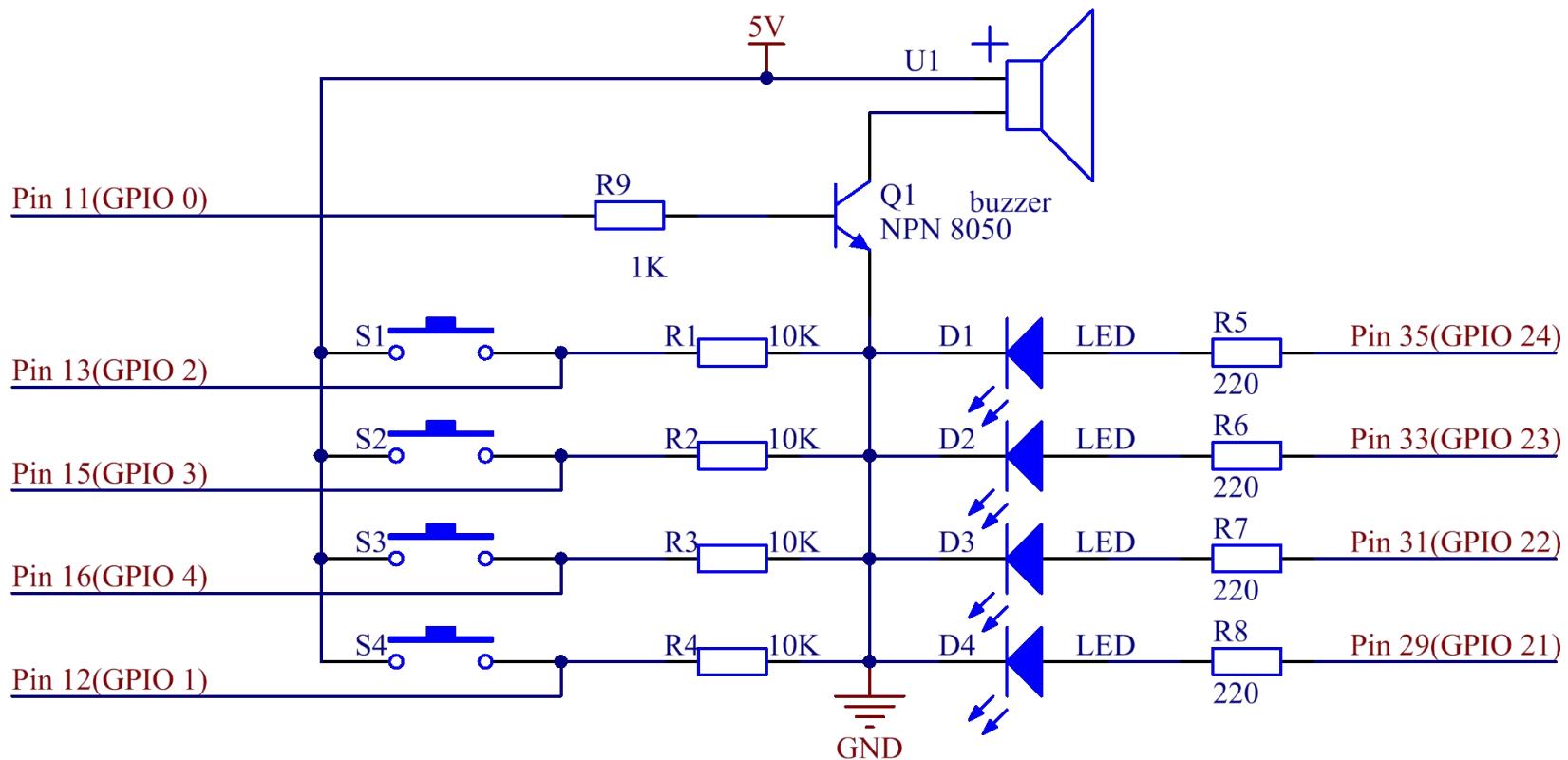
## Newly Added Components

1 * NPN S8050 	4 * Resistor (220Ω) 	1 * Resistor (1kΩ) 	4 * Resistor (10kΩ) 
4 * Led LED 	4 * Button 	1 * Active Buzzer 	

## Schematic Diagram

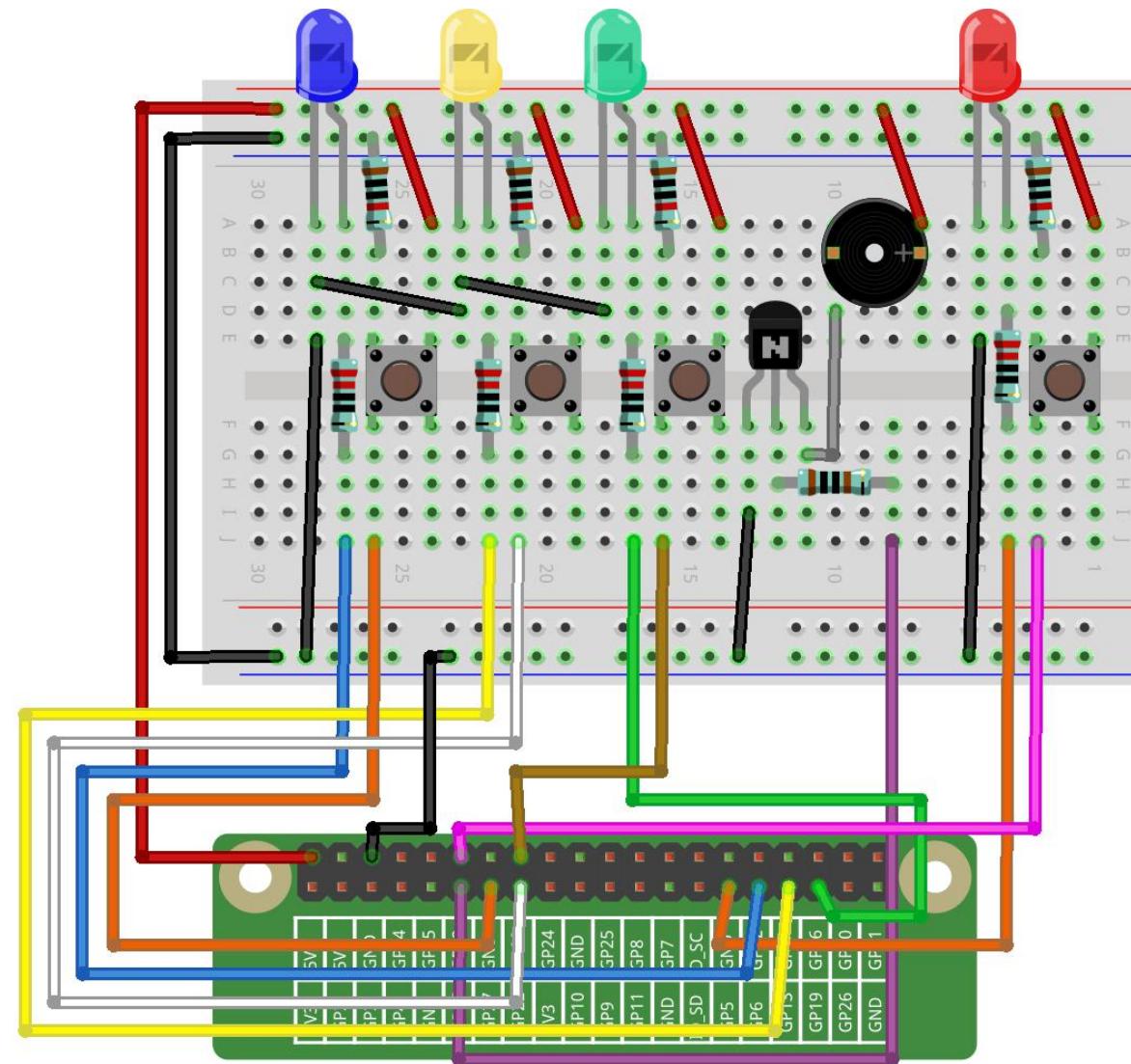
Button 2, 3 and 4 are answer buttons, and button 1 is the reset button. If button 2 is pressed first, the buzzer will beep, the corresponding LED will light up and all the other LEDs will go out. If you want to start another round, press button 1 to reset.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22
4	Pin16	23
1	Pin12	18
21	Pin29	5
22	Pin31	6
23	Pin33	13
24	Pin35	19



## Build the Circuit

Components	Raspberry Pi
Botton1	Pin 12
Botton2	Pin 13
Botton3	Pin 15
Botton4	Pin 16
LED1	Pin 29
LED2	Pin 31
LED3	Pin 33
LED4	Pin 35
NPN	Pin 11
GND	GND
Buzzer+	5V



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_14_AnswerMachine
```

2. Compile the code.

```
gcc 14_AnswerMachine.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

Now, first press button 4 to get started. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define BeepPin 0
5. #define ResetBtnPin 1
6. const int BtnPin[] = {2,3,4};
7. const int LedPin[] = {21,22,23,24};
8.
9. void Alarm()
```

```
10.{  
11.    for(int i=0;i<50;i++){  
12.        digitalWrite(BeepPin,HIGH); //the buzzer sound  
13.        delay(2); //delay 2ms  
14.        digitalWrite(BeepPin,LOW); //without sound  
15.        delay(2);  
16.    }  
17.}  
18.  
19.int main(void){  
20.    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen  
21.        printf("setup wiringPi failed !");  
22.        return 1;  
23.    }  
24.  
25.    pinMode(BeepPin, OUTPUT);  
26.    for(int j=1;j<4;j++)  
27.    {  
28.        pinMode(LedPin[j], OUTPUT);  
29.        digitalWrite(LedPin[j],LOW);  
30.    }  
31.    pinMode(LedPin[0], OUTPUT);  
32.    digitalWrite(LedPin[0],HIGH);  
33.    for(int k;k<3;k++)
```

```
34. {
35.     pinMode(BtnPin[k], INPUT);
36. }
37.
38. int flag = 1;
39.
40. while(1){
41.     // if reset button is pressed
42.     if(digitalRead(ResetBtnPin) == 1)
43.     {
44.         flag = 1;
45.         digitalWrite(LedPin[0], HIGH); //Reset Led turns on
46.         digitalWrite(LedPin[1], LOW);
47.         digitalWrite(LedPin[2], LOW);
48.         digitalWrite(LedPin[3], LOW);
49.     }
50.     if(flag==1)
51.     {
52.         //If the button1 press the first
53.         if(digitalRead(BtnPin[0]) == 1)
54.         {
55.             flag = 0;
56.             digitalWrite(LedPin[0], LOW);
57.             Alarm(); //buzzer sound
```

```
58.         digitalWrite(LedPin[1],HIGH); //turn the LED1 on only
59.         digitalWrite(LedPin[2],LOW);
60.         digitalWrite(LedPin[3],LOW);
61.         while(digitalRead(ResetBtnPin));
62.     }
63.     if(digitalRead.BtnPin[1]) == 1)
64.     {
65.         flag = 0;
66.         digitalWrite(LedPin[0],LOW);
67.         Alarm(); //buzzer sound
68.         digitalWrite(LedPin[1],LOW);
69.         digitalWrite(LedPin[2],HIGH); //turn the LED2 on only
70.         digitalWrite(LedPin[3],LOW);
71.         while(digitalRead(ResetBtnPin));
72.     }
73.     if(digitalRead.BtnPin[2]) == 1)
74.     {
75.         flag = 0;
76.         digitalWrite(LedPin[0],LOW);
77.         Alarm(); //buzzer sound
78.         digitalWrite(LedPin[1],LOW);
79.         digitalWrite(LedPin[2],LOW);
80.         digitalWrite(LedPin[3],HIGH); //turn the LED3 on only
81.         while(digitalRead(ResetBtnPin));
```

```
82.         }
83.     }
84. }
85. return 0;
86. }
```

## Code Explanation

```
9. void Alarm()
10. {
11.     for(int i=0;i<50;i++){
12.         digitalWrite(BeepPin,HIGH); //the buzzer sound
13.         delay(2);
14.         digitalWrite(BeepPin,LOW); //without sound
15.         delay(2);
16.     }
17. }
```

Define a function to control the buzzer. The buzzer rings when this function is called in the main function.

```
38. int flag = 1;
```

Define a flag to judge whether the answer device is in the state of answering. When flag = 0, it indicates that someone is currently scrambling, and others cannot continue to answer first; when flag = 1, it means that the reset button has been pressed, and a new round of answer rush can be conducted.

```
42.     if(digitalRead(ResetBtnPin) == 1)
43.     {
```

```
44.         flag = 1;
45.         digitalWrite(LedPin[0], HIGH); //Reset Led turns on
46.         digitalWrite(LedPin[1],LOW);
47.         digitalWrite(LedPin[2],LOW);
48.         digitalWrite(LedPin[3],LOW);
49.     }
```

If the reset button is detected to have been pressed, it means that the answer begins. Now set flag to 1 and let the referee LED light up, the rest of the LED lights out.

```
53.     if(digitalRead(BtnPin[0]) == 1)
54.     {
55.         flag = 0;
56.         digitalWrite(LedPin[0],LOW);
57.         Alarm(); //buzzer sound
58.         digitalWrite(LedPin[1],HIGH); //turn the LED1 on only
59.         digitalWrite(LedPin[2],LOW);
60.         digitalWrite(LedPin[3],LOW);
61.         while(digitalRead(ResetBtnPin));
62.     }
```

In the process of quick answering, if the first button is recognized to have been pressed, the flag is set to 0, and then no other buttons are detected. At this time, the buzzer alarms, indicating that someone has successfully responded, and the corresponding LED lights up. The identification codes of the remaining buttons are explained as above.

```
61.         while(digitalRead(ResetBtnPin));
```

Having executed the instruction of successful quick answer, it enters the loop to judge whether the button reset is pressed. Here, if the button reset is pressed, then the next round of quickfire answering begins.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 14_AnswerMachine.py
```

Now, first press button 4 to get started. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.

### Code

```
1. import RPi.GPIO as GPIO  
2. import time  
3.  
4. BeepPin = 17  
5. ResetBtnPin = 18  
6. BtnPin =(27,22,23)  
7. LedPin =(5,6,13,19)  
8.  
9. def setup():  
10.    GPIO.setmode(GPIO.BCM)
```

```
11.     GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
12.     GPIO.setup(ResetBtnPin, GPIO.IN)
13.     GPIO.setup(LedPin[0], GPIO.OUT, initial=GPIO.HIGH)
14.     for i in range(1,4):
15.         GPIO.setup(LedPin[i], GPIO.OUT, initial=GPIO.LOW)
16.     for i in range(0,3):
17.         GPIO.setup(BtnPin[i], GPIO.IN)
18.
19.def Alarm():
20.    for i in range(0,50):
21.        GPIO.output(BeepPin,GPIO.HIGH)
22.        time.sleep(0.003)
23.        GPIO.output(BeepPin,GPIO.LOW)
24.        time.sleep(0.003)
25.
26.def loop():
27.    flag = 1
28.    while True:
29.        if GPIO.input(ResetBtnPin) == 1:
30.            flag = 1
31.            GPIO.output(LedPin[0],GPIO.HIGH)
32.            GPIO.output(LedPin[1],GPIO.LOW)
33.            GPIO.output(LedPin[2],GPIO.LOW)
34.            GPIO.output(LedPin[3],GPIO.LOW)
```

```
35.     if flag == 1:  
36.         if GPIO.input.BtnPin[0]) == 1:  
37.             flag = 0  
38.             GPIO.output(LedPin[0],GPIO.LOW)  
39.             Alarm()  
40.             GPIO.output(LedPin[1],GPIO.HIGH)  
41.             GPIO.output(LedPin[2],GPIO.LOW)  
42.             GPIO.output(LedPin[3],GPIO.LOW)  
43.         elif GPIO.input.BtnPin[1]) == 1:  
44.             flag = 0  
45.             GPIO.output(LedPin[0],GPIO.LOW)  
46.             Alarm()  
47.             GPIO.output(LedPin[1],GPIO.LOW)  
48.             GPIO.output(LedPin[2],GPIO.HIGH)  
49.             GPIO.output(LedPin[3],GPIO.LOW)  
50.         elif GPIO.input.BtnPin[2]) == 1:  
51.             flag = 0  
52.             GPIO.output(LedPin[0],GPIO.LOW)  
53.             Alarm()  
54.             GPIO.output(LedPin[1],GPIO.LOW)  
55.             GPIO.output(LedPin[2],GPIO.LOW)  
56.             GPIO.output(LedPin[3],GPIO.HIGH)  
57.  
58. def destroy():
```

```
59.     # Turn off buzzer
60.     GPIO.output(BeepPin, GPIO.LOW)
61.     GPIO.output(LedPin[0],GPIO.LOW)
62.     GPIO.output(LedPin[1],GPIO.LOW)
63.     GPIO.output(LedPin[2],GPIO.LOW)
64.     GPIO.output(LedPin[3],GPIO.HIGH)
65.     # Release resource
66.     GPIO.cleanup()
67.
68.# If run this script directly, do:
69.if __name__ == '__main__':
70.    setup()
71.    try:
72.        loop()
73.    # When 'Ctrl+C' is pressed, the child program
74.    # destroy() will be executed.
75.    except KeyboardInterrupt:
76.        destroy()
```

## Code Explanation

```
19. def Alarm():
20.     for i in range(0,50):
21.         GPIO.output(BeepPin,GPIO.HIGH)
22.         time.sleep(0.003)
23.         GPIO.output(BeepPin,GPIO.LOW)
24.         time.sleep(0.003)
```

Define a function to control the buzzer. The buzzer rings when this function is called in the function **main**.

```
27.     flag = 1;
```

