

# Preface

## About SunFounder

SunFounder is a technology company focused on Raspberry Pi and Arduino open source community development. Committed to the promotion of open source culture, we strive to bring the fun of electronics making to people all around the world and enable everyone to be a maker. Our products include learning kits, development boards, robots, sensor modules and development tools. In addition to high quality products, SunFounder also offers video tutorials to help your own project. If you have interest in open source or making something cool, welcome to join us! Visit [www.sunfounder.com](http://www.sunfounder.com) for more!

## About the Super Kit 3.0

This super kit is suitable for the Raspberry Pi B, model B+ and Raspberry Pi 2 model B. It includes various components and chips that can show different interesting phenomena. You can make it happen by following the experiment instructions, and learn basic knowledge about them. Also you can explore more application after mastering the principle and code. Now get on the road!

In this book, we will show you circuits with both realistic illustrations and schematic diagrams. You can go to our official website [www.sunfounder.com](http://www.sunfounder.com) to download the related code by clicking **LEARN** --> **Get tutorials** and watch related videos by clicking VIDEO.

## Free Support



If you have any **TECHNICAL questions**, 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.

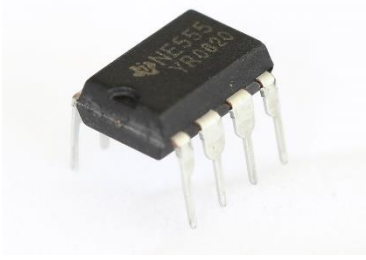

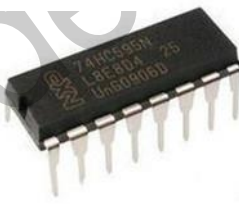

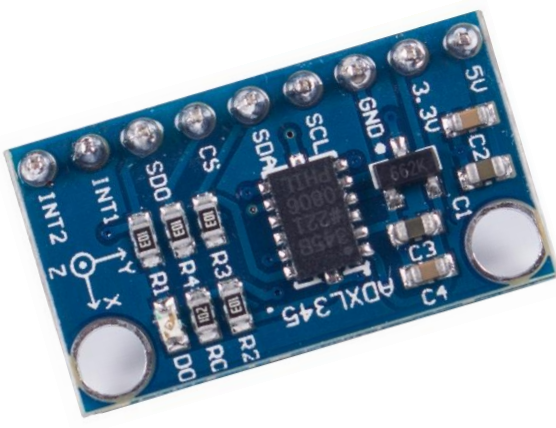
# Contents

Components List .....	1
Components Introduction.....	8
Resistor.....	8
Potentiometer.....	10
LED .....	11
RGB LED.....	12
Jumper Wires .....	13
Breadboard.....	13
Get Started.....	16
Preparation.....	16
Access to Raspberry Pi's Console.....	16
Using Console in GUI (terminal, recommended for beginners).....	16
Using Console Only .....	18
Introduction to Raspberry Pi.....	22
RAB Holder .....	22
Raspberry Pi.....	24
Raspberry Pi Pin Name .....	24
Extension Board.....	25
GPIO Libraries.....	28
WiringPi.....	28
Introduction.....	28
RPi.GPIO .....	29
Introduction.....	29
Download the Code.....	31
Lesson 1 Blinking LED .....	32
Lesson 2 Controlling an LED by a Button .....	41
Lesson 3 Flowing LED Lights .....	46

Lesson 4 Breathing LED.....	50
Lesson 5 RGB LED.....	54
Lesson 6 Buzzer.....	58
Lesson 7 Relay.....	63
Lesson 8 4N35.....	67
Lesson 9 Ne555.....	71
Lesson 10 Slide Switch.....	75
Lesson 11 How to Drive a DC Motor.....	79
Lesson 12 Rotary Encoder.....	84
Lesson 13 Driving LEDs by 74HC595.....	90
Lesson 14 Driving 7-Segment Display by 74HC595.....	96
Lesson 15 Driving Dot-Matrix by 74HC595.....	104
Lesson 16 LCD1602.....	111
Lesson 17 ADXL345.....	118

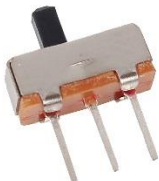
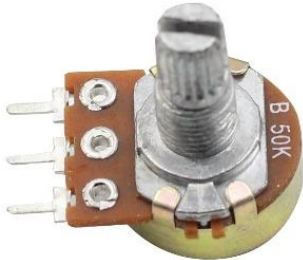
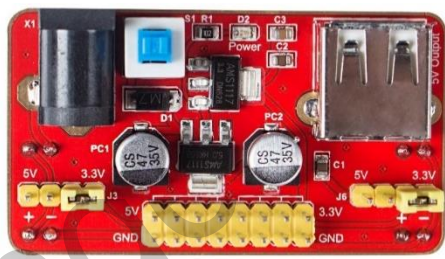



SunFounder

## Components List





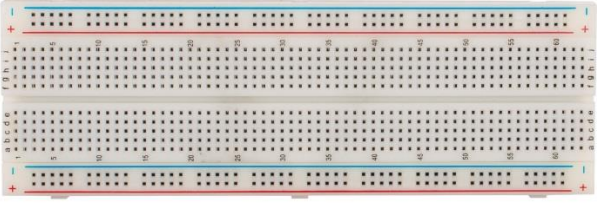

No.	Name	Quantity	Component
1	555 Timer IC	1	
2	Optocoupler (4N35)	2	
3	Shift Register (74HC595)	2	
4	L293D	1	
5	Accelerometer ADXL345	1	






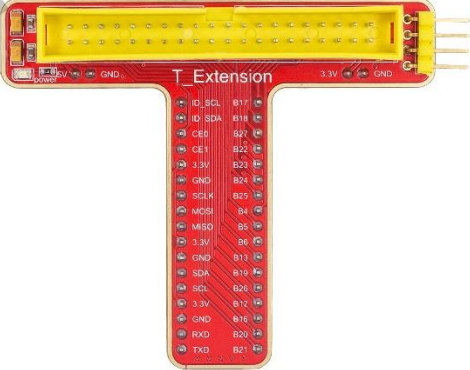
6	Rotary Encoder	1	
7	Button	5	
8	Resistor (220Ω)	8	 (red, red, black, black, brown)
9	Resistor (1kΩ)	8	 (brown, black, black, brown, brown)
10	Resistor (10kΩ)	4	 (brown, black, black, red, brown)
11	Resistor (100kΩ)	4	 (brown, black, black, orange, brown)
12	Resistor (1MΩ)	1	 (brown, black, green, gold)
13	Resistor (5.1MΩ)	1	 (green, brown, green, gold)
14	Diode Rectifier	4	

15	Switch	1	
16	Potentiometer (50k)	1	
17	Power Supply Module	1	
18	LCD1602	1	
19	Dot Matrix Display (8*8)	1	
20	7-Segment Display	2	

21	DC Motor	1	
22	RGB LED	1	
23	LED (red)	8	
24	LED (white)	4	
25	LED (green)	4	
26	LED (yellow)	4	

27	NPN Transistor (S8050)	2	
28	PNP Transistor (S8550)	2	
29	Capacitor Ceramic 100nF	4	
30	Capacitor Ceramic 10nF	4	
31	Breadboard	1	
32	Active Buzzer	1	

33	Relay	1	
34	Fan	1	
35	Male-to-Male Jumper Wire	65	
36	Female-to-Male Dupont Wire	20	
37	5-Pin Anti-reverse Cable	2	

38	9V Battery Buckle	1	
39	M3*10 Screw	2	
40	M2.5*6 Screw	4	
41	M3*6 Screw	6	
42	RAB Holder	1	
43	T-Extension Board	1	
44	40-Pin GPIO Cable	1	

### Notes:

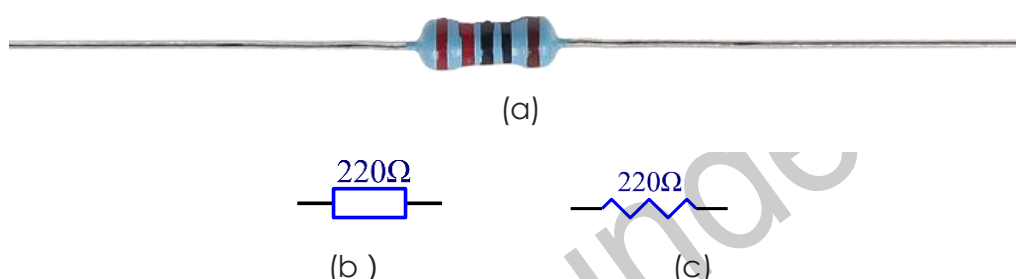
After unpacking, please check that the number of components is correct and that all components are in good condition.

# Components Introduction

## Resistor

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

The resistors in this kit are fixed ones. It is essential in the circuit to protect the connected components. Figure (a) below shows a  $220\Omega$  resistor.  $\Omega$  is the unit of resistance and the larger includes  $K\Omega$ ,  $M\Omega$ , etc. Their relationship can be shown as follows:  $1 M\Omega = 1000 K\Omega$ ,  $1 K\Omega = 1000 \Omega$ , which means  $1 M\Omega = 1000,000 \Omega = 10^6 \Omega$ . Figure (b) and (c) show two generally used circuit symbols for resistor. Normally, the resistance is marked on it. So if you see these symbols in a circuit, it stands for a resistor.



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. If you are not sure about the value, use the multimeter.

In the kit, a Resistor Color Code Calculator card is provided as shown below:

4-Band-Code

2%, 5%, 10%      560KΩ ± 5%

Color	1st Band	2nd Band	3rd Band	Multiplier	Tolerance
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8		± 0.05%
White	9	9	9		
Gold				0.1	± 5% (J)
Silver				0.01	± 10% (K)

5-Band-Code

0.1%, 0.25%, 0.5%, 1%      237Ω ± 1%

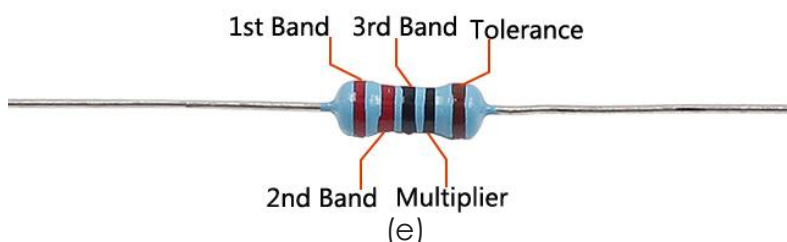
(d)



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

Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White	Gold	Silver
0	1	2	3	4	5	6	7	8	9	0.1	0.01

The 4- and 5-band resistors are frequently used, on which there are 4 and 5 chromatic bands. Let's see how to read the resistance value of a 5-band resistor as shown below. Normally, when you get a resistor, you may find it hard to decide which end to start for reading the color. The tip is that the gap between the 4<sup>th</sup> and 5<sup>th</sup> band will be comparatively larger. Therefore, you can observe the gap between the two chromatic bands at one end of the resistor; if it's larger than any other band gaps, then you can read from the opposite side.

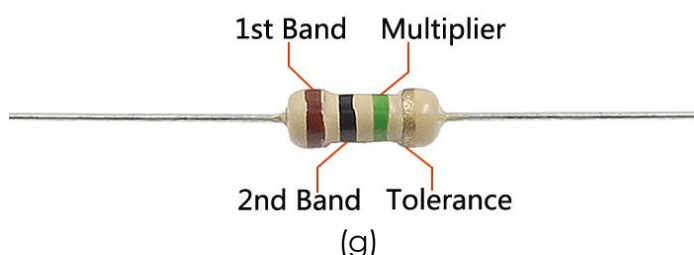


So for this resistor, the resistance should be read from left to right. The value should be in this format: **1<sup>st</sup> Band 2<sup>nd</sup> Band 3<sup>rd</sup> Band  $\times 10^{\text{Multiplier}} (\Omega)$**  and the permissible error is  **$\pm \text{Tolerance}\%$** . So the resistance value of this resistor is 2(red) 2(red) 0(black)  $\times 10^0(\text{black}) \Omega = 220 \Omega$ , and the permissible error is  $\pm 1\%$  (brown).

One more example. The resistance of the resistor below should be 1 (brown) 0 (black) 0 (black)  $\times 10^1(\text{brown}) \Omega = 100 \times 10 \Omega = 1000 \Omega = 1\text{K}\Omega$ , and the permissible error is  $\pm 1\%$  (brown). Now try it by yourself!



Now let's try a 4-band resistor. There are two 4-band resistors in the kit: a  $1\text{M}\Omega$  one and a  $5.1\text{M}\Omega$  one. You may not use such a large resistor in the experiments of the kit but you can use them in other projects. Unlike 5-band resistors, the third band of a 4-band one is not the 3<sup>rd</sup> band but the multiplier; its fourth band is Tolerance. So the resistance value of a 4-band resistor should be **1<sup>st</sup> band 2<sup>nd</sup> band  $\times 10^{\text{Multiplier}} (\Omega)$** , and the permissible error is  **$\pm \text{Tolerance}\%$** .





Read the resistance of the above resistor from left to right. The value is 1 (brown) 0 (black)  $\times 10^5$  (green) =  $10 \times 10^5 \Omega = 10^6 \Omega = 1 \text{ M}\Omega$  and the permissible error is  $\pm 5\%$  ( gold )

The resistance value of the resistor below is 5 (green) 1 (brown)  $\times 10^5$  (green) =  $51 \times 10^5 \Omega = 5.1 \times 10^6 \Omega = 5.1 \text{ M}\Omega$ , and the permissible value is  $\pm 5\%$  (gold).

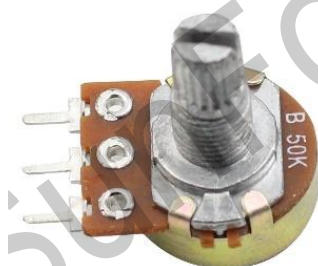


(h)

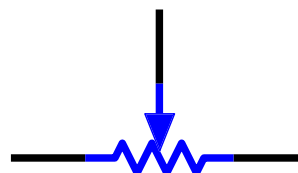
You can also use a multimeter to measure the resistance value of these resistors to double check whether you've read it correctly or not.

## 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. Figure (i) is the potentiometer and figure (j) is the corresponding circuit symbol. The middle pin in figure (i), represented by the arrow in Fig. (j) is the movable brush.



(i)



(j)

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 travel it has made.

### 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 within the travel of the moving contact.

### 3. Serving as a current controller

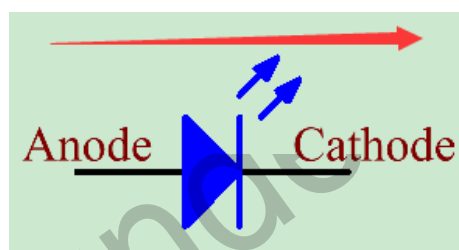
When the potentiometer acts as a current controller, the sliding contact terminal must be connected as one of the output terminals.

## LED

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



(k)



(l)

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

In this kit, LEDs of red, green, yellow and white are provided. An LED has two pins. The longer one is the anode, and shorter one, the 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 voltage 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.

## RGB LED

An RGB LED is provided in this kit. RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216 different colors.

RGB LEDs can be categorized into common anode and common cathode ones. In this experiment, the latter is used. The common cathode, or CC, means to connect the cathodes of the three LEDs. After you connect it with GND and plug in the three pins, the LED will flash the corresponding color. Its circuit symbol is shown as figure (n).



An RGB LED has 4 pins: the longest one is GND; the others are Red, Green and Blue. Touch its plastic shell and you will find a cut. The pin closest to the cut is the first pin, marked as Red, then GND, Green and Blue in turn.



(o)

Or you can distinguish them in another way. As GND is the longest one and can be defined directly, you just need to confirm the other three pins. You can test it by giving them a small voltage. The forward voltage drop from the three pins to the GND are respectively 1.8V (red), 2.5V (blue), and 2.3V (green). Thus, when you connect the same current limiting resistor with the three pins and supply them with the same voltage, the red one is the brightest, and then

comes the green and the blue one. Therefore, you may need to add a current limiting resistor with different resistances to the three pins for these colors.

## 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. Among others, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jumper 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. The reason we call it Male-to-Female is because it has the outstanding tip in one end as well as a sunk female end. Male-to-Male means both side are male and Female-to-Female means both ends are female.



Male-to-Female

(p)



Male-to-Male

(q)



Female-to-Female

(r)

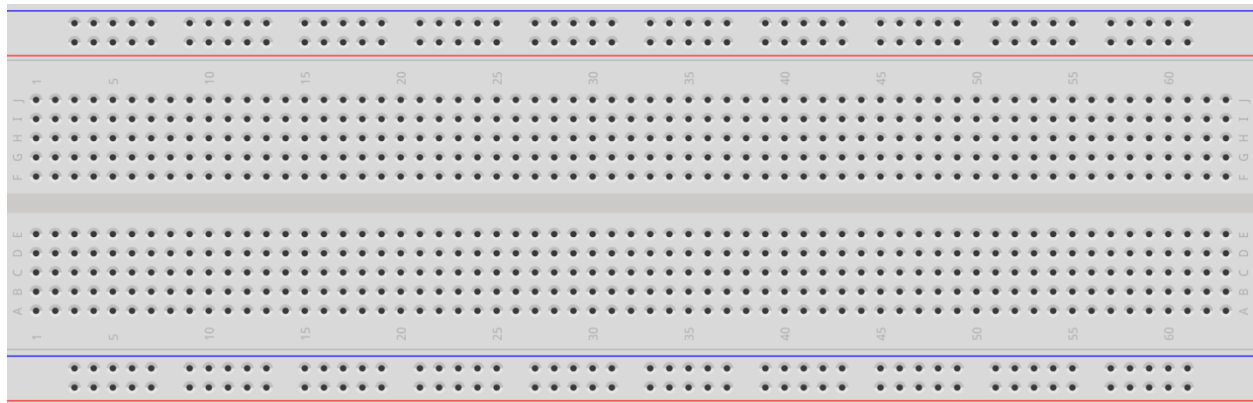
More than one type of them may be used in a project. The colors of the jumper wires are different but it doesn't mean their functions are different accordingly; it's just designed so to better identify the connection between each circuit. The Male-to-Male and Male-to-Female jumper wires are included in the kit. But actually only some Male-to-Male ones will be used in the experiments. You can use the Male-to-Female wires in other experiments.

## Breadboard

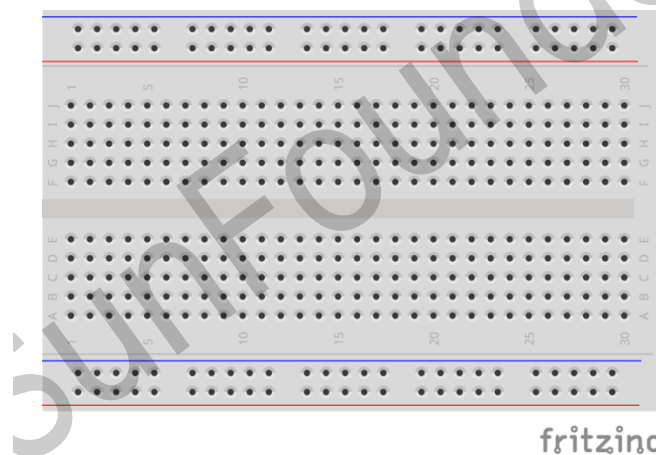
A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finalizing any circuit design. And it has many holes into which components like ICs and resistors as well as jumper wires mentioned above can be inserted. The breadboard allows you to easily plug in and remove components. If there is going to be many changes or if you just want to make a circuit quickly, it will be much quicker than

soldering up your circuit. Therefore, in lots of experiments, it is often used as a hub to connect two or more devices.

Normally, there are two types of breadboard: full+ and half+. You can tell their difference from the names. A half+ breadboard is half the size of a full+ one and their functions are the same. Here take the full+ breadboard.

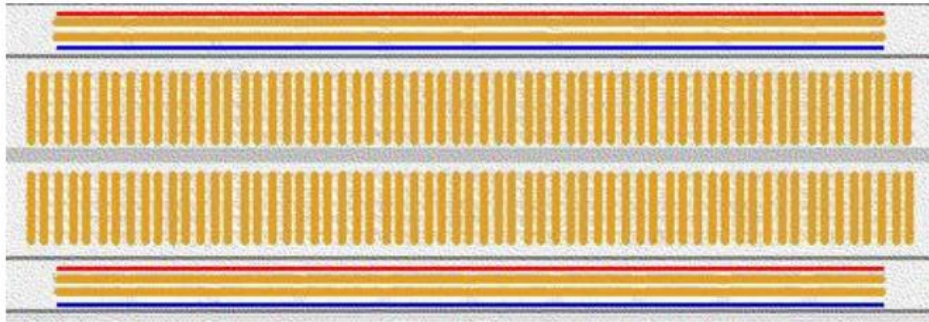


(s) Full+



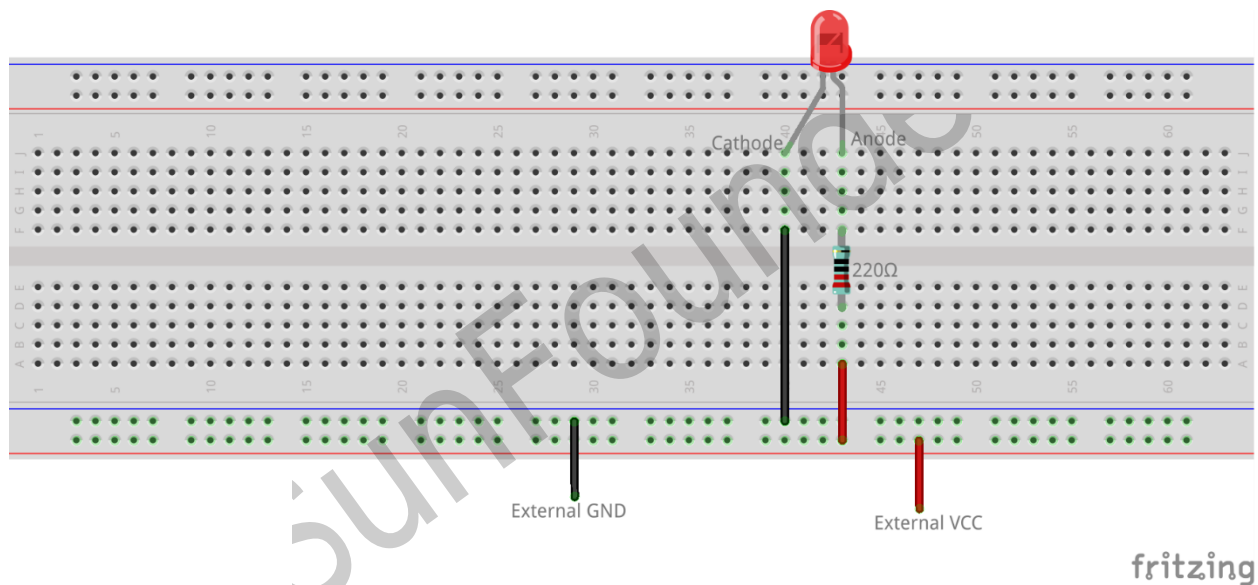
(t) Half+

This is the internal structure of a full+ breadboard. Although there are holes on the breadboard, internally some of them are connected with metal strips. Those holes are to insert pins of devices or wires. As shown in the fig. (t) below, there are four long metal strips on the long sides; the blue and red lines are marked just for clear observation. But you can take the blue line as the GND and red one as VCC for convenience. Every five holes in the middle are vertically connected with metal strips internally which don't connect with each other. You can connect them horizontally with wires or components. A groove is made in the middle on the breadboard for IC chips.



(u) Internal structure of the full+

Now let's make some simple experiment with the breadboard. Turn on an LED as shown in the figure below. You can have a try and the LED will light up. The breadboard makes it possible for you to plug and pull components at any time without welding, which is very convenient for tests.



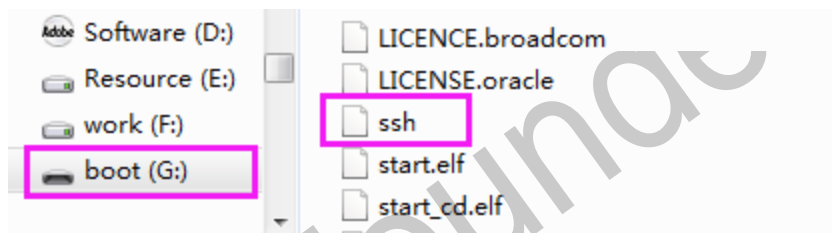
(v)

# Get Started

## Preparation

1. Prepare a MicroSD/TF card of no less than 8GB, a 5V 2A DC power adapter with a MicroUSB port, and a network cable (to connect your router and Raspberry Pi, or plug in the USB Wi-Fi adapter directly if you have one).
2. Download the image for the Raspbian system onto your computer. Refer to instructions through **DOWNLOADS->RASPBIAN** on the official website raspberrypi.org: <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>. Write the image into the microSD/TF card, and then plug the card into the slot on your Raspberry Pi.

**Note:** For 2016-11-25 release or above, SSH (a protocol securing remote login session and other network service) is Disabled by default. Therefore, when you need to log in remotely, you need to create a file named "ssh" under /boot/ to enable it.



## Access to Raspberry Pi's Console

In the subsequent tutorials, the console will be used from time to time. It is platform for interactions in Linux. Therefore, before starting the lessons, you may need to know how to access to Raspberry Pi's console.

### Using Console in GUI (**terminal, recommended for beginners**)

Using console in GUI is of great help for the beginners. You can not only compile and run the code in terminal, but also be able to do some simple file operating, code-downloading, etc. cooperating with GUI.

1. Preparations: a screen monitor, an HDMI cable (if your monitor only support VGA, use a VGA-HDMI converter), a USB mouse, a USB keyboard and a network cable or a USB Wi-Fi dongle.
2. Connect the monitor to power. Then connect it with the Raspberry Pi via the converter cable (HDMI cable). Connect the Ethernet cable or the USB Wi-Fi dongle, and the mouse and keyboard to USB ports. At last, connect a 5V 2A DC power to the RPi. Power on the screen if needed. Then you can see the display showing the Raspberry Pi icon as shown below.



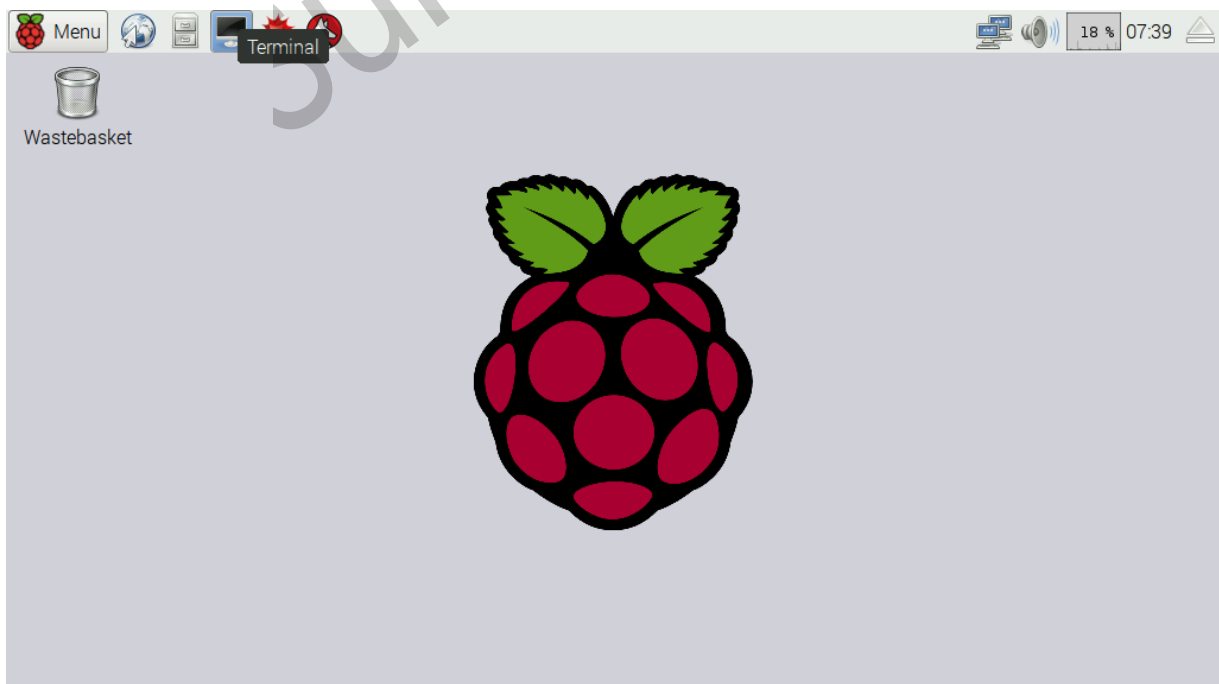
If the monitor displays colored texts with a black background after booting, and that is in console. You can just use this as a terminal (but not recommend for beginners), or change the option for automatically loading a graphic user interface (GUI). To activate GUI, you can type in `startx` with the keyboard and press **Enter**, and to always boot up to GUI, type in `sudo raspi-config` and go through **Boot Options** > **Desktop/Desk Autologin**, and reboot. Wait for a while and the GUI display will show up as below .



**Note:**

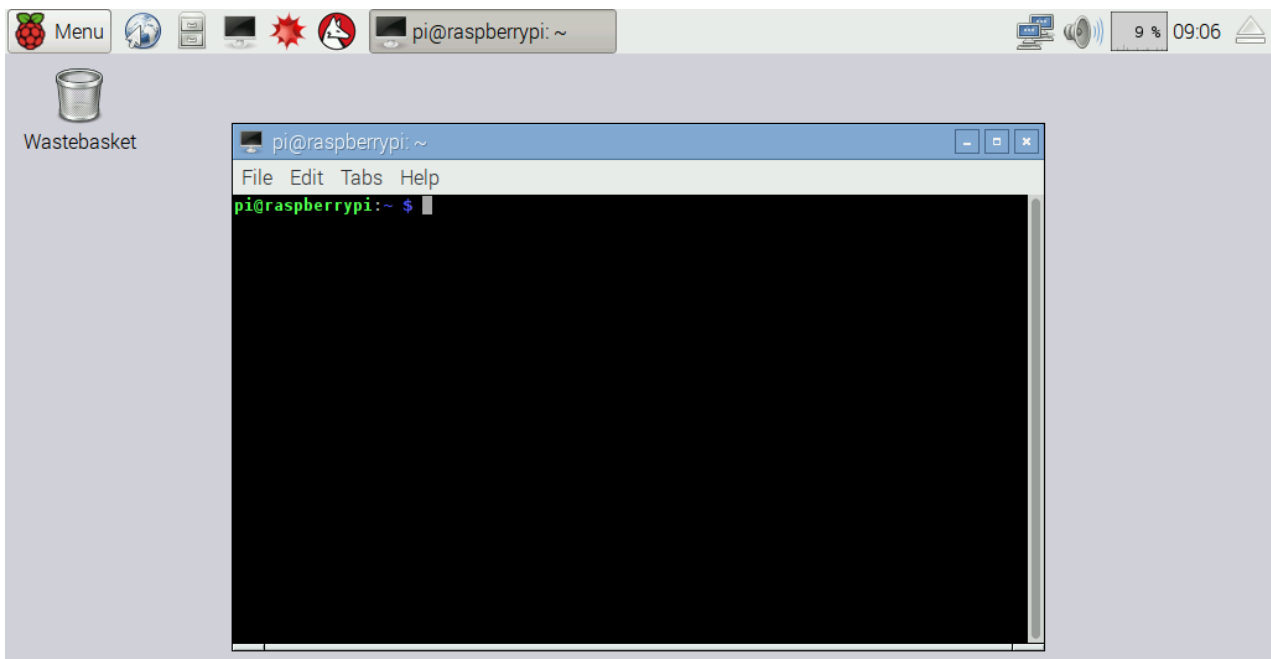
The screen monitor shown above is the 7 inch one we're using, available on our website [www.sunfounder.com](http://www.sunfounder.com) and our Amazon store. Check out now and use your Raspberry Pi in a most convenient way.

3. Now click the icon of **Terminal** on the screen, or press **CTRL+ALT+T** simultaneously.





4. Then a terminal will pop up as follows:



Here's the console we talk about before.

## Using Console Only

There are several ways to use the console only and they can be divided into mainly two ways: using directly and remotely.

### A. Directly

A screen is needed when you use the console directly.