Define a flag bit to judge whether the responder is in the state of answering. When **flag** = 0, it indicates that someone is currently scrambling, and others cannot continue to answer first; when **flag** = 1, it means that the reset button has been pressed, and a new round of answer rush can be conducted.

```
29.     if GPIO.input(ResetBtnPin) == 1:
30.         flag = 1
31.         GPIO.output(LedPin[0],GPIO.HIGH)
32.         GPIO.output(LedPin[1],GPIO.LOW)
33.         GPIO.output(LedPin[2],GPIO.LOW)
34.         GPIO.output(LedPin[3],GPIO.LOW)
```

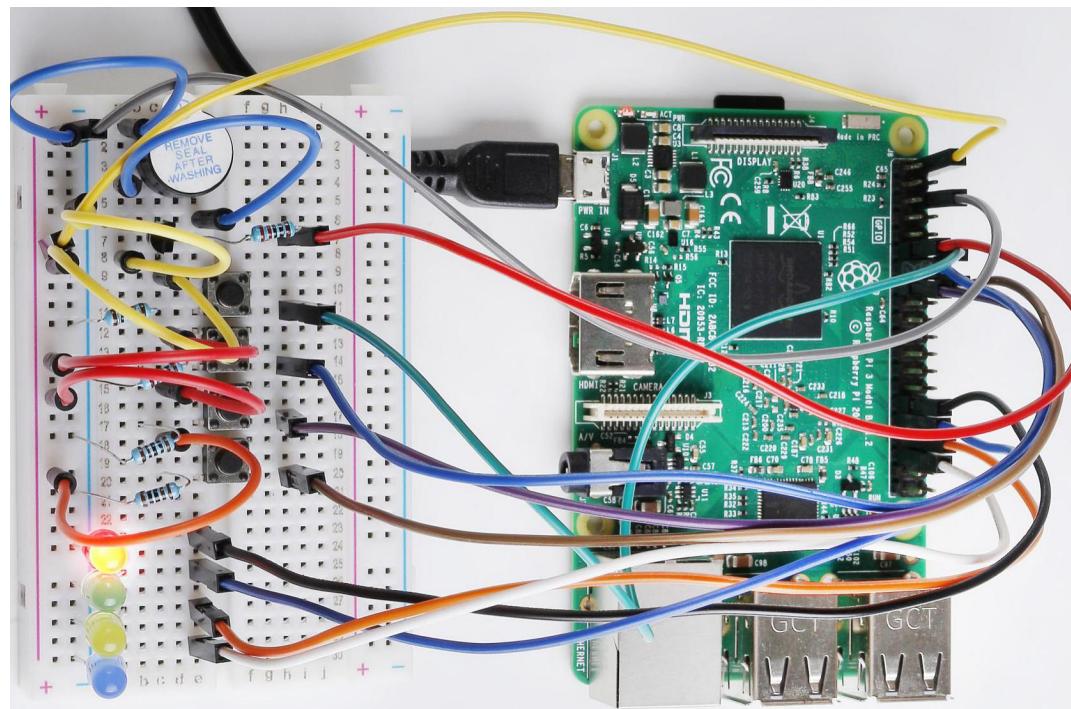
If the recognition that reset button has been pressed is done, it means that answer begins. Now, set flag to 1, and let the referee LED light up, other LEDs light out.

```
36.     if GPIO.input.BtnPin[0]) == 1:
37.         flag = 0
```

```
38.     GPIO.output(LedPin[0],GPIO.LOW)
39.     Alarm()
40.     GPIO.output(LedPin[1],GPIO.HIGH)
41.     GPIO.output(LedPin[2],GPIO.LOW)
42.     GPIO.output(LedPin[3],GPIO.LOW)
```

In the process of quick answering, if the first button is recognized to have been pressed, the flag is set to 0, and then no other buttons are detected. At this time, the buzzer alarms, indicating that there is a successful response, and the corresponding LED lights up. The identification codes of the remaining buttons are explained as above.

## Phenomenon Picture

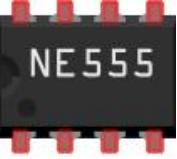


## Lesson 15 NE555 Timer

### Introduction

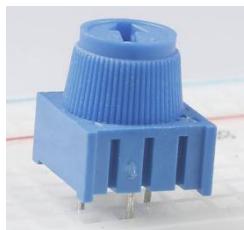
The NE555 Timer, a mixed circuit composed of analog and digital circuits, integrates analog and logical functions into an independent IC, thus tremendously expanding the applications of analog integrated circuits. It is widely used in various timers, pulse generators, and oscillators. In this experiment, the Raspberry Pi is used to test the frequencies of square waves generated by the 555 oscillating circuit and show them on terminal windows.

### Newly Added Components

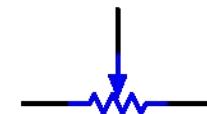
1 * Potentiometer (10KΩ)	2 * 104 ceramic capacitor	1 * NE555	1 * Resistor(10kΩ)
			

### Principle

#### Potentiometer



Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation. Potentiometer usually consists of resistor and movable brush. When the brush is moving along the resistor, there is a certain resistance or voltage output depending on the displacement.



The functions of the potentiometer in the circuit are as follows:

1. Serving as a voltage divider

Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the distance it moves.

2. Serving as a rheostat

When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value caused by moving contact.

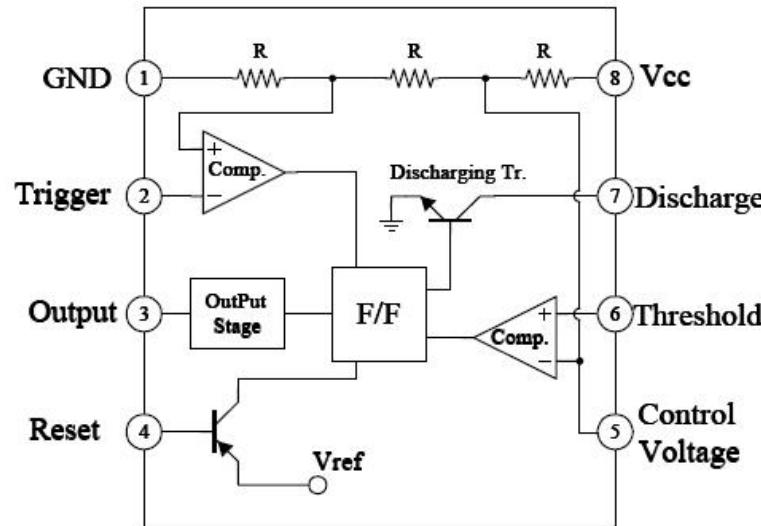
3. Serving as a current controller

When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

## 555 IC

The 555 IC was originally used as a timer, hence the name 555 time base circuit. It is now widely used in various electronic products because of its reliability, convenience, and low price. The 555 is a complex hybrid circuit with dozens of components such as a divider, comparator, basic R-S trigger, discharge tube, and buffer.

Pins and functions:



As shown in the picture, the pins are set dual in-line with the 8-pin package.

- Pin 1 (**GND**): the ground
- Pin 2 (**TRIGGER**): when the voltage at the pin reduces to 1/3 of the VCC (or the threshold defined by the control board), the output terminal sends out a High level

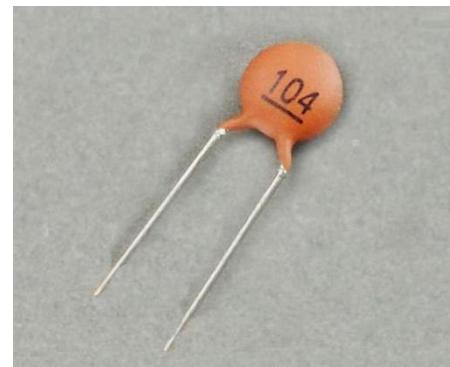
Pin 3 (**OUTPUT**): outputs High or Low, two states 0 and 1 decided by the input electrical level; maximum output current approx. 200mA at High

- Pin 4 (**RESET**): when a Low level is received at the pin, the timer will be reset and the output will return to Low level; usually connected to positive pole or neglected
- Pin 5 (**CONTROL VOLTAGE**): to control the threshold voltage of the chip (if it skips connection, by default, the threshold voltage is 1/3 VCC and 2/3 VCC)
- Pin 6 (**THRESHOLD**): when the voltage at the pin increases to 2/3 VCC (or the threshold defined by the

control board), the output terminal sends out a High level.

- Pin 7 (**DISCHARGE**): output synchronized with Pin 3, with the same logical level; but this pin does not output current, so pin 3 is the real High (or Low) when pin 7 is the virtual High (or Low); connected to the open collector (OC) inside to discharge the capacitor.
- Pin 8 (**VCC**): positive terminal for the NE555 timer IC, ranging +4.5V to +16V
- The NE555 timer works under the monostable, astable and bistable modes. In this experiment, apply it under the astable mode, which means it works as an oscillator, as shown below:

## Cap



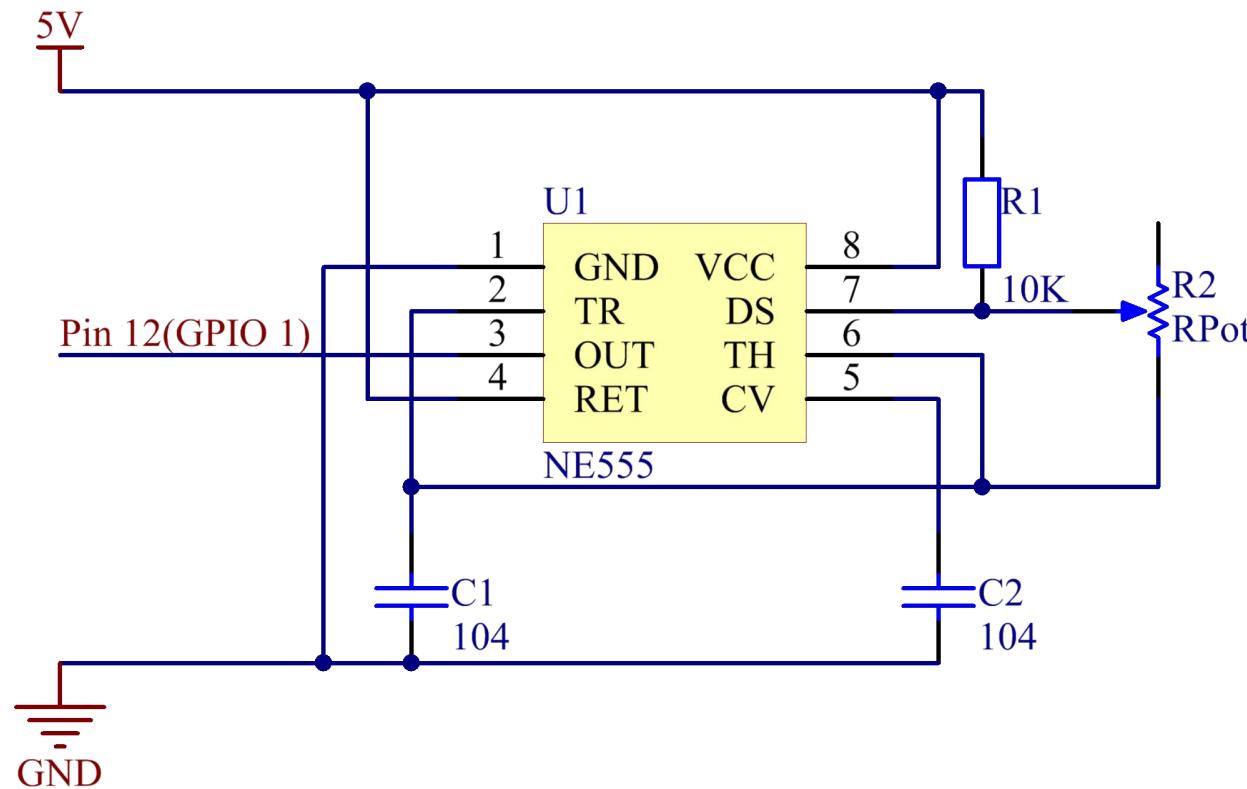
A ceramic capacitor is a capacitor that is made of ceramic material and works as a dielectric. It is coated with a metal film on the surface of the ceramic and sintered at a high temperature. The ceramic capacitor is commonly used in high-stability oscillator circuits as loops, bypass capacitors, and pad capacitors. It is a non-polar capacitor, so this capacitor does not need to distinguish between positive and negative during installation.

In the circuit of this lesson, the main function of the ceramic capacitor, high-frequency filtering is to remove some clutter that may occur in the working process of the NE555 chip, so that the waveform is more stable.

## Schematic Diagram

Build the circuit according to the following schematic diagram.

wiringPi	Physical	BCM
1	Pin12	18



## Working Process:

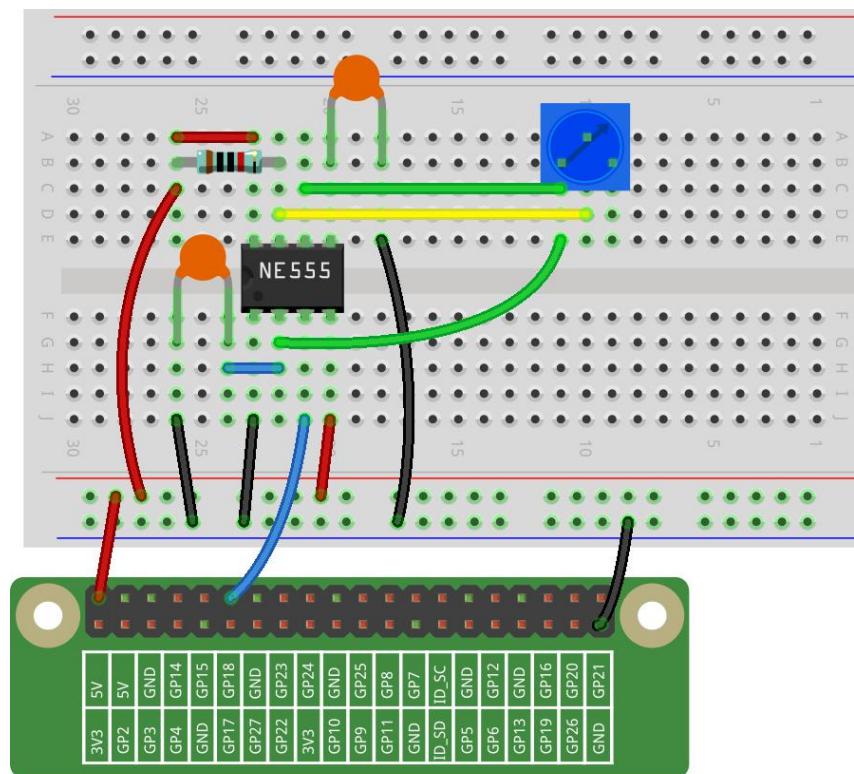
The oscillator starts to shake once the circuit is power on. During energizing, since the voltage at C1 cannot change abruptly, which means pin 2 is Low level initially, set the timer to 1, so pin 3 is **High level**. The capacitor C1 **charges** via R1 and R2 in a time span:

When the voltage at C1 reaches the threshold  $2/3V_{cc}$ , the timer is reset and pin 3 is **Low level**. Then C1 **discharges** via R2 till  $2/3V_{cc}$  in a time span:

Then the capacitor is recharged and the output voltage flips again:

## Build the Circuit

NE555	Components	Raspberry Pi
2	Cap1	GND
5	Cap2	GND
6	Potentiometer pin1	
7	Potentiometer pin2	
7	Res	5V
3		Pin 12
1	GND	GND
4,8		5V



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_15_NE555_Timer
```

2. Compile the code.

```
gcc 15_NE555_Timer.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

When the code is running, you will see the number of pulses on the display screen and the level of pin3 in NE555 at this time.

### Code

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <errno.h>
4. #include <stdlib.h>
5. #include <wiringPi.h>
6.
7. #define OutPin 1
8.
9. static volatile int globalCounter = 0 ;
```

```
10.  
11. void exInt0_ISR(void) //GPIO 1 interrupt service routine  
12. {  
13.     ++globalCounter;  
14. }  
15.  
16. int main (void)  
17. {  
18.     if(wiringPiSetup() < 0){  
19.         fprintf(stderr, "Unable to setup wiringPi:%s\n", strerror(errno));  
20.         return 1;  
21.     }  
22.  
23.     delay(2000);  
24.     pinMode(OutPin,INPUT);  
25.     pullUpDnControl(OutPin,PUD_UP);  
26.     wiringPiISR(OutPin, INT_EDGE_FALLING, &exInt0_ISR);  
27.  
28.     while(1){  
29.         printf("Current pluse number is : %d, %d\n", globalCounter,digitalRead(OutPin));  
30.         delay(100);  
31.     }  
32.     return 0;  
33. }
```

## Code Explanation

9. `static volatile int globalCounter = 0 ;`

Define a variable to record the number of pulses, and initialize the number of pulses to **0**.

11. `void exInt0_ISR(void)`  
12. {  
13. `++globalCounter;`  
14. }

Set an external interrupt function and **globalCounter** will automatically +1 when an interrupt occurs.

24. `pinMode(OutPin,INPUT);`  
25. `pullUpDnControl(OutPin,PUD_UP);`

Set the out pin of NE555 to **INPUT** mode, then let the pin be in pull-up state (1).

26. `wiringPiISR(OutPin, INT_EDGE_FALLING, &exInt0_ISR);`

Set an interrupt in **OutPin**, when the value of **OutPin** changes from 1 to 0. Then call the `exInt0_ISR()` function to let the variable **globalCounter** add 1.

29. `printf("Current pluse number is : %d, %d\n", globalCounter,digitalRead(OutPin));`

Print out the number of pulses, **globalCounter** and the value of **OutPin** at this time.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 15_NE555.py
```

When the code is running, you can see the number of pulses on the display.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. SigPin = 18
5.
6. g_count = 0
7.
8. def count(ev=None):
9.     global g_count
10.    g_count += 1
11.
12. def setup():
13.     GPIO.setmode(GPIO.BCM)
```

```
14.     GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
15.     GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # wait for rising
16.
17. def main():
18.     while True:
19.         print ('g_count = %d' % g_count)
20.         time.sleep(0.01)
21.
22. def destroy():
23.     GPIO.cleanup()      # Release resource
24.
25. if __name__ == '__main__':    # Program start from here
26.     setup()
27.     try:
28.         main()
29.     except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
30.         destroy()
```

## Code Explanation

```
6.     g_count = 0
```

Define a variable to record the number of pulses, and initialize the number of pulses to **0**.