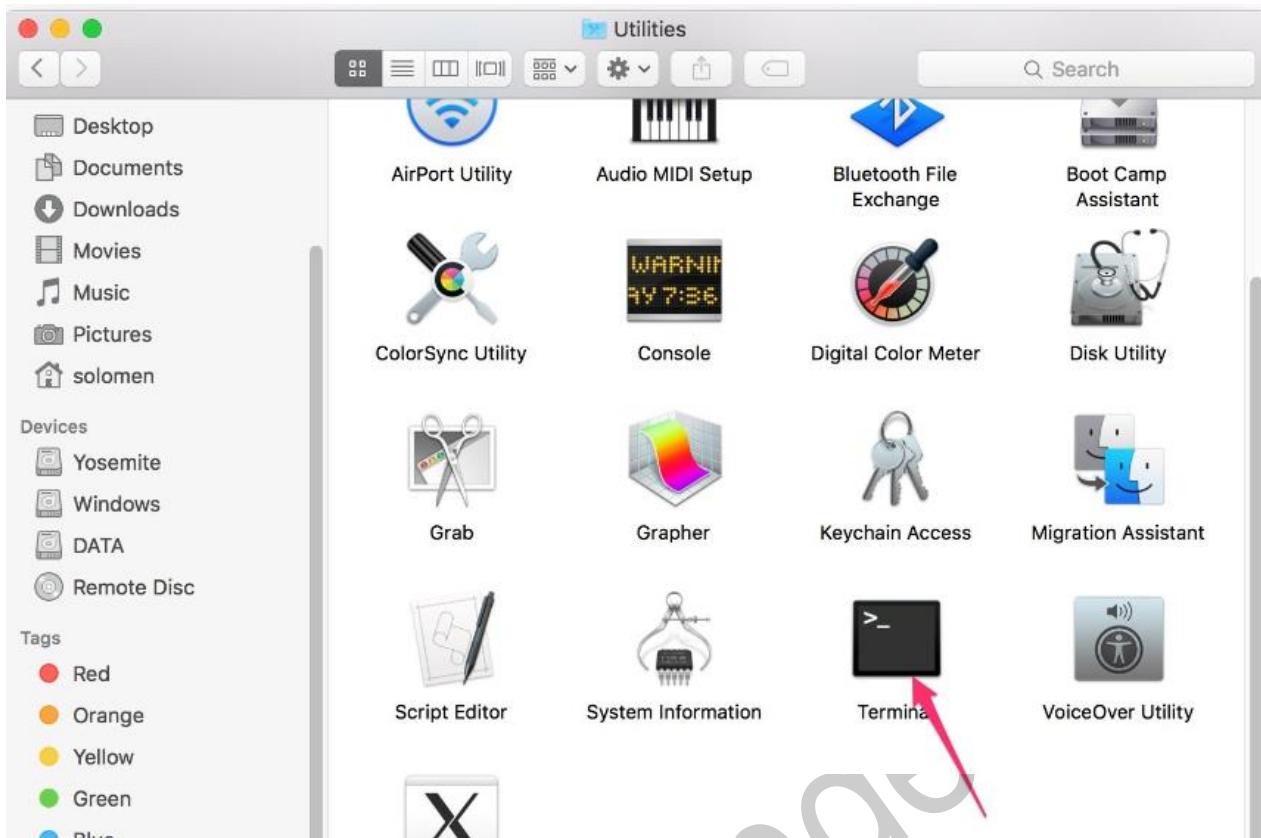
1. Preparations: a screen monitor, an HDMI cable (if your monitor only support VGA, use a VGA-HDMI converter), a USB keyboard and a network cable or a USB Wi-Fi dongle. Yes, you don't need a mouse, or you cannot use mouse.
2. Connect the monitor to power. Then connect it with the Raspberry Pi via the converter cable. Connect the Ethernet cable or the USB Wi-Fi dongle, and keyboard. At last, connect a 5V 2A DC power to the RPi. Power on the screen if needed. Then you can see the console full screen. If you boot into GUI instead, Open a terminal, type in `sudo raspi-config` and press enter, go through **Boot Options > Console/Console Autologin**, and reboot.

### B. Remotely

For three platforms: **Windows**, **Mac** and **Linux**, it might be a little bit different to do this.

**Linux** and **Mac** users can easily log into the Raspberry Pi via ssh.

On Linux or Mac, find **Terminal** and open it.



Type in `ssh pi@<ip_address>`

– `ssh` is the tool for remote login; `pi`, the user name, and `<ip_address>` as the name suggests, your RPi's IP address. For example:

```
ssh pi@192.168.0.1
```

Press **Enter** to confirm.

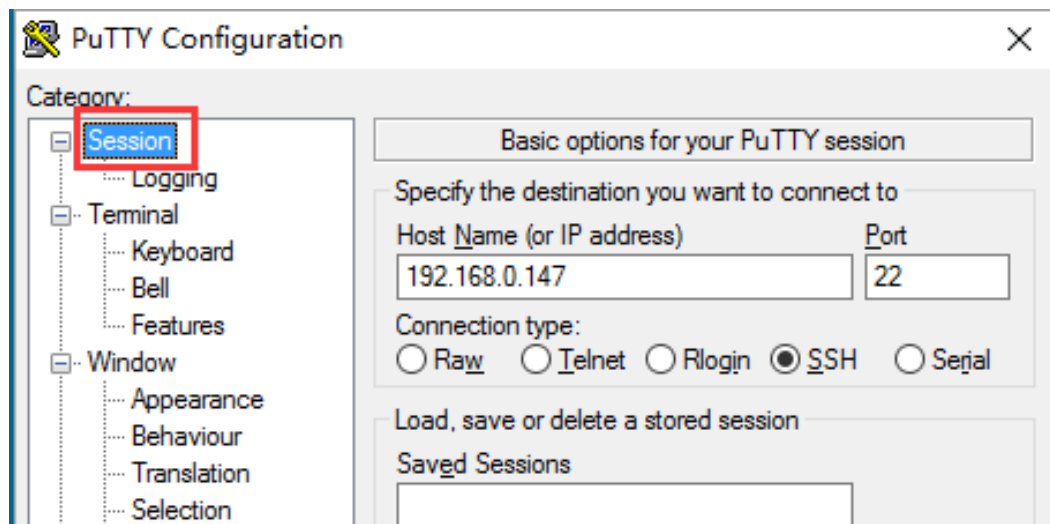
If you get a prompt that no `ssh` is found, you need to install a `ssh` tool like Ubuntu and Debian by yourself:

```
sudo apt-get install ssh
```

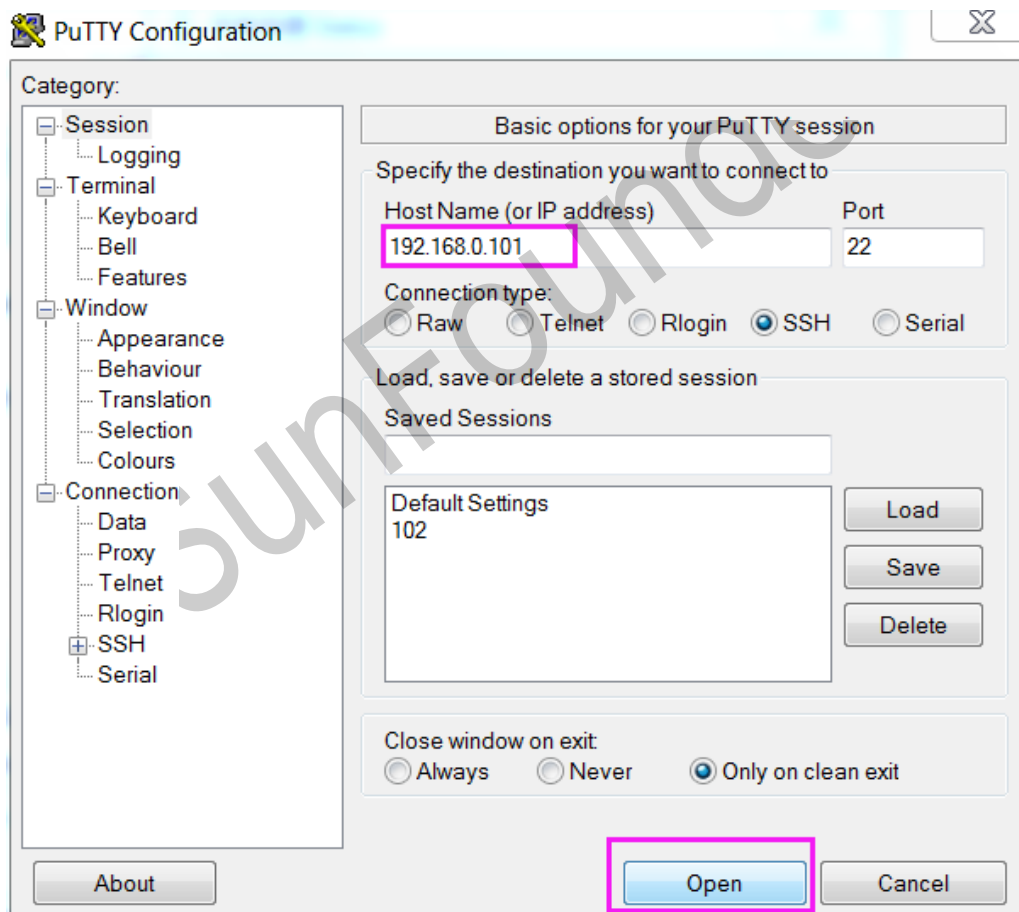
For other platforms, please contact your supplier.

For **Windows users**, you may use a `ssh` tool to log into Raspberry Pi remotely, like PuTTY.

1. Similarly plug the TF card into the Raspberry Pi (RPi), power the RPi with a 5V 2A DC power and connect the Ethernet cable (Better not a USB Wi-Fi dongle). Now). Now the Raspberry Pi is ready.
2. Then you need to know the IP address of the RPi. You can find it on the settings interface of your router. For more details, please refer to your router provider's page.
3. Open PuTTY and click **Session** on the left tree-alike structure (generally it's collapsed upon PuTTY startup):

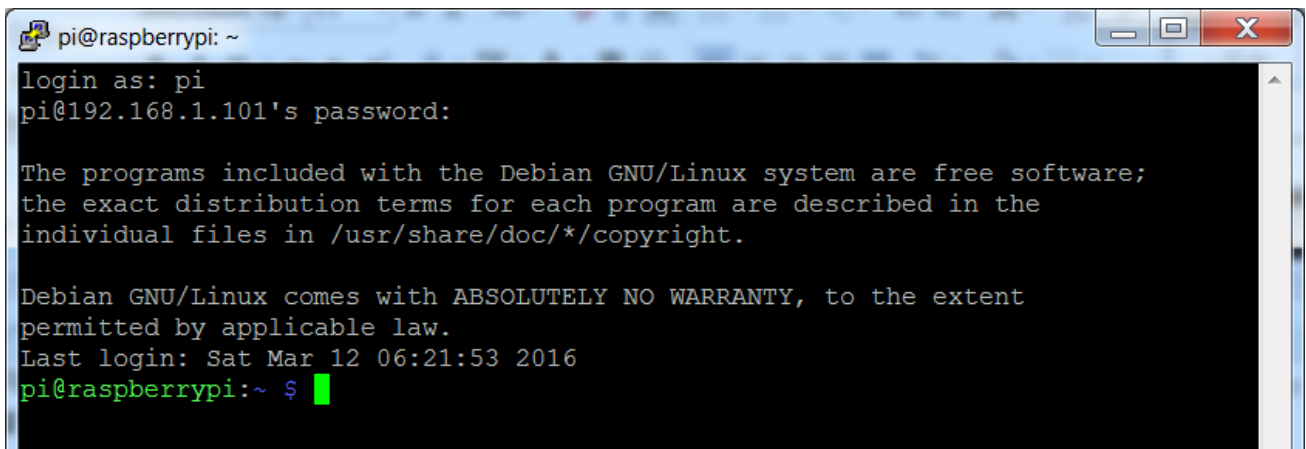


4. Enter the IP address of the RPi you just got in the textbox under Host Name (or IP address) and 22 under Port (by default it is 22)



5. Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, you'll be prompted with a security reminder. Just click **Yes**. When the PuTTY window prompts login as: type in the user name of the RPi: pi, and password: raspberry (the default one, if you haven't changed it).

**Note:** when you're typing the password in, the window shows nothing just null, but you're in fact is typing things in. So just focus on typing it right and press **Enter**. After you log in the RPi successfully, the window will display as follows.



```
pi@raspberrypi: ~
login as: pi
pi@192.168.1.101's password:

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: Sat Mar 12 06:21:53 2016
pi@raspberrypi:~ $
```

Now , no matter what method you take, you can get in the console of your Raspberry Pi.

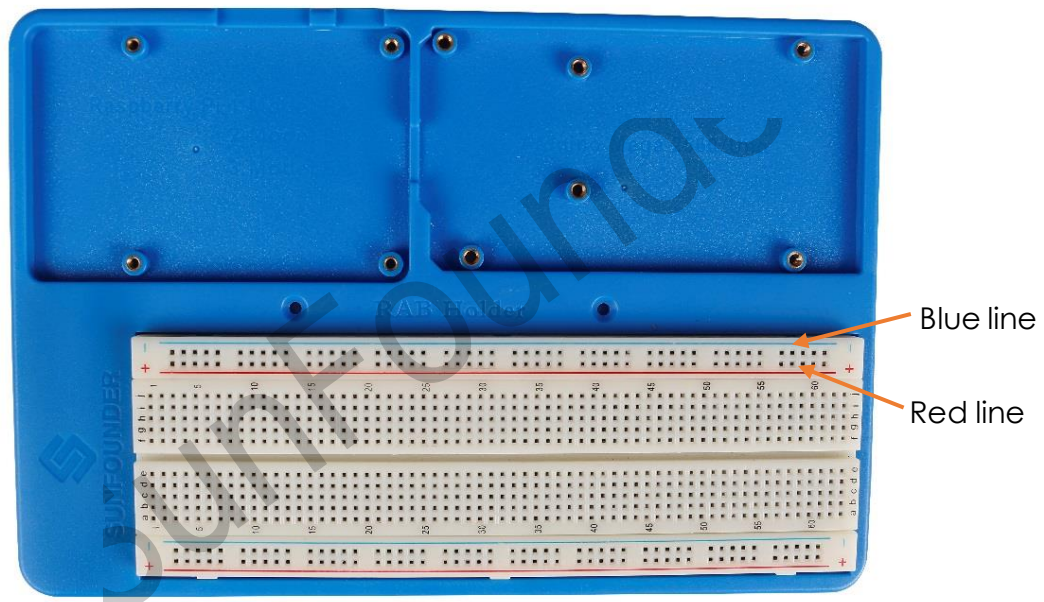
# Introduction to Raspberry Pi

## RAB Holder

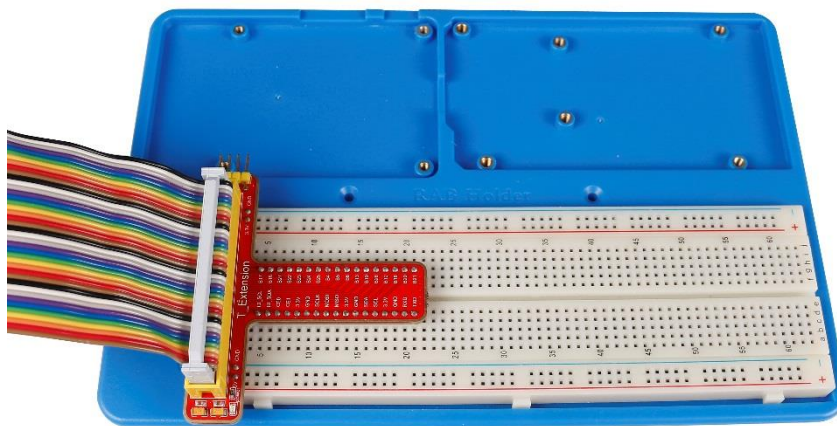
RAB Holder is a basic but indispensable component for your experiment. It makes your experiment easier and can be used for fixing a bread board, an Arduino board like Uno board or Mega2560, or a Raspberry Pi board.

Before starting the experiment, you need to fix the Raspberry Pi and the breadboard on the RAB Holder first.

Remove the sticker from the back of the breadboard first, and fix the breadboard on the RAB Holder. Pay attention to place it in such a position as shown in the figure below, so that the 3.3V pin and 5V pin on the T-Extension Board align with the bus strips besides the two red lines when we insert the T-Extension Board later.

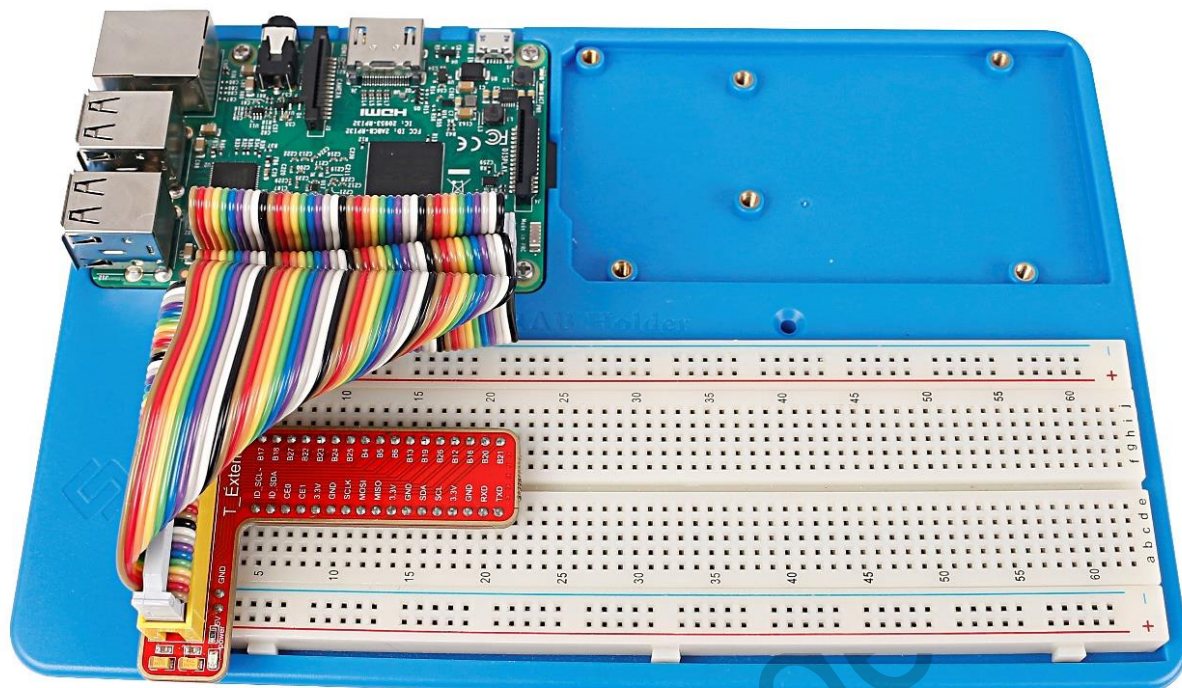


Then insert the T-Extension Board into the Breadboard, and insert the 40-pin GPIO Cable into the board.

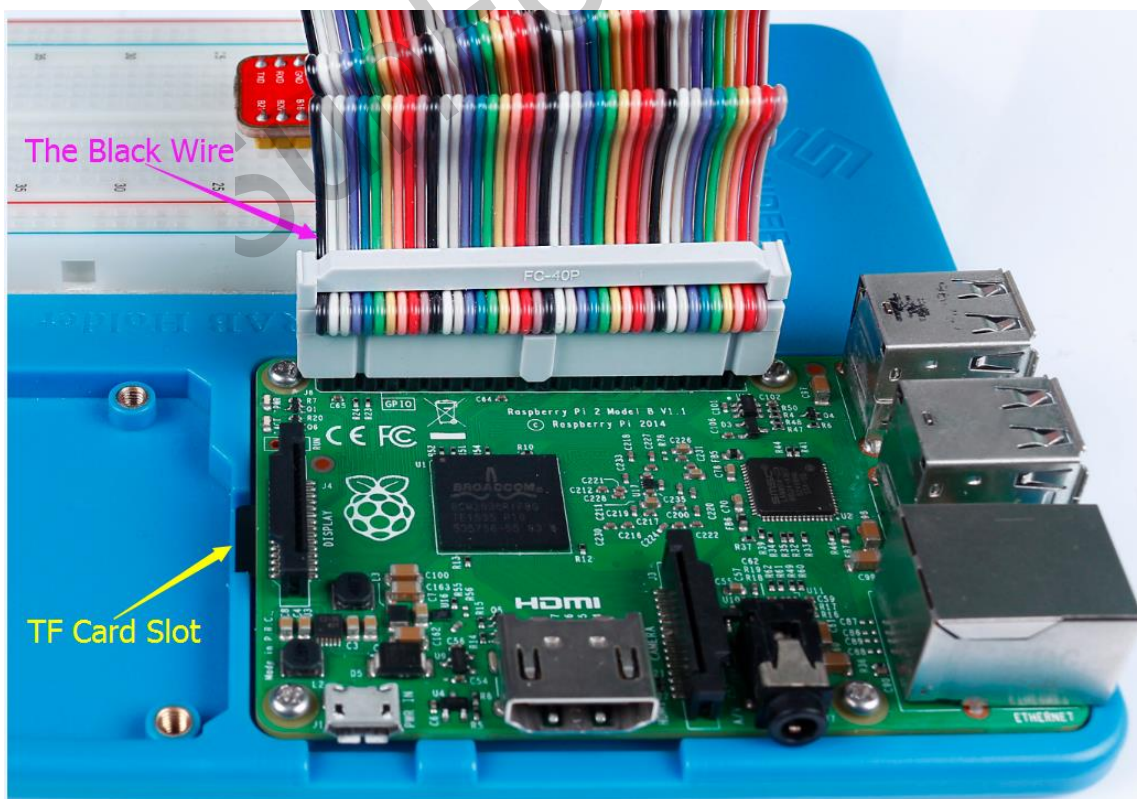




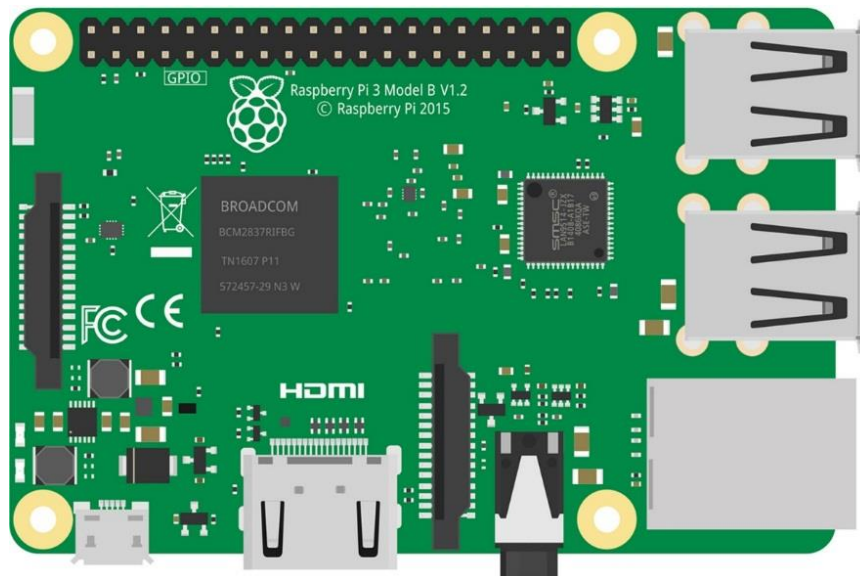
Then place the Raspberry Pi in the holder, fasten it with M2.5x5 screws. Since it may be a little difficult to fasten it, be careful to operate.



Pay attention to the direction when plugging in the 40-pin GPIO Cable into the Raspberry Pi pins. The black wire at the edge should be close to the TF card slot. **DO NOT** connect them inversely, or the Raspberry Pi will be short cut!



## Raspberry Pi



A Raspberry Pi is an indispensable component for the kit, the control device. But it is not included in the kit and you need to prepare one yourself.

The **Raspberry Pi** is a kind of minicomputer for users like amateurs, teachers, students and small businesses, etc. With a pre-installed Linux system, it is credit card-sized and equipped with an ARM architecture processor, whose operational performance is similar to that of smart phone. As for ports, the Raspberry Pi provides USB ports for mouse and keyboard. In addition, there are also ports for the Fast Ethernet, SD card and HDMI display or TV's connection. Being low cost and low consumption, the Raspberry Pi is very suitable for embedded projects. Many people have been able to apply the Pi to a variety of projects including some simple ones for children and complex ones with more advanced functions. You can apply it like a PC for spreadsheets-making, word-processing and games, or to play HD videos of up to 1080p.

It can't be better describing the Raspberry Pi as "**though small, perfectly formed**". It has the same ability compared with the PC and carries ports for the USB, Ethernet, HDMI, RCA, and 3.5mm stereo jack. Moreover, it can control GPIOs, while the PC cannot. Through this kit, you will learn how to use the GPIOs to make simple experiments and how to program.

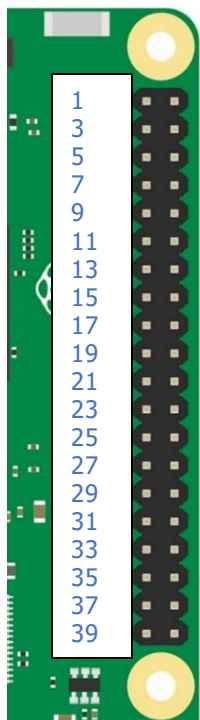
The Raspberry Pi evolves through many versions including the latest (so far) Raspberry Pi 3 Model B, 2 model B, 1 Model B+, Zero, and 1 Model A+. Certainly, **the newer, the more powerful**. The 3 model B now even supports Bluetooth and Wi-Fi. You can choose according to actual needs.

## Raspberry Pi Pin Name

There are no pins printed on the latest Raspberry Pi, which may bring various troubles to new users. The following is the actual pins definition and form definition of the Raspberry Pi. The



actual pins are corresponding to the Physical on the right. There are 40 pins in total from left to right. The columns near Physical are the V (voltage), Mode (Input or Output), Name (original name), wPi (wiringPi, C language based on), BCM (Python based on) of the pins.



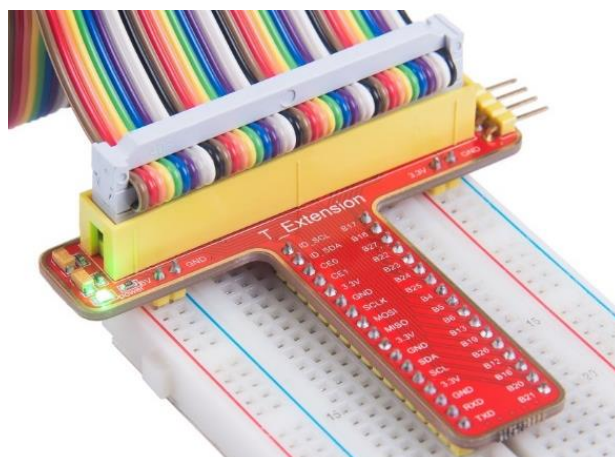
Pi 3											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
2	8	3.3v		1	1	2		5v			
3	9	SDA.1	ALT0	1	3	4		5V			
4	7	SCL.1	ALT0	1	5	6		0v			
		GPIO. 7	IN	0	7	8	1	TxD	15	14	
		0v			9	10	1	RxD	16	15	
17	0	GPIO. 0	IN	0	11	12	0	GPIO. 1	1	18	
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	GPIO.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

Besides the original name of the pins, there are other three ways of naming including physical, wiringPi and BCM. If you see a pin being defined as 0 in the C language, its original name is GPIO 0. In Python code, it's 17 (BCM) or 11 (physical). So you need to know the name, physical, wiringPi and BCM of a pin.

## Extension Board

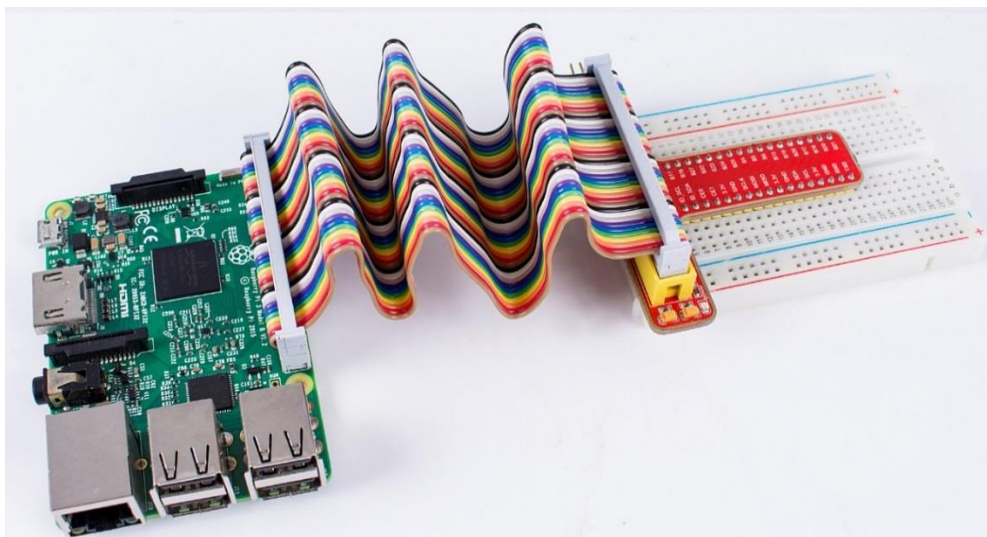
The function of the extension board is to lead out pins of the Raspberry Pi to breadboard by GPIO Extension Board to avoid GPIO damage caused by frequent plugging in or out. For plugging convenience, we designed it in T-shape and name it T-Shape Extension Board.

### 40-pin GPIO Extension Board





This is our 40-pin GPIO Extension Board and GPIO cable for Raspberry Pi model B+, 2 model B and 3 model.

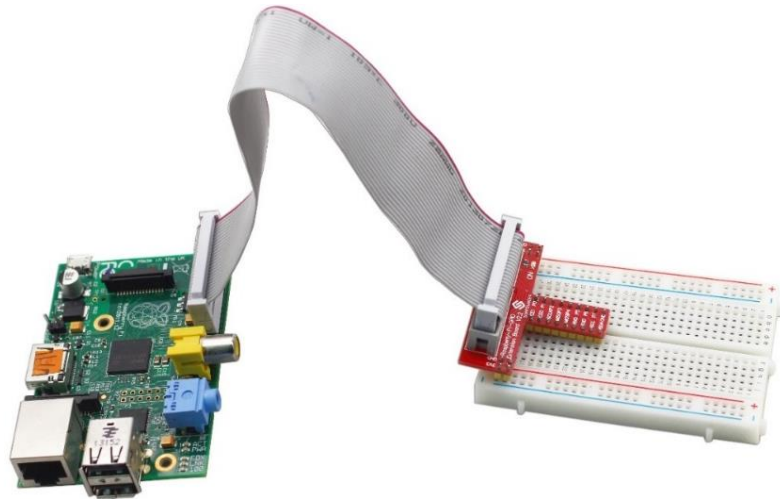


For your better understanding of every pins, we have drawn a table for you to know the Name, BCM and wiring pi of each pin.

Name	wiringPi	BCM		BCM	wiringPi	Name	
T Extension							
ID_SCL	31	1	ID_SCL	B17	17	0	GPIO0
ID_SDA	30	0	ID_SDA	B18	18	1	GPIO1
CEO	30	8	CEO	B27	27	2	GPIO2
CE1	11	7	CE1	B22	22	3	GPIO3
3.3V	3.3V	3.3V	3.3V	B23	23	4	GPIO4
0V	GND	GND	GND	B24	24	5	GPIO5
SCLK	14	11	SCLK	B25	25	6	GPIO6
MOSI	12	10	MOSI	B4	4	7	GPIO7
MISO	13	9	MISO	B5	5	21	GPIO21
3.3V	3.3V	3.3V	3.3V	B6	6	22	GPIO22
0V	GND	GND	GND	B13	13	23	GPIO23
SDA	8	2	SDA	B19	19	24	GPIO24
SCL	9	3	SCL	B26	26	25	GPIO25
3.3V	3.3V	3.3V	3.3V	B12	12	26	GPIO26
0V	GND	GND	GND	B16	16	27	GPIO27
RXD	16	15	RXD	B20	20	28	GPIO28
TXD	15	14	TXD	B21	21	29	GPIO29

## 26-pin GPIO Extension Board

26-pin GPIO Extension Board and GPIO cable for Raspberry Pi model B and Raspberry Pi model A.



Also, we have drawn a table of the corresponding BCM, wiringPi and Name of each pins.

Name	wiringPi Pin	BCM GPIO	BCM GPIO	wiringPi Pin	Name
Raspberry-Pi-GPIO Extension Board V2.2					
CE1	11	7	CE1 P0	17	0 GPIO0
CE0	10	8	CE0 P1	18	1 GPIO1
SCLK	14	11	SCLK P2	21	2 GPIO2
MISO	13	9	MISO P3	22	3 GPIO3
MOSI	12	10	MOSI P4	23	4 GPIO4
RXD	16	15	RXD P5	24	5 GPIO5
TXD	15	14	TXD P6	25	6 GPIO6
SCL0	9	1	SCL P7	4	7 GPIO7
SDA0	8	0	SDA GND	0V	0V 0V

# GPIO Libraries

## WiringPi

### Introduction

WiringPi is a GPIO library for C applied to the Raspberry Pi. 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.

Now the Raspbian Jessie 2016-05-27 has wiringPi pre-installed, you can use it directly.

Test whether wiringPi is installed or not.

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

Raspberry Pi Details:
Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Embest
* Device tree is enabled.
* This Raspberry Pi supports user-level GPIO access.
  -> See the man-page for more details
  -> ie. export WIRINGPI_GPIOMEM=1
```

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

Use the command below to see the GPIO layout

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

## RPi.GPIO

If you are a Python user, you can program GPIOs with API provided by RPi.GPIO and use BCM numbering method to control the GPIOs of Raspberry Pi. Please note that it differs from the way that using wiringPi numbering method to control the GPIO on a Raspberry Pi in C language.

### Introduction

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/>.

Now the Raspbian Jessie 2016-05-27 has RPi.GPIO pre-installed, you can use it directly, too.

Test whether RPi.GPIO is installed or not:

Type in **python** to python CLI:

```
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, Type in:

```
import RPi.GPIOimport RPi.GPIO
```

If no error prompts, it means RPi.GPIO is installed.

```
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
>>> 
```

Then, type in

```
RPi.GPIO.VERSION
```

```
>>> RPi.GPIO.VERSION
'0.6.2'
>>> 
```

If it shows its version like above, your Pi is ready to go!

**So in this kit, please note that the example code is ONLY test on Raspbian.**

SunFoundation

# Download the Code

We provide two methods for download:

## Method 1: Use git clone (Recommended)

Log into Raspberry Pi's console, just as previously shown.

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/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi.git
```

The advantage of this method is that, you can update the latest code any time you want, using `git pull` under the folder.

## Method 2: Download the code.

Download the source code from our website, [www.sunfounder.com](http://www.sunfounder.com)

Click **LEARN->Get Tutorials**, find **Super Kit V3.0 for Raspberry Pi** and click to download the **Super Kit for Raspberry Pi.zip** file, unzip it, and then move the folder to the directory `/home/pi/`.

This is a simple way to do it.

# Lesson 1 Blinking LED

## Introduction

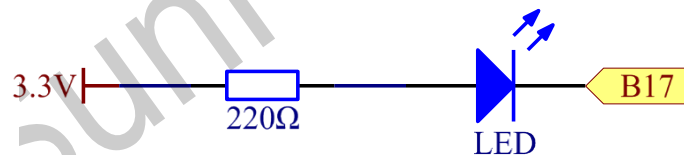
In this lesson, we will learn how to program Raspberry Pi to make an LED blink. You can play numerous tricks with an LED as you want. Now get to start and you will enjoy the fun of DIY at once!

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* T-Extension Board
- 1 \* 40-Pin Cable
- 1 \* LED
- 1 \* Resistor (220Ω)
- Jumper wires

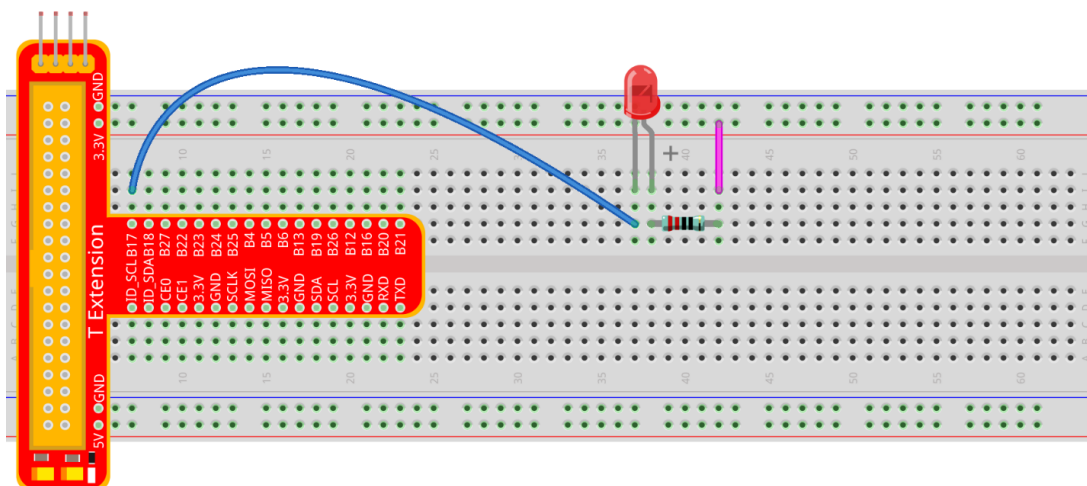
## Principle

In this experiment, connect a 220Ω resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to B17 of Raspberry Pi. We can see from the schematic diagram that the anode of LED connects to a current-limiting resistor and then to 3.3V. **Therefore**, to turn on an LED, we need to make B17 low (0V) level. It can be realized by programming.



## Experimental Procedures

### Step 1: Build the circuit



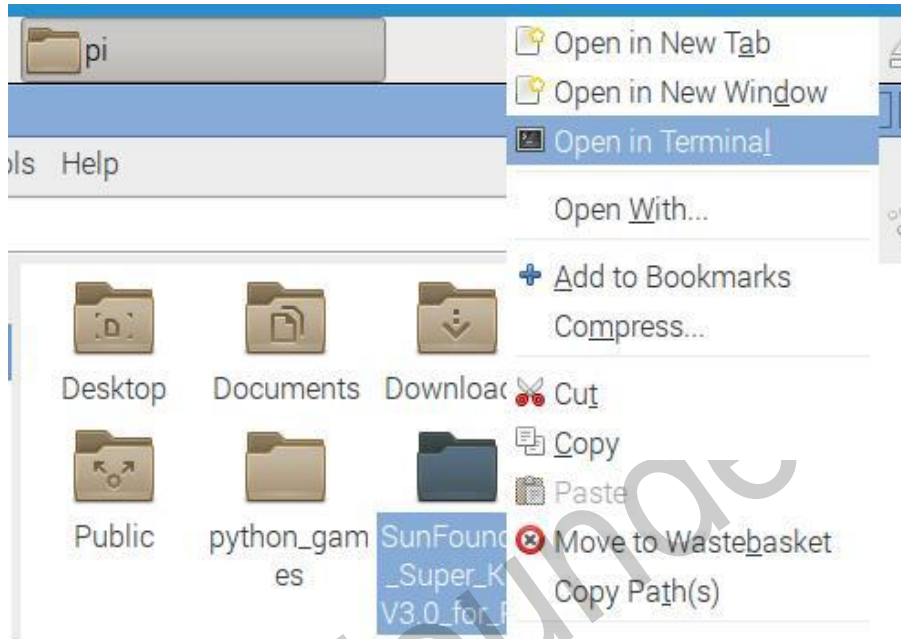
**For C language users:**

**Step 2:** Go to the folder of the code.

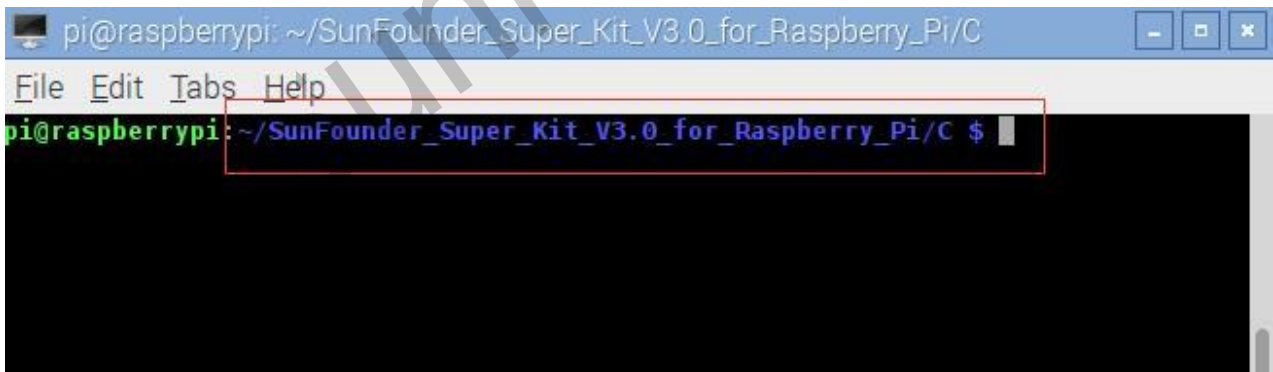
**If you use a monitor, you're recommended to take the following steps.**

Go to `/home/pi/` and find the folder `SunFounder_Super_Kit_V3.0_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 `01_blinkLed.c`



In the lessons later, we will show how to get into the folder of the code in commandway, not with the display. You only need to find out the code file and right click **Open in Terminal**. You can back to lesson 1 to check if you forgot. Certainly, the terminal can be opened if you're using display, and then use `cd` command directly to go to the code path.

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

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

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



```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
login as: pi
pi@192.168.0.131's password:
Server refused to set all environment variables

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: Thu Jul 21 06:42:51 2016 from 192.168.0.130
pi@raspberrypi:~ $ cd ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C/
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $
```

In either way, you now are in the folder C. The subsequent procedures under the two methods are the same. Let's move on.

### Step 3: Compile the Code

```
gcc 01_blinkLed.c -o 01_blinkLed -lwiringPi
```

**Note:** `gcc` is GNU Compiler Collection. Here, it functions like compiling the C language file `01_blinkLed.c` and outputting an executable file `01_blinkLed`. In the command, `-o` means outputting and `-lwiringPi` is to load the library `wiringPi` (`l` is short for library). If you want to write your own C code and compile to run it, you need to master `gcc`.

Since the `gcc` command is too long, you can use `make` to test the experimental effect of the kit to make the process quicker and more convenient.

```
make 01_blinkLed
```

**Note:** The `make` command will compile according to the rules in the Makefile. Two files will be generated after compiling: `*.o` and an executable file.

We use `makefile`, in essence, is to write the compilation method of `gcc` into the automated script. If you have written your own program in C language, you need to write and modify the `makefile` so as to use `make` command to compile your C code.

### Step 4: Run the executable file output in the previous step:

```
sudo ./01_blinkLed
```

**Note:** To control the GPIO, you need to access to led with the permission of superuser (`sudo` is not needed to control the GPIO for the raspbian system after 2016-5-27), namely, by the command `sudo`. In the command `./` indicates the current directory. The whole command is to run the `01_blinkLed` in the current directory.

```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
File Edit Tabs Help
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 01_blinkLed
[cc] 01_blinkLed.c
[link]
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./01_blinkLed

=====
Blink LED
-----
LED connect to GPIO0
LED will Blink at 500ms
SunFounder|
=====

...LED on
LED off...
...LED on
LED off...
...LED on
```

If you want to view the code `01_blinkLed.c`, press **Ctrl + C** to stop running the code. Then type the following command to open it:

```
nano 01_blinkLed.c
```

**Note:** nano is a text editor tool. The command is to open the code file `01_edblinkLed.c` by this tool.

```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
File Edit Tabs Help
GNU nano 2.2.6 File: 01_blinkLed.c
*****
* Filename : blinkLed.c
* Description : Make an led blinking.
* Author : Robot
* E-mail : support@sunfounder.com
* website : www.sunfounder.com
* Update : Caven 2016/07/01
*****
#include <wiringPi.h>
#include <stdio.h>

#define LedPin 0

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

    pinMode(LedPin, OUTPUT);
}
```

## Code Explanation

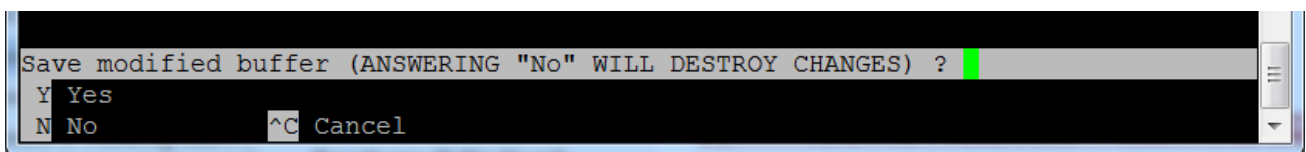
`#include <wiringPi.h>`//The hardware drive library designed for the C language of Raspberry Pi. Adding this library is convenient for hardware initialization, I/O ports, PWM outputs, etc.

`#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.

```
#define LedPin    0 // Pin B17 of the T_Extension Board is corresponding to the pin0 in
wiringPi, namely, GPIO 0 of the raspberry Pi. Assign GPIO 0 to LedPin, LedPin represents
GPIO 0 in the code later.

pinMode(LedPin, OUTPUT)// Set LedPin as output to write value to it.
digitalWrite(LedPin, LOW)// Set GPIO0 as 0V (low level). Since the cathode of LED is
connected to GPIO0, thus the LED will light up if GPIO0 is set low. On the contrary, set
GPIO0 as high level, digitalWrite (LedPin, HIGH): LED will go out.
```

Press **Ctrl+X** to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in **Y** (save) or **N** (don't save). Then press **Enter** to exit. Repeat **Step 3** and **Step 4** to see the effect after modifying.



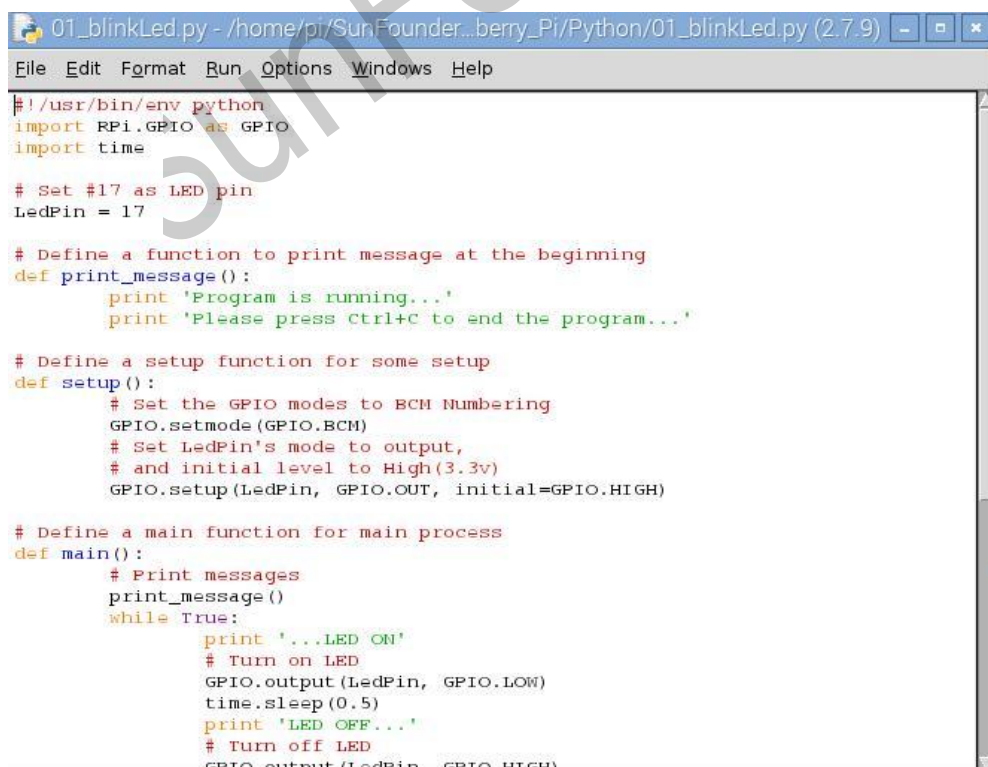
### For Python users

**Step 2:** Go to the folder of the code and run it.

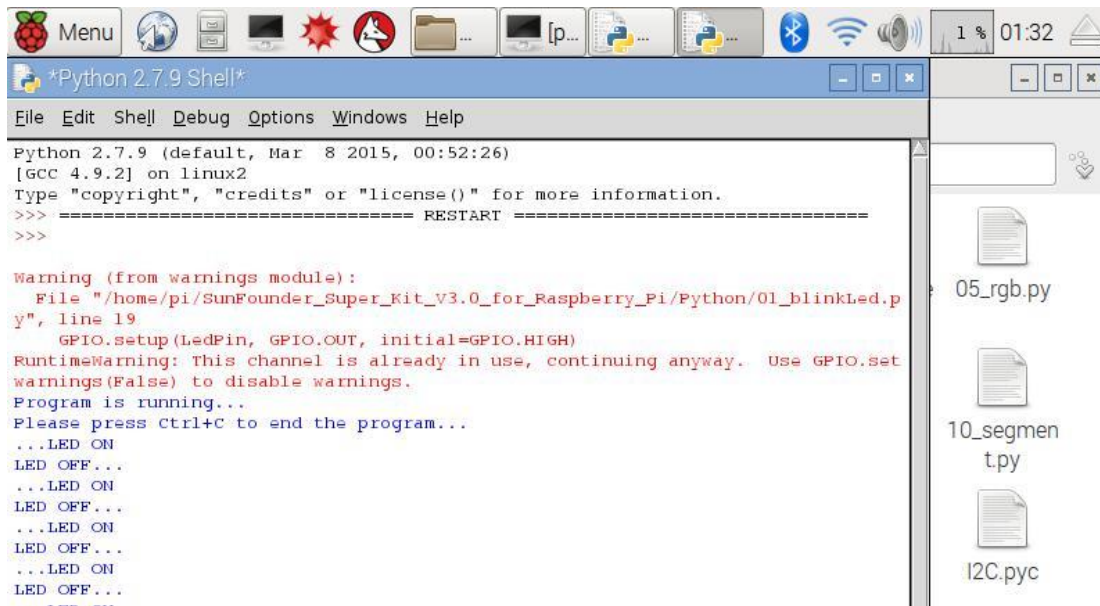
Open the downloaded folder *SunFounder\_Super\_Kit\_V3.0\_for\_Raspberry\_Pi/Python* and you can see them.

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

Find *01\_blinkLed.py* and double click it to open. Now you're in the file.



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



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

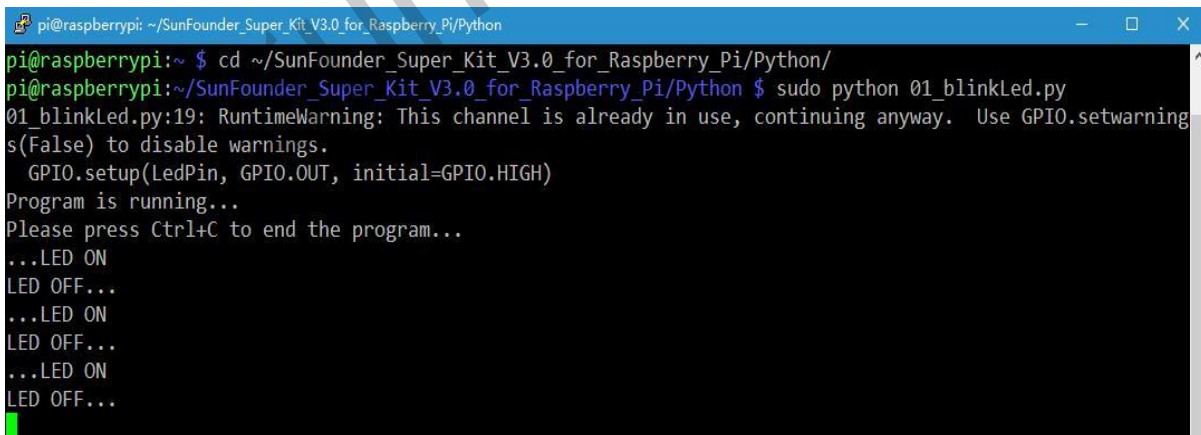
**If you want to log into the Raspberry Pi remotely, type in the command:**

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

Run the code:

```
sudo python 01_blinkLed.py
```

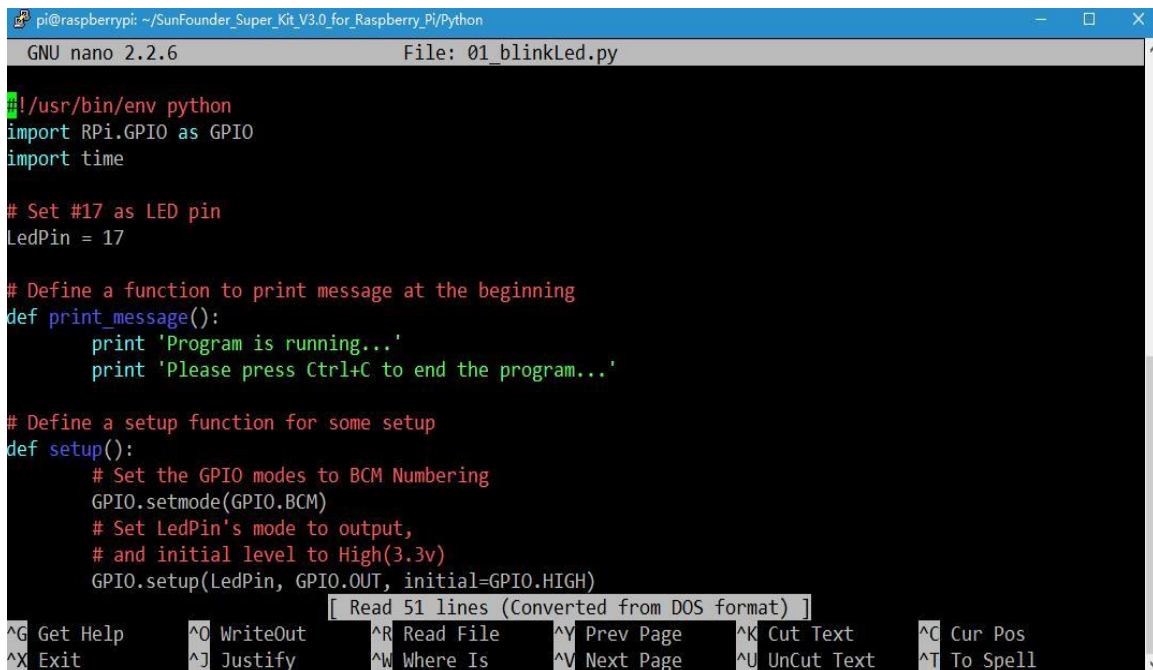
**Note:** Here `sudo` – superuser do, and `python` means to run the file by Python.



If you want to view the code `01_blinkLed.py`, press **Ctrl + C** to stop running the code. Then type the following command to open it:

```
nano 01_blinkLed.py
```

**Note:** `nano` is a text editor tool. The command is to open the code file `01_blinkLed.c` by this tool.



```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
GNU nano 2.2.6 File: 01_blinkLed.py

#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

# Set #17 as LED pin
LedPin = 17

# Define a function to print message at the beginning
def print_message():
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'

# 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 High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)

[ Read 51 lines (Converted from DOS format) ]

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

## Code Explanation

#!/usr/bin/env python:

When the system detects this, it will search the installation path of python in the env setting, then call the corresponding interpreter to complete the operation. It's to prevent the user not installing the python onto the /usr/bin default path.

import RPi.GPIO as GPIO # import RPi.GPIO package, thus python code control GPIO easily with it.

import time # import time package, for time delay function in the following program.

LedPin = 17 # LED connects to the B17 of the T-shape extension board, namely, the GPIO 0 of the Raspberry Pi.

# Define a setup function for some setup

def setup():

GPIO.setmode(GPIO.BCM) # Set the GPIO modes to BCM Numbering

# Set LedPin's mode to output, and initial level to High (3.3v)

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

# 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.LOW)



```

        # delay 0.5 second, which is equals to the delay in C language, using second as
the unit,

        time.sleep(0.5)
        print 'LED OFF...'

        # Turn off LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Turn off LED
    GPIO.output(LedPin, GPIO.HIGH)

    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()

        # When 'Ctrl+C' is pressed, the child program destroy () will be executed.
    except KeyboardInterrupt:
        destroy()

```

```

pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
GNU nano 2.2.6 File: 01_blinkLed.py

#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

# Set #17 as LED pin
LedPin = 17

# Define a function to print message at the beginning
def print_message():
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'

# 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 High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)

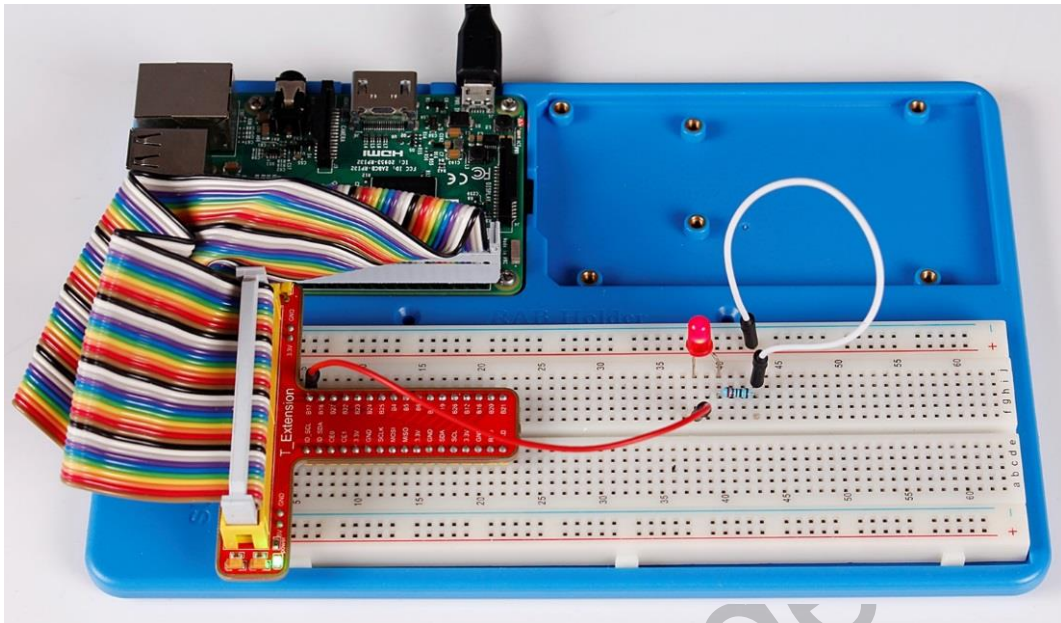
[ Read 51 lines (Converted from DOS format) ]
^G Get Help ^O WriteOut ^R Read File ^V Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^N Next Page ^U UnCut Text ^T To Spell

```

Press **Ctrl+X** to exit. If you have modified the code, there will be a prompt asking whether to save the changes or not. Type in **Y** (save) or **N** (don't save).

Then press **Enter** to exit. Type in `nano 01_blinkLed.py` again to see the effect after the change.

Run the code to make it work. It will be like below:



### Further Exploration

If you want the LED to speed up the blinking, just change the delay time. For example, change the time to `delay (200)` (for C) or `time.sleep(0.2)` (for python) in the program, recompile and run, and then you will see the LED blink faster.

### Summary

Raspberry Pi packages many low-level detail designs, which ease your way to explore your own apps. Maybe that is the charm of Raspberry Pi. Now you have already learnt how to use the Raspberry Pi GPIO to blink an LED. Keep moving to the next contents.

### FAQ

If you haven't modified the code, you do not need to run `make 01_blinkLed` again.

```
make 01_blinkLed
```

Or a message will appear: `make: '01_blinkLed' is up to date`

```
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ gcc 01_blinkLed.c -o 01_blinkLed -lwiringPi
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 01_blinkLed
make: '01_blinkLed' is up to date.
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $
```

It will not appear only when you run the make command after having changed the code and saved it.

**Tips:** For any **TECHNICAL** questions, add a topic under **FORUM**  section on our website [www.sunfounder.com](http://www.sunfounder.com) and we'll reply as soon as possible.



## Lesson 2 Controlling an LED by a Button

### Introduction

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

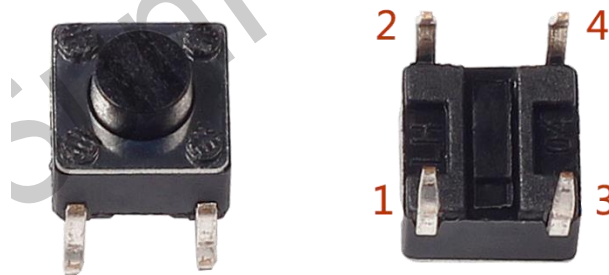
### Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* LED
- 1 \* Button
- 1 \* Resistor (220Ω)
- Jumper wires
- 1 \* T-Extension Board
- 1 \* 40-Pin Cable

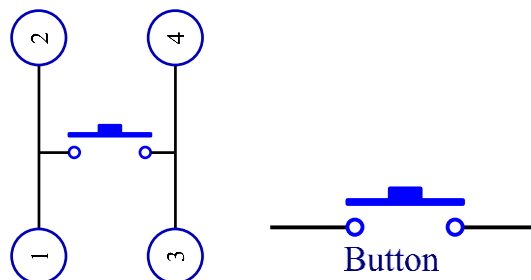
### Principle

#### Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect 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. Pin 1 is connected to pin 2 and pin 3 to pin 4. So you just need to connect either of pin 1 and pin 2 to pin 3 or pin 4.

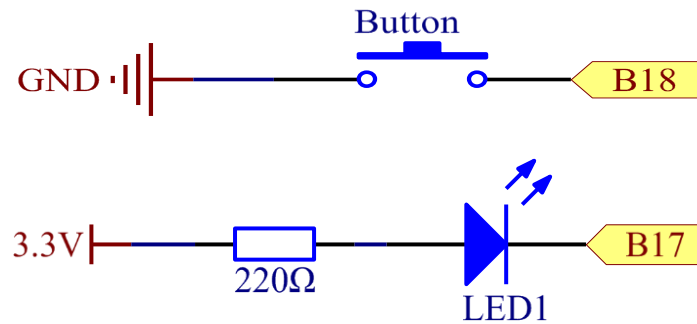


The following is the internal structure of a button. Since the pin 1 is connected to pin 2, and pin 3 to pin 4. The symbol on the right below is usually used to represent a button in circuits.



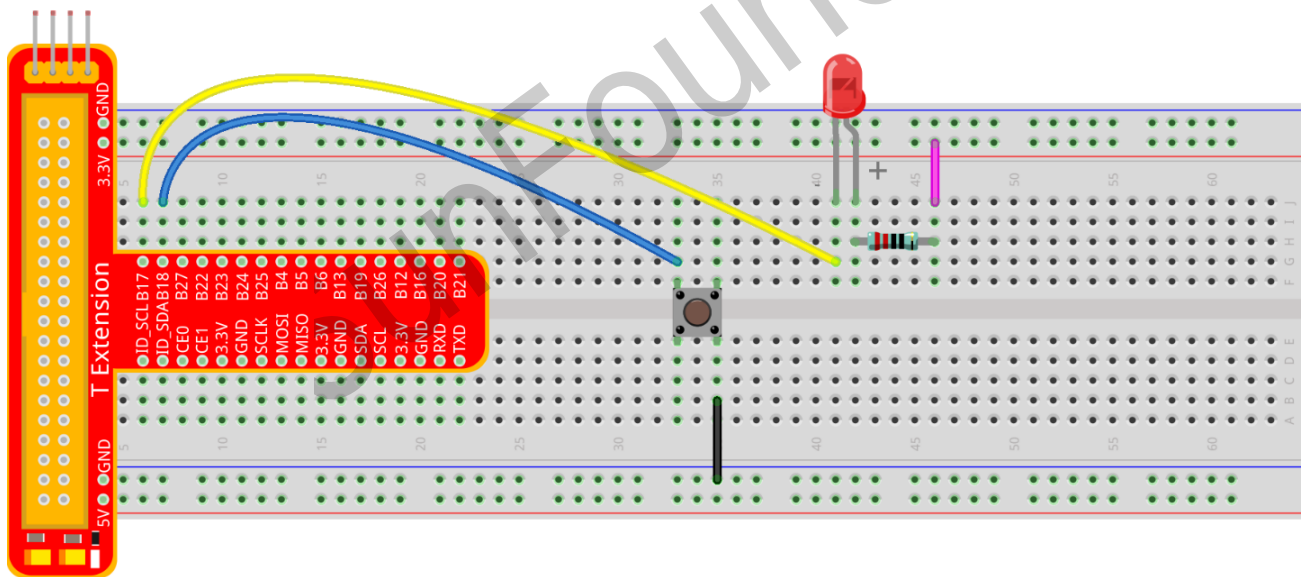
When the button is pressed, the 4 pins are connected, thus closing the circuit.

Use a normally open button as the input of Raspberry Pi, the detailed connection is as shown in the schematic diagram below. When the button is pressed, the B18 will turn into low level (0V). We can detect the state of the B18 through programming. That is, if the B18 turns into low level, it means the button is pressed. You can run the corresponding code when the button is pressed, and then the LED will light up. **Note:** The longer pin of the LED is the anode and the shorter one is the cathode.



## Experimental Procedures

### Step 1: Build the circuit



### For C language users:

**Step 2:** Open the code file:

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

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

**Step 3:** Compile the Code

```
gcc 02_buttonControlled.c -o 02_buttonControlled -lwiringPi
```

or

```
make 02_buttonControlled
```

**Step 4:** Run the executable file above

```
sudo ./02_buttonControlled
```

**Step 5:** Check the code

```
nano 02_buttonControlled.c
```

### Code Explanation

```
#define LedPin    0  Pin B17 in the T_Extension Board connects to the GPIO0. GPIO0
corresponds to pin0 in the wiringPi pin figure. So in C program, LedPin is defined as 0.
#define ButtonPin 1 Pin B18 in the T_Extension Board connects to the GPIO8. GPIO8
corresponds to pin1 in the wiringPi pin figure. So in C program, LedPin is defined as 1.
pinMode(LedPin, OUTPUT) Set LedPin as output to assign value to it.
pinMode(ButtonPin, INPUT) Set ButtonPin as input to read the value of ButtonPin.
pullUpDnControl(ButtonPin, PUD_UP) Set the ButtonPin as pull-up input. When the button
is not pressed, the I/O port is 3.3V. When the button is pressed, the I/O port connects
to GND (0V). You can judge the button status by reading the level value of the I/O port.
while(1){
    // indicate that button has pressed down
    if(digitalRead(ButtonPin) == 0){
        // LED on
        digitalWrite(LedPin, LOW);
        printf("...LED on\n");
    }
    else{
        // LED off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
    }
}
```

```
digitalWrite (LedPin, HIGH) in while: close the LED. if (digitalRead (ButtonPin) == 0:
check whether the button has been pressed. Execute digitalWrite(LedPin, LOW) when
pressed to light up LED.
```

Press **Ctrl+X** to exit, if you have modified the code, there will be a prompt asking whether to save the changes or not. Type in **Y** (save) or **N** (don't save). Then press **Enter** to exit. Repeat **Step 3** and **Step 4** to see the effect after modifying.