```
7. def count(ev=None):
8.     global g_count
```

```
9.     g_count += 1
```

This function will change the value of the global variable **g\_count**.

```
14.     GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Set the **SigPin** to input mode and pull up to high level(3.3V).

```
15.     GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count)
```

Set an interrupt in **SigPin**, when the value of **SigPin** changes from 0 to 1. Then call the **count()** function to let the variable **g\_count** add 1.

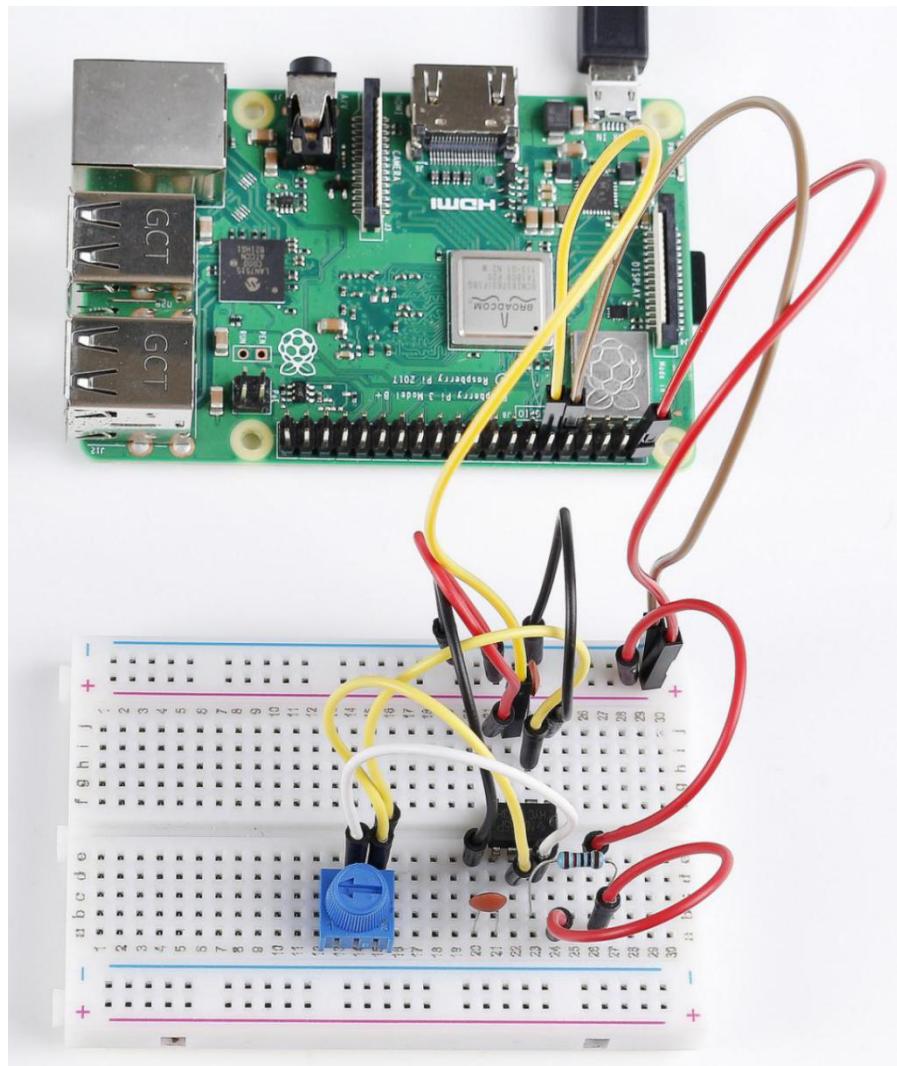
```
18.     while True:
```

```
19.         print ('g_count = %d' % g_count)
```

```
20.         time.sleep(0.01)
```

Print out the value of the number of pulse **g\_count** at an interval of **0.01s**.

## Phenomenon Picture



# Lesson 16 Servo

## Introduction

Servo is a type of geared motor that can only rotate 180 degrees. It is controlled by sending electrical pulses from your board. These pulses tell the servo what position it should move to.

A servo has three wires: the brown wire is GND, the red one is VCC, and the orange one is signal line.

## Newly Added Components



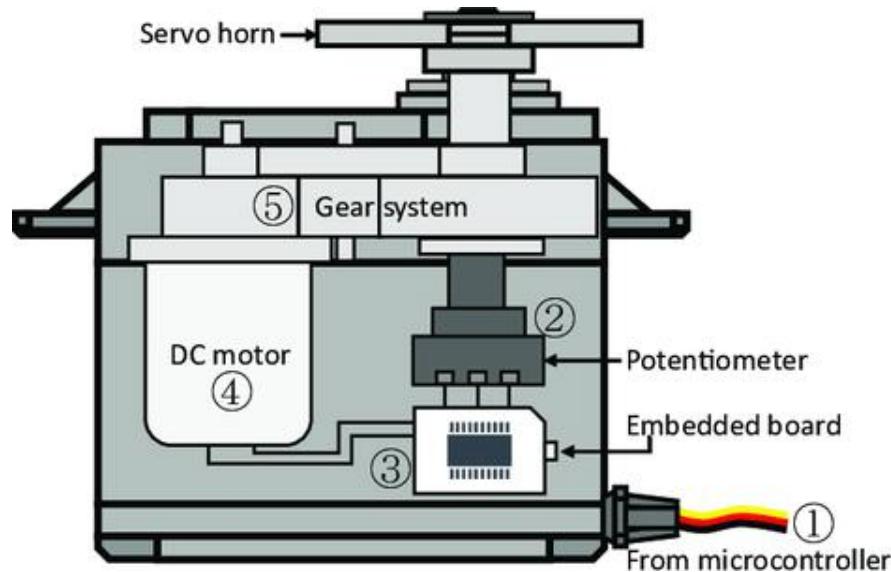
## Principle

### Servo

A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

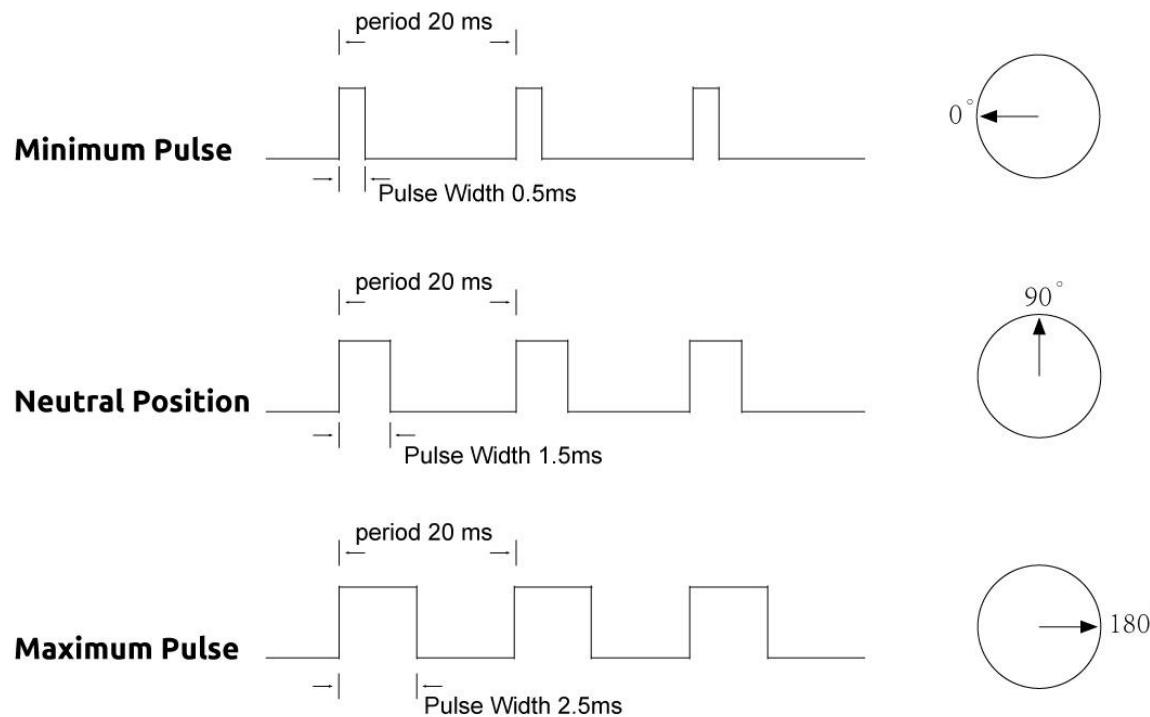


It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft and potentiometer of the servo are connected together. When the shaft rotates, it drives the potentiometer, so the potentiometer outputs a voltage signal to the embedded board. Then the board determines the direction and speed of rotation based on the current position, so it can stop exactly at the right position as defined and hold there.



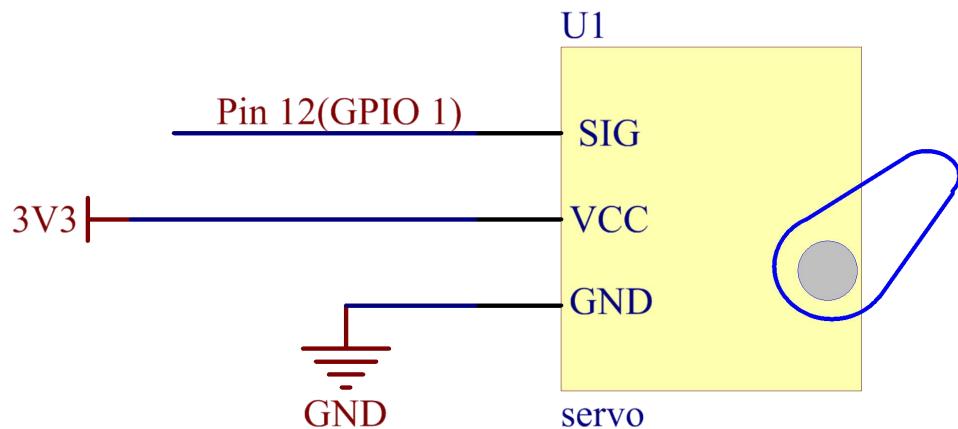
The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position).

When a pulse is sent to a servo that is less than 1.5 ms, the servo rotates to a position and holds its output shaft some number of degrees counterclockwise from the neutral point. When the pulse is wider than 1.5 ms the opposite occurs. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. **Generally the minimum pulse will be about 0.5 ms wide and the maximum pulse will be 2.5 ms wide.**



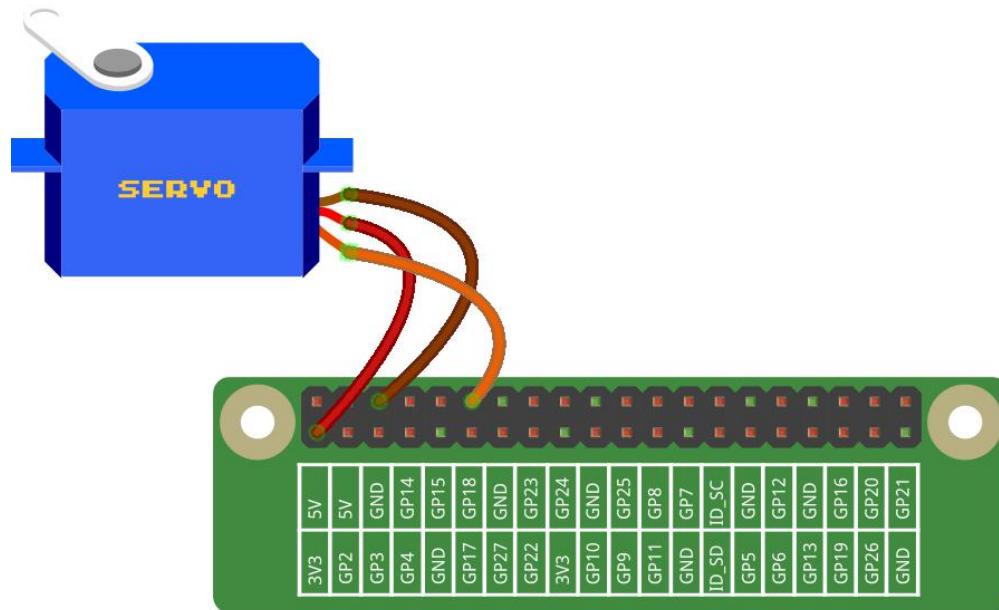
## Schematic Diagram

wiringPi	Physical	BCM
1	Pin12	18



## Build the Circuit

Note: Connect the brown to GND, Red to VCC, Orange to pin12 of the control board.



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_16_Servo
```

2. Compile the code.

```
gcc 16_Servo.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

### Code

```
1. #include <wiringPi.h>
2. #include <softPwm.h>
3. #include <stdio.h>
4.
5. #define servoPin    1
6. long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
7.     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
8. }
9. void servoWrite(int pin, int angle){ //Specif a certain rotation angle (0-180) for the servo
```

```
10.    if(angle < 0)
11.        angle = 0;
12.    if(angle > 180)
13.        angle = 180;
14.    softPwmWrite(pin,map(angle,0,180,5,25));
15. }
16.
17. int main(void)
18. {
19.     int i;
20.
21.     if(wiringPiSetup() == -1){ //when initialize wiring failservo,print message to screen
22.         printf("setup wiringPi failed !");
23.         return 1;
24.     }
25.     softPwmCreate(servopin, 0, 200);      //initialize PWM pin of servo
26.     while(1){
27.         for(i=0;i<181;i++){
28.             servoWrite(servopin,i);
29.             delay(1);
30.         }
31.         delay(500);
32.         for(i=181;i>-1;i--){
33.             servoWrite(servopin,i);
```

```
34.         delay(1);  
35.     }  
36.     delay(500);  
37. }  
38. return 0;  
39. }
```

## Code Explanation

```
6. long map(long value, long fromLow, long fromHigh, long toLow, long toHigh){  
7.     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;  
8. }
```

Create a **map()** function to map value in the following code.

```
9. void servoWrite(int pin, int angle){ //Specif a certain rotation angle (0-180) for the servo  
10.    if(angle < 0)  
11.        angle = 0;  
12.    if(angle > 180)  
13.        angle = 180;  
14.    softPwmWrite(pin,map(angle,0,180,5,25));  
15. }
```

Define a function to limit the angle of the servo to 0 to 180 in order to set the angle of servo.

```
softPwmWrite(pin,map(angle,0,180,5,25));
```

This function can change the duty cycle of the PWM pin.

To make the servo rotate to  $0 \sim 180^\circ$ , the pulse width should change within the range of  $0.5\text{ms} \sim 2.5\text{ms}$  when the period is  $20\text{ms}$ ; in the function, **softPwmCreate()**, we have set that the period is  $200 \times 100\text{us} = 20\text{ms}$ , thus we need to map  $0 \sim 180$  to  $5 \times 100\text{us} \sim 25 \times 100\text{us}$ .

```
25. softPwmCreate(servoPin, 0, 200);
```

The function is to use softwares to create a PWM pin, **servoPin**, then the initial pulse widths of them are set to **0**, and the period of PWM is **200x100us**.

```
27. for(i=0;i<181;i++){  
28.     servoWrite(servoPin,i);  
29.     delay(1);  
30. }
```

In a **for** loop, we want servo to rotate from 0 degrees to 180 degrees.

```
32. for(i=181;i>-1;i--){  
33.     servoWrite(servoPin,i);  
34.     delay(1);  
35. }
```

In a **for** loop, we want servo to rotate from 180 degrees to 0 degrees.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 16_Servo.py
```

After the program is executed, the servo will rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees, circularly.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3.
4. servoPin = 12
5.
6. def map( value, fromLow, fromHigh, toLow, toHigh):
7.     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
8.
9. def setup():
10.    global p
11.    GPIO.setmode(GPIO.BRD)
12.    GPIO.setup(servoPin, GPIO.OUT)
```

```
13.     GPIO.output(servoPin, GPIO.LOW)
14.
15.     p = GPIO.PWM(servoPin, 50)
16.     p.start(0)
17.
18. def servowrite(angle):      # make the servo rotate to specific angle (0-180 degrees)
19.     if(angle<0):
20.         angle = 0
21.     elif(angle > 180):
22.         angle = 180
23.     p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))
24.
25. def loop():
26.     while True:
27.         for i in range(0, 181, 1):
28.             servowrite(i)
29.             time.sleep(0.001)
30.             time.sleep(0.5)
31.         for i in range(180, -1, -1):
32.             servowrite(i)
33.             time.sleep(0.001)
34.             time.sleep(0.5)
35.
36. def destroy():
```

```
37.     p.stop()
38.     GPIO.cleanup()
39.
40. if __name__ == '__main__':      #Program start from here
41.     setup()
42.     try:
43.         loop()
44.     except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will be executed.
45.         destroy()
```

## Code Explanation

```
6. def map( value, fromLow, fromHigh, toLow, toHigh):
7.     return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow
```

Create a **map()** function to map value in the following code.

```
15.     p = GPIO.PWM(servoPin, 50)
16.     p.start(0)
```

Set the **servoPin** to PWM pin, then the frequency to **50hz**, and the period to 20ms.