### For Python users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run the code

```
sudo python 02_buttonControlled.py
```

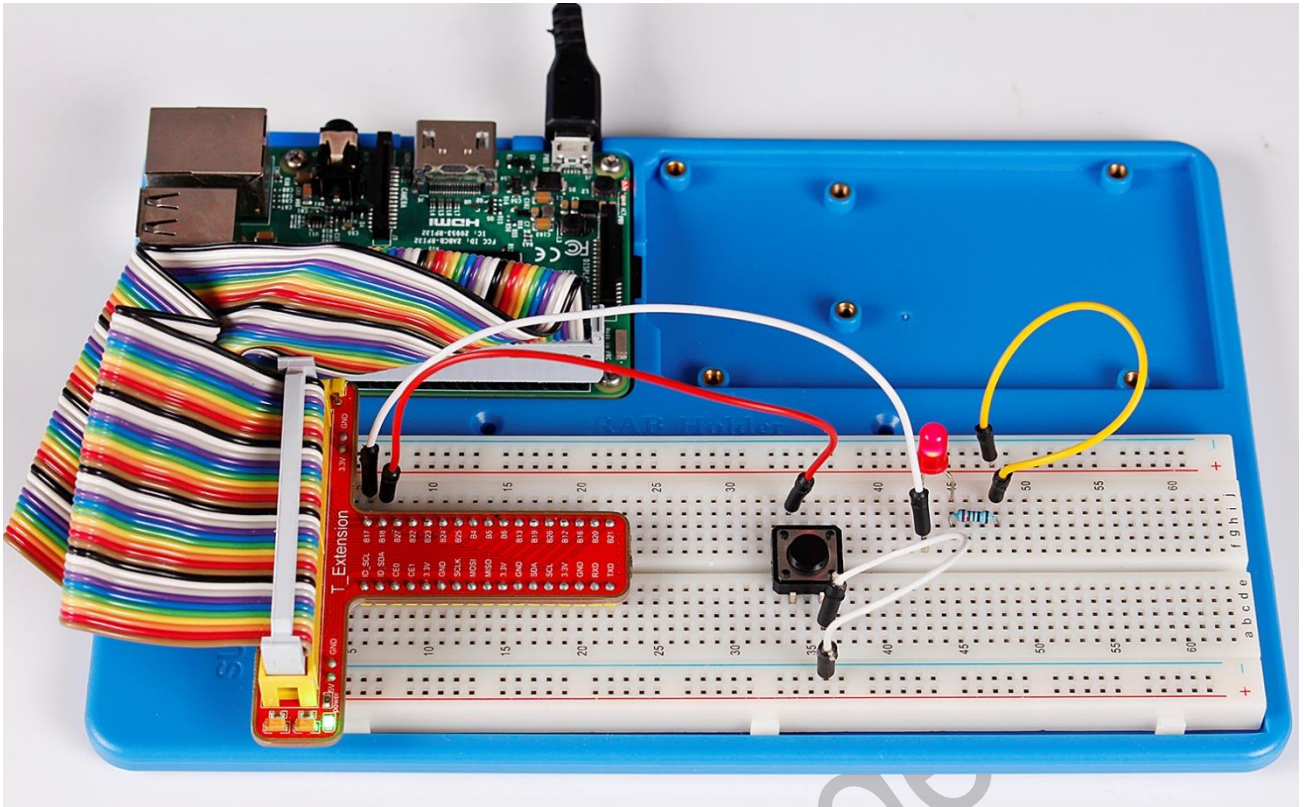
**Step 4:** Check the code

```
nano 02_buttonControlled.py
```

### Code Explanation

```
LedPin = 17 # Set #17 as LED pin
BtnPin = 18 # Set #18 as button pin
# Set up a falling detect on BtnPin, and callback function to swLED
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLED)
# Define a callback function for button callback, execute the function after the
callback of the interrupt.
def swLed(ev=None):
    global Led_status
    # Switch Led status (on-->off; off-->on)
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
    if Led_status:
        print 'LED OFF...'
    else:
        print '...LED ON'
```

Now, press the button, and the LED will light up; press the button again, and the LED will go out. At the same time, the state of the LED will be printed on the screen.



## Summary

Through this experiment, you have learnt how to control the GPIOs of the Raspberry Pi by programming.

# Lesson 3 Flowing LED Lights

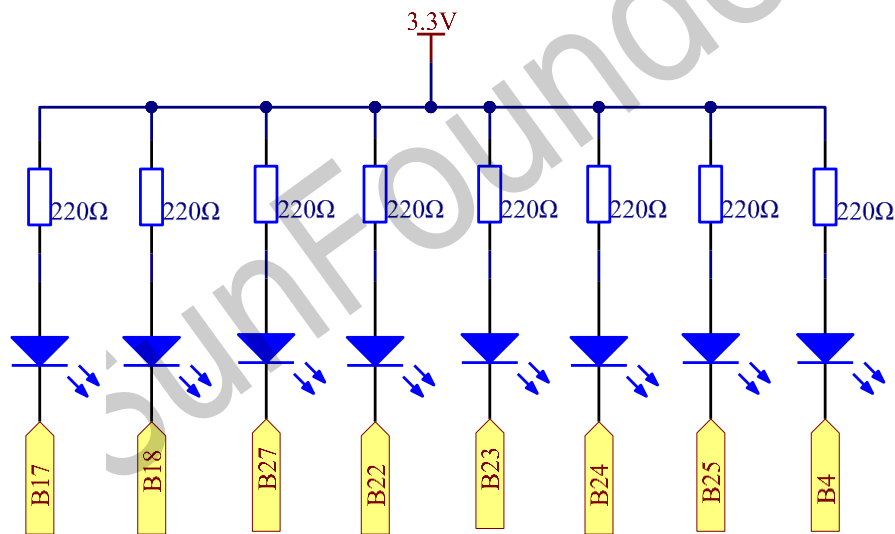
## Introduction

In this lesson, we will learn how to make eight LEDs blink in various effects as you want based on Raspberry Pi.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 8 \* LED
- 8 \* Resistor (220Ω)
- Jumper wires
- 1 \* T-Extension Board
- 1 \* 40-Pin Cable

## Principle

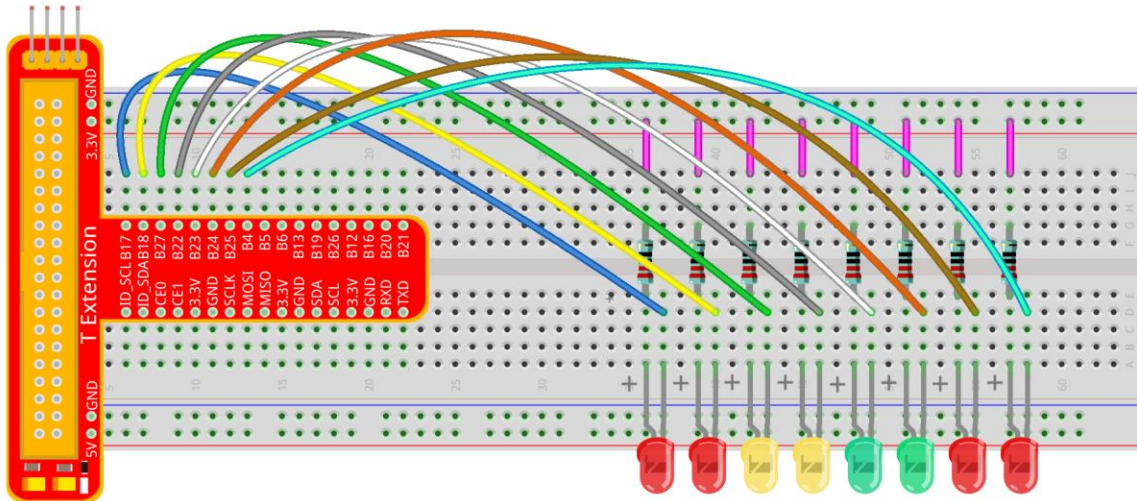


**Principle:** Judging from the schematic diagram, we can know that a LED and a current-limiting resistor have been connected to B17, B18, B27, B22, B23, B24, B25, and B4 respectively. The current-limiting resistor has been connected to the 3.3V power supply on other side. Therefore, if we want to light up one LED, we only need to set the GPIO of the LED as low level. So in this experiment, set B17, B18, B27, B22, B23, B24, B25, and B4 to low level in turn by programming, and then LED0-LED7 will light up in turn. You can make eight LEDs blink in different effects by controlling their delay time and the order of lighting up.

## Experimental Procedures

**Step 1:** Build the circuit





**For C language users:**

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Note:** Use the cd command to switch to the code path of this experiment.

**Step 3:** Compile the Code

```
gcc 03_8Led.c -o 03_8Led -lwiringPi
```

or

```
make 03_8Led
```

**Note:** gcc is a linux command which can realize compiling and generating the C language program file **03\_8Led.c** to the executable file **03\_8Led**.

make is a linux command which can compiling and generating the executable file according to the rule inside the makefile.

**Step 4:** Run the executable file above

```
sudo ./03_8Led
```

**Note:** Here the Raspberry Pi will run the executable file 03\_8Led compiled previously.

```
pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ make 03_8Led
[CC] 03_8Led.c
[link]
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./03_8Led

=====
      8 LEDs
-----
LED0 connect to GPIO0
LED1 connect to GPIO1
LED2 connect to GPIO2
LED3 connect to GPIO3
LED4 connect to GPIO4
LED5 connect to GPIO5
LED6 connect to GPIO6
LED7 connect to GPIO7

      Flow LED effect
                               SunFounder
=====

From left to right.
From right to left.
From left to right.
From right to left.
From left to right.
From right to left.
```



## Code Explanation

```
void Led_on(int channel): This is a subfunction with a formal parameter int channel for
importing the numbers of the controlled pins. Its function body is digitalWrite(channel,
LOW); Set the I/O port of channel as low level(0V), the LED on this port lights up. void
led_off(int channel) is to turn off the LED. Setting function simplifies the input for
the repeated content.

for(i=0;i<8;i++){          //make 8 pins' mode is output
    pinMode(i, OUTPUT);
}

//The cathodes of the 8 LEDs connect to B17, B18, B27, B22, B23, B24, B25, and B4 of
the T-shape extension board respectively, corresponding to 0,1,2,3,4,5,6,7. It is to set
the 8 LEDs as output separately. Use for loop to make it more concise and efficient.

for(i=0;i<8;i++){ //make LED on from left to right
    Led_on(i);    // turn the LED i on
    delay(100);   // keep the LED i lighting for 100ms.
    Led_off(i);   // Turn the LED i off
}

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

for(i=7;i>=0;i--){ //make LED off from right to left
    led_on(i); // turn the LED i on
    delay(100); // keep the LED i lighting for 100ms
    led_off(i); //turn the LED i off
}

// In this for loop, light up and turn off the LED in GPIO7 to GPIO0 successively,
making a flowing LED light from left to right.
```

### For Python users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 03_8Led.py
```

## Code Explanation

```
LedPins = [17, 18, 27, 22, 23, 24, 25, 4] # The cathodes of the 8 LEDs connect to B17,
B18, B27, 22, 23, 24, 25, 4 of the T-shape extension board. In BCM, these pins are
corresponding to 17, 18, 27, 22, 23, 24, 25, and 4.

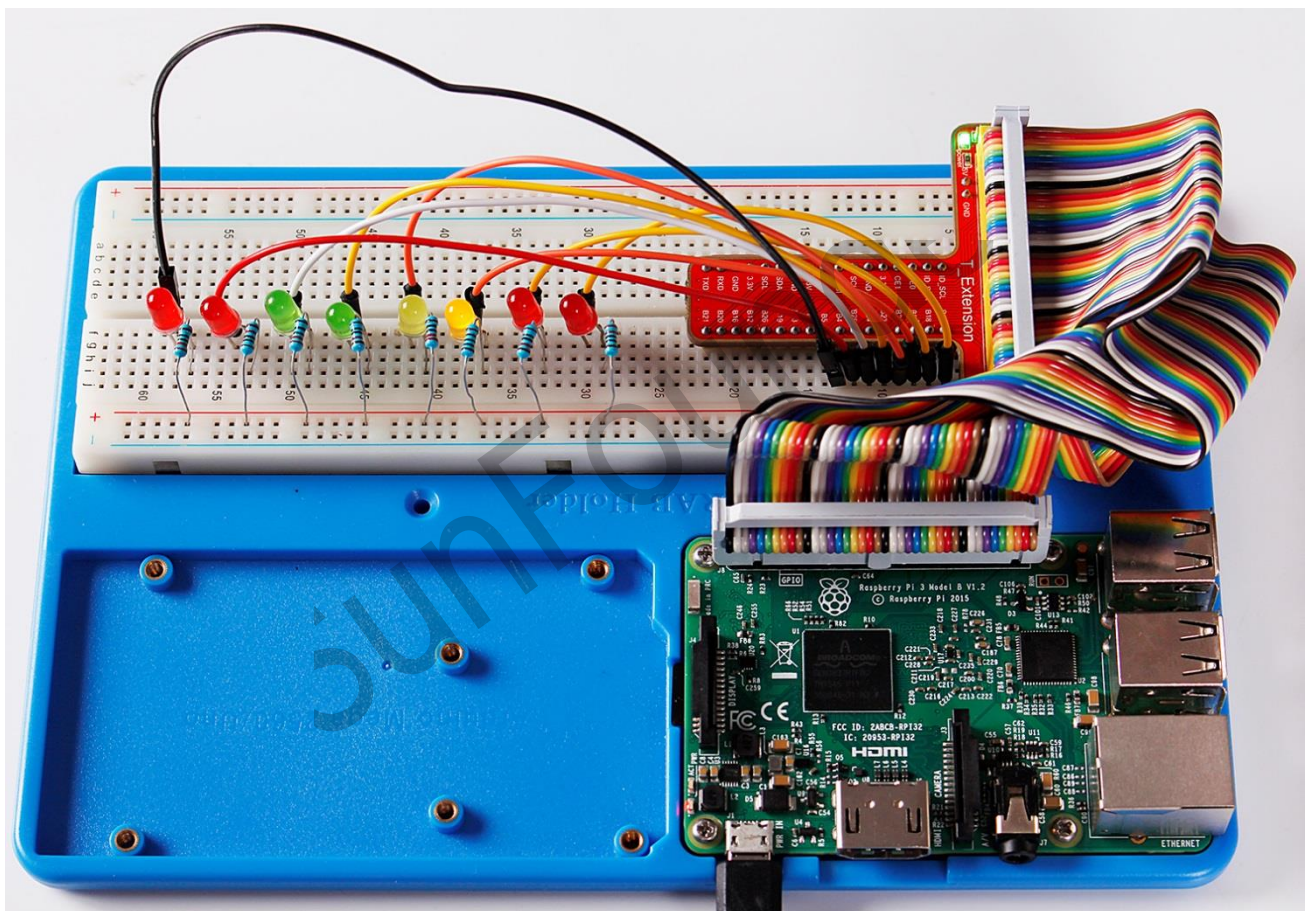
leds = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- '] # the array to print out the status of
the 8 LEDs
```

```

for pin in LedPins # Assign the element in pins list to pin variable one by one. In
GPIO.setup (pin, GPIO.OUT), set the pins in list as output one by one.
    GPIO.output(pin, GPIO.LOW) # Set each element in the pins list as low level to
light up the LEDs
    leds[LedPins.index(pin)] = 0 # Show which LED is on
    time.sleep(0.1) # wait for 0.1s
    GPIO.output(pin, GPIO.HIGH)) # After delaying, set it as high level to light up or
turn off the LED.
    leds[LedPins.index(pin)] = '-' # Show the led is off

```

You will see the eight LEDs lighten up one by one, and then dim in turn.



## Further Exploration

You can write the blinking effects of LEDs in an array. If you want to use one of these effects, you can call it in the *main()* function directly.

# Lesson 4 Breathing LED

## Introduction

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

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* LED
- 1 \* Resistor (220Ω)
- Jumper wires
- 1 \* T-Extension Board
- 1 \* 40-Pin Cable

## 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 (3.3 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 change, or modulate, that 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 3.3v 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:

$$D = \frac{T}{P} \times 100\%$$

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.



### Step 3: Compile the Code

```
make 04_breathLed
```

### Step 4: Run the executable file above

```
sudo ./04_breathLed
```

### Code Explanation

```
pinMode(LedPin, PWM_OUTPUT); // Set the I/O as pwn output

for(i=0;i<1024;i++){ // i,as the value of pwm, increases progressively during 0-1024.
    pwmWrite(LedPin, i); // Write i into the LEDPin
    delay(2); // wait for 2ms, interval time between the changes indicates the speed of
breathing.
} // the value of pwm add 1 every 2ms, when the value of pwm increases, the luminance of
the LED increases.

for(i=1023;i>=0;i--){
    pwmWrite(LedPin, i);
    delay(2);
} // the value of pwm minus 1 every 2ms, when the value of pwm decreases, the luminance
of the LED decreases.
```

### For Python users:

#### Step 2: Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

#### Step 3: Run

```
sudo python 04_breathLed.py
```

### Code Explanation

```
GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW) # Set LedPin as OUTPUT, initialize the
pin as low level.
pLED = GPIO.PWM(LedPin, 1000) # use PWM in the RPi.GPIO library. Set LedPin as analog
PWM output, the frequency as 1000Hz, assign these configurations to pLed.
pLed.start(0) # Start pLed with 0% pulse width
time.sleep(0.05)
while True:
    # Increase duty cycle from 0 to 100
    for dc in range(0, 101, step): # set dc from 0 to 100 in for loop. Set step to
cycle.
        # Change duty cycle to dc
```



```

pLed.ChangeDutyCycle(dc) # ChangeDutyCycle() function in pLED output
pulse width 0~100% according to the variable dc.

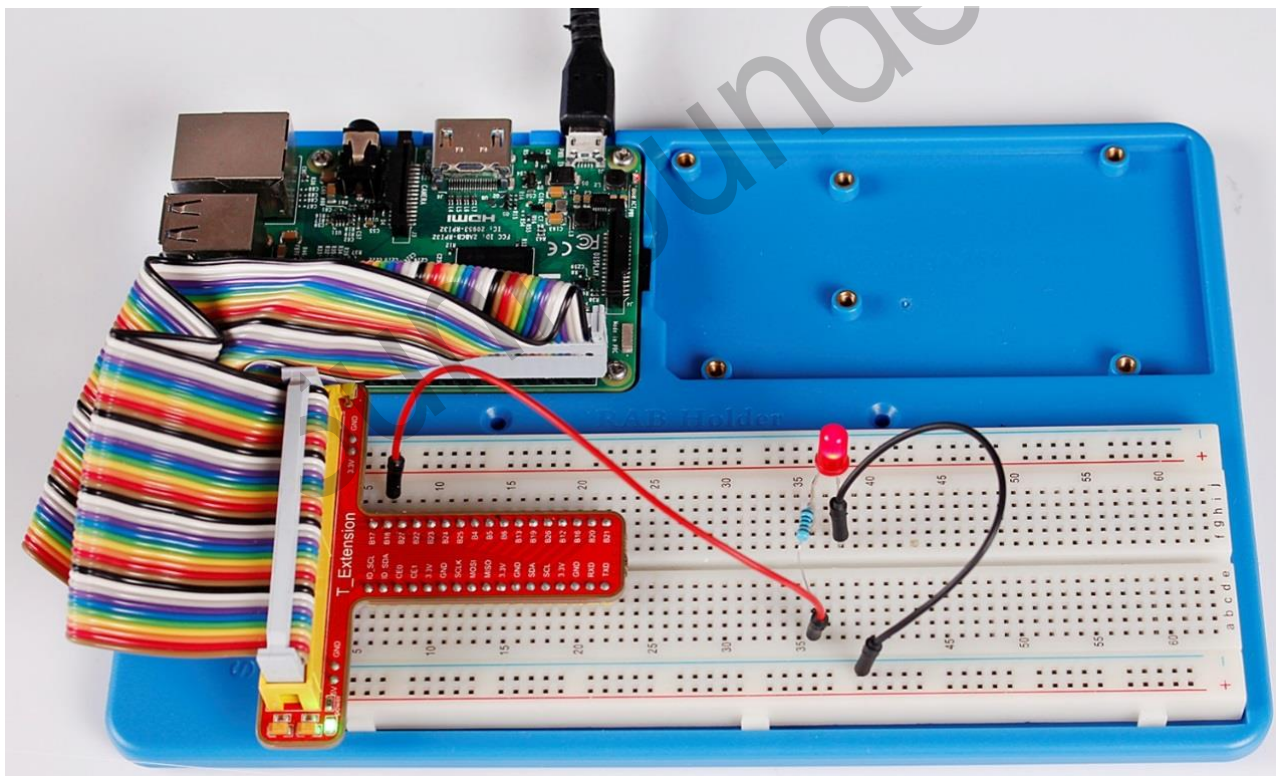
print " ++ Duty cycle: %s"%dc # print information
time.sleep(delay) # it will delay after changing the pulse width for
each time, this parameter can be modified to change the LED's lighting and dimming
speed.

time.sleep(1)
# decrease duty cycle from 100 to 0
for dc in range(100, -1, -step): # the luminance of the LED decreases with each
cycle.

    # Change duty cycle to dc
    pLED.ChangeDutyCycle(dc) # same as the last for loop
    print " -- Duty cycle: %s"%dc
    time.sleep(delay)

```

Now you will see the gradual change of the LED luminance, between bright and dim.



## Summary

Through this experiment, you should have mastered the principle of PWM and how to program Raspberry Pi with PWM. You can try to apply this technology to DC motor speed regulation later.

# Lesson 5 RGB LED

## Introduction

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

## Components

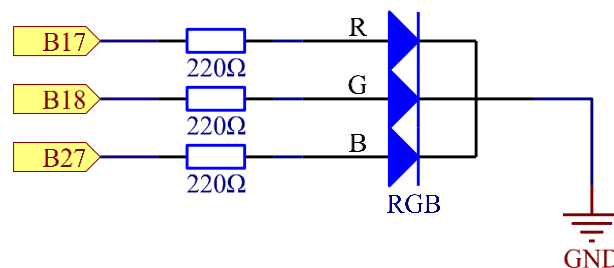
- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* RGB LED
- 3 \* Resistor (220Ω)
- Several jumper wires

## Principle

In this experiment, we will use a RGB. For details of RGB, please refer to the introduction of RGB LED in **Components Introduction**.



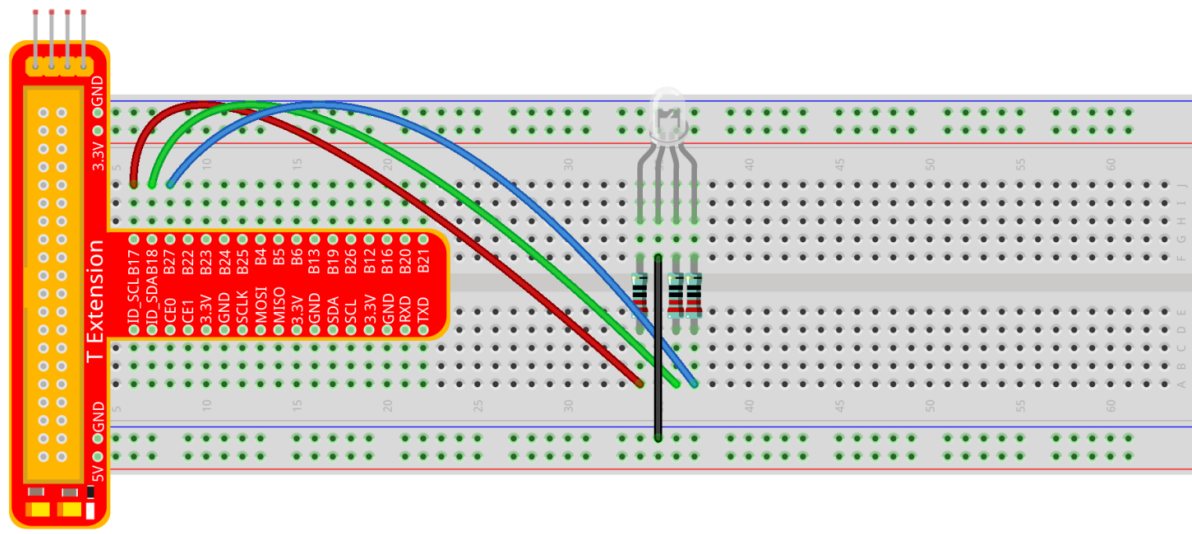
The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with 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.



## Experimental Procedures



## Step 1: Build the circuit



## For C language users:

### Step 2: Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

### Step 3: Compile the Code

```
make 05_rgb
```

### Step 4: Run the executable file above

```
sudo ./05_rgb
```

## Code Explanation

```
#include <softPwm.h> // library used for realizing the pwm function of the software.
void ledInit(void){ // define function used for initializing I/O port to output for
pwm.
    // LedPinX refers to one pin. 0 is the minimum value and 100 is the maximum (as a
percentage). The function is to use software to create a PWM pin, set its value between
0-100%.
    softPwmCreate(LedPinRed, 0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
void ledColorSet(uchar r_val, uchar g_val, uchar b_val){ // 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 three formal parameters'
different values corresponds to various colors. You can modify the 3 formal parameters
randomly to verify.
    softPwmWrite(LedPinRed, r_val);
    softPwmWrite(LedPinGreen, g_val);
```

```

    softPwmWrite(LedPinBlue, b_val);
}

ledColorSet(0xff,0x00,0x00);    // red calls 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 users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 05_rgb.py
```

### Code Explanation

```

# Set up a color table in Hexadecimal
COLOR = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]

# Set pins' channels with dictionary
pins = {'Red':17, 'Green':18, 'Blue':27}

p_R = GPIO.PWM(pins['Red'], 2000) # the same as the last lesson, here we configure
the channels and frequencies of the 3 PWM.
p_G = GPIO.PWM(pins['Green'], 2000)
p_B = GPIO.PWM(pins['Blue'], 2000)

p_R.start(0) # the same as the last lesson, the PWM of the 3 LEDs begin with 0.
p_G.start(0)
p_B.start(0)

# Define a MAP function for mapping values. Like from 0~255 to 0~100
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def setColor(color): # configures the three LEDs' luminance with the inputted
color value .

    R_val = (color & 0xFF0000) >> 16 # these three lines are used for analyzing the
col variables

    G_val = (color & 0x00FF00) >> 8 # assign the first two values of the
hexadecimal to R, the middle two assigned to G

    B_val = (color & 0x0000FF) >> 0 # assign the last two values to B, please
refer to the shift operation of the hexadecimal for details.

```

```

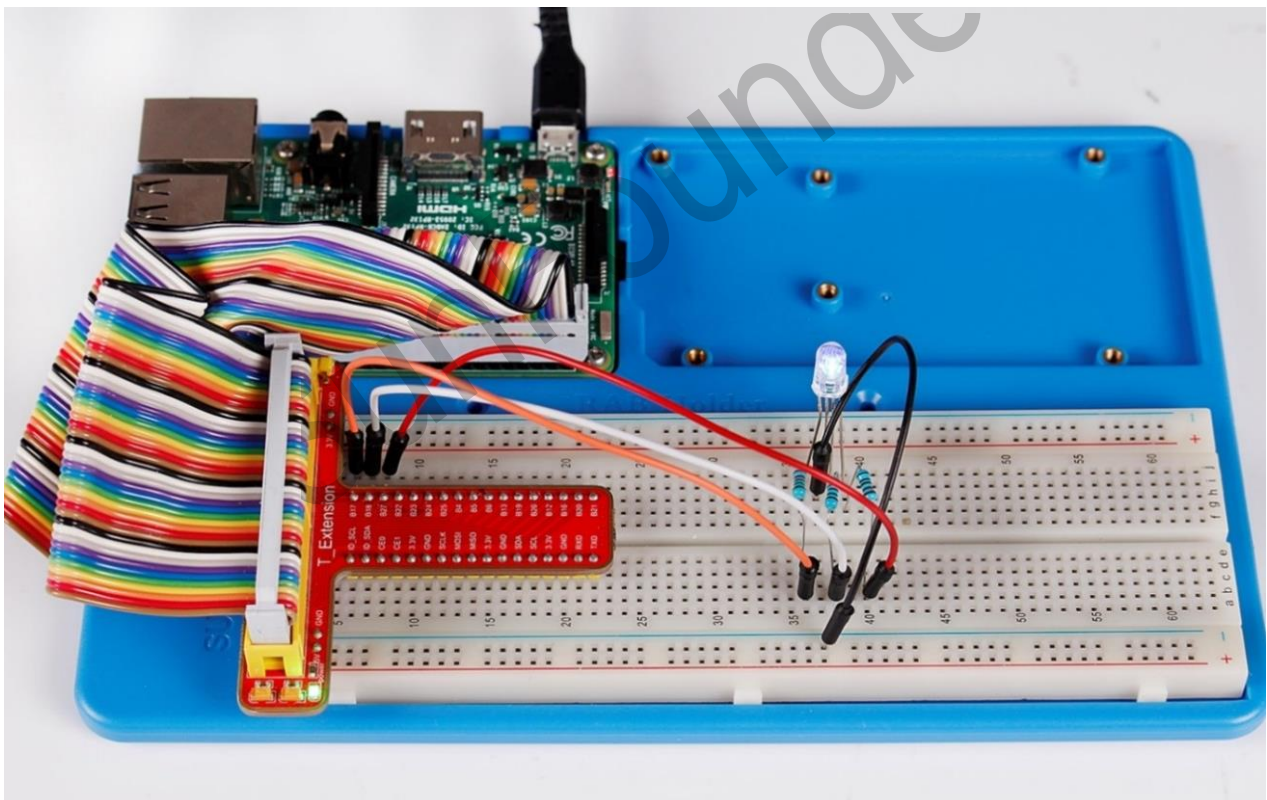
R_val = MAP(R_val, 0, 255, 0, 100)    # use map function to map the R,G,B
value among 0~255 into PWM value among 0~100.
G_val = MAP(G_val, 0, 255, 0, 100)
B_val = MAP(B_val, 0, 255, 0, 100)

p_R.ChangeDutyCycle(R_val)    # Assign the mapped duty cycle value to the
corresponding PWM channel to change the luminance.
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)

for color in COLOR:           # Assign every item in the COLOR list to the color
respectively and change the color of the RGB LED via the setColor() function.
    setColor(color) # change the color of the RGB LED
    time.sleep(0.5)    # set delay for 0.5s after each color changing. Modify this
parameter will changed the LED's color changing rate.

```

Here you should see the RGB LED flash different colors in turn.



### Further Exploration

You can modify the parameters of the function `ledColorSet()` by yourself, and then run the code to see the color changes of the RGB LED.

# Lesson 6 Buzzer

## Introduction

In this lesson, we will learn how to drive an active buzzer to beep with a PNP transistor.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* Buzzer (Active)
- 1 \* PNP transistor (8550)
- 1 \* Resistor (1K $\Omega$ )
- Jumper wires

## Principle

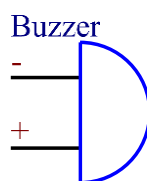
### Buzzer

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



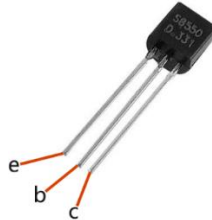
The difference between an active buzzer and a passive buzzer is: An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not beep if DC signals are used; instead, you need to use square waves whose frequency 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.

The following is the electrical symbol of a buzzer. It has two pins with positive and negative poles. With a + in the surface represents the anode and the other is the cathode.



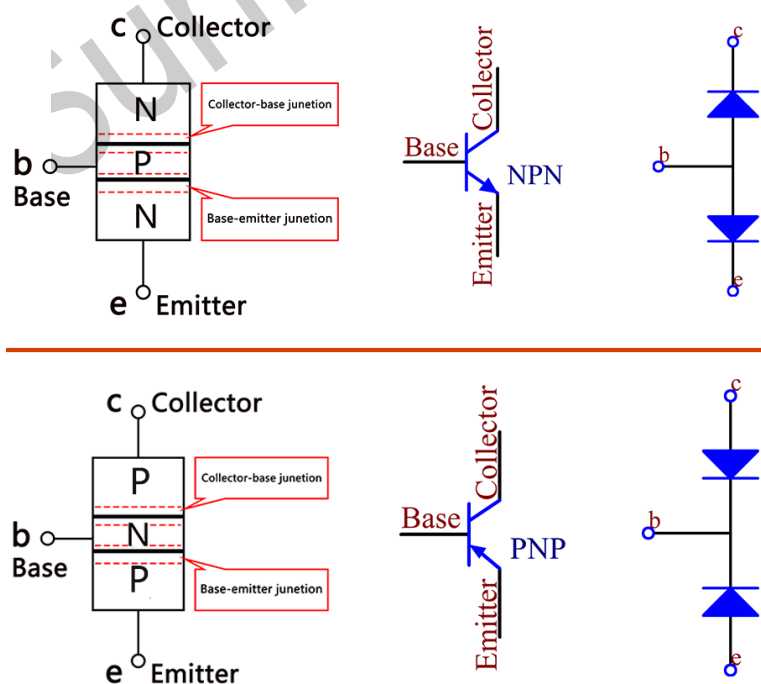
You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

## Transistor



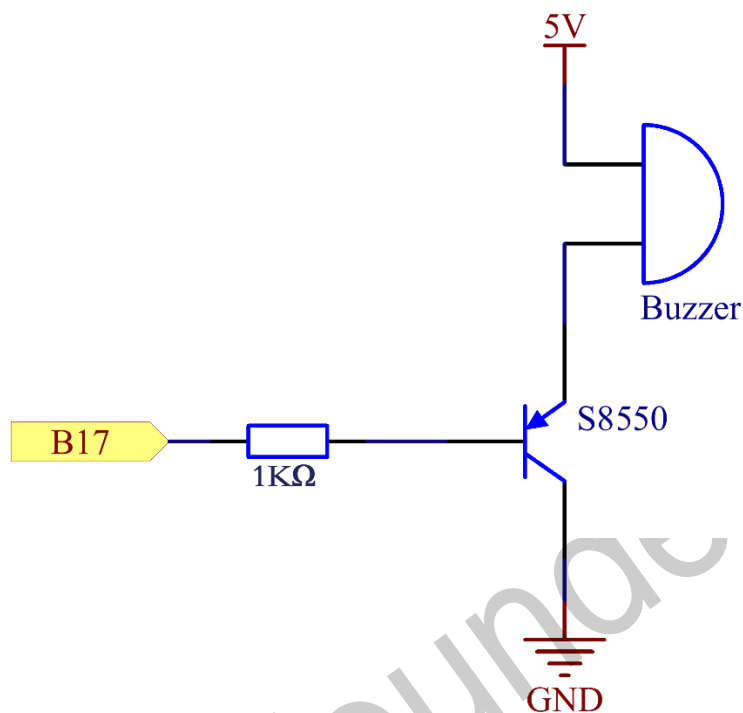
The transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used as a non-contact switch. A transistor is a three-layer structure composed of P-type or N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are all N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when 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 arrow in the circuit symbol indicates the direction of emitter junction. Transistors can be divided into two kinds: the NPN and PNP one. 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.

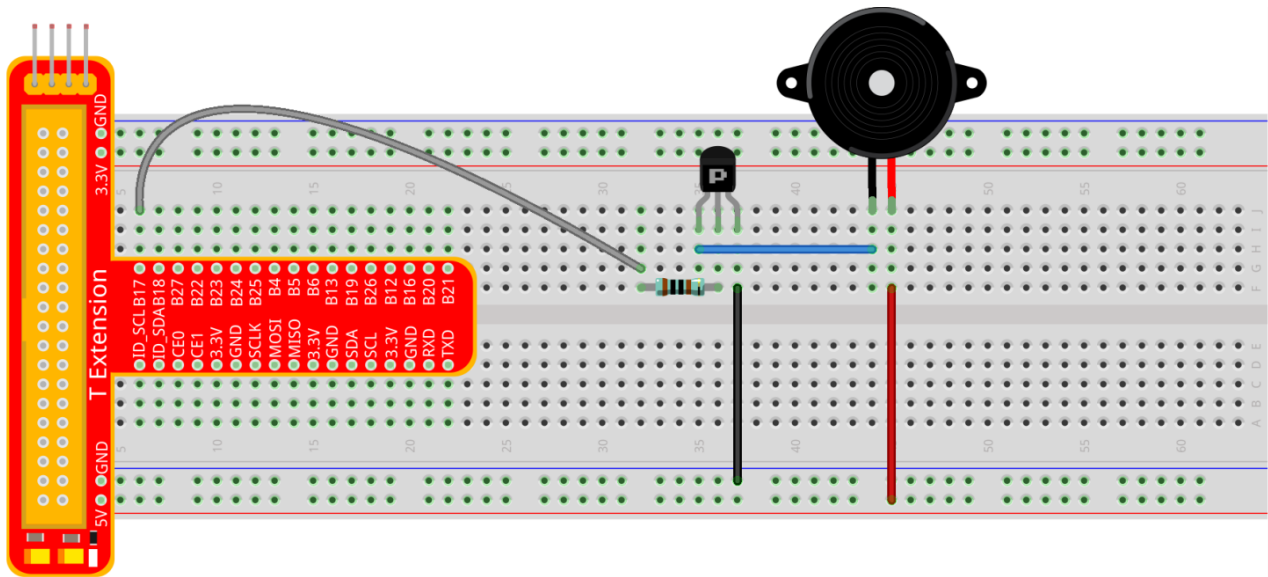
The schematic diagram:



**Principle:** In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the B17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and the buzzer will make sounds. But when high level is supplied to the IO of Raspberry Pi, the transistor will be cut off and the buzzer will not make sounds.

### Experimental Procedures

**Step 1:** Build the circuit (Pay attention to poles of the buzzer: The one with + label is the positive pole and the other is the negative.)



### For C language users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile the Code

```
make 06_beep
```

**Step 4:** Run the executable file above.

```
sudo ./06_beep
```

### Code Explanation

```
digitalWrite(BeepPin, LOW); // We use an active buzzer in this experiment, so it will
make sound automatically when connecting to the direct current. This sketch is to set
the I/O port as low level (0V), thus to manage the transistor and make the buzzer beep.
digitalWrite(BeepPin, HIGH); // To set the I/O port as high level(5V), thus the
transistor is not energized and the buzzer doesn't beep.
```

### For Python users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

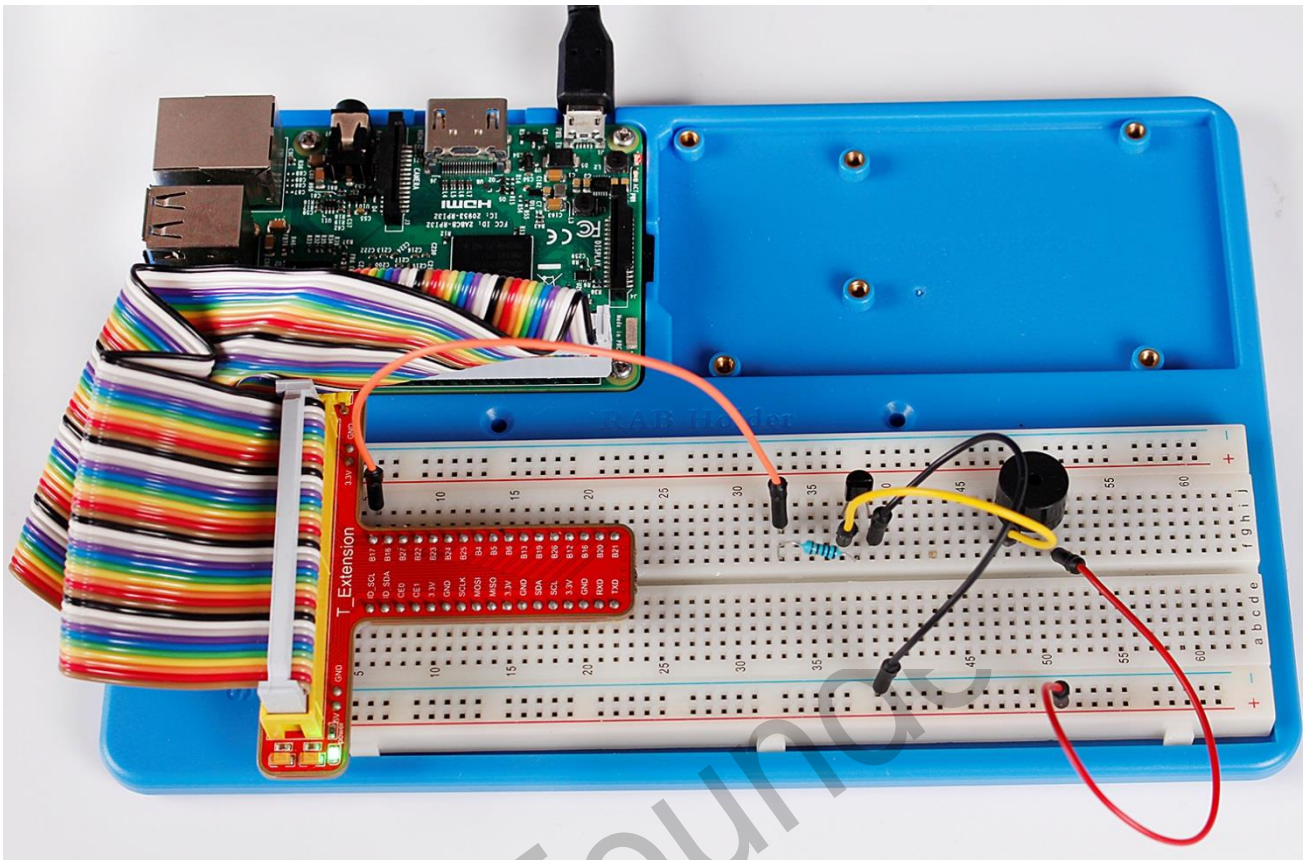
```
sudo python 06_beep.py
```

### Code Explanation

```
GPIO.output(BeepPin, GPIO.LOW)# Set the buzzer pin as low level.
time.sleep(0.1)# Wait for 0.1 second. Change the switching frequency by changing this
parameter. Note: Not the sound frequency. Active Buzzer cannot change sound frequency.
GPIO.output(BeepPin, GPIO.HIGH) # close the buzzer
```



```
time.sleep(0.1)
```

 Now, you should hear the buzzer make sounds.

### Further Exploration

If you have a passive buzzer in hand, you can replace the active buzzer with it. Now you can make a buzzer sound like “do re mi fa so la si do” with just some basic knowledge of programming. Give a try!

# Lesson 7 Relay

## Introduction

As we know relay is a device which is used to provide connection between two or more points or device in response to the input signal applied. In another words relay provide isolation between the controller and the device as we know devices may work on AC as well as on DC. However, they receive signals from microcontroller which works on DC hence we require 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.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* Relay
- 1 \* LED
- 1 \* Resistor ( $220\Omega$ )
- 1 \* Resistor ( $1K\Omega$ )
- 1 \* NPN Transistor
- 1 \* Diode (Rectifier)
- Several jumper wires

## Principle

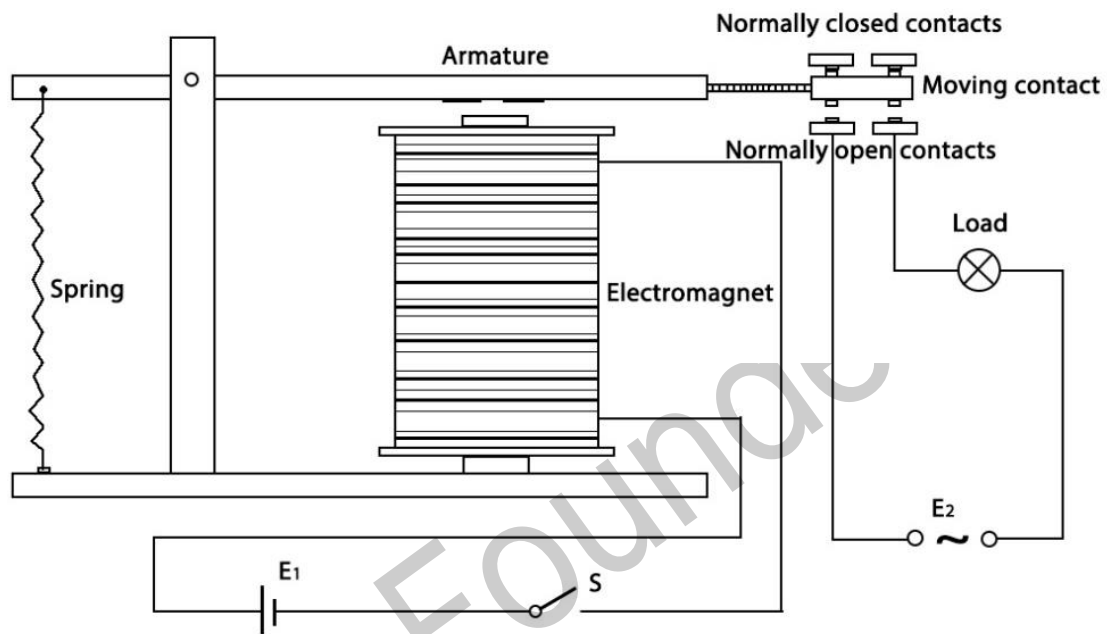
### Relay

There are 5 parts in every relay:

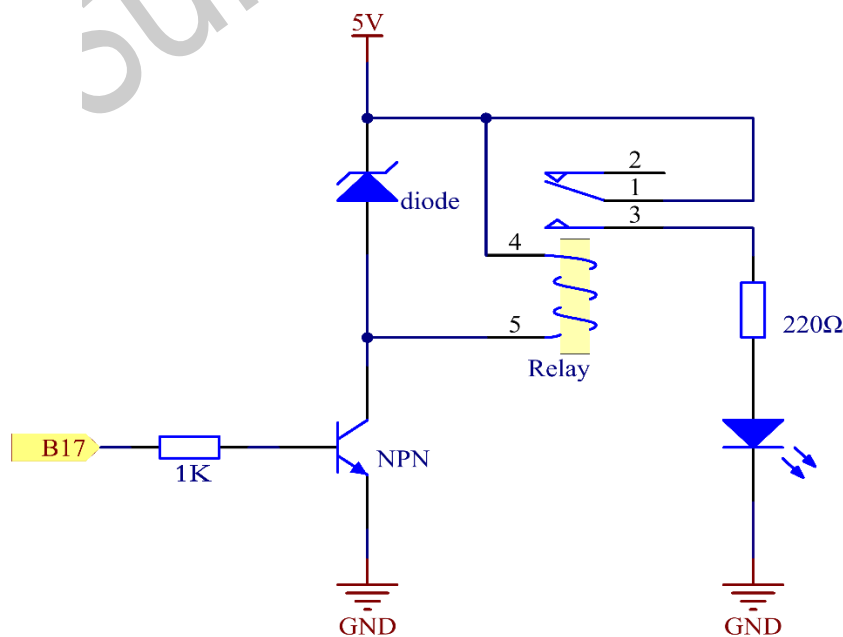
1. **Electromagnet** – It consists of an iron core wound by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.
2. **Armature** – The movable magnetic strip is known as armature. When current flows through them, the coil is energized thus producing 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.

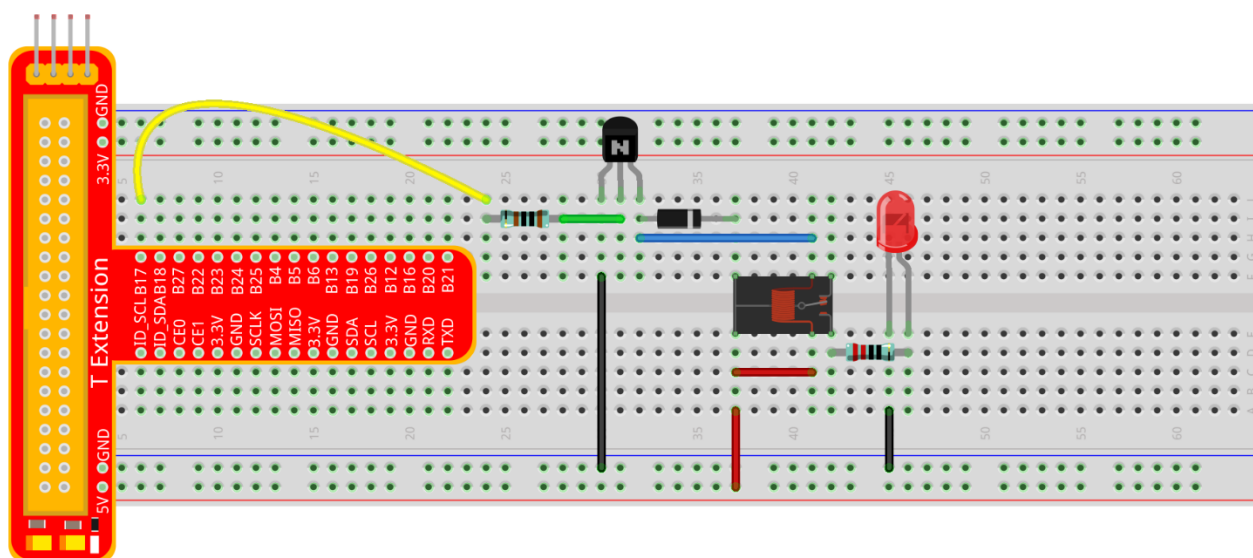


Schematic diagram:



## Experimental Procedures

### Step 1: Build the circuit



fritzing

**For C language users:**

## Step 2: Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

### Step 3: Compile the Code

```
make 07 relay
```

**Step 4:** Run the executable file above

```
sudo ./07_relay
```

## Code Explanation

```
digitalWrite(relayPin, LOW); // set the I/O port as low level (0V) to energize the
transistor. The coil of the relay is powered and generate electromagnetic force, and the
relay closes.
```

```
digitalWrite(relayPin, HIGH); // Set the I/O port as high level (5V), thus the
transistor is not energized and the coil is not powered. There is no electromagnetic
force, so the relay opens.
```

### For Python users:

### Step 2: Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

### Step 3: Run

```
sudo python 07_relay.py
```

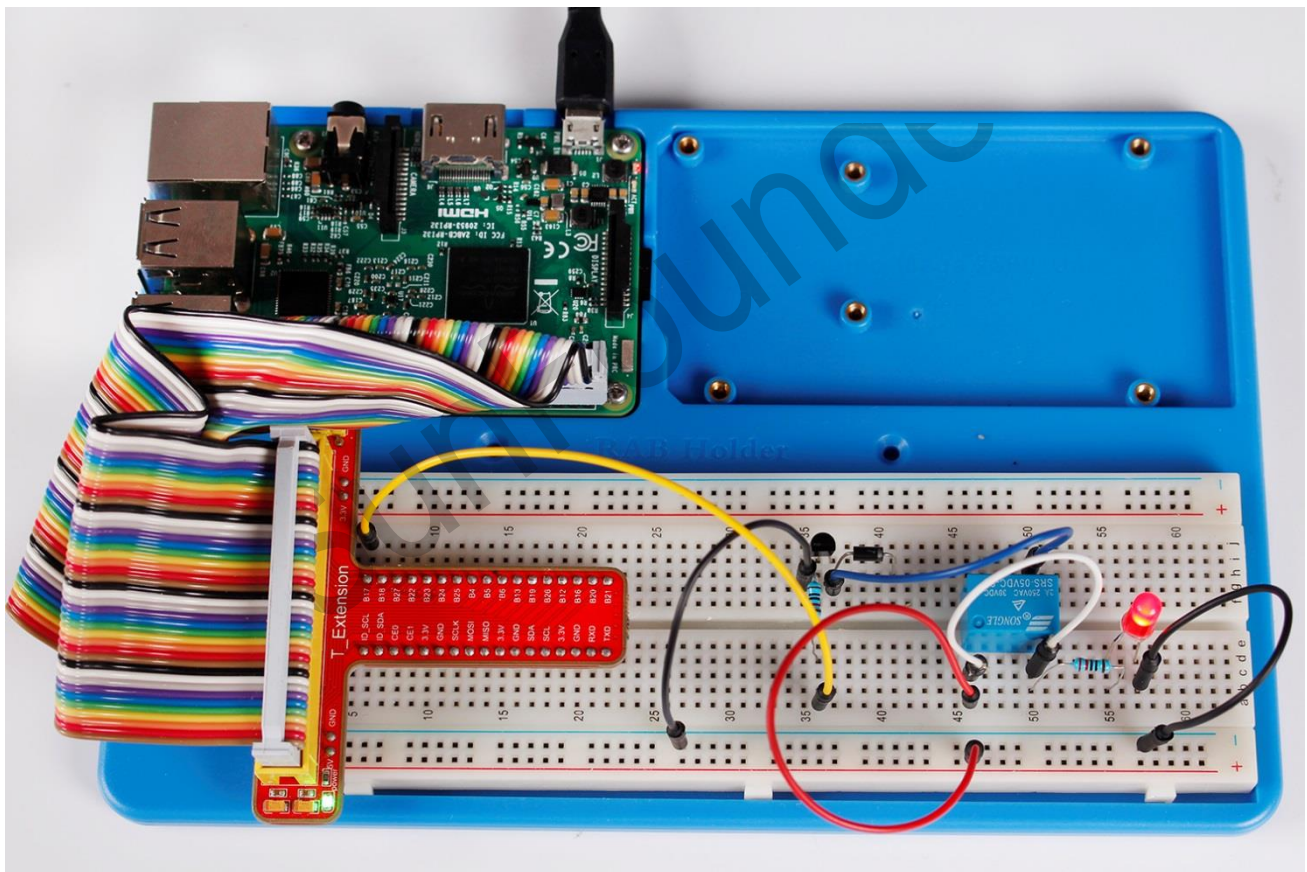
### Code Explanation

```

GPIO.output(relayPin, GPIO.LOW)  # Set the pins of transistor as low level to actuate
the relay.
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.
GPIO.output(relayPin, GPIO.HIGH)  # Set the pins of the transistor as low level to
let the relay open.
time.sleep(1)

```

Now, connect a device of high voltage, and the relay will close and the LED will light up; connect one of low voltage, and it will open and the LED will go out. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.





## Lesson 8 4N35

### Introduction

The 4N35 is an optocoupler for general purpose application. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor. When the input signal is applied to the LED in the input terminal, the LED lights up. After receiving the light signal, the light receiver then converts it into electrical signal and outputs the signal directly or after amplifying it into a standard digital level. Thus, the transition and transmission of electricity-light-electricity is completed. Since light is the media of the transmission, meaning the input terminal and the output one are isolated electrically, this process is also be known as electrical isolation.

### Components

- 1 \* 4N35
- 1 \* LED
- 1 \* 220 Ohm Resistor
- 1\* 1k Ohm Resistor
- Some jump wires

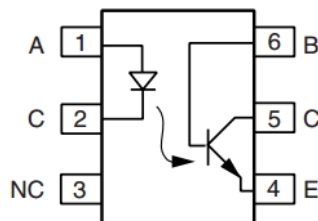
### Principle

#### 4N35



The 4N35 is an optocoupler for general purpose application. 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. In other words, it is used to prevent interference from external electrical signals. 4N35 can be used in AV conversion audio circuits. Broadly it is widely used in electrical insulation for a general optocoupler.

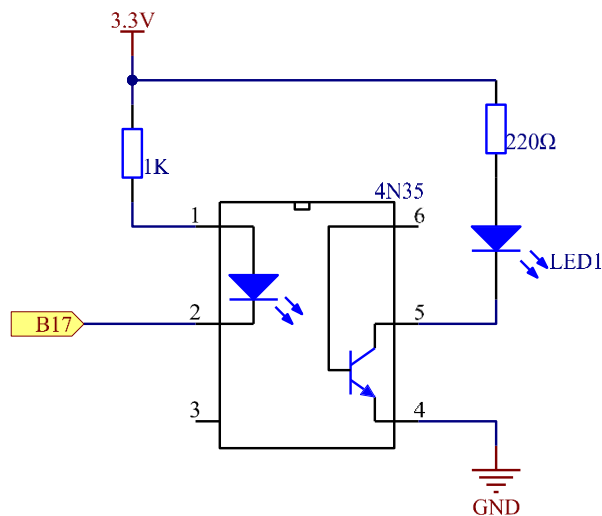


See the internal structure of 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays. This can be done to control the load connected to the phototransistor. Even when



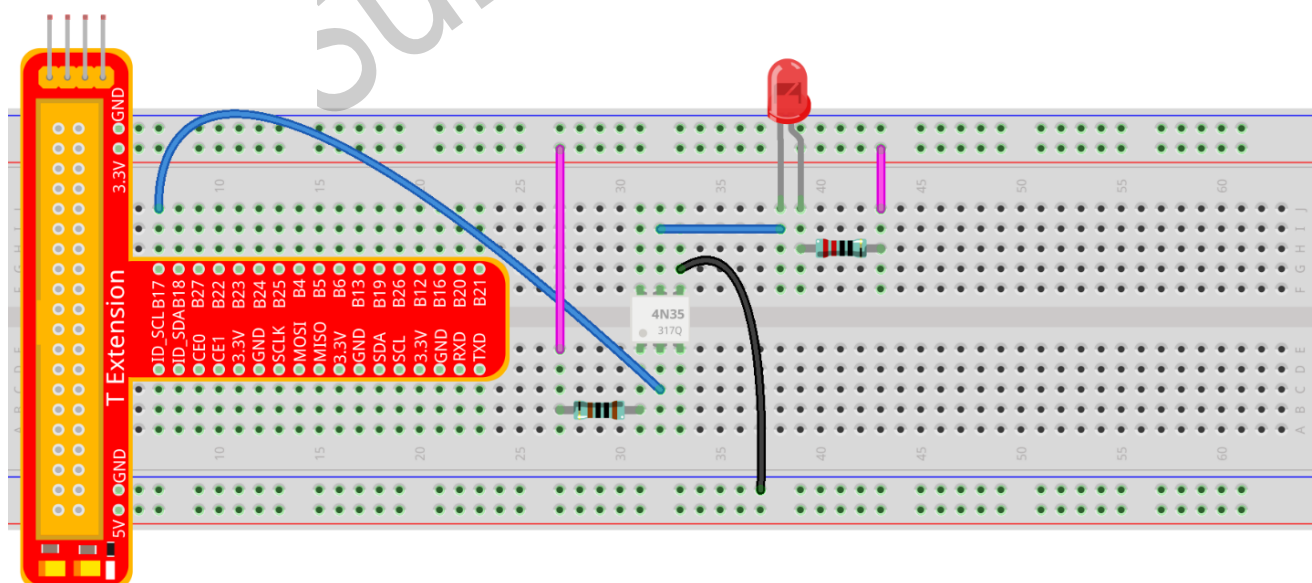
the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

The schematic diagram:



**Principle:** In this experiment, use an LED as the load connected to the NPN phototransistor. Connect pin 2 of 4N35 to pin B17, pin 1 connects a 1K current-limiting resistor and then a 3.3V. Connect pin 4 to GND, and pin 5 to the cathode of the LED. Then hook the anode of the LED to 3.3V after connecting with a 220 Ohm resistor. When in program, a LOW level is given to pin B17, the infrared LED will emit infrared rays. Then the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus turning on the LED. Also you can control the LED by circuits only – connect pin 2 to ground and it will brighten.

**Step 1:** Build the circuit



fritzing

## For C language users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile the Code

```
make 08_4N35
```

**Step 4:** Run the executable file above.

```
sudo ./08_4N35
```

## Code Explanation

```
digitalWrite(_4N35Pin, LOW); // set the I/O port as low level (0V), thus the optocoupler
is energized, and the pin connected to LED conducts to the 0V. Then the LED lights up.
delay(500); // optocoupler is a kind of electronic device and there is no limitation on
its on-off frequency.
digitalWrite(_4N35Pin, HIGH); // set I/O port as high level (3.3V), thus the optocoupler
is not energized ,and the pin connected to LED cannot conduct to the 0V. Then the LED
goes out.
```

## For Python users:

**Step 2:** Open the code file

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

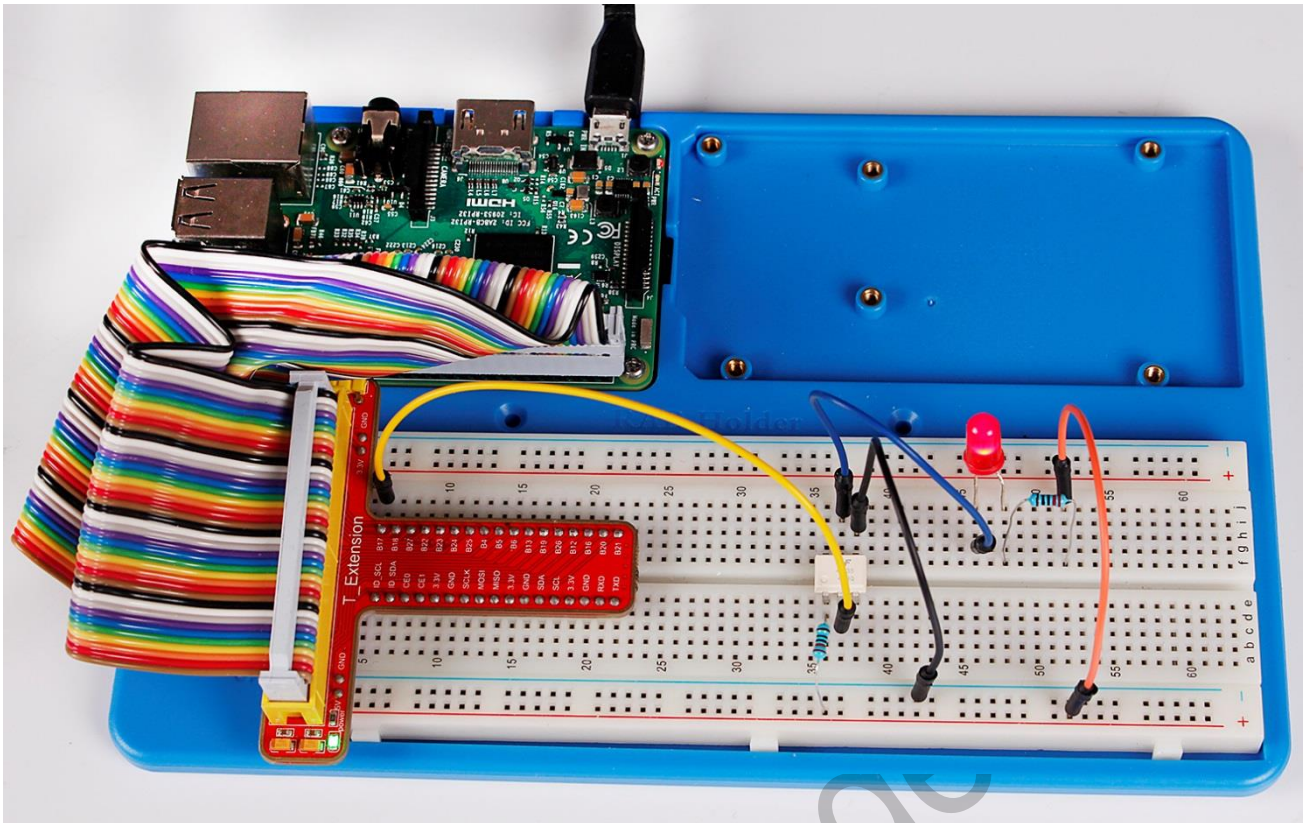
```
sudo python 08_4N35.py
```

## Code Explanation

```
GPIO.output(Pin_4N35, GPIO.LOW) # set the pins of optocoupler as low level, thus the
optocoupler is energized, and the pin connected to LED conducts to the 0V. Then the LED
lights up.

time.sleep(0.5) #wait for 0.5 second. The on-off frequency of the
optocoupler can be changed by modifying this parameter.
GPIO.output(Pin_4N35, GPIO.HIGH) # set the pins of optocoupler as high level, thus the
optocoupler is disconnected, and the pin connected to LED break the connection to the
0V. Then the LED goes out.
time.sleep(0.5)
```

You will see the LED blinks.



## Exploration

4N35 is an optocoupler that usually used for driving relay as well as motor circuits. As there is no direct connection between the input and output, even if a short circuit at the output end occurs, the control board will not be burnt. Have a try!

# Lesson 9 Ne555

## Introduction

If you ask anyone in the know to rank the most commonly and widely used IC, the famous 555 time base IC would certainly be at the top of the list. The 555 – a mixed circuit composed of analog and digital circuits – integrates analogue and logical functions into an independent IC, and hence tremendously expands the application range of analog integrated circuits. The 555 is widely used in various timers, pulse generators, and oscillators.

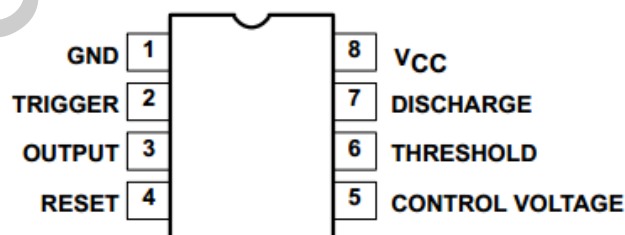
## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* NE555
- 2 \* 104 ceramic capacitor
- 1 \* Potentiometer (50K $\Omega$ )
- 1 \* Resistor (10K $\Omega$ )
- 1 \* USB cable
- Jumper wires

## Principle

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.

555 chip pins are introduced as follows:

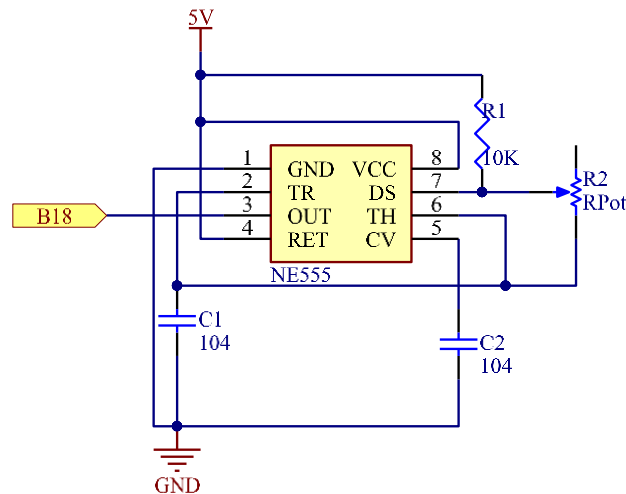


As shown in the picture, the 555 IC is dual in-line with the 8-pin package. Thus:

- Pin 1 (GND): the ground;
- Pin 2 (TRIGGER): the input of lower comparator;
- Pin 3 (OUTPUT): having two states of 0 and 1 decided by the input electrical level;
- Pin 4 (RESET): output low level when supplied a low one;
- Pin 5 (CONTROL VOLTAGE): changing the upper and lower level trigger values;
- Pin 6 (THRESHOLD): the input of upper comparator;

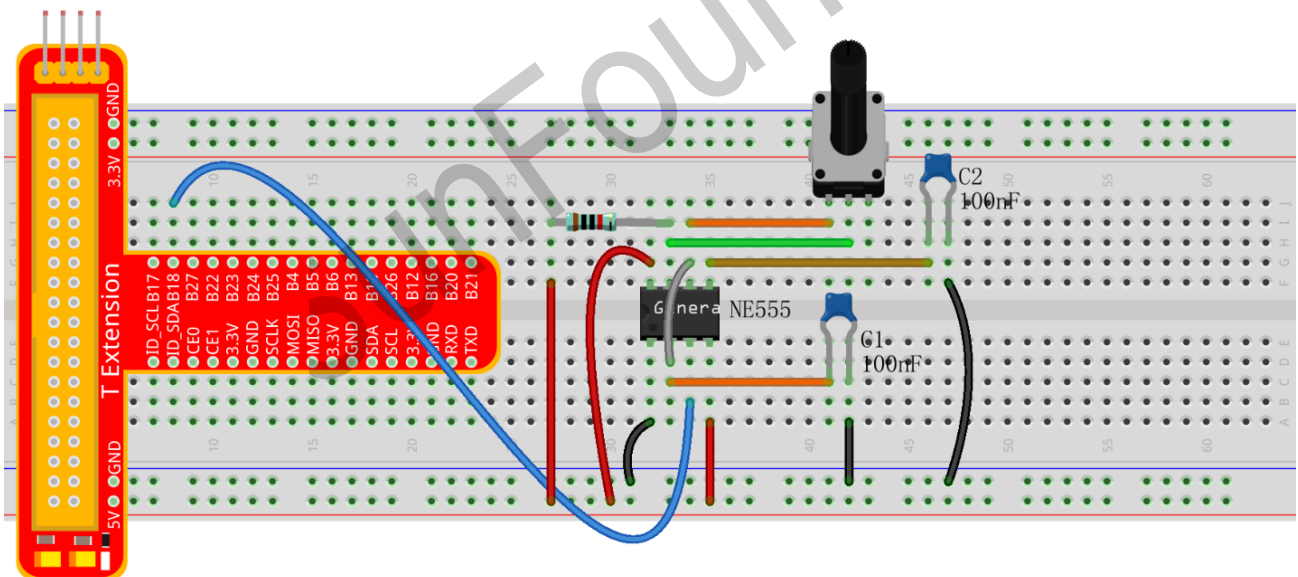
- Pin 7 (DISCHARGE): having two states of suspension and ground connection also decided by input, and the output of the internal discharge tube;
- Pin 8 (VCC): the power supply;

The schematic diagram



## Experimental Procedures

### Step 1: Build the circuit



### For C language users:

#### Step 2: Go to the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

#### Step 3: Compile

```
make 09_ne555
```

#### Step 4: Run the executable file above.