**p.start(0):** Run the PWM function, and set the initial value to **0**.

```
18. def servoWrite(angle):
19.     if(angle<0):
20.         angle = 0
```

```
21.     elif(angle > 180):  
22.         angle = 180  
23.         p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))#map the angle to duty cycle and output it
```

Create a function, **servoWrite()** to write angle that ranges from 0 to 180 into the servo.

```
p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))
```

This function can change the duty cycle of the PWM.

To render a range **0 ~ 180°** to the servo, the pulse width of the servo is set to **0.5ms-2.5ms**.

In the previous codes, the period of PWM was set to 20ms, thus the duty cycle of PWM is  $(0.5/20)\%-(2.5/20)\%$ , and the range 0 ~ 180 is mapped to **2.5 ~ 12.5**.

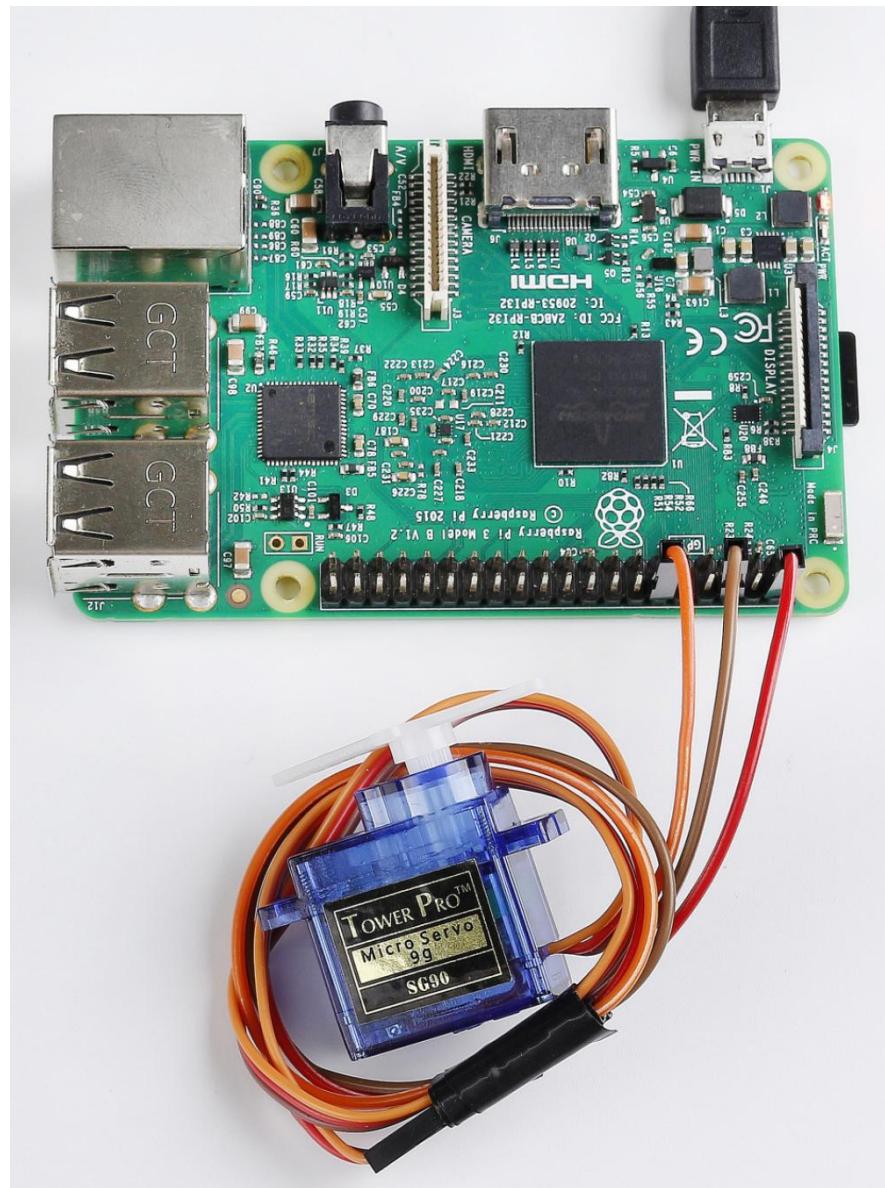
```
24.     for i in range(0, 181, 1): #make servo rotate from 0 to 180 deg  
25.         servoWrite(i) # Write to servo  
26.         time.sleep(0.001)
```

In a **for** loop, we want servo to rotate from **0** degrees to **180** degrees.

```
27.     for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg  
28.         servoWrite(i)  
29.         time.sleep(0.001)
```

In a **for** loop, we want servo to rotate from **180** degrees to **0** degrees.

## Phenomenon Picture



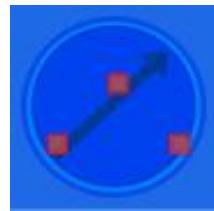
# Lesson 17 LCD1602

## Introduction

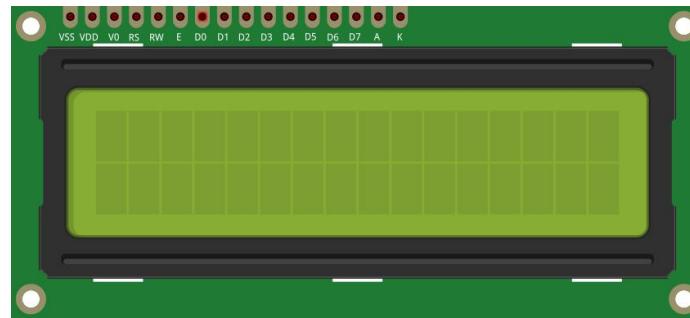
In this lesson, we will learn how to use an LCD1602 to display characters and strings. LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters and so on. It's composed of 5x7 or 5x11 dot matrix positions; each position can display one character. Now let's check more details!

## Newly Added Components

1 \* Potentiometer (10kΩ)

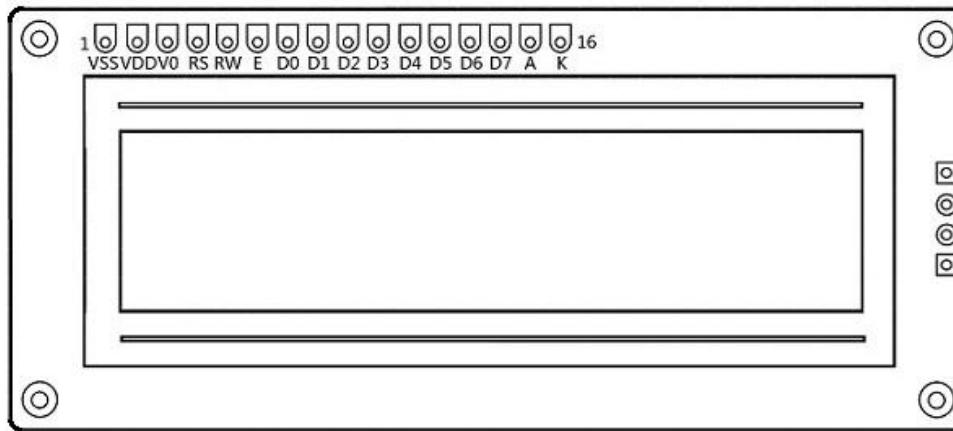


1 \* LCD1602



## Principle

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the Raspberry Pi are almost completely occupied. If you want to connect more sensors, there will be no ports available. Therefore, the four-port connection is used here for better application.



## Pins of LCD1602 and their Functions

**VSS:** connected to ground.

**VDD:** connected to a +5V power supply.

**VO:** to adjust the contrast.

**RS:** A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

**R/W:** A Read/Write pin to select between reading and writing mode.

**E:** An enabling pin that reads the information when High level (1) is received. The instructions are run when the signal changes from High level to Low level.

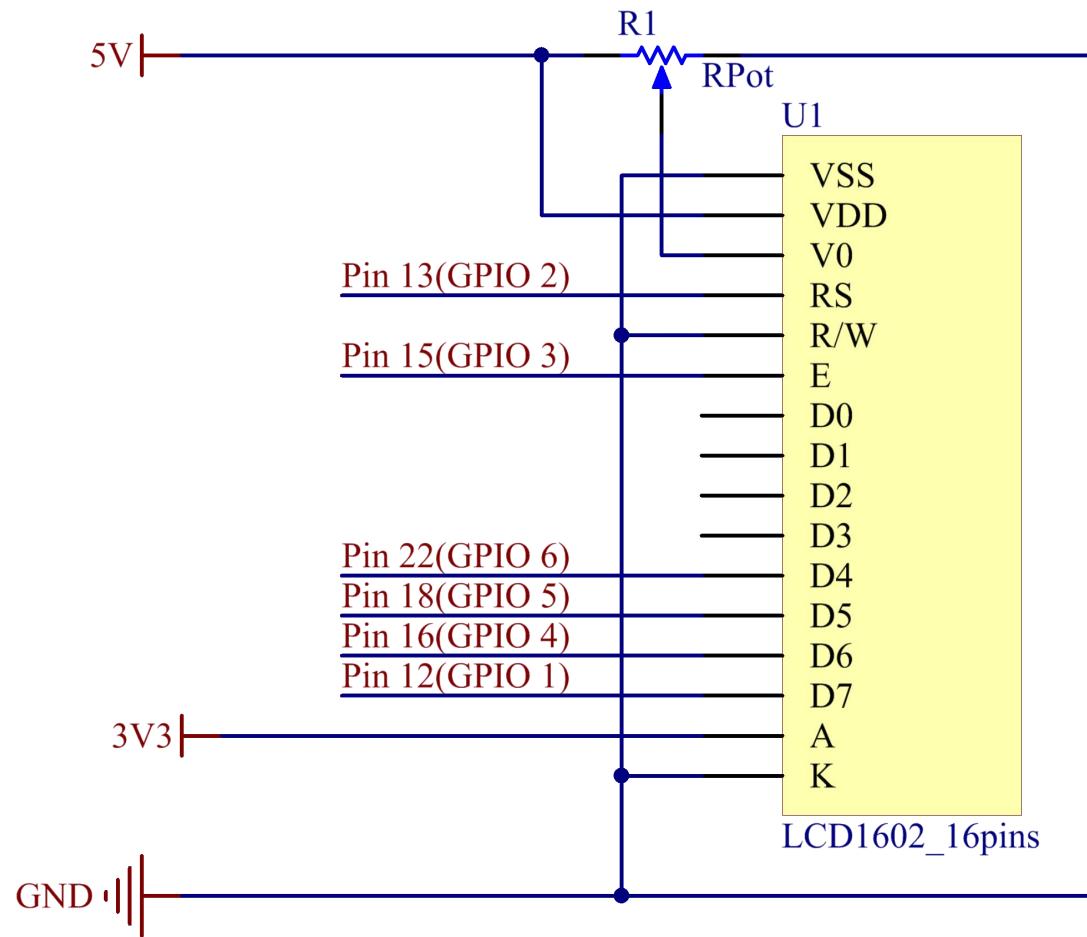
**D0-D7:** to read and write data.

**A and K:** Pins that control the LCD backlight. Connect K to GND and A to 3.3v. Open the backlight and you will see clear characters in a comparatively dark environment.

## Schematic Diagram

Connect **K** to **GND** and **A** to **3.3 V**, and then the backlight of the LCD1602 will be turned on. Connect **VSS** to **GND** and the **LCD1602** to the power source. Connect **VO** to the middle pin of the potentiometer - with it you can adjust the contrast of the screen display. Connect **RS** to **Pin 13** and **R/W** pin to **GND**. Connect **E** to **Pin 15** and the characters displayed on the LCD1602 are controlled by **D4-D7**. For programming, it is optimized by calling function libraries.

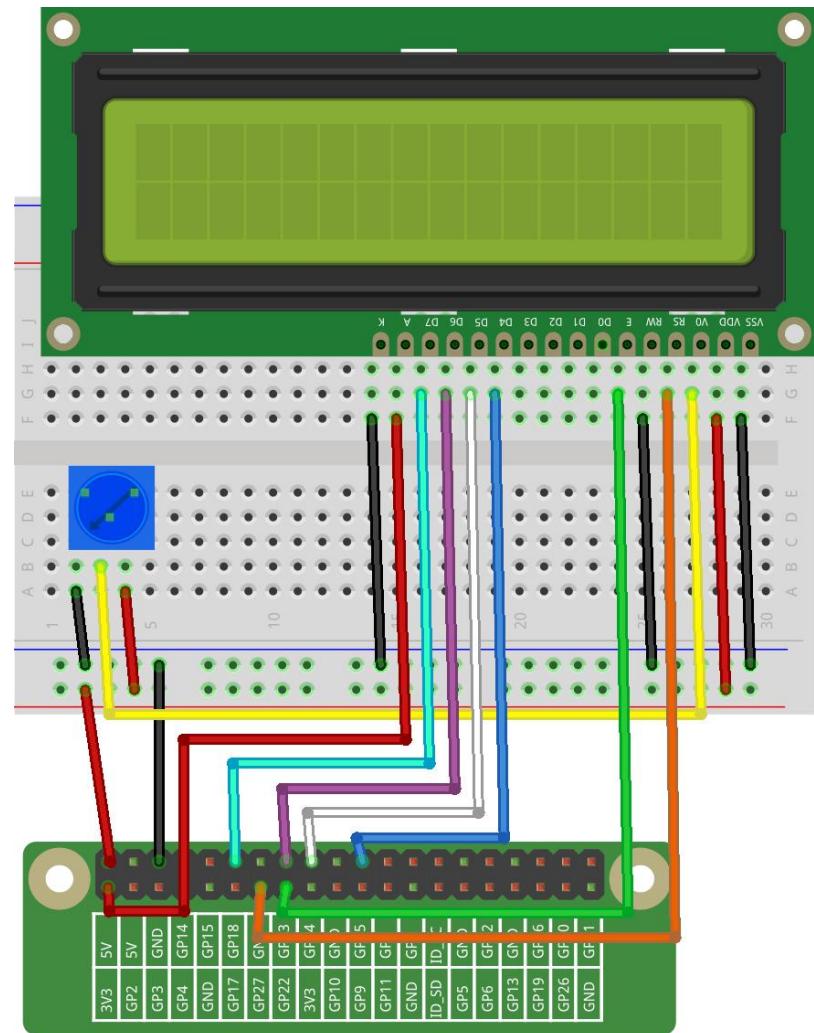
wiringPi	Physical	BCM
1	Pin12	18
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24
6	Pin22	25



## Build the Circuit

Note: Make sure the pins are connected correctly. Otherwise, characters will not be displayed properly. You may need to adjust the potentiometer till the LCD1602 can display clearly.

LCD1602	Components	RPi
VSS		GND
VDD		5V
V0	Potentiometer pin2	
RS		Pin 13
R/W		GND
E		Pin 15
D4		Pin 22
D5		Pin 18
D6		Pin 16
D7		Pin 12
A		3V3
K		GND
	Potentiometer pin1	5V
	Potentiometer pin3	GND



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_17_LCD1602
```

2. Compile the code.

```
gcc 17_Lcd1602.c -lwiringPiDev -lwiringPi
```

**Note:** In order to use the LCD driver in the wiringPi devLib, you need to use -lwiringPiDev at compile time.

3. Run the executable file.

```
sudo ./a.out
```

You may see the "SunFounder" and "hello, world" appear one by one on the LCD.

### Code

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <wiringPi.h>
4. #include <lcd.h>
5.
6. const unsigned char Buf[] = "---SUNFOUNDER---";
7. const unsigned char myBuf[] = " sunfounder.com";
8.
9. int main(void)
```

```
10. {
11.     int fd;
12.     int i;
13.
14.     if(wiringPiSetup() == -1){
15.         exit(1);
16.     }
17.
18.     fd = lcdInit(2,16,4,2,3, 0,0,0,0,6,5,4,1); //see /usr/local/include/lcd.h
19.     printf("%d", fd);
20.     if (fd == -1){
21.         printf("lcdInit 1 failed\n");
22.         return 1;
23.     }
24.
25.     delay(1000);
26.     lcdClear(fd);
27.     lcdPosition(fd, 0, 0);
28.     lcdPuts(fd, "Welcome To--->");
29.     lcdPosition(fd, 0, 1);
30.     lcdPuts(fd, "sunfounder.com");
31.     delay(1000);
32.     lcdClear(fd);
33.
```

```
34.     while(1){  
35.         lcdClear(fd);  
36.         for(i=0; i<16; i++){  
37.             lcdPosition(fd, i, 0);  
38.             lcdPutchar(fd, *(myBuf+i));  
39.             delay(100);  
40.         }  
41.         for(i=0;i<sizeof(Buf)-1;i++){  
42.             lcdPosition(fd, i, 1);  
43.             lcdPutchar(fd, *(Buf+i));  
44.             delay(200);  
45.         }  
46.         delay(500);  
47.     }  
48.     return 0;  
49. }
```

## Code Explanation

```
4. #include <lcd.h>
```

This is a library that integrates LCD1602 functional functions, in which functions are defined such as `LcdClear()`, `LcdPosition()`, `LcdPuts()`, and so on. These functions can be called directly after importing into the library.

```
18. fd = lcdInit(2,16,4,2,3, 0,0,0,0,6,5,4,1); //see /usr/local/include/lcd.h  
19. printf("%d", fd);
```

```
20.     if (fd == -1){  
21.         printf("lcdInit 1 failed\n") ;  
22.     return 1;
```

Initialize the lcd1602. The prototype of `lcdInit()` is as follows:

```
int lcdInit (int rows, int cols, int bits, int rs, int strb,int d0, int d1, int d2, int d3, int  
d4, int d5, int d6, int d7) ;
```

This is the main initialisation function and must be called before you use any other LCD functions.