```
sudo ./09_ne555
```



## Code Explanation

```
static volatile int globalCounter = 0 ; // a static integer variable to store the pulse
count

void exInt0_ISR(void) { //GPIO0 interrupt service routine
    ++globalCounter;
}

wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR); // set an interrupt here and the
signal is falling edge for Pin 0. When the interrupt happens, execute the function
exInt0_ISR(), and the pulse count will add 1.

while(1){ // if no interrupt happens, the pulse count will stay and just print it.
    printf("Current pulse number is : %d\n", globalCounter);
}
```

### For Python users:

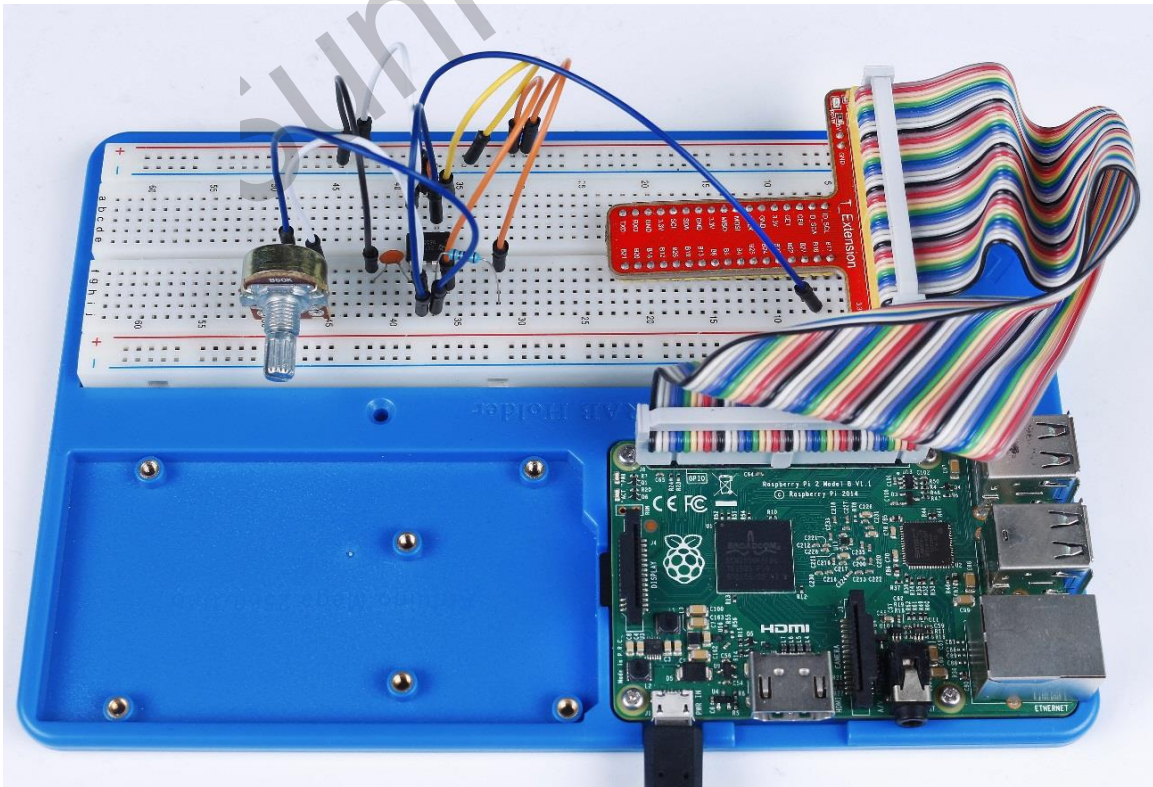
**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 09_ne555.py
```

Now you can see the number of square waves printed. Spin the potentiometer and the value will decrease or increase.





## Code Explanation

```
g_count = 0    # a global variable used to store the pulse count
def count(ev=None):    # Define a function to be run when an interrupt happens
    global g_count    # this function will change the value of the global variable g_count,
    # thus here we add global before it.
    g_count += 1

GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # set an interrupt here and
# the interrupt signal is a rising edge for Pin Sig. It will run the function count()
# accordingly
while True:    # wait for the interrupt
    print 'g_count = %d' % g_count    # print the information
    time.sleep(0.001)
```

SunFounder

# Lesson 10 Slide Switch

## Introduction

In this lesson, we will learn how to use a Slide Switch. Usually, the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard is to show its function.

## Components

- 1 \* SunFounder Uno board
- 1 \* Breadboard
- 1 \* Slide Switch
- 2 \* LED
- 2 \* Resistors (220Ω)
- 1 \* Resistors (10kΩ)
- 1 \* USB cable
- Jumper wires

## Principle

### Slide Switch

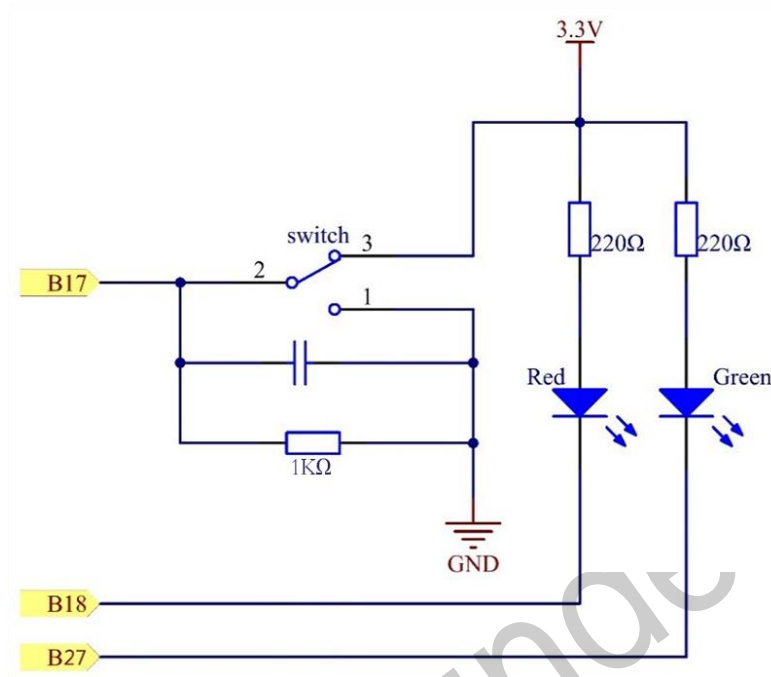
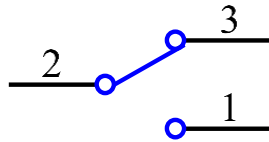


A slide switch, just as its name implies, is to slide the switch bar to **connect** or **break** the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The Slide Switch is commonly used in low-voltage circuit. It features flexibility and stability, and widely applies in electric instruments and electric toys.

How it works: Use the middle pin as the fixed one. When you pull the slide to the left, the left two pins are connected; to the right, the right two pins connected. Thus, it connects and disconnects circuits as a switch. See the figure below:



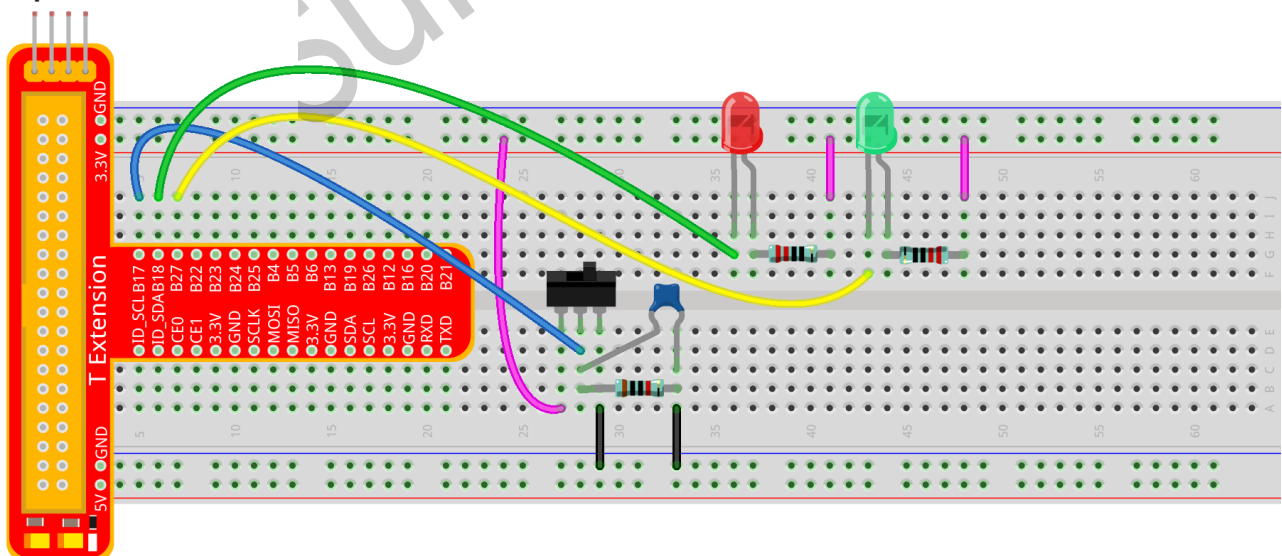
The circuit symbol of the slide switch is as shown below. 2 in the figure means the middle pin.



**Principle:** Connect the middle pin of the Slide Switch to B17, and two LEDs to pin B18 and B27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

## Experimental Procedures

**Step 1:** Build the circuit



**For C language users:**

**Step 2:** Go to the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

### Step 3: Compile

```
make 10_slideSwitch
```

### Step 4: Run the executable file above.

```
sudo ./10_slideSwitch
```

## Code Explanation

```
// 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
if(digitalRead(slidePin) == 1){
    digitalWrite(led1, LOW);
    digitalWrite(led2, HIGH);
    printf("LED1 on\n");
}

// 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
if(digitalRead(slidePin) == 0){
    digitalWrite(led2, LOW);
    digitalWrite(led1, HIGH);
    printf(".....LED2 on\n");
}
```

### For Python users:

#### Step 2: Get into the folder of the code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

#### Step 3: Run

```
sudo python 10_slideSwitch.py
```

## Code Explanation

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.

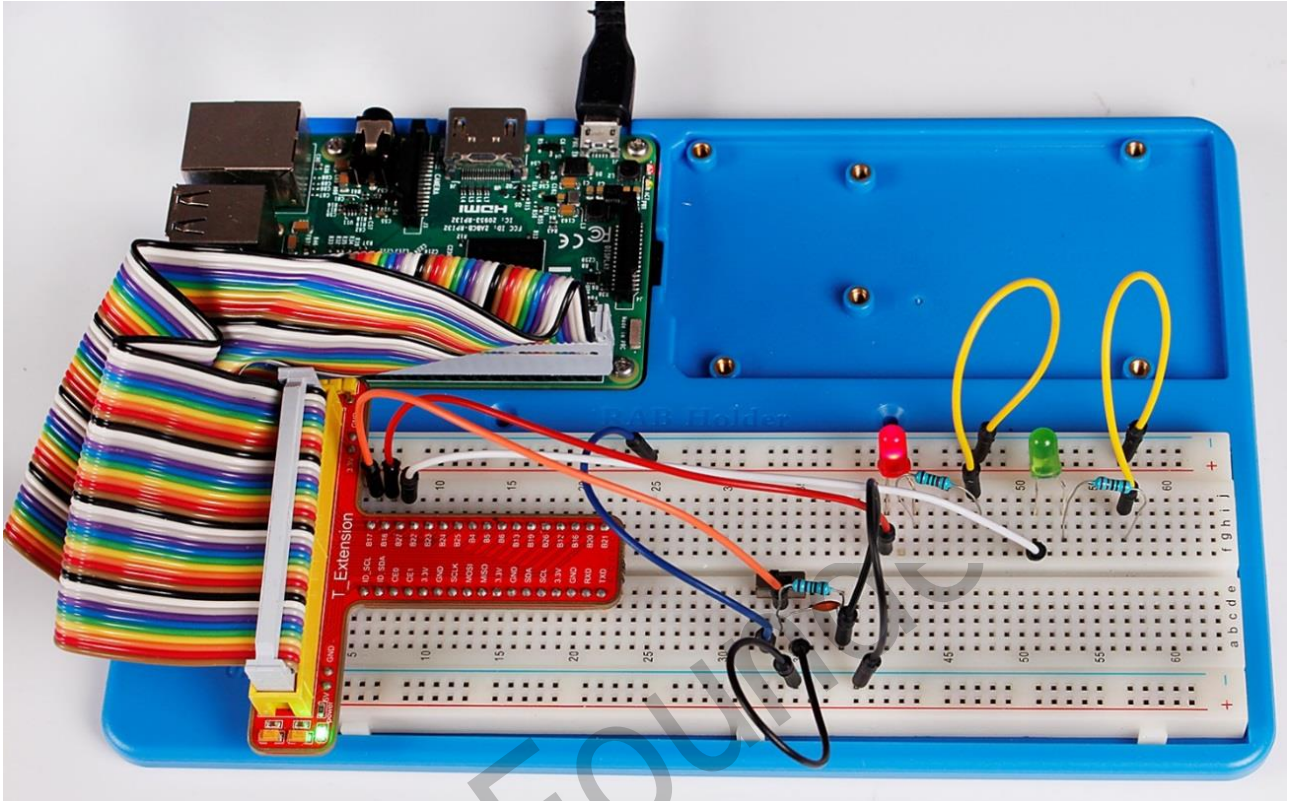
```
if GPIO.input(slidePin) == 1:
    print 'LED1 ON (High Value)'
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.HIGH)
```

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.

```
if GPIO.input(slidePin) == 0:
    print 'LED2 ON (Low Value)'
```

```
GPIO.output(led2Pin, GPIO.LOW)
GPIO.output(led1Pin, GPIO.HIGH)
```

Now pull the slide, and you can see the two LEDs light up alternately.



# Lesson 11 How to Drive a DC Motor

## Introduction

In this lesson, we will learn to how to use L293D to drive a DC motor and make it rotate clockwise and counterclockwise. [Since the DC Motor needs a larger current, for safety purpose, here we use the Power Supply Module to supply motors.](#)

## Components

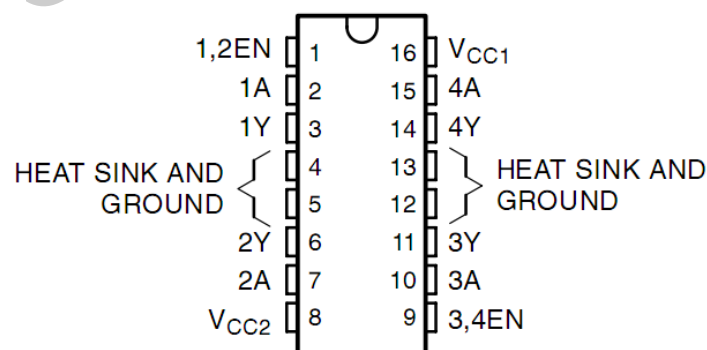
- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* L293D
- 1 \* DC motor
- 1 \* Power Module
- Jumper wires

## Principle

### L293D

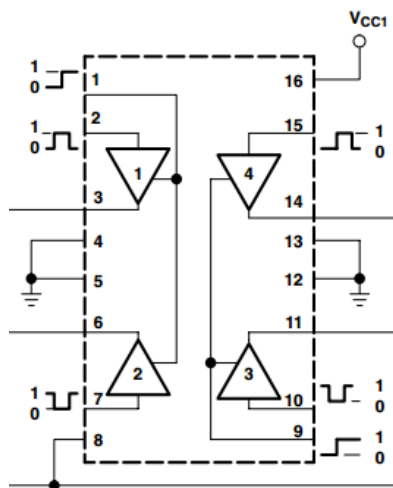
L293D is a 4-channel motor driver integrated by chip with high voltage and high current. It's designed to connect to standard DTL, TTL logic level, and drive inductive loads (such as relay coils, DC, stepping motors) and power switching transistors etc. DC Motors are devices that turn DC electrical energy into mechanical energy. They are widely used in electrical drive for their superior speed regulation performance.

See the figure of pins below. L293D has two pins (Vcc1 and Vcc2) for power supply. Vcc2 is used to supply power for the motor, while Vcc1 to supply for the chip. Since a small-sized DC motor is used here, connect both pins to +5V. If you use a higher power motor, you need to connect Vcc2 to an external power supply.



The following is the internal structure of L293D. Pin **EN** is an enable pin and only works with high level; **A** stands for input and **Y** for output. You can see the relationship among them at the right bottom. When pin **EN** is High level, if **A** is High, **Y** outputs high level; if A is Low, Y outputs Low level. When pin **EN** is Low level, the L293D does not work.





INPUTS†		OUTPUT Y
A	EN	
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,  
Z = high impedance (off)

In this experiment, it just needs to drive one motor, so here only half of the L293D will be used.

## DC Motor



This is a 5V DC motor. It will rotate when you give the two terminals of the copper sheet one high and one low level. For convenience, you can weld the pins to it.

Size: 25\*20\*15MM

Free-run current (3V): 70m

Stall current (3V): 800mA

Operation Voltage: 1-6V

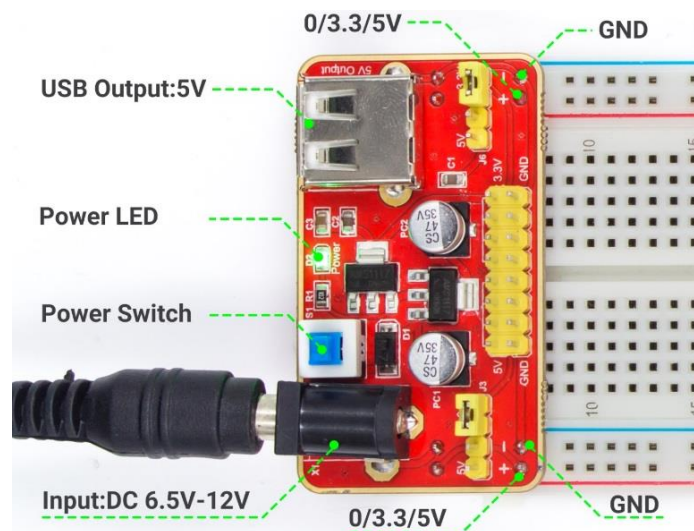
A Free-run speed (3V): 13000RPM

Shaft diameter: 2mm

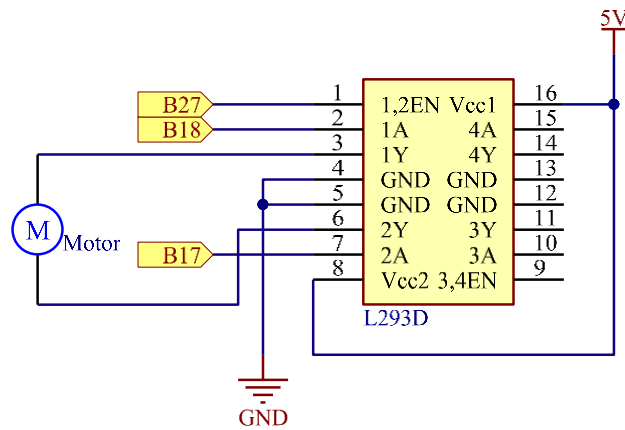
## Power Supply Module

In this experiment, it needs large currents to drive the motor especially when it starts and stops, which will severely interfere with the normal work of Raspberry Pi. Therefore, we separately supply power for the motor by this module to make it run safely and steadily.

You can just plug it in the breadboard to supply power. It provides a voltage of 3.3V and 5V, and you can connect either via a jumper cap included.



Schematic diagram:

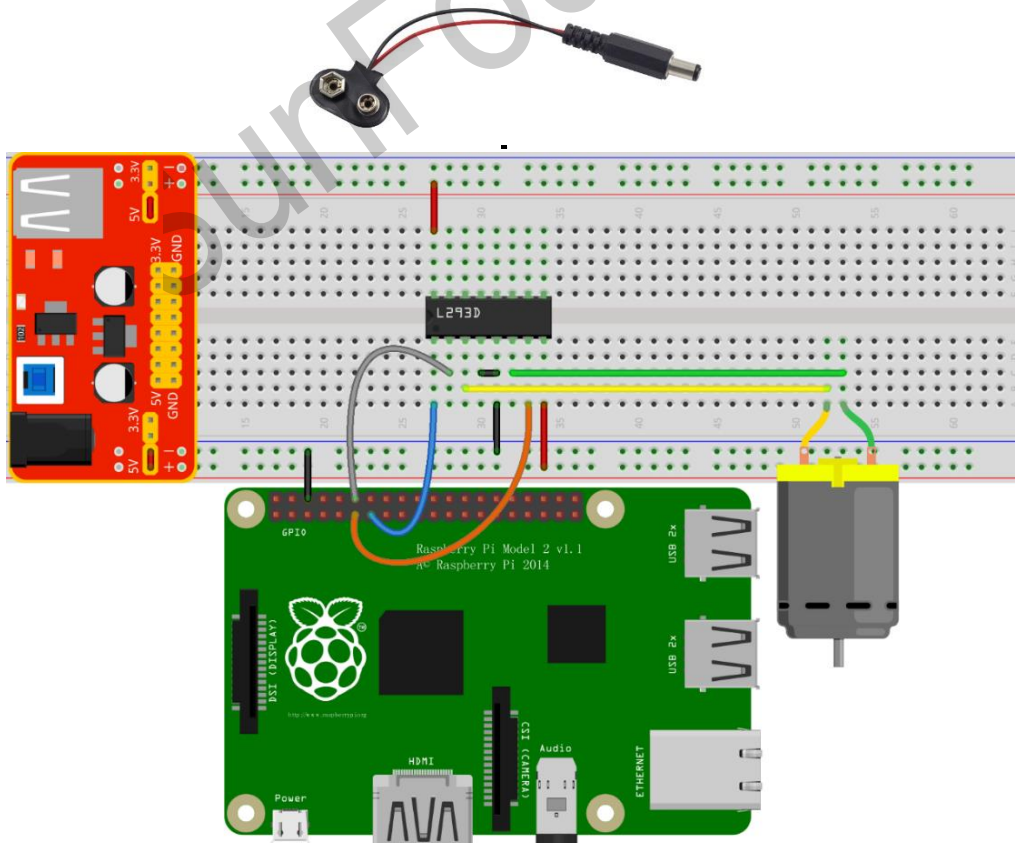


**Principle:** Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to B27, and set it as high level. Connect pin2 to B18, and pin7 to B27, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

## Experimental Procedures

**Step 1:** Build the circuit. Since the power supply module and T-cable are incompatible, we will not use the T-Cable in this experiment.

**Note:** The power module can apply a 9V battery with the 9V Battery Buckle in the kit. Insert the jumper cap of the power module into the 5V bus strips of the breadboard.



### For C language users:

**Step 2:** Get into the folder of the code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile

```
make 11_motor
```

**Step 4:** Run the executable file above.

```
sudo ./11_motor
```

### Code Explanation

```
digitalWrite(MotorEnable, HIGH): // Enable the L239D
digitalWrite(MotorPin1, HIGH):) // Set a high level for 2A(pin 7); since 1,2EN(pin 1) is
in high level, 2Y will output high level
digitalWrite(MotorPin2, LOW): // Set a low level for 1A, then 1Y will output low level,
and the motor will rotate.
for(i=0;i<3;i++){
    delay(1000);
} // this loop is to delay for 3*1000ms
digitalWrite(MotorEnable, LOW) // If 1,2EN (pin1) is in low level, L293D does not work.
Motor stops rotating.
digitalWrite(MotorPin1, LOW)
digitalWrite(MotorPin2, HIGH) // Reverse the current flow of the motor, then the motor
will rotate reversely.
```

### For Python users:

**Step 2:** Get into the folder of the code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 11_motor.py
```

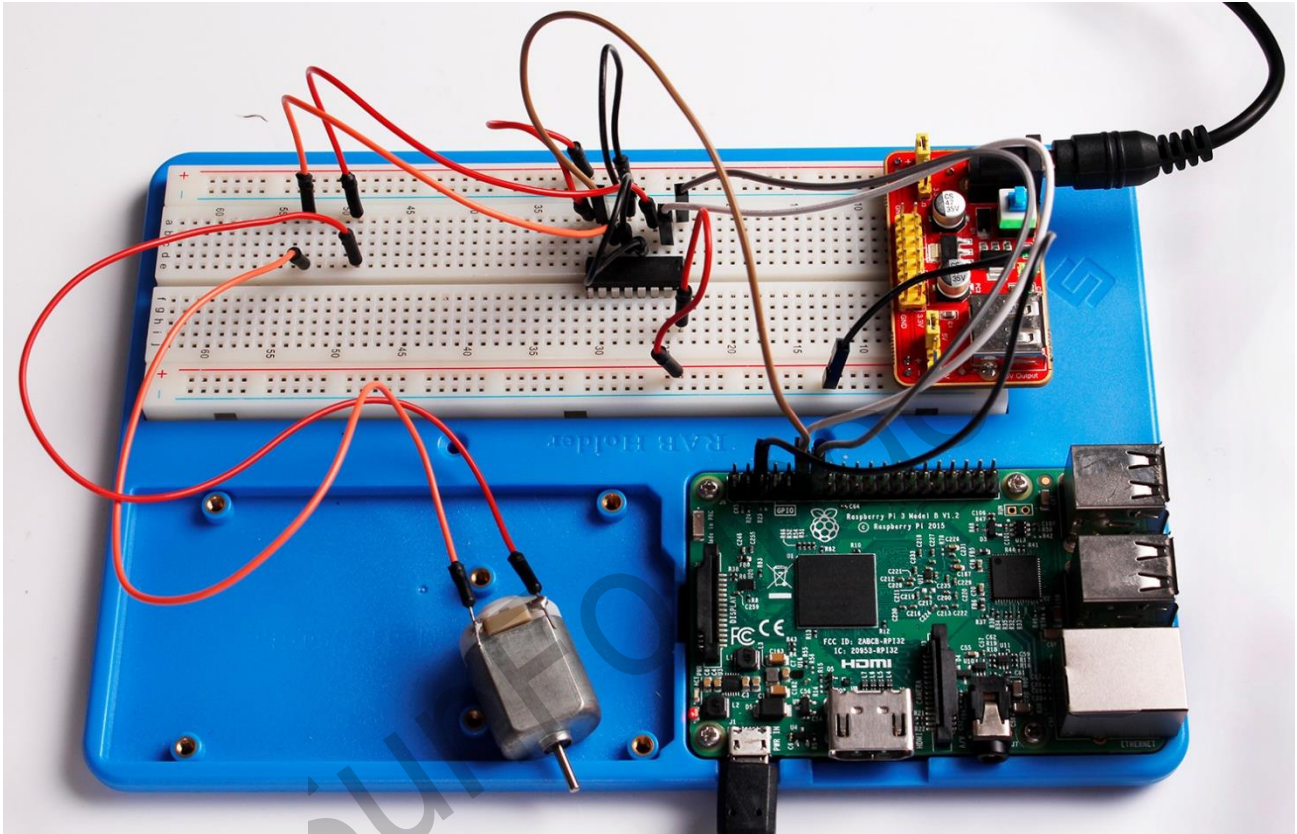
### Code Explanation

```
GPIO.setup(MotorPin1, GPIO.OUT) # Set pin1 and pin2 for motor's rotation direction
as output pin
GPIO.setup(MotorPin2, GPIO.OUT)
GPIO.setup(MotorEnable, GPIO.OUT) # Set pins for motor's working condition as
output pin
GPIO.output(MotorEnable, GPIO.LOW) # Set the motor low level for initial state

GPIO.output(MotorEnable, GPIO.HIGH) # Set the motor in high level
GPIO.output(MotorPin1, GPIO.HIGH) # Set pin1 in high level and pin2 in low level
```

```
GPIO.output(MotorPin2, GPIO.LOW) # Make the motor rotate clockwise
time.sleep(5)                     # rotate for 5 seconds
GPIO.output(MotorEnable, GPIO.LOW) # Stop the motor
time.sleep(5)                     #wait for 5 seconds
Code for motor counter-clockwise rotation is similar to sketch above
```

Now, you should see the motor blade rotating.



### Further Exploration

You can use buttons to control the clockwise and counterclockwise rotation of the motor blade based on the previous lessons. Also you can apply the PWM technology to control the rotation.

# Lesson 12 Rotary Encoder

## Introduction

A rotary encoder is an electro-mechanical device that converts the angular position or motion of a shaft or axle to analog or digital code. Rotary encoders are usually placed at the side which is perpendicular to the shaft. They act as sensors for detecting angle, speed, length, position, and acceleration in automation field.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 4 \* Jumper wires (Male to Male, 2 red and 2 black)
- 1 \* Network cable (or USB wireless network adapter)
- 1 \* Rotary Encoder module
- 1 \* 5-Pin anti-reverse cable

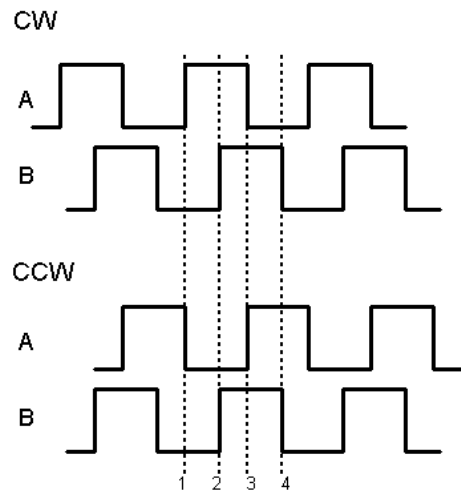
## Experimental Principle



A rotary encoder is an electronic switch with a set of regular pulses with strictly timing sequence. When used with IC, it can achieve increment, decrement, page turning, and other operations such as mouse scrolling, menu selection, and so on.

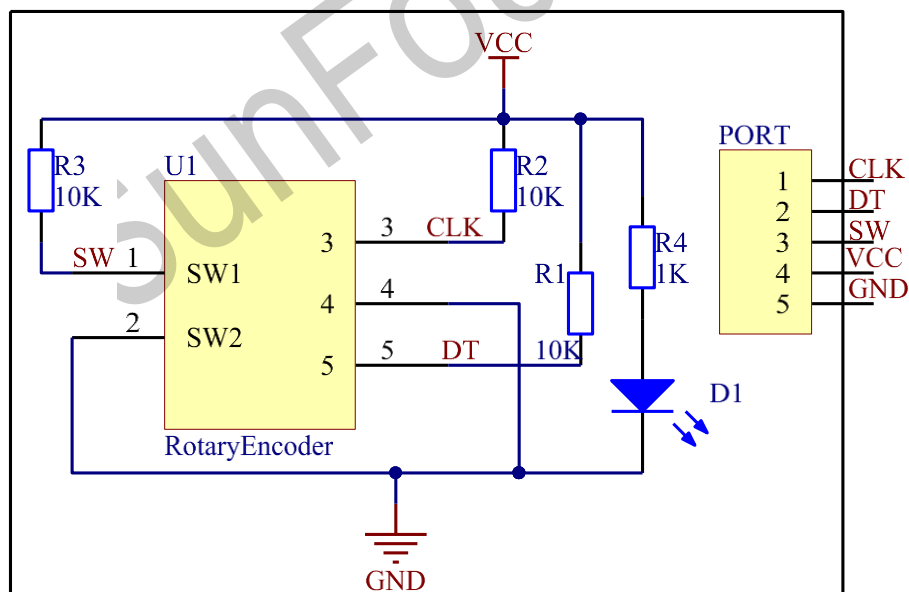
There are mainly two types of rotary encoders: absolute and incremental (relative) encoders. Here we use an incremental (relative) encoders.

Most rotary encoders have 5 pins with three functions of turning left & right and pressing down. Pin 1 and pin 2 are switch wiring terminals used to press. Pin 4 is generally connected to ground. Pin 3 and pin 5 are first connected to pull-up resistor and connect to VCC. Pin 3 and pin 5 generate two-phase square waves whose phase difference is 90°. Usually the two-phase square waves are called channel A and channel B as shown below:



We can see from the figure above: If channel A is in low level, and channel B converts from high level to low, it indicates the Rotary Encoder has spun clockwise (CW). If channel A is in low level, and channel B converts from low level to high, it indicates the Rotary Encoder has spun counter-clockwise (CCW). Thus when channel A is in low level, we can know the direction that Rotary Encoder spun by channel B.

The schematic diagram of the Rotary Encoder is shown as below. We can see that pin 3 on the Rotary Encoder is CLK of the module, while pin 5 is DT. Then we can know the Rotary's rotating direction by the value of CLK and DT.

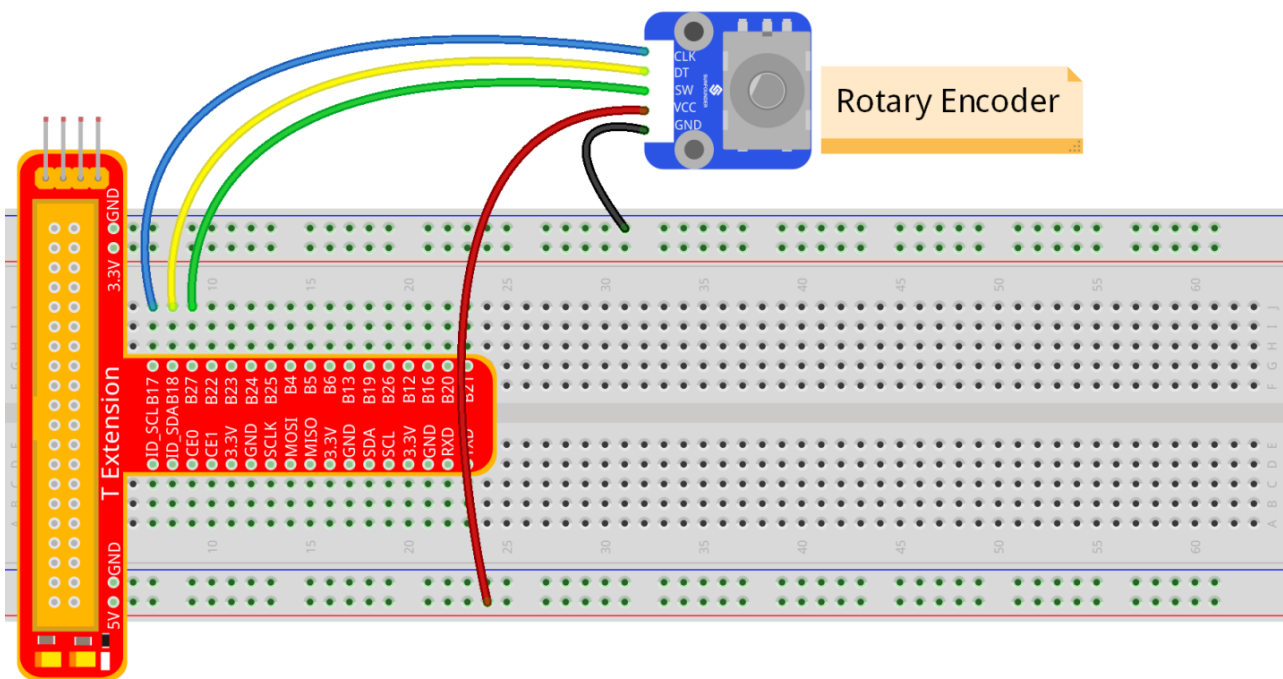


It is summarized by using oscilloscope to observe the output waveform of CLK and DT and operating the rotary encoder. You can try yourself.

## Experimental Procedures

**Step 1:** Build the circuit





**For C language users:**

**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile

```
make 12_rotaryEncoder
```

**Step 4:** Run the executable file above

```
sudo ./12_rotaryEncoder
```

## Code Explanation

```
#define RoAPin 0 // CLK connects to B17, define B17 as 0 in wiring Pi.
#define RoBPin 1 // DT connects to GPIO1, define B18 as 1 in wiring Pi.
#define SWPin 2 // SW connects to GPIO2

void rotaryDeal(void) : // Pi detects the pulse when spinning the rotary encoder, and
judge the spinning direction, then increase or decrease the value of globalCounter to
record the angular displacement.