**Rows** and **cols** are the rows and columns on the display (e.g. 2, 16 or 4,20). **Bits** is the number of bits wide on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the display RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the Pi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the ‘handle’ to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault. (Usually incorrect parameters)

```
26.     lcdClear(fd);
```

This function is used to clear the lcd screen. After calling this function, all information displayed on the screen will be cleared.

```
27.     lcdPosition(fd, 0, 0);
```

Set the position of the cursor at row 0 and col 0 (in fact it's the first line and first column) for subsequent text entry.

The prototype of **lcdpostion** function is as follows:

```
lcdPosition (int handle, int x, int y) ;
```

Set the position of the cursor for subsequent text entry. **x** is the column and **0** is the left-most edge. **y** is the line and **0** is the top line.

```
28.     lcdPuts(fd, "Welcome To--->");
```

Display "**Welcome To--->**" at the specified location of LCD1602.

```
36.     for(i=0; i<16; i++){
37.         lcdPosition(fd, i, 0);
38.         lcdPutchar(fd, *(myBuf+i));
39.         delay(100);
40.     }
```

Use the **lcdPosition()** function to place the cursor at col **i** and row 0(the top line ) for subsequent text entry. Then the characters in the array **myBuf []** are displayed one by one to the LCD1602.

\* is the address of myBuf, the real address of characters stored in memory. After calling `lcdPutchar(fd, *(myBuf+ i))`, the program will find the real address of the character, read the information stored in the address, and display it on the LCD screen.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 17_Lcd1602.py
```

You may see the "SunFounder" and "hello, world" appear one by one on the LCD.

### Code

```
1. from time import sleep
2.
3. def main():
4.     global lcd
5.     lcd = LCD()
6.     line0 = " sunfounder.com"
7.     line1 = "---SUNFOUNDER---"
8.
9.     lcd.clear()
10.    lcd.message("Welcome to --->\n sunfounder.com")
11.    sleep(3)
12.
13.    while True:
```

```
14.     lcd.begin(0, 2)
15.     lcd.clear()
16.     for i in range(0, len(line0)):
17.         lcd.setCursor(i, 0)
18.         lcd.message(line0[i])
19.         sleep(0.1)
20.     for i in range(0, len(line1)):
21.         lcd.setCursor(i, 1)
22.         lcd.message(line1[i])
23.         sleep(0.1)
24.     sleep(1)
25.
26.if __name__ == '__main__':
27.    try:
28.        main()
29.    except KeyboardInterrupt:
30.        lcd.clear()
31.        lcd.destroy()
```

Note: Because the source code contains so many definitions, we only list few code here. Please download the complete code from the address marked in the document.

## Code Explanation

```
6.     line0 = " sunfounder.com"  
7.     line1 = "---SUNFOUNDER---"
```

Define 2 lines of characters that will be displayed on the LCD 1602.

```
10.    lcd.message("Welcome to --->\n sunfounder.com")
```

On LCD1602, "**Welcome to --->\n sunfounder.com**" pops up.

```
15.    lcd.begin(0, 2)
```

Initializes the LCD screen and specifies the dimensions (width and height) of the display. begin() function needs to be called before any other LCD library commands.

```
16.    lcd.clear()
```

This function is used to clear the lcd screen. After calling this function, all information displayed on the screen will be cleared.

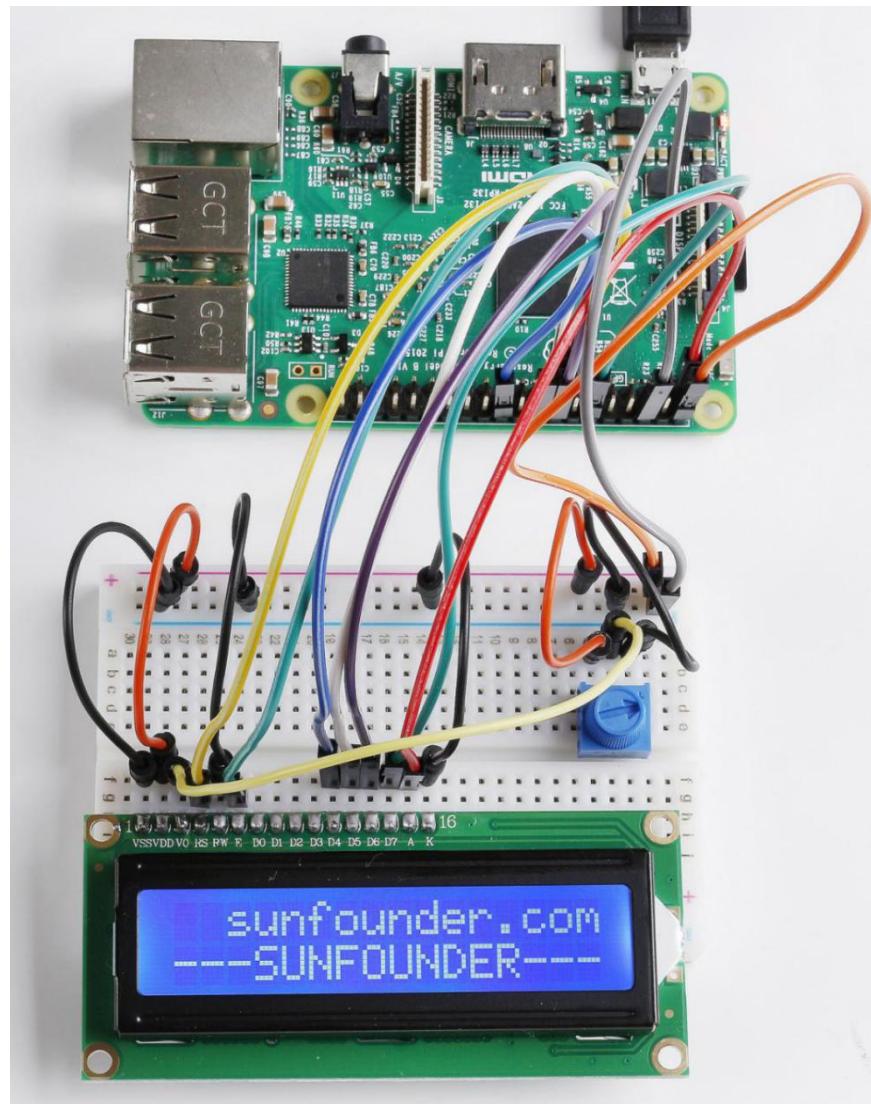
```
18.    lcd.setCursor(i, 0)
```

Set the position of the cursor at col i and row 0 (the first line) for subsequent text entry.

```
19.    lcd.message(line0[i])
```

The characters in the array **line0[]** will be displayed at the specified location one by one.

## Phenomenon Picture

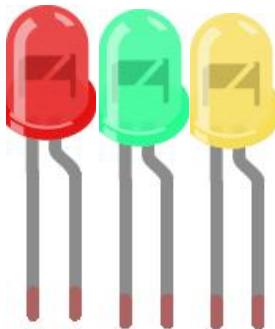


# Lesson 18 Driving LEDs by 74HC595

## Introduction

In this lesson, we will learn how to use 74HC595 to make eight LEDs blink regularly. Now let's get started!

## Newly Added Components

8 * LED	1 * 74HC595	8 * Resistor (220Ω)
		

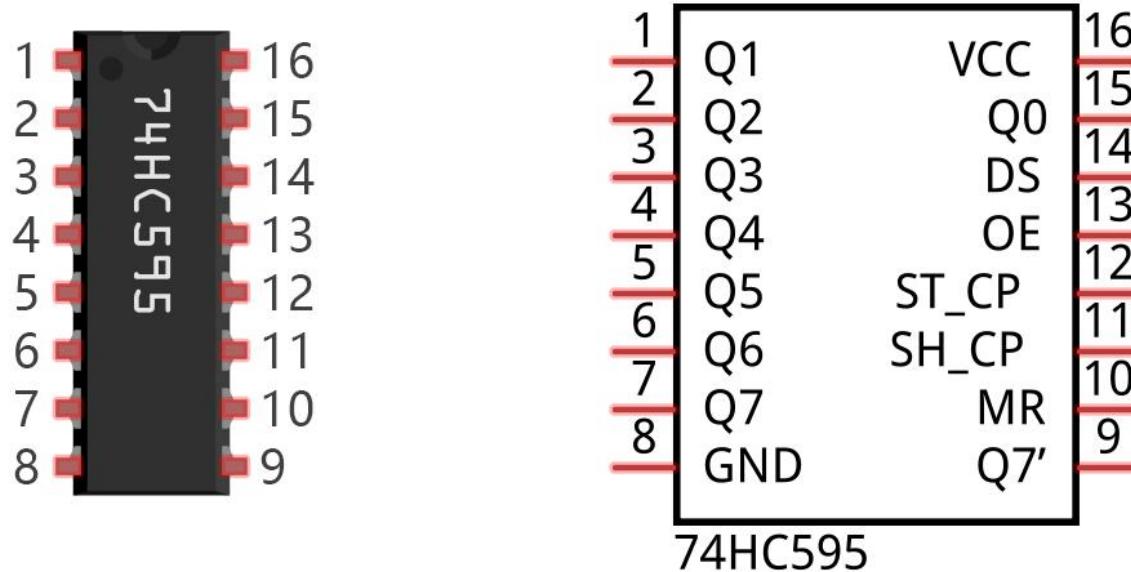
## Principle

### 74HC595

The 74HC595 consists of an 8 – bit shift register and a storage register with three – state parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU.

When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp. If the two clocks are connected together, the shift register is always one pulse earlier than the memory register. There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register. The memory register outputs a

Bus with a parallel 8-bit and in three states. When OE is enabled (low level), the data in memory register is output to the bus.



### Pins of 74HC595 and their Functions:

**Q0-Q7:** 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-Segment Display directly.

**Q7':** Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series.

**MR:** Reset pin, active at low level;

**SHcp:** Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

**STcp**: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

**OE**: Output enable pin, active at low level.

**DS**: Serial data input pin.

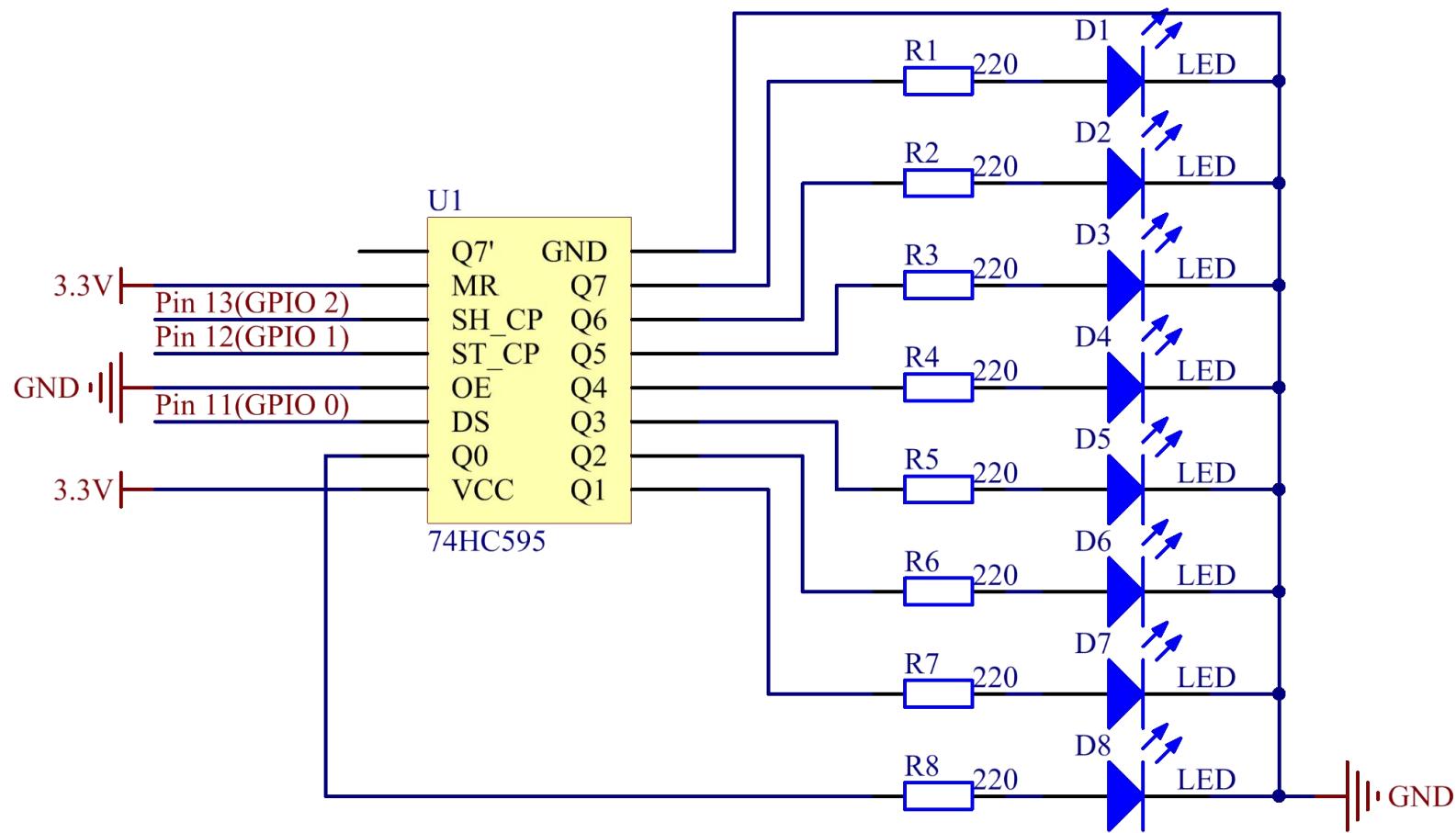
**VCC**: Positive supply voltage.

**GND**: Ground.

## Schematic Diagram

In the experiment **MR** is connected to **3.3V** (HIGH Level) and **OE** to **GND** (LOW Level). Therefore, the data is input into the rising edge of **SHcp** and enters the memory register through the rising edge. In the rising edge of the **SHcp**, the data in the shift register moves successively one bit in one time, i.e. data in **Q1** moves to **Q2**, and so forth. In the rising edge of **STcp**, data in the shift register moves into the memory register. All data will be moved to the memory register 8 times. Then the data in the memory register is output to the bus (**Q0-Q7**).

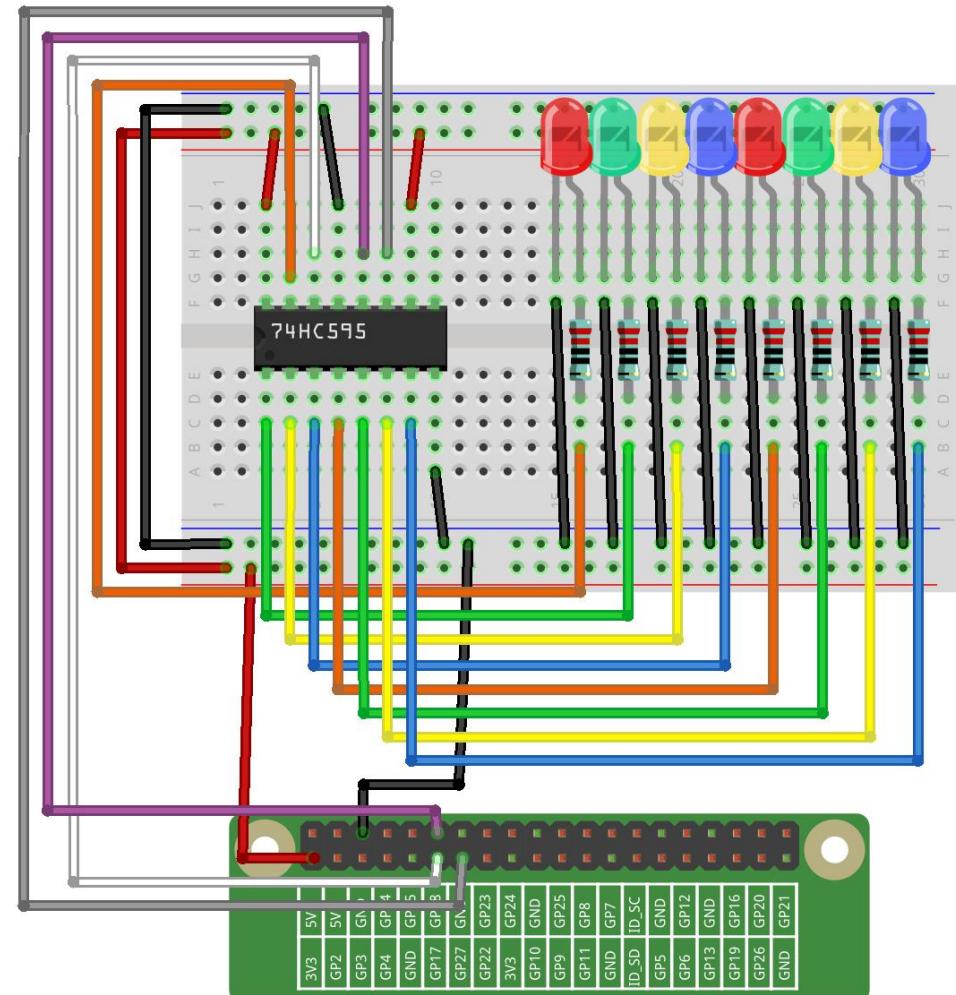
wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27



## Build the Circuit

Note: Recognize the direction of the chip according to the concave on it.

LED	74HC595	Raspberry Pi
LED1	Q7	
LED2	Q6	
LED3	Q5	
LED4	Q4	
LED5	Q3	
LED6	Q2	
LED7	Q1	
LED8	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_18_Driving_Leds_by_74hc595
```