    Last_RoB_Status = digitalRead(RoBPin); // Read the value of DT
    while(!digitalRead(RoAPin){ // If CLK is low, run the program below.
        Current_RoB_Status = digitalRead(RoBPin); // Read the value of DT, and store it
in Current_RoB_Status.
        flag = 1;
    }
    if(flag == 1) // If CLK outputs low level, then flag=1
        {flag = 0;
```

```

        if((Last_RoB_Status == 0)&&(Current_RoB_Status == 1)){ // If DT value converts
from low to high, the globalCounter adds 1.
        }
        if((Last_RoB_Status == 1)&&(Current_RoB_Status == 0)){ //If DT value converts
from high to low,
            globalCounter --; // the globalCounter
decreases 1.
        }
    }
    printf("globalCounter : %d\n",globalCounter); // Print the value of globaCounter.
    void btnISR(void): // If the rotary encoder is pressed down, reset the value.

```

### For Python users:

**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 12_rotaryEncoder.py
```

### Code Explanation

```

globalCounter = 0 # Set a global variable to count
flag = 0 # Set a flag for reverse spinning.
Last_RoB_Status = 0 # Set a variable to store the previous state of pinB
Current_RoB_Status = 0 # Set a variable to store the present state of pinB

# Define a function to deal with rotary encoder
def rotaryDeal():
    global counter
    global Last_RoB_Status, Current_RoB_Status
    flag = 0
    Last_RoB_Status = GPIO.input(RoBPin) # Store channel B state
    # When RoAPin level changes
    while(not GPIO.input(RoAPin)): # When channel A is not in low, exit the while
loop
        Current_RoB_Status = GPIO.input(RoBPin)
        flag = 1
    if flag == 1: # If flag value is 1, the rotary encoder is CW rotating
        # Reset flag
        flag = 0

```

```

        if (Last_RoB_Status == 0) and (Current_RoB_Status == 1):
            counter = counter + 1

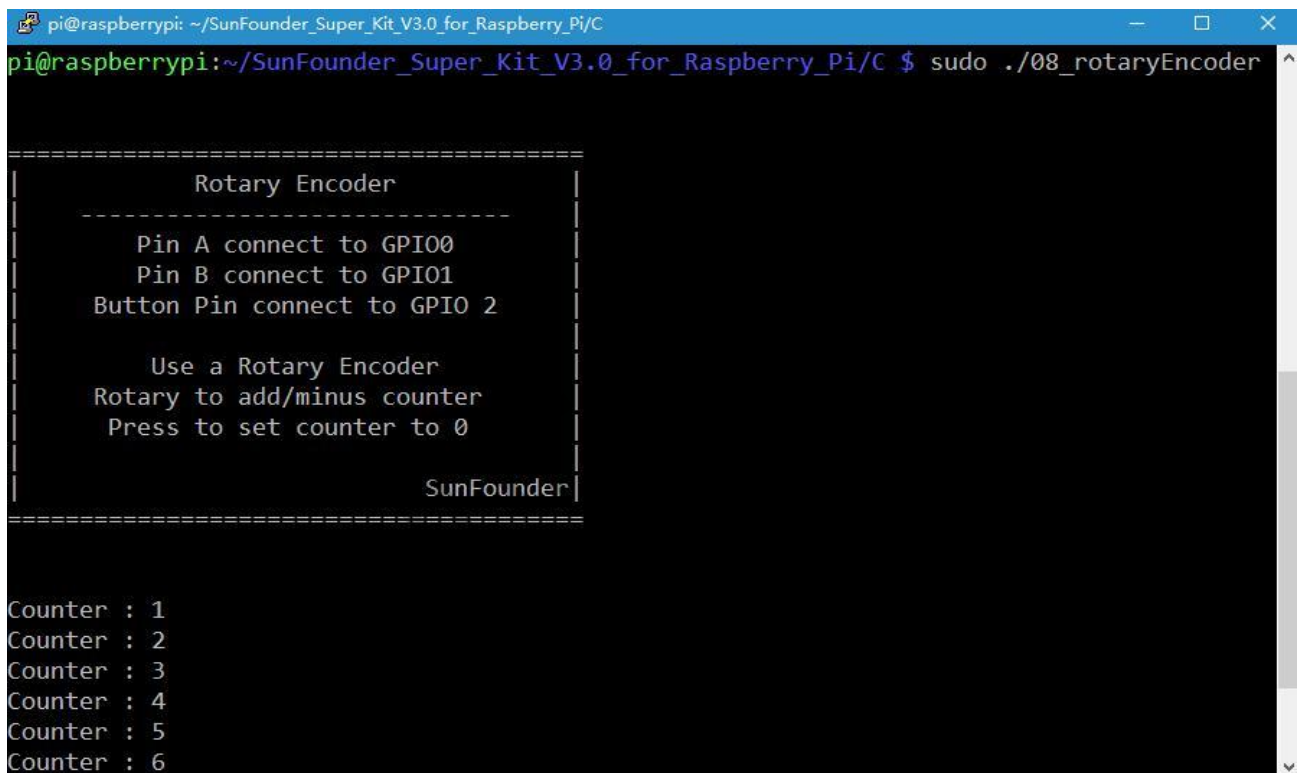
        if (Last_RoB_Status == 1) and (Current_RoB_Status == 0):
            counter = counter - 1

        print 'counter = %d' % counter

# Define a callback function on switch, to clean "counter"
def clear(ev=None):
    global counter
    counter = 0

```

Now, gently rotate the encoder to change the value of the variable in the above program, and you will see the value printed on the screen. Rotate the encoder clockwise, the value will increase; or rotate it counterclockwise, the value will decrease.



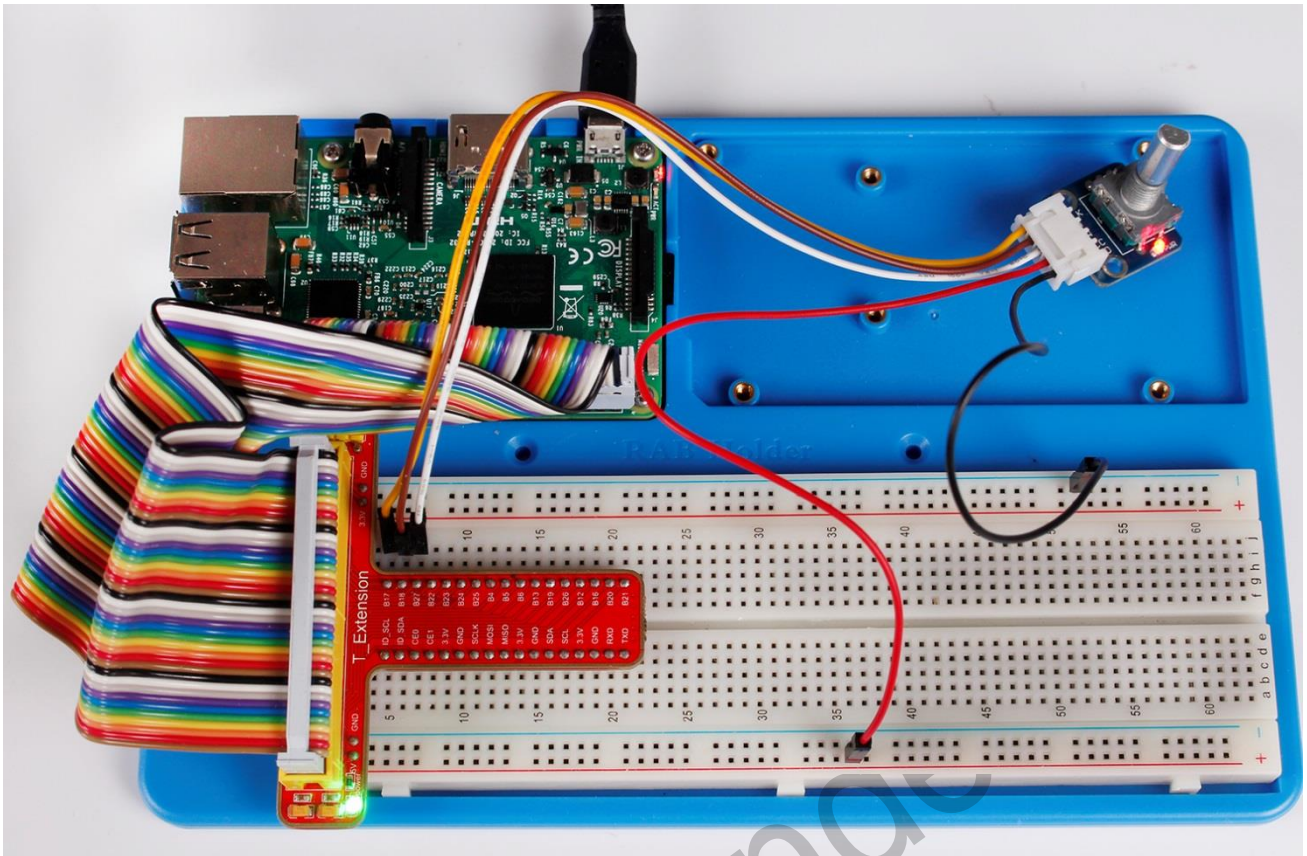
```

pi@raspberrypi: ~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
pi@raspberrypi:~/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C $ sudo ./08_rotaryEncoder

=====
|               Rotary Encoder               |
|-----|
| Pin A connect to GPIO0                    |
| Pin B connect to GPIO1                    |
| Button Pin connect to GPIO 2              |
|
| Use a Rotary Encoder                      |
| Rotary to add/minus counter               |
| Press to set counter to 0                 |
|
|                                         SunFounder |
|-----|
=====

Counter : 1
Counter : 2
Counter : 3
Counter : 4
Counter : 5
Counter : 6

```



### Further Exploration

In this experiment, the pressing down function of rotary encoder is not involved. Try to explore this function by yourself!

# Lesson 13 Driving LEDs by 74HC595

## Introduction

In this lesson, we will learn how to use 74HC595 to make eight LEDs blink regularly.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* 74HC595
- 8 \* LED
- 8 \* Resistor (220Ω)
- Jumper wires

## 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 that you can save I/O ports of an MCU. The 74HC595 is widely used to indicate multipath LEDs and drive multi-bit segment displays. "Three-state" mentioned above refers to the fact that you can set the output pins as either high, low or high impedance. With data latching, the instant output will not be affected during the shifting; with data output, you can cascade 74HC595s more easily. Compatible with low voltage TTL circuit, 74HC595 can transform serial input of 8-bit data into parallel output of 8-bit data. So it is often used to extend GPIO for embedded system and drive low power devices.

1	Q1	VCC	16
2	Q2	Q0	15
3	Q3	DS	14
4	Q4	CE	13
5	Q5	STcp	12
6	Q6	SHcp	11
7	Q7	MR	10
8	GND	Q7'	9

## 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; here it is directly connected to 5V to keep the chip from resetting.

**SH\_CP:** 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.

**ST\_CP:** Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

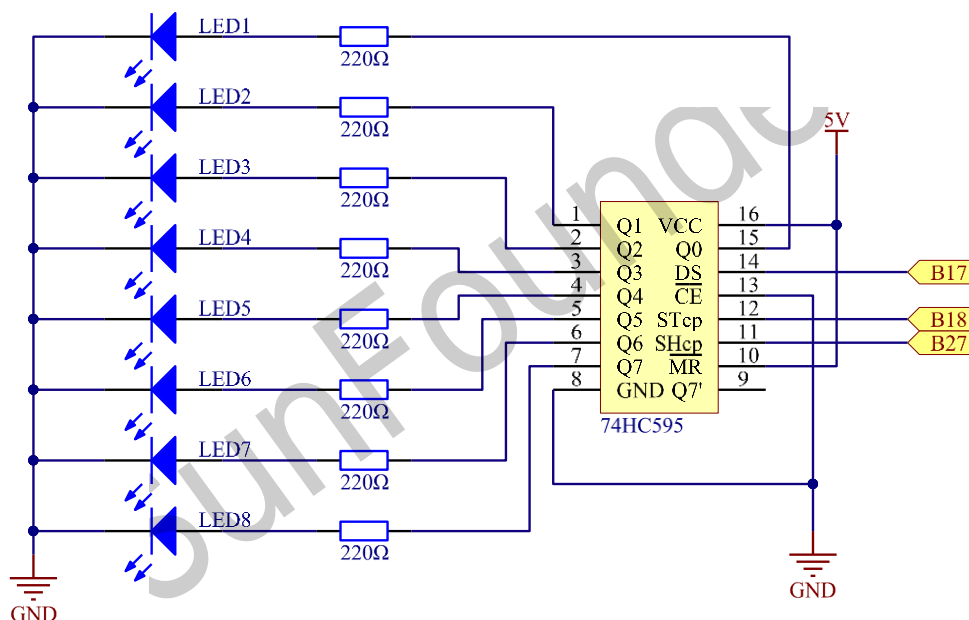
**OE:** Output enable pin, active at low level; here connected to GND to keep 74HC595 in output enable state.

**DS:** Serial data input pin

**VCC:** Positive supply voltage

**GND:** Ground

The schematic diagram is shown as below:

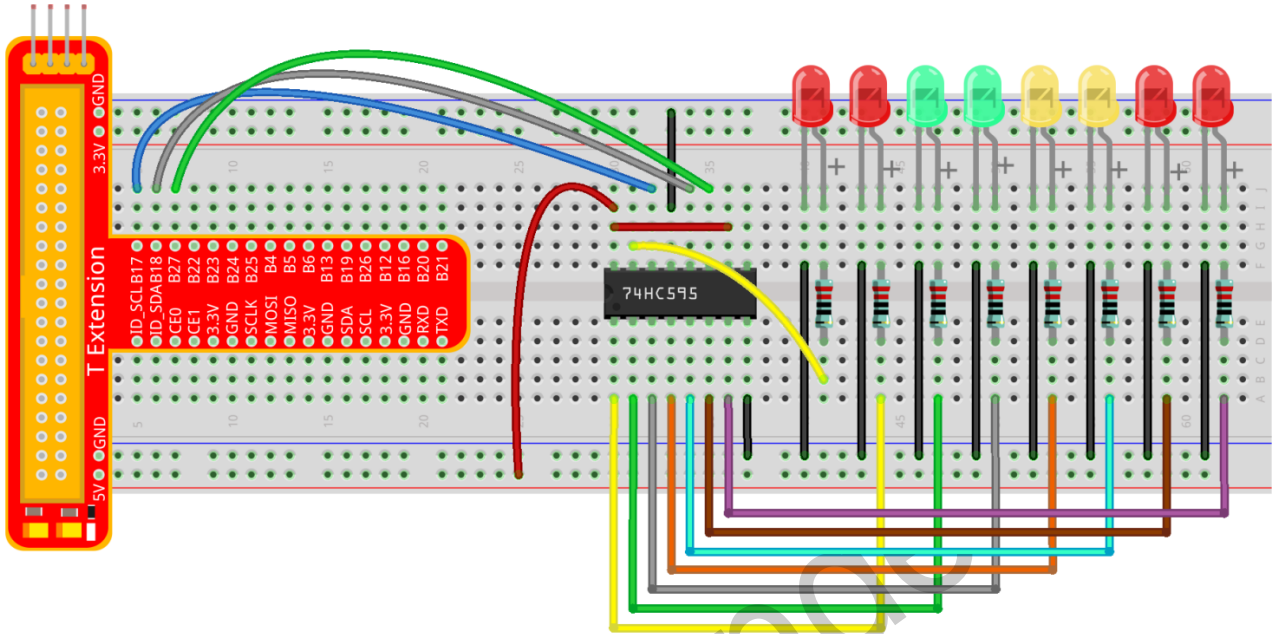


**Principle:** In this experiment, connect 74HC595's ST\_CP to Raspberry Pi's B18, SH\_CP to B27, and DS to B17; connect a current-limit resistor and then a LED to Q0-Q7 respectively; connect MR and VCC to 5V, CE and GND to GND. Input data in DS pin to the shift register when SH\_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST\_CP (the clock input of the memory) is at the rising edge, and output to Q0-Q7. Then you can control the states of SH\_CP and ST\_CP via Raspberry Pi GPIO to transform serial input data into parallel output data so as to save Raspberry Pi GPIOs.



## Experimental Procedures

**Step 1:** Build the circuit. If you want to take out the chip from the breadboard, DO NOT pull it in one direction forcefully, for fear that the pins on it may be bent and you may get hurt. Try to use a sharp tool to cross the notch of the breadboard to remove the chip.



**For C language users:**

**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile

```
make 13_74HC595_LED
```

**Step 4:** Run the executable file above.

```
sudo ./13_74HC595_LED
```

## Code Explanation

unsigned char LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80}; // This array is to store the output values of Q0-Q7. For example, 0x01 in binary format is 0000 0001, thus Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 are 0 0 0 0 0 0 0 1 respectively, that is Q0=1, and the LED connected to Q0 will light up. Thus we can light up the eight LEDs separately in this way.

```
void pulse(int pin){ // generate a rising edge
    digitalWrite(pin, 0);
    digitalWrite(pin, 1);
}
```

```
void SIPO(unsigned char byte){ // Assign the char byte to the SDI bit by bit
    int i;
```

```

for(i=0;i<8;i++){
    digitalWrite(SDI, ((byte & (0x80 >> i)) > 0)); // Use the for loop to count 8
times in cycle, and write a 1-bit data to the SDI each time. The data is a result of the
AND operation. (0x80 >> i) is to implement the operation from left to right by bit, so
each time one of the eight bits in byte (0000 0001).

    pulse(SRCLK); // the shift register generates a rising edge pulse, and data in
DS will shift to the shift register.

} // This part is to assign the data in byte to SDI(DS) by bits, thus when the shift
register generates a rising edge pulse, data in SDI(DS) will transfer to it by bits.
}

void init(void): // Set DS, ST_CP, SH_CP as output, and low level as the initial state
for(i=0;i<8;i++){
    SIPO(LED[i]); // Assign the value in the LED[i] array to SDI(DS). When i=1,
LED[0]=0x01 shifts to the shift register.

    pulse(RCLK); // RCLK (ST_CP) generates a rising edge pulse, and the data of the
shift register is stored in the RCLK (ST_CP) storage register, and output at Q0-Q7.

    delay(150);
} After 8 cycles, Q0-Q7 will output 0x01 to 0x10 in sequence, that is to light up the
LEDs connected to Q0-Q7 in turn.

```

Sketch in later part not explained here is to light up 8 LEDs together, and dim them; then light up LEDs connected to Q7-Q0 one by one, and all 8 LEDs light up, dim in the end. Thus, a cycle completes. You can observe the LEDs' state.

### For Python users:

**Step 2:** Get into the folder of the code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 13_74HC595_LED.py
```

### Code Explanation

```

LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80] # Define some LED blinking
modes. Convert hexadecimal value to binary value will be more intuitionistic. For
instance, 0x01 is binary 00000001, meaning the last LED lighting up; 0x80 is binary
10000000, representing the first LED lighting up.

LED1 = [0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff] # blink mode 1
LED2 = [0x01,0x05,0x15,0x55,0xb5,0xf5,0xfb,0xff] # blink mode 2
LED3 = [0x02,0x03,0x0b,0x0f,0x2f,0x3f,0xbf,0xff] # blink mode 3

# Shift the data to 74HC595

```

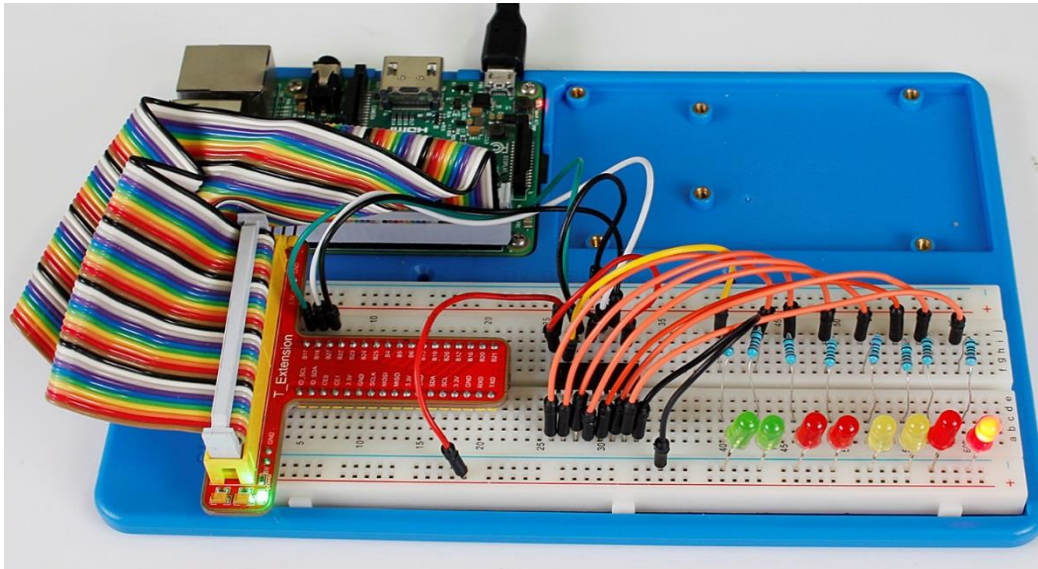
```

def hc595_shift(dat): # Shift the data to 74HC595
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit)) # Assign dat data to SDI pins of HC595 by
bits
        GPIO.output(SRCLK, GPIO.HIGH) # Every SRCLK adds one, the shift register moves
one bit.
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH) # Everytime RCLK adds one, the HC595 updates output.
        time.sleep(0.001)
        GPIO.output(RCLK, GPIO.LOW)
leds = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- '] # the array storing the LED state, used
for command line printing.
while True:
    # Change LED status from mode
    print "    mode"
    for onoff in mode: # Assign value to variable onoff by mode[] list
        hc595_shift(onoff)
        leds[mode.index(onoff)] = 1 # Show which led is on
        print leds
        time.sleep(sleeptime)
        leds[mode.index(onoff)] = '- ' # Show the led is off
        # for loops in later part work similarly, lighting up LED by list.

```

Input a 2-bit hexadecimal parameter `dat` via `hc595_in(dat)` to control 8 LEDs state, and `hc595_out()` will output state to 8 LEDs. In `while True`, the `for` loop will shift the LED blinking list to the `hc595_in(dat)` function, thus we can see the LED light flowing.

Here you should see eight LEDs light up one by one, and then all light up and dim after a while; then eight LEDs will light up from reverse direction one by one, and then all light up and then dim after a while. This cycle will keep running.



### Further Exploration

In this experiment, three Raspberry Pi GPIOs are used to separately control 8 LEDs based on 74HC595. In fact, 74HC595 has another powerful function – cascade. With cascade, you can use a microprocessor to control more peripherals. We'll check more details later.

# Lesson 14 Driving 7-Segment Display by 74HC595

## Introduction

Since we've got some knowledge of the 74HC595 in the previous lesson, now let's try to use it and drive a 7-segment display to show a figure from 0 to 9 and A to F.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* 74HC595
- 1 \* 7-segment display
- 1 \* Resistor (220Ω)
- Jumper wires

## Principle

### 7-Segment Display

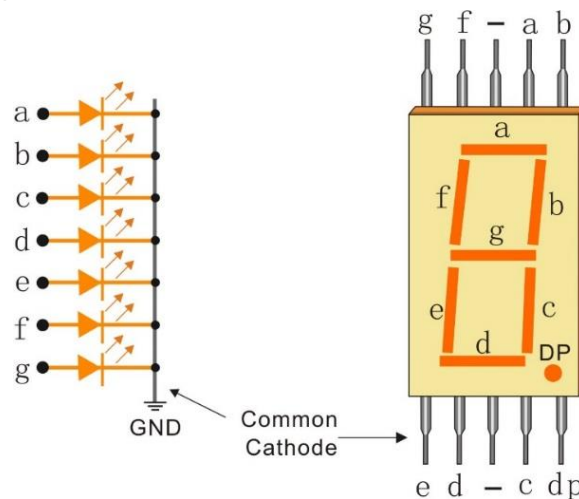
A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral (both decimal and hexadecimal) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal point (DP) when two or more 7-segment displays are connected together to display numbers greater than ten.



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.

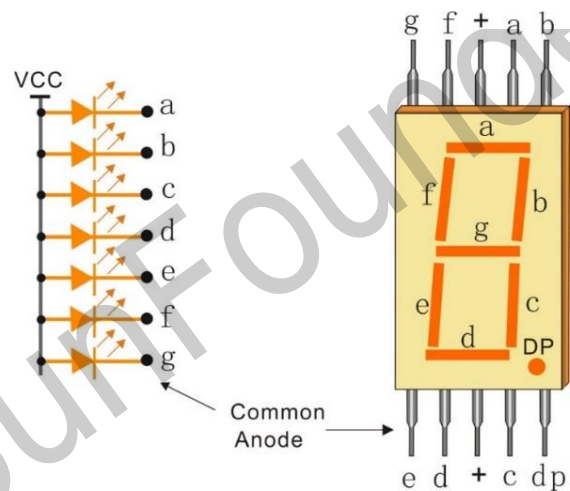
The common pin of the display generally tells its type. There are two types of pin connection: a pin of connected cathodes and one of connected anodes, indicating Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected.

## Common Cathode 7-Segment Display



In a common cathode display, the cathodes of all the LED segments are connected to the logic "0" or ground. Then an individual segment (a-g) is energized by a "HIGH", or logic "1" signal via a current limiting resistor to forward bias the anode of the segment.

## Common Anode 7-Segment Display

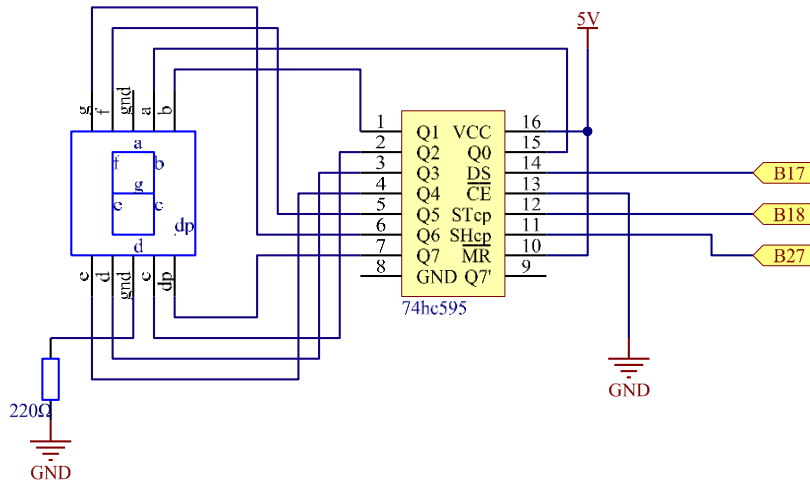


In a common anode display, the anodes of all the LED segments are connected to the logic "1". Then an individual segment (a-g) is energized by a ground, logic "0" or "LOW" signal via a current limiting resistor to the cathode of the segment.

In this experiment, a common cathode 7-segment display is used. It should be connected to ground. When the anode of an LED in a certain segment is at high level, the corresponding segment will light up; when it is at low, the segment will stay dim.

The schematic diagram is shown as below:

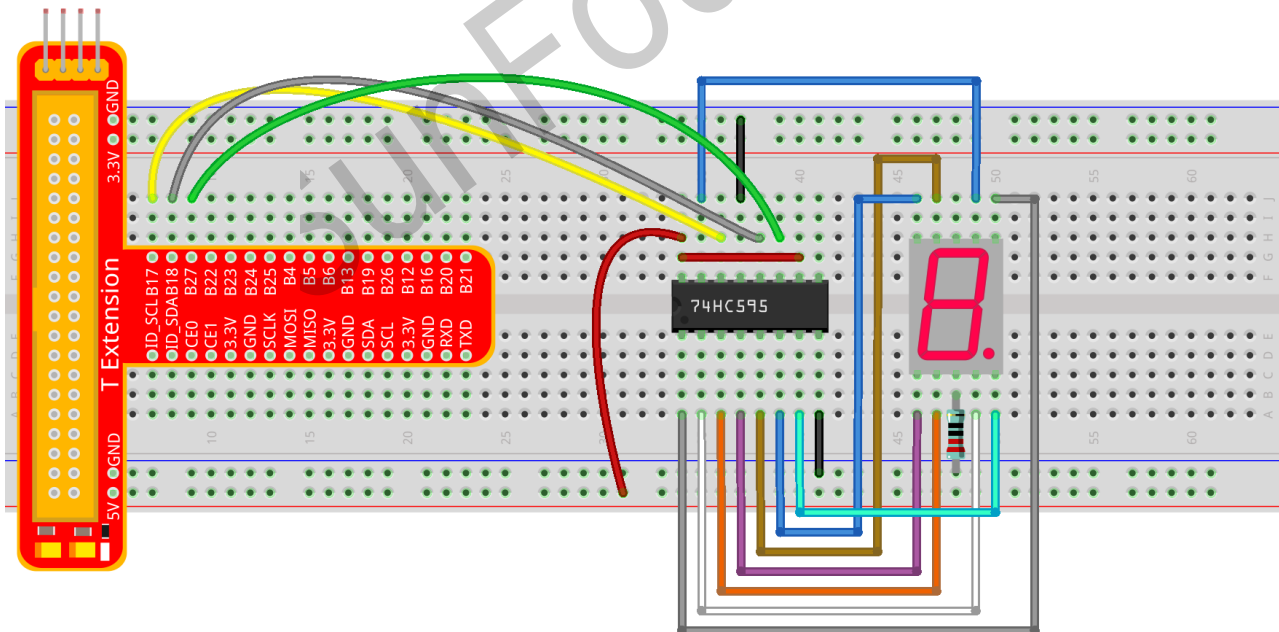




**Principle:** Connect pin ST\_CP of 74HC595 to Raspberry Pi B18, SH\_CP to B27, DS to B17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH\_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST\_CP (the clock input of the memory) is at the rising edge. Then you can control the states of SH\_CP and ST\_CP via the Raspberry Pi GPIOs to transform serial data input into parallel data output so as to save Raspberry Pi GPIOs and drive the display.

## Experimental Procedures

### Step 1: Build the circuit



### For C language users:

#### Step 2: Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

#### Step 3: Compile

```
make 14_segment
```

**Step 4:** Run the executable file above.