2. Compile the code.

```
gcc 18_74hc595.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

As the code runs, you can see these eight LEDs are lit up from left to right, and then all LEDs light up and flash 3 times. After that, these eight LEDs are lit from right to left, then they all turn on before flashing 3 times. This loop continues in this way.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define SDI 0 //serial data input
5. #define RCLK 1 //memory clock input(STCP)
6. #define SRCLK 2 //shift register clock input(SHCP)
7.
8. unsigned char LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

```
9.  
10.void pulse(int pin){  
11.    digitalWrite(pin, 0);  
12.    digitalWrite(pin, 1);  
13.}  
14.  
15.void SIPO(unsigned char byte){  
16.    int i;  
17.    for(i=0;i<8;i++){  
18.        digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));  
19.        pulse(SRCLK);  
20.    }  
21.}  
22.  
23.void init(void){  
24.    pinMode(SDI, OUTPUT);  
25.    pinMode(RCLK, OUTPUT);  
26.    pinMode(SRCLK, OUTPUT);  
27.  
28.    digitalWrite(SDI, 0);  
29.    digitalWrite(RCLK, 0);  
30.    digitalWrite(SRCLK, 0);  
31.}  
32.
```

```
33.int main(void){  
34.    int i;  
35.  
36.    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen  
37.        printf("setup wiringPi failed !");  
38.        return 1;  
39.    }  
40.  
41.    init();  
42.  
43.    while(1){  
44.        for(i=0;i<8;i++){  
45.            SIPO(LED[i]);  
46.            pulse(RCLK);  
47.            delay(150);  
48.        }  
49.        delay(500);  
50.  
51.        for(i=0;i<3;i++){  
52.            SIPO(0xff);  
53.            pulse(RCLK);  
54.            delay(100);  
55.            SIPO(0x00);  
56.            pulse(RCLK);
```

```
57.         delay(100);
58.     }
59.     delay(500);
60.
61.     for(i=0;i<8;i++){
62.         SIPO(LED[8-i-1]);
63.         pulse(RCLK);
64.         delay(150);
65.     }
66.     delay(500);
67.
68.     for(i=0;i<3;i++){
69.         SIPO(0xff);
70.         pulse(RCLK);
71.         delay(100);
72.         SIPO(0x00);
73.         pulse(RCLK);
74.         delay(100);
75.     }
76.     delay(500);
77. }
78. return 0;
79. }
```

## Code Explanation

```
10. void pulse(int pin){  
11.     digitalWrite(pin, 0);  
12.     digitalWrite(pin, 1);  
13. }
```

Define an pulse function to generate an pulse.

```
15. void SIPO(unsigned char byte){  
16.     int i;  
17.     for(i=0;i<8;i++){  
18.         digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));  
19.         pulse(SRCLK);  
20.     }  
21. }
```

The function **SIPO** is used to assign the byte data to **SDI(DS)** by bits.

Among them, the inequality in statement **digitalWrite()((byte & (0x80>>i))>0)** is used to confirm each value written into the register and it realizes the function by Shift operator ( $>>$ ).

For example, if byte=0x01:

When the condition "i=0" is met,  $0x80(1000\ 0000)>>0$  becomes  $0x80(1000\ 0000)$ , if  $byte \& 0x80 = 0$ , the inequality is false, and output 0 (false).

If "i=1" is true,  $0x80>>1$  changes into  $0x40(0100\ 0000)$ ; when  $byte \& 0x40 = 0$ , output 0.

Deduce the rest from this, when and only when "i=8" is met,  $0x80 >> 8$  is  $0x01(0000\ 0001)$ ,  $\text{byte} \& 0x01 = 1$ , and output 1(true).

Pulse(SRCLK) generates a rising edge pulse on input pin of shift register to shift the 8 bit data on SDI to shift register successively.

In a word, this **for** loop produces 8 times to shift the 8 bits of 0000 0001 to shift register.

```
23. void init(void){  
24.     pinMode(SDI, OUTPUT);  
25.     pinMode(RCLK, OUTPUT);  
26.     pinMode(SRCLK, OUTPUT);  
27.  
28.     digitalWrite(SDI, 0);  
29.     digitalWrite(RCLK, 0);  
30.     digitalWrite(SRCLK, 0);  
31.}
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

```
44. for(i=0;i<8;i++){  
45.     SIPO(LED[i]);  
46.     pulse(RCLK);  
47.     delay(150);  
48. }
```

Use the **for** loop to count 8 times in cycle, and write a 1-bit data to the SDI each time.

When  $i=0$ ,  $\text{LED}[0]=0x01(0000\ 0001)$ , through the function  $\text{SIPO}(\text{LED}[0])$ , shifts the 8 bits of  $0x01$  to shift register successively.  $\text{Pulse}(\text{SRCLK})$  generates a rising edge signal on input pin of storage register to shift the  $0x01$  on shift register to storage register at once. Then the data in the memory register are output to the bus ( $Q7-Q0$ ), so you'll see the LED on  $Q0$  is lit up. After loops, output all eight elements in the array  $\text{LED}[i]$  to the bus ( $Q7-Q0$ ), and you'll see eight LEDs turning on from left to right.

```
51.     for(i=0;i<3;i++){  
52.         SIPO(0xff);  
53.         pulse(RCLK);  
54.         delay(100);  
55.         SIPO(0x00);  
56.         pulse(RCLK);  
57.         delay(100);  
58.     }
```

In this part, the **for** loop is used to three times repeat the program in **for()** statement.  $\text{SIPO}(0xff)$  means 8 LEDs are lit up,  $\text{SIPO}(0x00)$  represents 8 LEDs turn off. That is, let 8 LEDs turn off 3 times simultaneously.

```
61.     for(i=0;i<8;i++){  
62.         SIPO(LED[8-i-1]);  
63.         pulse(RCLK);  
64.         delay(150);  
65.     }
```

By the same token, this for loop allows 8 LEDs be lit up one by one in reverse order. Here,  $i$  gradually increases from 0, and  $8-i-1$  gradually decreases.  $\text{SIPO}(\text{LED}[8-i-1])$  can be used to call the data in the  $\text{LED}[]$  array from back to front so that you can get 8 LEDs lit up one by one in reverse order.

```
68.     for(i=0;i<3;i++){  
69.         SIPO(0xff);  
70.         pulse(RCLK);  
71.         delay(100);  
72.         SIPO(0x00);  
73.         pulse(RCLK);  
74.         delay(100);  
75.     }
```

Then, make the eight LEDs turn on or off 3 times simultaneously.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo python 18_74HC595.py
```

As the code runs, you can see these eight LEDs are lit up from left to right, and then all LEDs light up and flash 3 times. After that, these eight LEDs are lit from right to left, then they all turn on before flashing 3 times. This loop continues in this way.

### Code

```
1. import RPi.GPIO as GPIO
```

```
2. import time
3.
4. SDI    = 17
5. RCLK   = 18
6. SRCLK = 27
7.
8. LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80]      #original mode
9. BLINK = [0xff,0x00,0xff,0x00,0xff,0x00]                #blink
10.
11. def setup():
12.     GPIO.setmode(GPIO.BCM)
13.     GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
14.     GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
15.     GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
16.
17. # Shift the data to 74HC595
18. def hc595_shift(dat):
19.     for bit in range(0, 8):
20.         GPIO.output(SDI, 0x80 & (dat << bit))
21.         GPIO.output(SRCLK, GPIO.HIGH)
22.         time.sleep(0.001)
23.         GPIO.output(SRCLK, GPIO.LOW)
24.         GPIO.output(RCLK, GPIO.HIGH)
25.         time.sleep(0.001)
```

```
26.     GPIO.output(RCLK, GPIO.LOW)
27.
28. def main():
29.     print_message()
30.     mode = LED0
31.     sleeptime = 0.15
32.     blink_sleeptime = 0.15
33.
34.     while True:
35.         # Change LED status from mode
36.         for onoff in mode:
37.             hc595_shift(onoff)
38.             time.sleep(sleeptime)
39.
40.         for onoff in BLINK:
41.             hc595_shift(onoff)
42.             time.sleep(blink_sleeptime)
43.
44.         # Change LED status from mode reverse
45.         for onoff in reversed(mode):
46.             hc595_shift(onoff)
47.             time.sleep(sleeptime)
48.
49.         for onoff in BLINK:
```

```
50.         hc595_shift(onoff)
51.         time.sleep(blink_sleeptime)
52.
53.def destroy():
54.     GPIO.cleanup()
55.
56.if __name__ == '__main__':
57.     setup()
58.     try:
59.         main()
60.     except KeyboardInterrupt:
61.         destroy()
```

## Code Explanation

```
8. LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80]      #original mode
```

Use array to define LED flashing mode, you can also customize several hexadecimals to light up 8 LEDs.

```
11. def setup():
12.     GPIO.setmode(GPIO.BCM)      # Number GPIOs by its BCM location
13.     GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
14.     GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
15.     GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LED lights are set to output mode, default low level.

```
18. def hc595_shift(dat):
```

Define a function **hc595\_shift()** to output the 8 bits of **dat** to Q0-Q7.

```
19. for bit in range(0, 8):  
20.     GPIO.output(SDI, 0x80 & (dat << bit))  
21.     GPIO.output(SRCLK, GPIO.HIGH)  
22.     time.sleep(0.001)  
23.     GPIO.output(SRCLK, GPIO.LOW)
```

Assign the **dat** to SDI(DS) according to bits. Pin **SRCLK** will convert from low to high, and generate a rising edge pulse, then shift the data in pin SDI to shift register. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

```
24. GPIO.output(RCLK, GPIO.HIGH)  
25. time.sleep(0.001)  
26. GPIO.output(RCLK, GPIO.LOW)
```

Pin **RCLK** converts from low to high and generate a rising edge, then shift data from shift register to storage register. Finally the data in the memory register is output to the bus (Q0-Q7).

```
36. for onoff in mode:  
37.     hc595_shift(onoff)  
38.     time.sleep(sleepetime)
```

Here we use a **onoff** variable to control the LED that changes within the range of mode, and **hc595\_shift** (**onoff**) means lighting up LED one by one. For example, when mode is the first datum in **LED0**, or **0x01**, **onoff = mode = 0x01 = 00000001**. In this course, the LED is lit by high level. To put it another way, it is

Hc595\_shift (onoff) = hc595\_shift (00000001) that lights up the last LED. Along the same vein, when the value of mode is the second datum of LED0 (onoff = 0x02 = 00000010), the second last LED turns on.

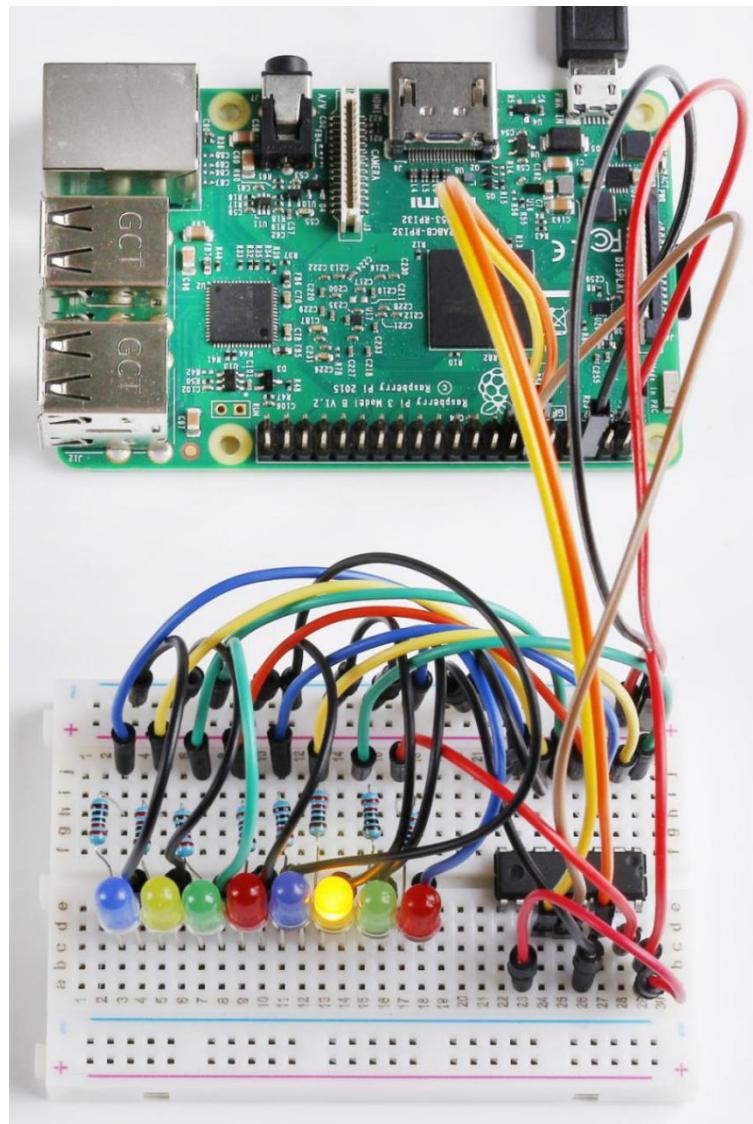
```
45.     for onoff in reversed(mode):  
46.         hc595_shift(onoff)  
47.         time.sleep(sleepetime)
```

According to the same principle, a reversed is used here to get LEDs lit up in reverse order.

```
49.     for onoff in BLINK:  
50.         hc595_shift(onoff)  
51.         time.sleep(blink_sleeptime)
```

In the same way, light up 8 LEDs; exactly, 8 LEDs are turned on or off 3 times synchronously in the same pattern as that of the LEDs in the BLINK array.

## Phenomenon Picture

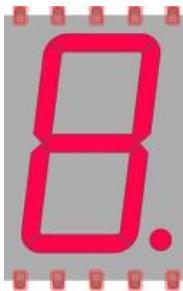


## Lesson 19 7-segment

### Introduction

Generally, there are two ways to drive a single 7-Segment Display. One way is to connect its 8 pins directly to eight ports on the Raspberry Pi. Or you can connect the 74HC595 to three ports of the Raspberry Pi and then the 7-segment display to the 74HC595. In this experiment, we will use the latter. By this way, we can save five ports – considering the board's limited ports, and this is very important. Now let's get started!

### Newly Added Components

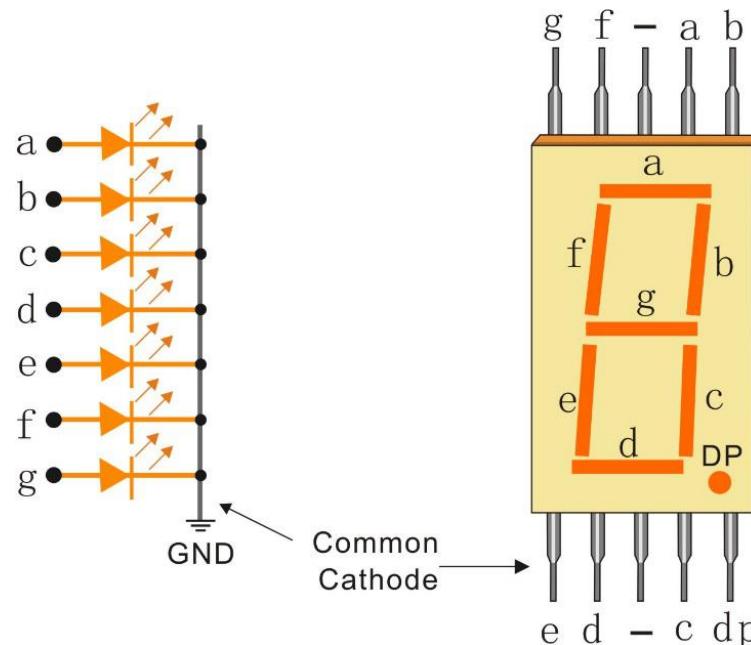
1 * 7-Segment Display	1 * 74HC595	1 * Resistor (220Ω)
		

## Principle

### 7-Segment Display

A 7-Segment Display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral to be displayed.

There are two types of pin connection: Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected. In this kit, we use the former.



Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from "a" through to "g" representing each individual LED. The other LED pins are connected together forming a common pin. So by forward biasing the appropriate

pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

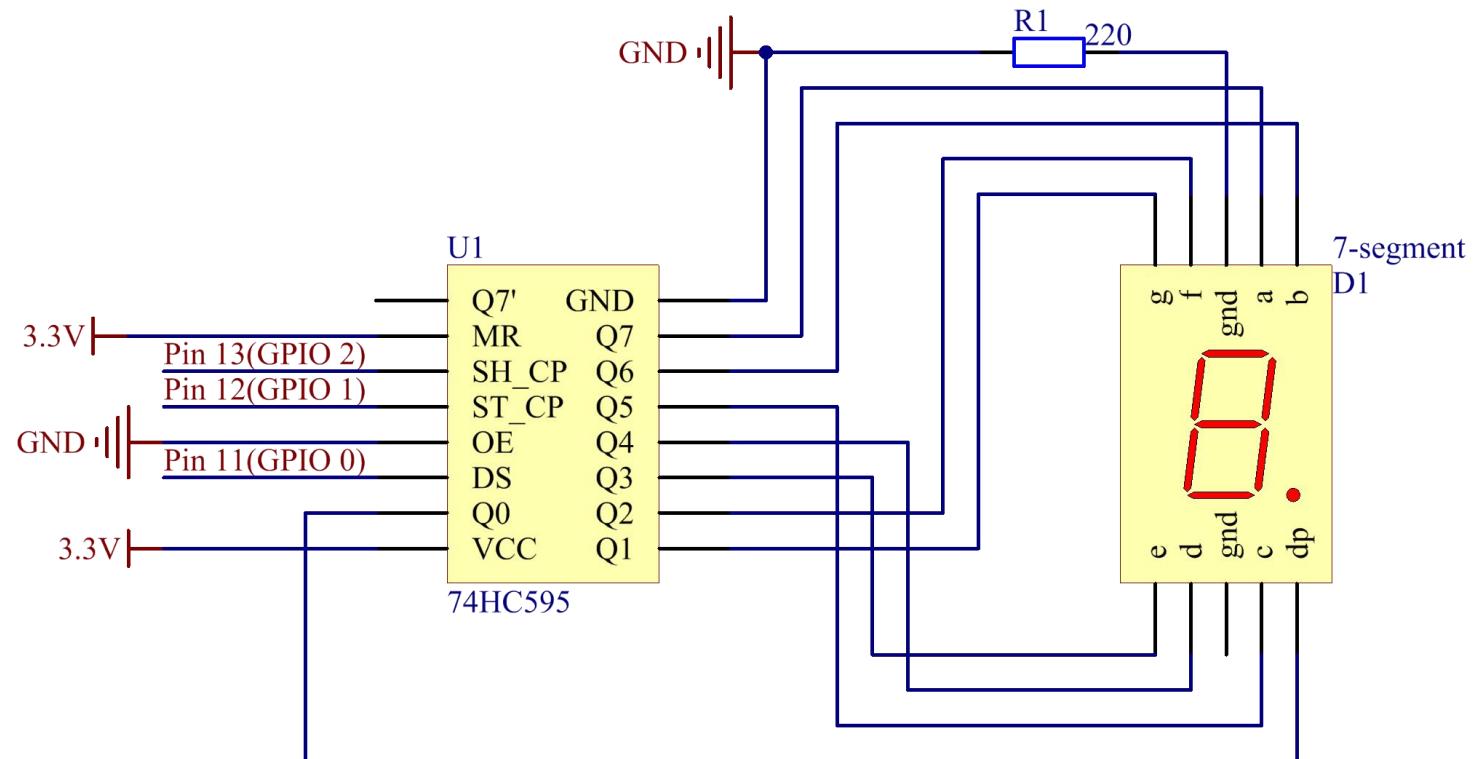
## Display Codes

To help you get to know how 7-Segment Displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-Segment Display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-Segment Display, while HEX Code corresponds to hexadecimal number.

Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39
3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d	.	10000000	0x80
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

## Schematic Diagram

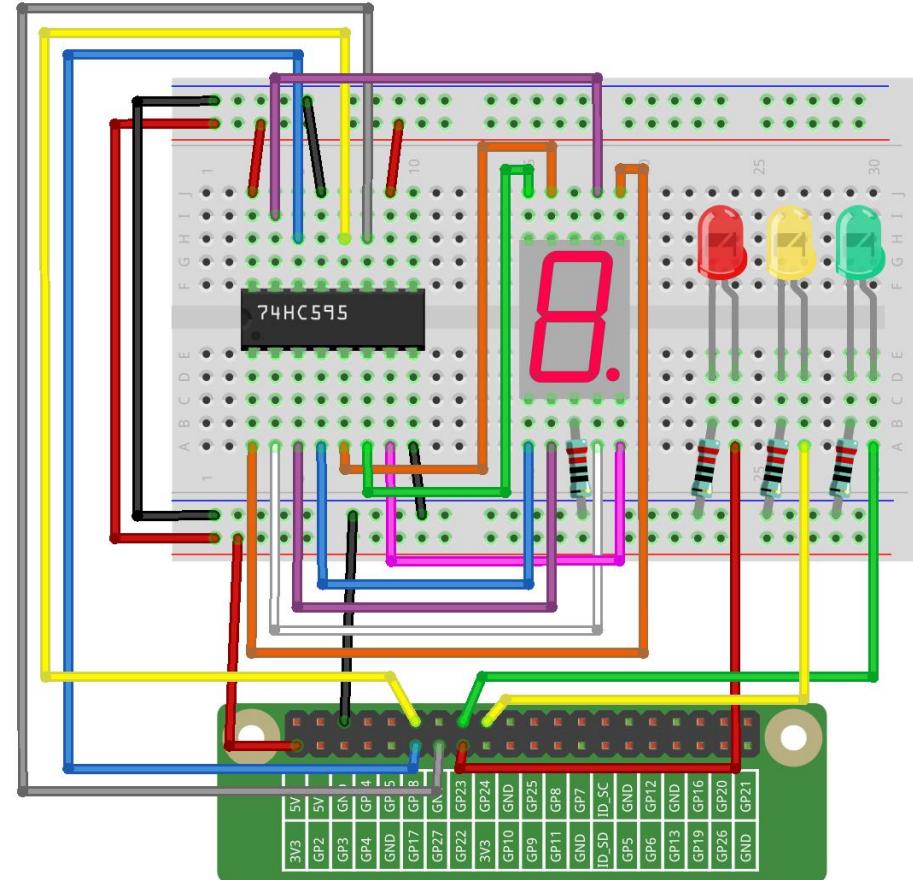
wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27



## Build the Circuit

Note: Recognize the direction of the chip according to the concave on it.

7-Segment	74HC595	Raspberry Pi
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_19_7-segment
```

2. Compile the code.

```
gcc 19_7-Segment.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

You may see 0 to 9 and A to F on the 7-Segment Display.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3.
4. #define SDI 0 //serial data input
5. #define RCLK 1 //memory clock input(STCP)
6. #define SRCLK 2 //shift register clock input(SHCP)
7.
8. unsigned char SegCode[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,
   0x79,0x71,0x80};
9.
```

```
10. void init(void){  
11.     pinMode(SDI, OUTPUT);  
12.     pinMode(RCLK, OUTPUT);  
13.     pinMode(SRCLK, OUTPUT);  
14.  
15.     digitalWrite(SDI, 0);  
16.     digitalWrite(RCLK, 0);  
17.     digitalWrite(SRCLK, 0);  
18. }  
19.  
20. void hc595_shift(unsigned char dat){  
21.     int i;  
22.     for(i=0;i<8;i++){  
23.         digitalWrite(SDI, 0x80 & (dat << i));  
24.         digitalWrite(SRCLK, 1);  
25.         delay(1);  
26.         digitalWrite(SRCLK, 0);  
27.     }  
28.  
29.     digitalWrite(RCLK, 1);  
30.     delay(1);  
31.     digitalWrite(RCLK, 0);  
32. }  
33.
```

```
34. int main(void){  
35.     int i;  
36.  
37.     if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen  
38.         printf("setup wiringPi failed !");  
39.         return 1;  
40.     }  
41.  
42.     init();  
43.  
44.     while(1){  
45.         for(i=0;i<17;i++){  
46.             hc595_shift(SegCode[i]);  
47.             delay(500);  
48.         }  
49.     }  
50.     return 0;  
51. }
```

## Code Explanation

```
10. void init(void){  
11.     pinMode(SDI, OUTPUT);  
12.     pinMode(RCLK, OUTPUT);  
13.     pinMode(SRCLK, OUTPUT);
```

```
14.  
15.     digitalWrite(SDI, 0);  
16.     digitalWrite(RCLK, 0);  
17.     digitalWrite(SRCLK, 0);  
18. }
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

```
19. void hc595_shift(unsigned char dat)
```

To assign 8 bit value to 74HC595's shift register.

```
22. for(i=0;i<8;i++){  
23.     digitalWrite(SDI, 0x80 & (dat << i));  
24.     digitalWrite(SRCLK, 1);  
25.     delay(1);  
26.     digitalWrite(SRCLK, 0);  
27. }
```

Assign the **dat** value to SDI(DS) by bits. Then shift them to the shift register by bits. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

```
29.     digitalWrite(RCLK, 1);  
30.     delay(1);  
22.     digitalWrite(RCLK, 0);
```

Pin RCLK converts from low to high and generates a rising edge, then shifts data from shift register to storage register. Finally the data in the memory register are output to the bus (Q0-Q7).

```
45.     for(i=0;i<17;i++){  
46.         hc595_shift(SegCode[i]);  
47.         delay(500);  
48.     }
```

In the **for** loop, output 16 values from array **Segcode[]** to 7-Segment Display.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo 19_7-Segment.py
```

You may see 0 to 9 and A to F on the 7-Segment Display.

### Code

```
1. import RPi.GPIO as GPIO  
2. import time  
3.  
4. # Set up pins  
5. SDI    = 17
```

```
6. RCLK = 18
7. SRCLK = 27
8.
9. segCode = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]
10.
11.
12. def setup():
13.     GPIO.setmode(GPIO.BCM)
14.     GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
15.     GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
16.     GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
17.
18. # Shift the data to 74HC595
19. def hc595_shift(dat):
20.     for bit in range(0, 8):
21.         GPIO.output(SDI, 0x80 & (dat << bit))
22.         GPIO.output(SRCLK, GPIO.HIGH)
23.         time.sleep(0.001)
24.         GPIO.output(SRCLK, GPIO.LOW)
25.         GPIO.output(RCLK, GPIO.HIGH)
26.         time.sleep(0.001)
27.         GPIO.output(RCLK, GPIO.LOW)
28.
29. def main():
```

```
30.     while True:  
31.         # Shift the code one by one from segCode list  
32.         for code in segCode:  
33.             hc595_shift(code)  
34.             time.sleep(0.5)  
35.  
36. def destroy():  
37.     GPIO.cleanup()  
38.  
39. if __name__ == '__main__':  
40.     setup()  
41.     try:  
42.         main()  
43.     except KeyboardInterrupt:  
44.         destroy()
```

## Code Explanation

```
12. def setup():  
13.     GPIO.setmode(GPIO.BCM)  
14.     GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)  
15.     GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)  
16.     GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level.

```
19. def hc595_shift(dat):
    To assign 8 bit value to 74HC595's shift register.
20.     for bit in range(0, 8):
21.         GPIO.output(SDI, 0x80 & (dat << bit))
22.         GPIO.output(SRCLK, GPIO.HIGH)
23.         time.sleep(0.001)
24.         GPIO.output(SRCLK, GPIO.LOW)
```

Assign the **dat** value to SDI(DS) by bits. Then shift them to the shift register by bits. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

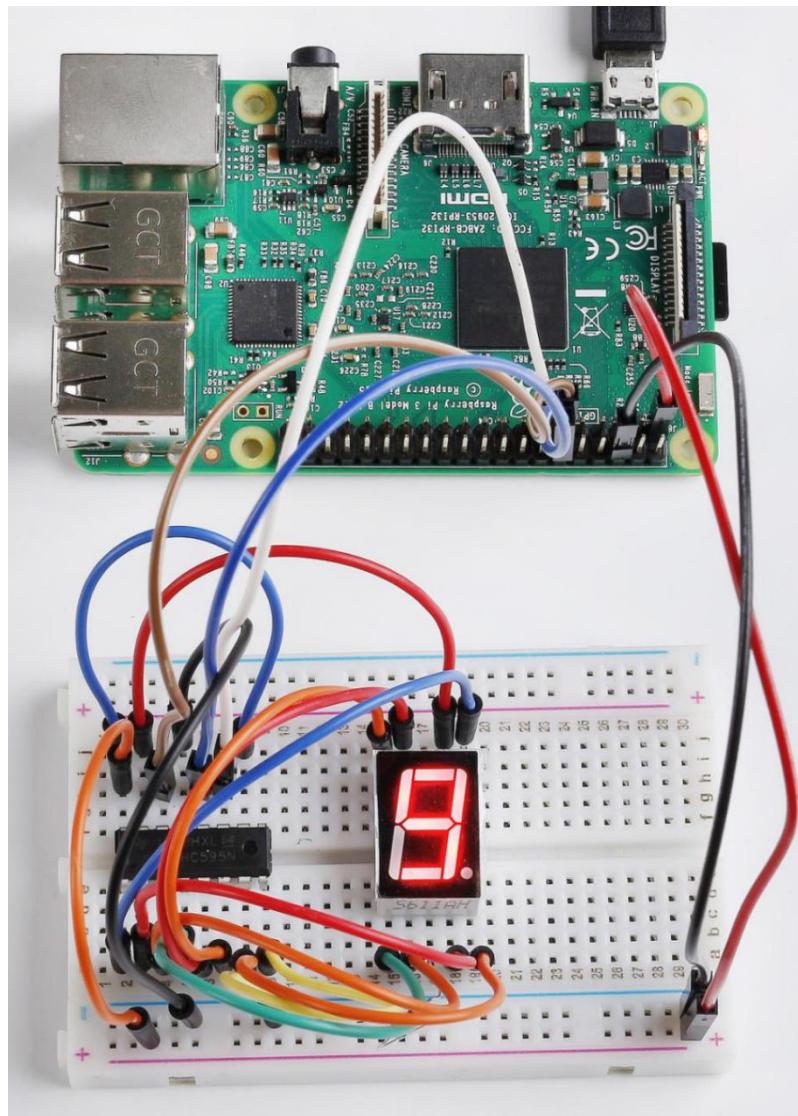
```
25.     GPIO.output(RCLK, GPIO.HIGH)
26.     time.sleep(0.001)
27.     GPIO.output(RCLK, GPIO.LOW)
```

Pin **RCLK** converts from low to high and generates a rising edge, then shifts data from shift register to storage register. Finally the data in the memory register are output to the bus (Q0-Q7).

```
32.     for code in segCode:
33.         hc595_shift(code)
34.         time.sleep(0.5)
```

In the **for** loop, output 16 values from array **Segcode []** to 7-Segment Display.

## Phenomenon Picture

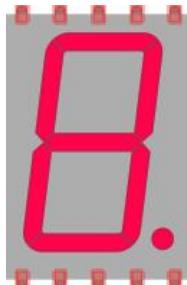
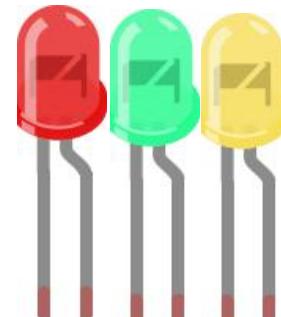


# Lesson 20 Traffic Light

## Introduction

In last lesson, we learned how to use a 74HC595 chip to drive a 7-Segment Display. Based on that, we can apply it more widely now, such as making a simple traffic light. Now let's get started!

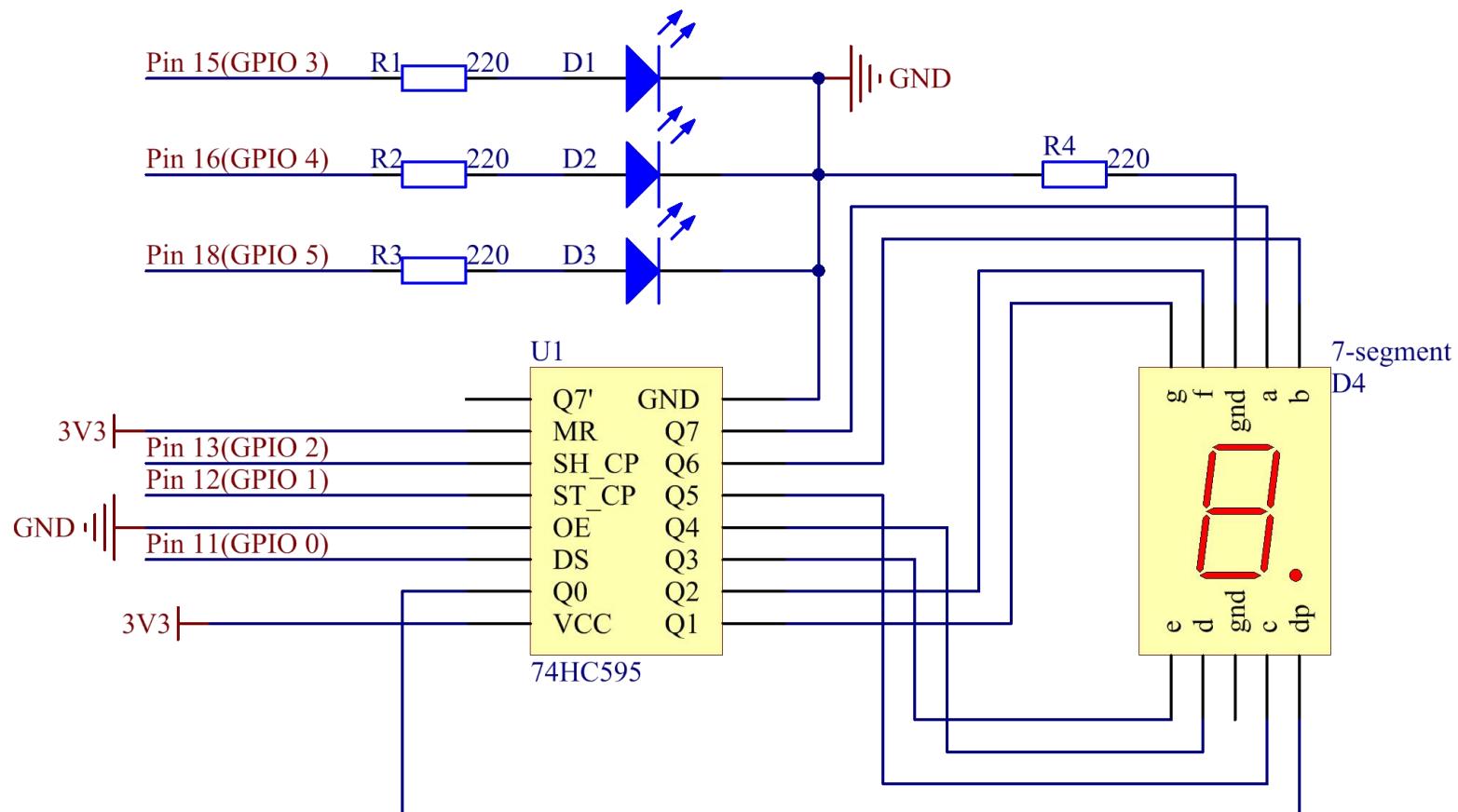
## Newly Added Components

1 * 7-Segment Display	1 * 74HC595	4 * Resistor (220Ω)	3 * LED
			

## Schematic Diagram

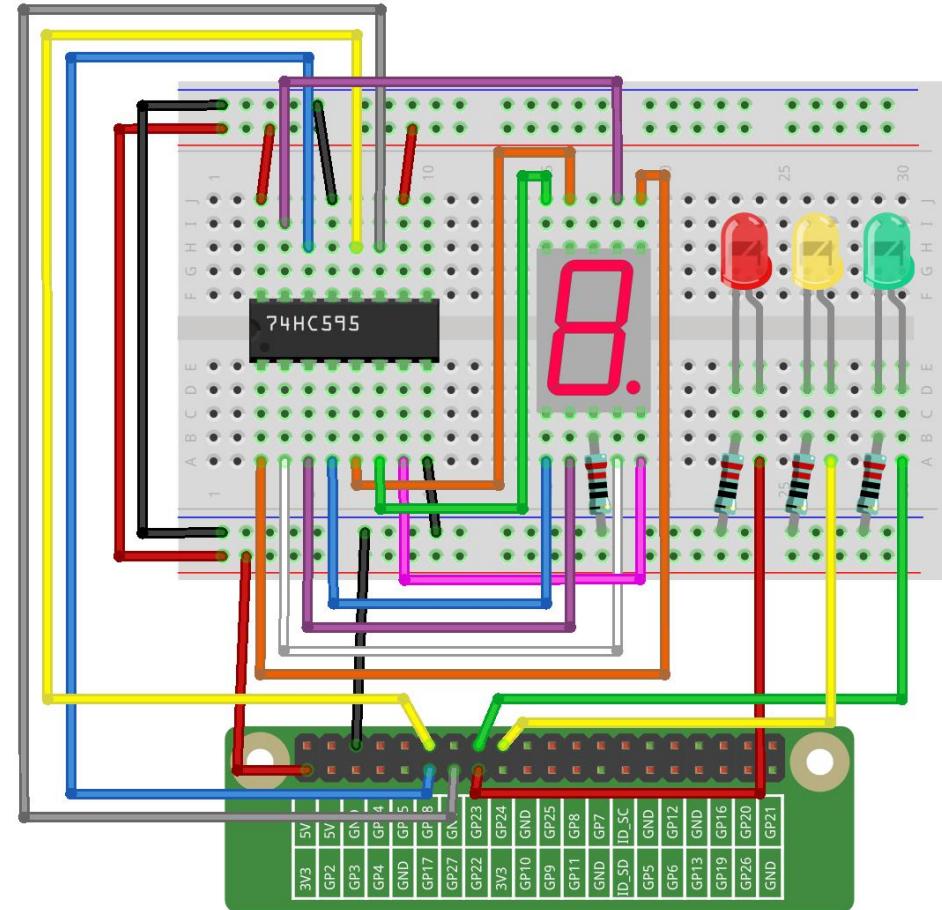
wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27
3	Pin 15	22

4	Pin16	23
5	Pin18	24



## Build the Circuit

7-Segment	74HC595	Raspberry Pi
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND
Red LED		Pin 15
Green LED		Pin 16
Yellow		Pin 18



## ➤ For C Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/c/Lesson_20_TrafficLight
```

2. Compile the code.

```
gcc 20_TrafficLight.c -lwiringPi
```

3. Run the executable file.

```
sudo ./a.out
```

You can see the following phenomenon of traffic lights. The red LED lights up for 9 seconds, green LED for 5s, and yellow LED for 3s.

### Code

```
1. #include <wiringPi.h>
2. #include <stdio.h>
3. #include <wiringShift.h>
4. #include <signal.h>
5. #include <unistd.h>
6. #define SDI 0 //serial data input(DS)
7. #define RCLK 1 //memory clock input(STCP)
8. #define SRCLK 2 //shift register clock input(SHCP)
9. const int ledPin[]={3,4,5}; //Define 3 LED pin(Red, Green, Yellow)
```

```
10.unsigned char SegCode[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,
 0x79,0x71,0x80};

11.

12.int greentime = 5;
13.int yellowtime = 3;
14.int redtime = 9;
15.int colorState = 0;
16.char *lightColor[]={ "Red", "Green", "Yellow"};
17.int counter = 9;

18.

19.void init(void){
20.    pinMode(SDI, OUTPUT);
21.    pinMode(RCLK, OUTPUT);
22.    pinMode(SRCLK, OUTPUT);
23.
24.    digitalWrite(SDI, 0);
25.    digitalWrite(RCLK, 0);
26.    digitalWrite(SRCLK, 0);
27.
28.    for(int i=0;i<3;i++){
29.        pinMode(ledPin[i],OUTPUT);
30.        digitalWrite(ledPin[i],LOW);
31.    }
32.}
```

33.

```
34.void hc595_shift(unsigned char dat){  
35.    int i;  
36.    for(i=0;i<8;i++){  
37.        digitalWrite(SDI, 0x80 & (dat << i));  
38.        digitalWrite(SRCLK, 1);  
39.        delay(1);  
40.        digitalWrite(SRCLK, 0);  
41.    }  
42.    digitalWrite(RCLK, 1);  
43.    delay(1);  
44.    digitalWrite(RCLK, 0);  
45.}  
46.  
47.void timer(int sig){      //Timer function  
48.    if(sig == SIGALRM){  
49.        counter --;  
50.        alarm(1);  
51.        if(counter == 0){  
52.            if(colorState == 0) counter = greentime;  
53.            if(colorState == 1) counter = yellowtime;  
54.            if(colorState == 2) counter = redtime;  
55.            colorState = (colorState+1)%3;  
56.        }  
57.    }  
58.}
```

```
57.         printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);
58.     }
59. }
60.
61.void display(int num)
62.{
63.     hc595_shift(SegCode[num%10]);
64.     delay(1);
65. }
66.
67.void lightup(int state)
68.{
69.     for(int i=0;i<3;i++){
70.         digitalWrite(ledPin[i],LOW);
71.     }
72.     digitalWrite(ledPin[state],HIGH);
73. }
74.
75.int main(void)
76.{
77.     int i;
78.
79.     if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
80.         printf("setup wiringPi failed !");
```

```
81.         return 1;
82.     }
83.
84.     init();
85.
86.     signal(SIGALRM,timer); //configure the timer
87.     alarm(1);           //set the time of timer to 1s
88.     while(1){
89.
90.         display(counter);
91.         lightup(colorState);
92.     }
93.     return 0;
94. }
```

## Code Explanation

```
12. int greentime = 5;
13. int yellowtime = 3;
14. int redtime = 9;
```

Define the duration of lighting of three LEDs. Since what we use is a 7-Segment Display here, we shorten the length of seconds of lighting of traffic lights, setting the green LED to light up for 5 seconds, the yellow LED to 3 seconds, and the red LED to 9 seconds.

```
15. int colorState = 0;
16. int counter = 9;
```

The variable **colorState** corresponds to the state of the traffic lights, and we only need to do a simple calculation of **colorState** to indicate the order change of the state of the traffic lights. The Variable counter is used to count down the time to each traffic light status and will be output on a 7-Segment Display.

```
19. void init(void){  
20.     pinMode(SDI, OUTPUT);  
21.     pinMode(RCLK, OUTPUT);  
22.     pinMode(SRCLK, OUTPUT);  
23.  
24.     digitalWrite(SDI, 0);  
25.     digitalWrite(RCLK, 0);  
26.     digitalWrite(SRCLK, 0);  
27.  
28.     for(int i=0;i<3;i++){  
29.         pinMode(ledPin[i],OUTPUT);  
30.         digitalWrite(ledPin[i],LOW);  
31.     }  
32. }
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

```
47. void timer(int sig){  
48.     if(sig == SIGALRM){  
49.         counter --;  
50.         alarm(1);
```

```
51.     if(counter == 0){  
52.         if(colorState == 0) counter = greentime;  
53.         if(colorState == 1) counter = yellowtime;  
54.         if(colorState == 2) counter = redtime;  
55.         colorState = (colorState+1)%3;  
56.     }  
57.     printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);  
58. }  
59. }
```

On this timer, counter decreases gradually with every second passing, and when it goes to 0, the state of the traffic light changes accordingly.

```
67. void lightup(int state)  
68. {  
69.     for(int i=0;i<3;i++){  
70.         digitalWrite(ledPin[i],LOW);  
71.     }  
72.     digitalWrite(ledPin[state],HIGH);  
73. }
```

The function is to turn off all the lights first, and then light up the corresponding LED according to the value of the traffic light state.

## ➤ For Python Language Users

### Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit-for-raspberry-pi/python
```

2. Run the code.

```
sudo 20_TrafficLight.py
```

You can see the following phenomenon of traffic lights. The red LED lights up for 9 seconds, green LED for 5s, and yellow LED for 3s.

### Code

```
1. import RPi.GPIO as GPIO
2. import time
3. import threading
4.
5. #define the pins connect to 74HC595
6. SDI    = 17      #serial data input(DS)
7. RCLK   = 18      #memory clock input(STCP)
8. SRCLK = 27      #shift register clock input(SHCP)
9. number = (0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80)
10.
11.ledPin =(22,23,24)
12.
```

```
13.greenLight = 5
14.yellowLight = 3
15.redLight = 9
16.lightColor=("Red", "Green", "Yellow")
17.
18.colorState=0
19.counter = 9
20.t = 0
21.
22.def setup():
23.    GPIO.setmode(GPIO.BCM)
24.    GPIO.setup(SDI, GPIO.OUT)
25.    GPIO.setup(RCLK, GPIO.OUT)
26.    GPIO.setup(SRCLK, GPIO.OUT)
27.    for pin in ledPin:
28.        GPIO.setup(pin,GPIO.OUT)
29.
30.
31.def hc595_shift(dat):
32.    for bit in range(0, 8):
33.        GPIO.output(SDI, 0x80 & (dat << bit))
34.        GPIO.output(SRCLK, GPIO.HIGH)
35.        GPIO.output(SRCLK, GPIO.LOW)
36.        GPIO.output(RCLK, GPIO.HIGH)
```

```
37.     GPIO.output(RCLK, GPIO.LOW)
38.
39.def display(num):
40.    hc595_shift(0xff)
41.    hc595_shift(number[num%10])
42.    time.sleep(0.003)
43.
44.
45.def timer():          #timer function
46.    global counter
47.    global colorState
48.    global t
49.    t = threading.Timer(1.0,timer)
50.    t.start()
51.    counter-=1
52.    if (counter is 0):
53.        if(colorState is 0):
54.            counter= greenLight
55.        if(colorState is 1):
56.            counter=yellowLight
57.        if (colorState is 2):
58.            counter=redLight
59.        colorState=(colorState+1)%3
60.    print ("counter : %d      color: %s "%(counter,lightColor[colorState]))
```

```
61.  
62. def lightup(state):  
63.     for i in range(0,3):  
64.         GPIO.output(ledPin[i], GPIO.LOW)  
65.     GPIO.output(ledPin[state], GPIO.HIGH)  
66.  
67. def loop():  
68.     global t  
69.     global counter  
70.     global colorState  
71.     t = threading.Timer(1.0,timer)  
72.     t.start()  
73.     while True:  
74.         display(counter)  
75.         lightup(colorState)  
76.  
77. def destroy():    # When "Ctrl+C" is pressed, the function is executed.  
78.     global t  
79.     GPIO.cleanup()  
80.     t.cancel()      #cancel the timer  
81.  
82. if __name__ == '__main__': # Program starting from here  
83.     setup()  
84.     try:
```

```
85.     loop()
86. except KeyboardInterrupt:
87.     destroy()
```

## Code Explanation

```
13. greenLight = 5
14. yellowLight = 3
15.redLight = 9
```

Define the duration of lighting of three LEDs. Since what we use is a 7-Segment Display here, we shorten the length of seconds of lighting of traffic LEDs, setting the green LED to light up for 5 seconds, the yellow LED to 3 seconds, and the red LED to 9 seconds.

```
18. colorState=0
19.counter = 9
```

The variable **colorState** corresponds to the state of the traffic LEDs, and we only need to do a simple calculation of **colorState** to indicate the order change of the state of the traffic LEDs.

**counter** is used to count down the time to each traffic LED status and will be output on a 7-Segment Display.

```
45. def timer():          #timer function
46.     global counter
47.     global colorState
48.     global t
49.     t = threading.Timer(1.0,timer)
50.     t.start()
```

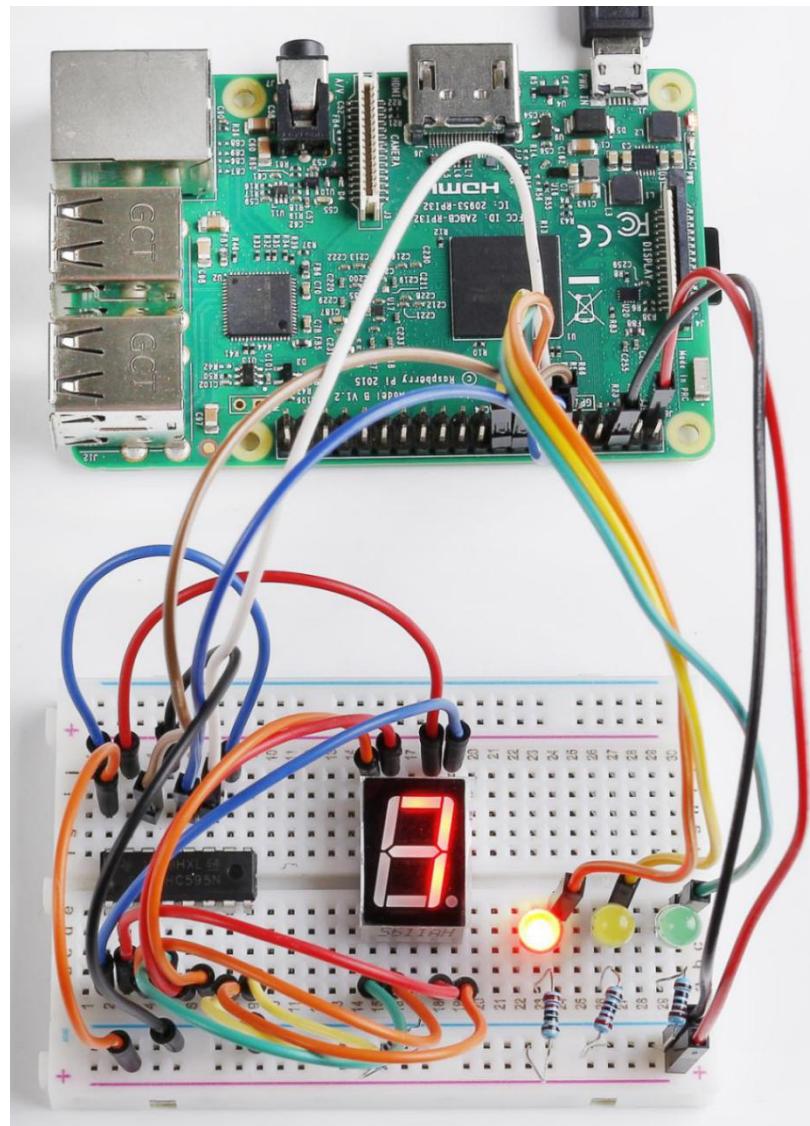
```
51.     counter-=1
52.     if (counter is 0):
53.         if(colorState is 0):
54.             counter= greenLight
55.         if(colorState is 1):
56.             counter=yellowLight
57.         if (colorState is 2):
58.             counter=redLight
59.         colorState=(colorState+1)%3
60.     print ("counter : %d      color: %s "%(counter,lightColor[colorState]))
```

On this timer, counter decreases gradually with every second passing, and when it goes to 0, the state of the traffic LED changes accordingly.

```
62. def lightup(state):
63.     for i in range(0,3):
64.         GPIO.output(ledPin[i], GPIO.LOW)
65.     GPIO.output(ledPin[state], GPIO.HIGH)
```

The function is to turn off all the LEDs first, and then light up the corresponding LED according to the value of the traffic LED state.

## Phenomenon Picture



## Copyright Notice

All contents including but not limited to texts, images, and code in this manual are owned by the SunFounder Company. You should only use it for personal study, investigation, enjoyment, or other non-commercial or nonprofit purposes, under the related regulations and copyrights laws, without infringing the legal rights of the author and relevant right holders. For any individual or organization that uses these for commercial profit without permission, the Company reserves the right to take legal action.