```
sudo ./14_segment
```

### Code Explanation

```
unsigned char SegCode[17] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80};
// display array from 0 to F.
void init(void){} // Initialize the function, set ds, st_cp, sh_cp three pins to low
level, and the initial state as 0.
void hc595_shift(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i)); // Assign the dat data to SDI(DS) by
bits. Here we assume dat=0x3f(0011 1111, when i=0, 0x3f will shift right(<<) 0 bits,
0x3f & 0x80 = 1000 0000,
        digitalWrite(SRCLK, 1); // SH_CP will convert from low to high, and generate a
rising edge pulse, then shift the DS date to shift register.
        delay(1);
        digitalWrite(SRCLK, 0);
    } // to assign 8 bit value to 74HC595's shift register

    digitalWrite(RCLK, 1); // ST_CP converts from low to high and generate a rising
edge, then shift data from shift register to storage register.
    delay(1);
    digitalWrite(RCLK, 0);
} // Transfer data in shift register to data register to update the displayed data.
```

**For Python users:**

**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 14_segment.py
```

### Code Explanation

```
# Define a segment code from 0 to F in Hexadecimal
# Common cathode
segCode =
[0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]
# Common anode
```

```

# segCode =
[0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e]
# Shift the data to 74HC595
def hc595_shift(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)
for code in segCode: # Input item in segCode list to hc595_shift()function, to display
the character.
    hc595_shift(code)

```

If you want to display a number, use the `hc595_shift()` function, `segCode` list and decimal value `x` in the sketch:

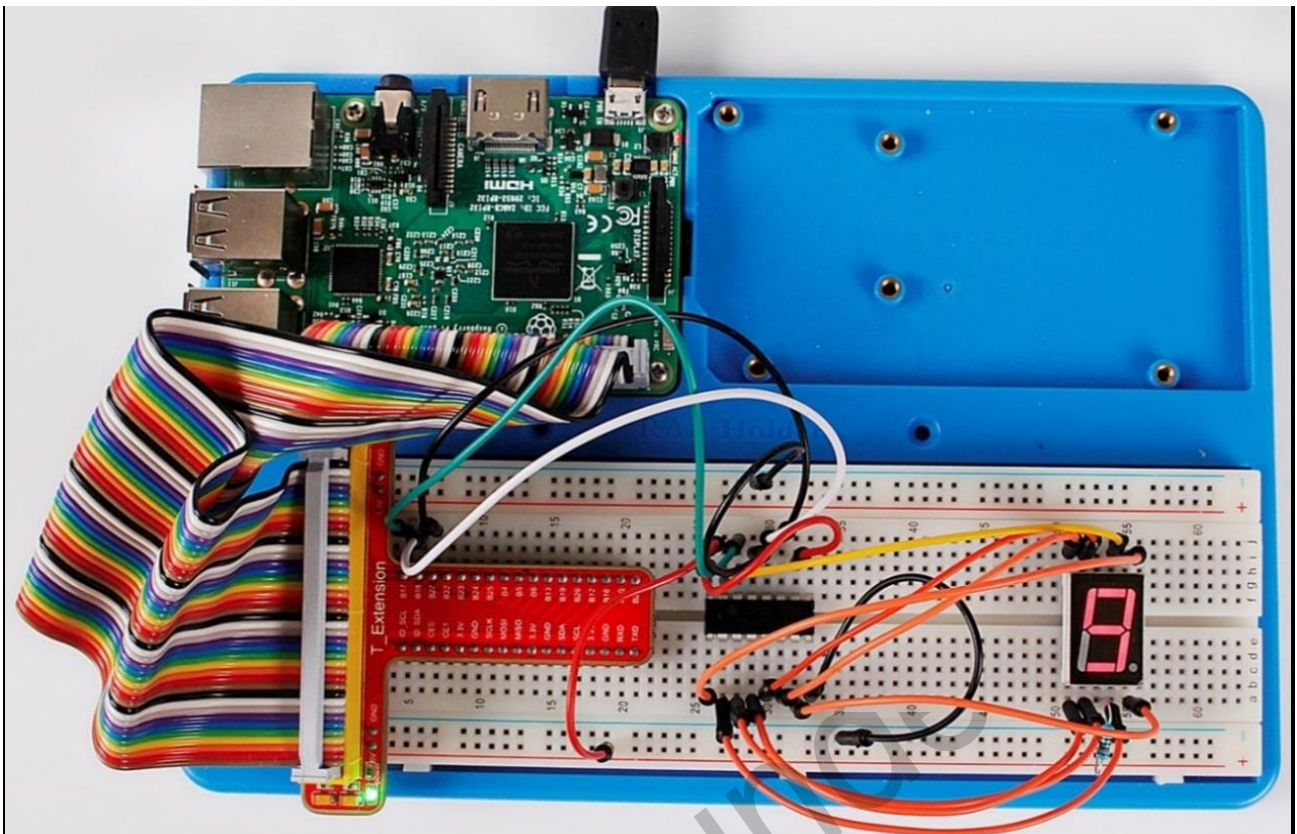
```

hc595_shift(segCode[x]) # x is a number needs to be displayed ranging from 0~15, and it
will be converted and displayed by 0~F in hexadecimal.

```

**Note:** The hexadecimal format of number 0~15 are (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

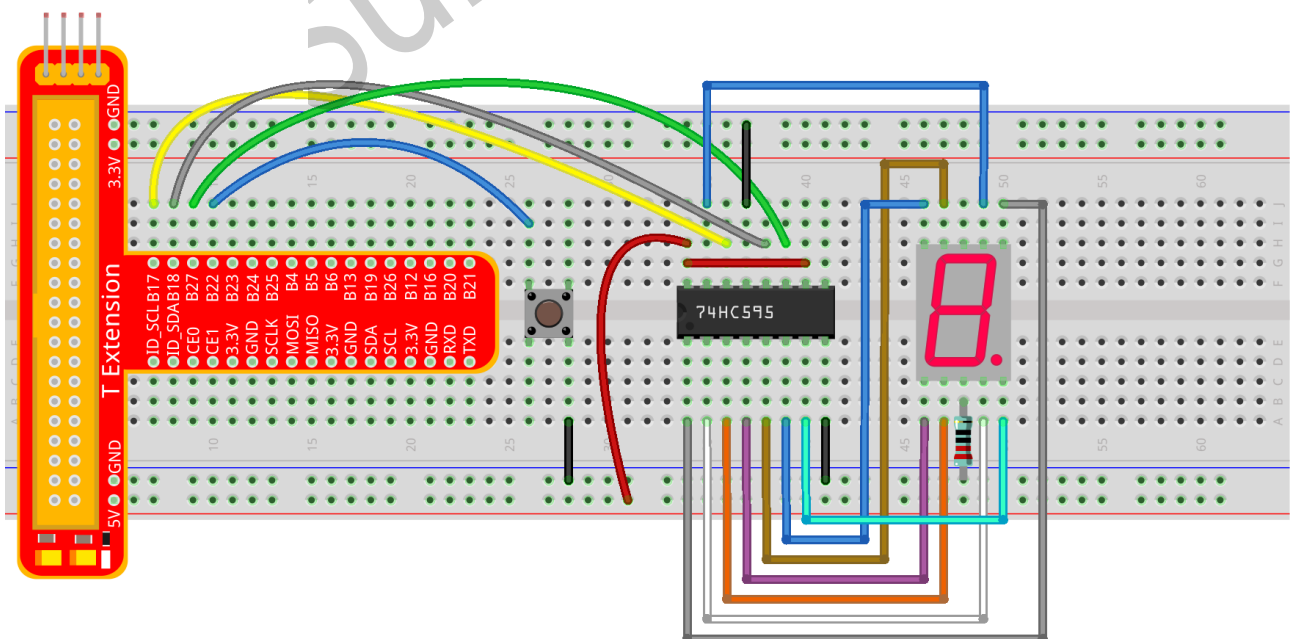
You should see the 7-segment display from 0 to 9 and A to F.



### Further Exploration

You can slightly modify the hardware and software based on this experiment to make a dice. For hardware, add a button to the original board.

### Build the circuit:



Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

Next, Compile the Code

```
make 14_dice
```

Run

```
sudo ./14_dice
```

## Code Explanation

```
void randomISR(void){ // An interrupt function, run when the interrupt happens
    flag = 1; // flag represents the state of the button
}
if(wiringPiISR(TouchPin, INT_EDGE_FALLING, &randomISR)){ //Set an interrupt here as the
falling edge for TouchPin. When the interrupt happens, execute the function randomISR().
    printf("Unable to setup ISR : %s\n", strerror(errno));
    return 1;
}
srand(time(NULL));
num = rand() % 6;
// Two functions here: One is the srand function, which is used before calling function
rand() and used as seed for the random number generator; while the other is rand(),
which is a function to generate the random number. Usually, these two functions are used
together to generate the random number. Thus a random number of 0-6 will be displayed on
the 7-segment display.
```

**For Python users:**

**Step 2:** Get into the folder of the code.

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 14_dice.py
```

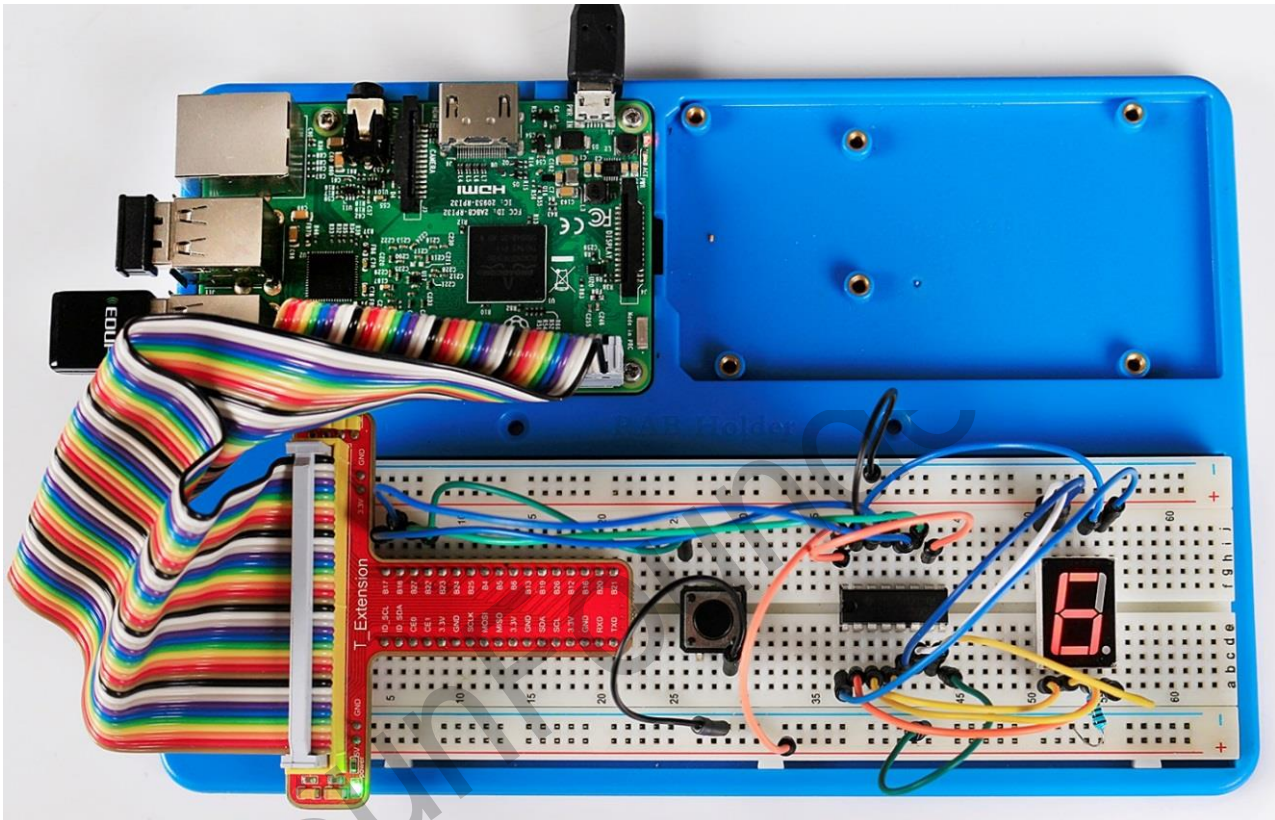
## Code Explanation

```
import random # use this function to generate the random number
SegCode = [0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d] # Define a segment code from 1 to 6
in Hexadecimal
GPIO.add_event_detect(TouchPin, GPIO.RISING, callback = randomISR, bouncetime = 20) #
Set an interrupt, and the rising edge for TouchPin. When the interrupt happens, execute
the function randomISR(). Set bouncetime for button to 20ms.
def randomISR(channel): # Interrupt calling the function
    global flag
```



```
flag = 1  
num = random.randint(1,6) # Generate a random number from 1~6.  
hc595_shift(SegCode[num-1]) # Output the hexadecimal values in list by 74HC595.
```

Now you should see a number flashing between 0 and 6 quickly on the segment display. Press the button on the breadboard, and the display will statically display a random number between 0 and 6 for 2 seconds and then circularly flash randomly between 0 and 6 again.



## Summary

Through this lesson, you may have mastered the basic principle and programming for 7-segment display based on Raspberry Pi, as well as more knowledge about using 74HC595. Now you can apply what you've learnt and put it into practice to create your own works!



# Lesson 15 Driving Dot-Matrix by 74HC595

## Introduction

As the name suggests, an LED dot matrix is a matrix composed of LEDs. The lighting up and dimming of the LEDs formulate different characters and patterns.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 2 \* 74HC595
- 1 \* Dot-Matrix
- Jumper wires

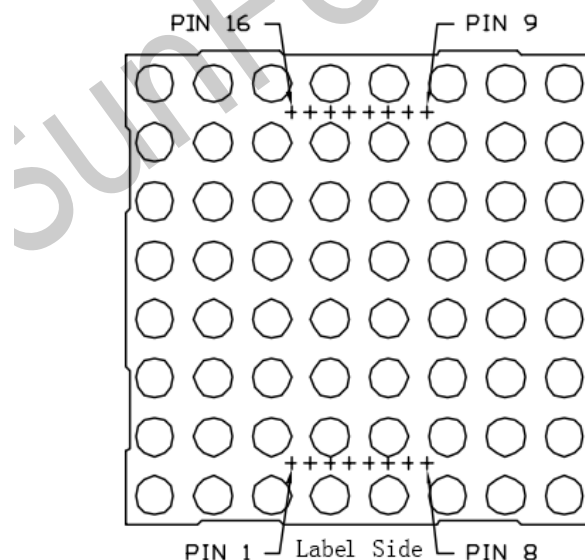
## Principle

### Dot Matrix

Generally, dot matrix can be categorized into two types: common cathode (CC) and common anode (CA). They look much alike, but internally the difference lies. You can tell by test. A CA one is used in this kit. You can see **788BS** labeled at the side.

See the figure below. The pins are arranged at the two ends at the back. Take the label side for reference: pins on this end are pin 1-8, and on the other are pin 9-16.

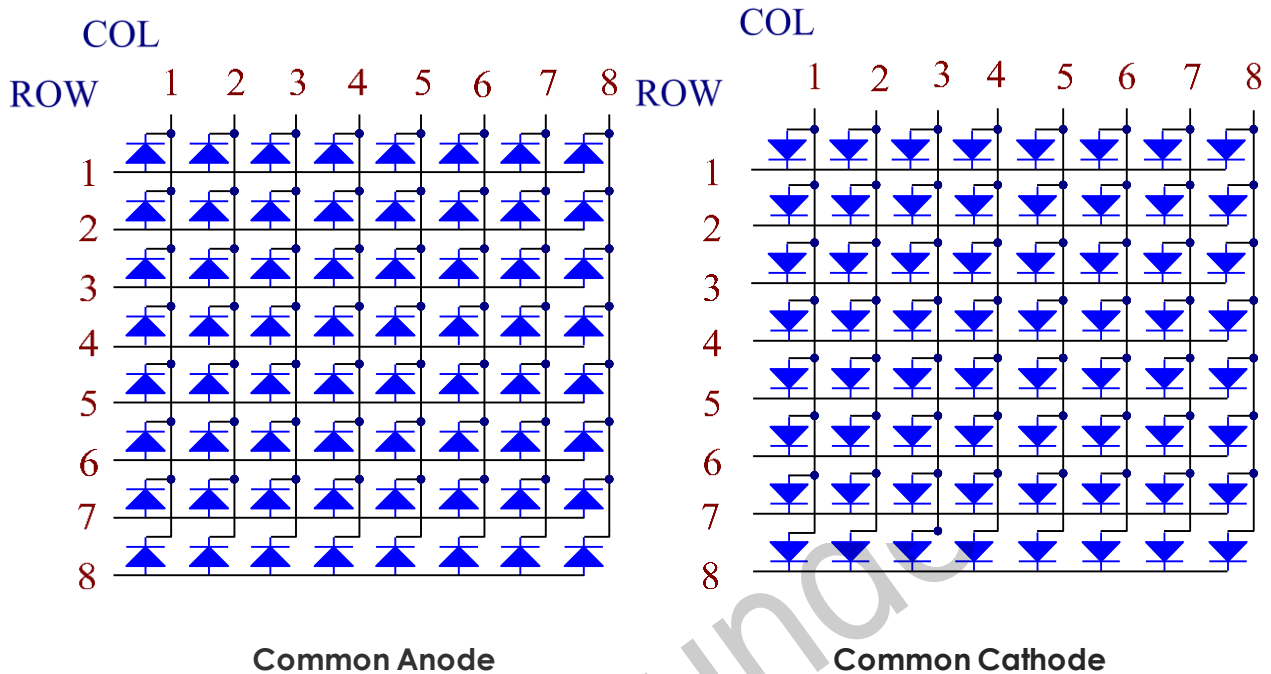
The external view:



Below the figures show their internal structure. You can see in a CA matrix, ROW represents the anode of the LED, and COL is cathode; it's contrary for a CC one. One thing in common: for both types, pin 13, 3, 4, 10, 6, 11, 15, and 16 are all COL, when pin 9, 14, 8, 12, 1, 7, 2, and 5 are all ROW. If you want to turn on the first LED at the top left corner, for a CA matrix, just set pin 9 as High and pin 13 as Low, and for a CC one, set pin 13 as High and pin 9 as Low. If

you want to light up the whole first column, for CA, set pin 13 as Low and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as High, when for CC, set pin 13 as High and ROW 9, 14, 8, 12, 1, 7, 2, and 5 as Low. Consider the following figures for better understanding.

The internal view:

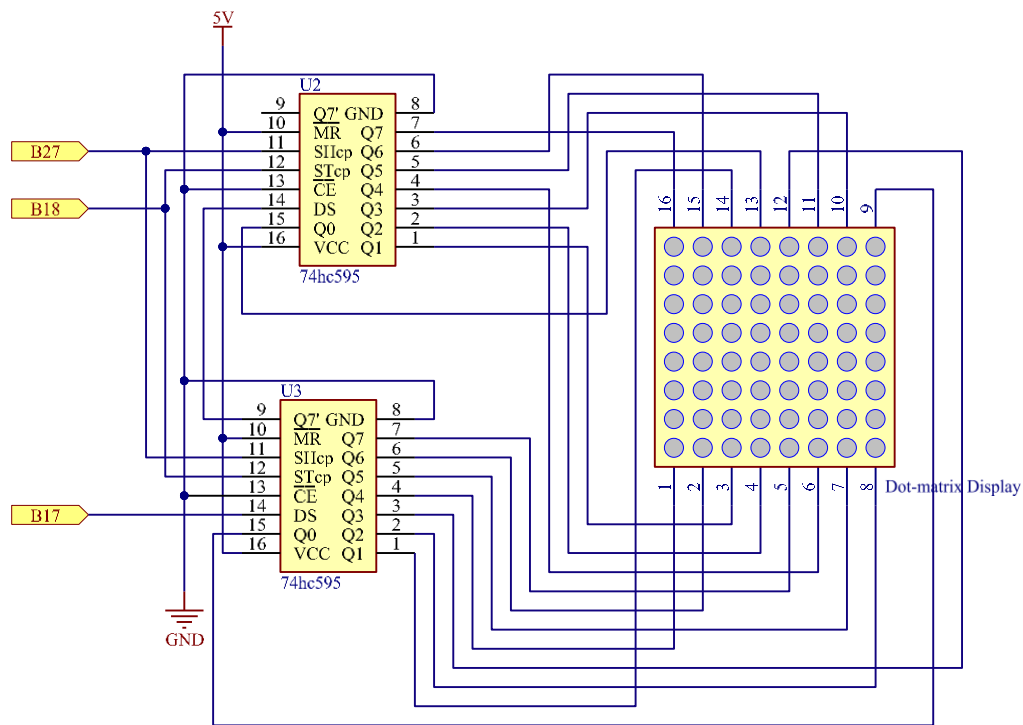


Pin numbering corresponding to the above rows and columns:

COL	1	2	3	4	5	6	7	8
Pin No.	13	3	4	10	6	11	15	16
ROW	1	2	3	4	5	6	7	8
Pin No.	9	14	8	12	1	7	2	5

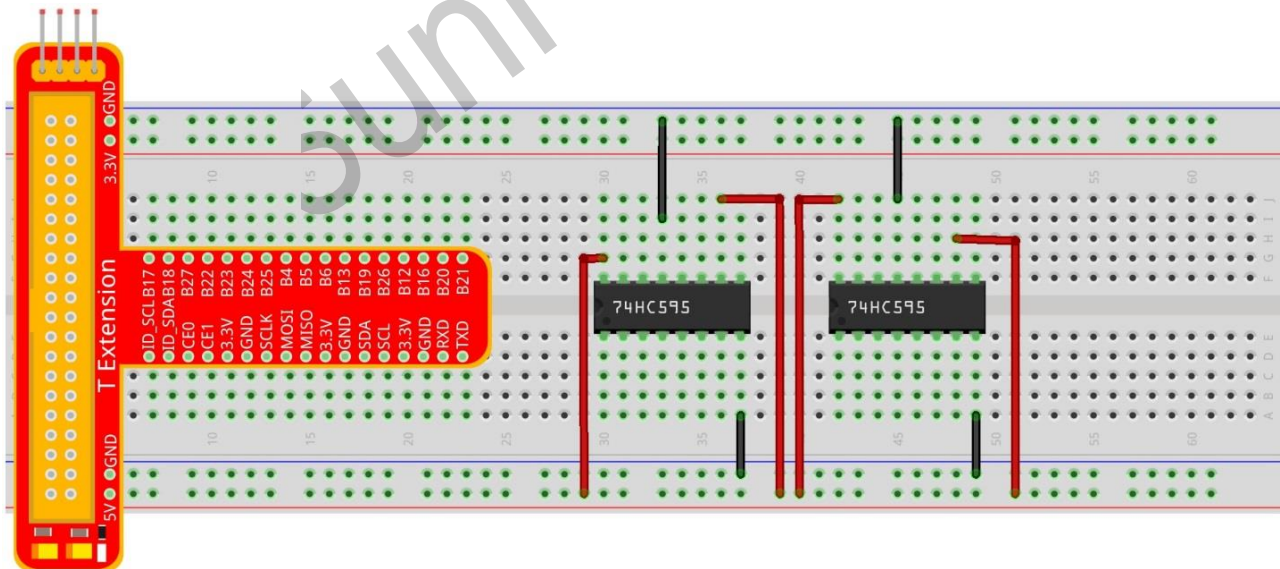
In this experiment, a CA dot matrix is used. You can see the label ends with "BS". The wiring and code are done for the CA matrix. Therefore, if you happen to have a CC matrix, you need to change the wiring and code. In addition, two 74HC595 chips are used here. One is to control the rows of the dot matrix while the other, the columns.

The schematic diagram

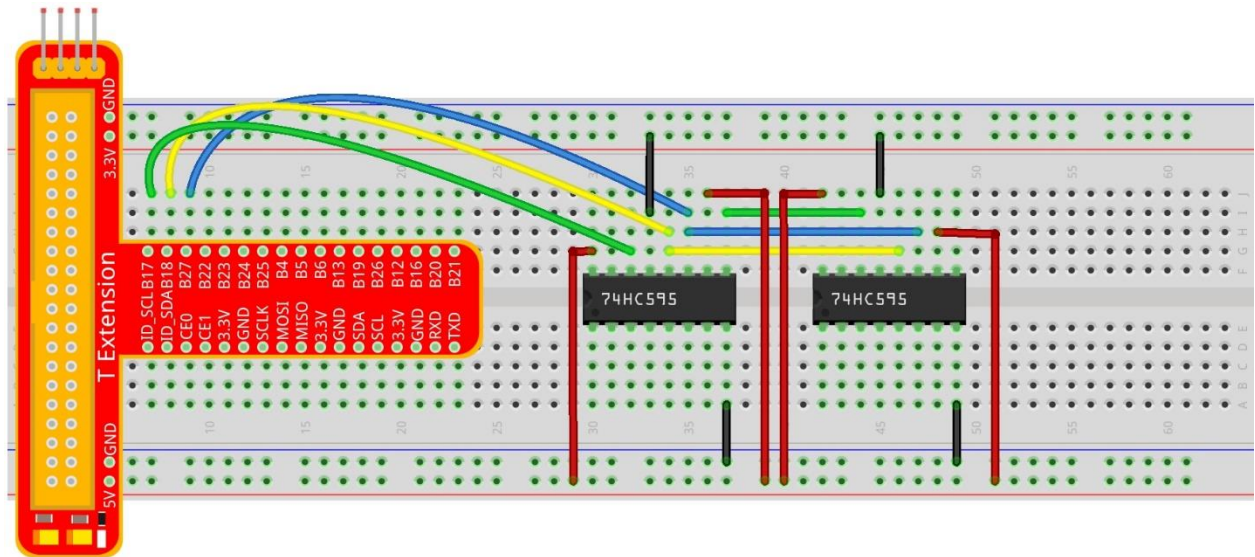


## Experimental Procedures

**Step 1:** Build the circuit. Since the wiring is complicated, let's make it step by step. First, insert the T-Cobbler and two 74HC595 chips into breadboard. Connect the 5V and GND of the T-Cobbler to holes on the two sides of the board, then hook up pin16 and 10 of the two 74HC595 chips to VCC and pin 13 respectively, and pin 8 to GND.

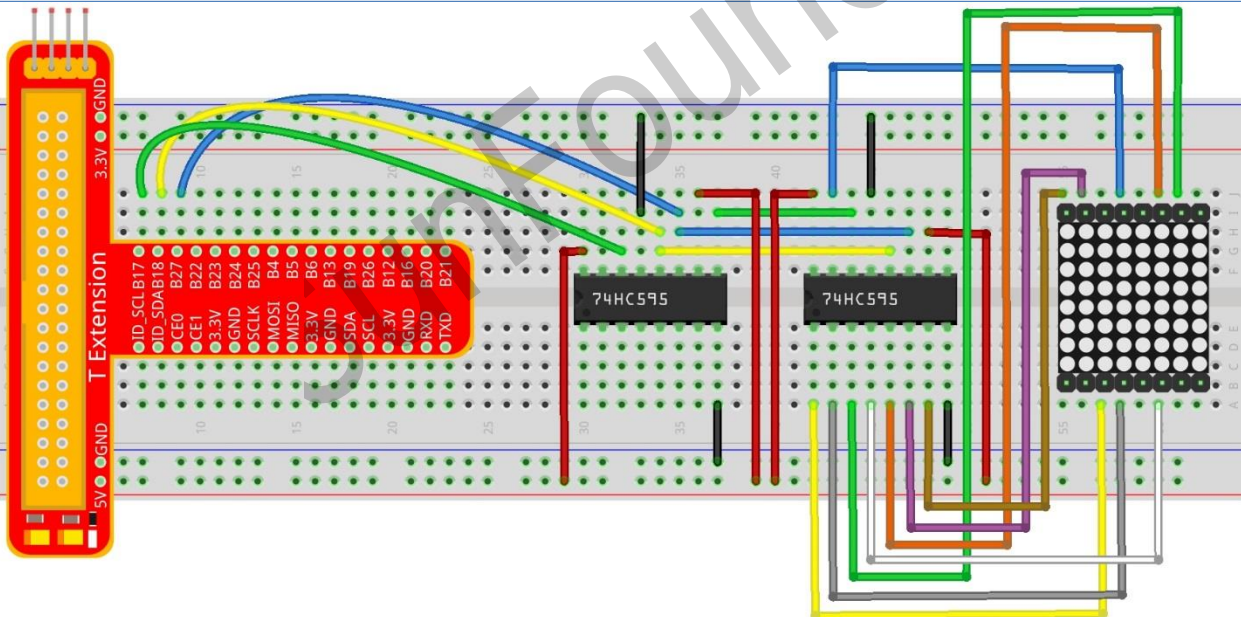


**Step 2:** Connect pin 11 of the two 74HC595 together, and then to GPIO27; then pin 12 of the two chips, and to GPIO18; next, pin 14 of the 74HC595 on the left side to GPIO17 and pin 9 to pin 14 of the other 74HC595.



**Step 3:** Insert the dot matrix onto the breadboard. The 74HC595 on the right side is to control columns of the matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

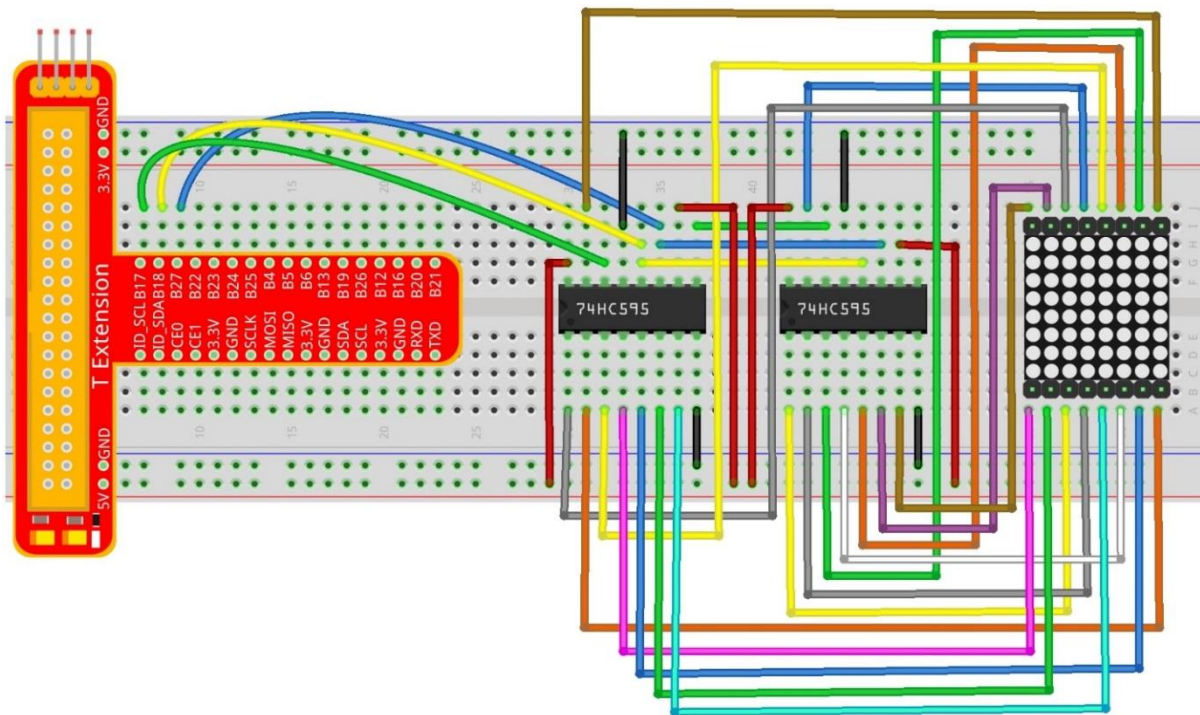
COL	1	2	3	4	5	6	7	8
Pin No.	13	3	4	10	6	11	15	16



**Step 4:** Now connect the ROWs of the dot matrix. The 74HC595 on the left controls ROW of the matrix. See the table below for the mapping. We can see, Q0-Q7 of the 74HC595 on the left are mapped with pin 9, 14, 8, 12, 1, 7, 2, and 5 respectively.

ROW	1	2	3	4	5	6	7	8
Pin No.	9	14	8	12	1	7	2	5





**Note:** PLEASE connect devices correctly. DO NOT wire up insufficiently. DO NOT connect to the wrong side of the dot matrix. In the Fritzing image above, the side with label is at the bottom.

**For C language users:**

**Step 2:** Get into the folder of code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile

```
make 15_dotMatrix
```

**Step 4:** Run

```
sudo ./15_dotMatrix
```

## Code Explanation

```
void hc595_in(unsigned char dat){ // Write an 8-bit data to the shift register of the
74HC595
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i)); // Write the value of dat to pin SDI of
the HC595 bit by bit
        digitalWrite(SRCLK, 1); // Everytime SRCLK adds one, the shift register moves 1
bit
        delay(1);
        digitalWrite(SRCLK, 0);
    }
}
```

```

}
void hc595_out(){    // Update the output data of the 74HC596
    digitalWrite(RCLK, 1); // Everytime RCLK adds 1, the HC595 updates the output.
    delay(1);
    digitalWrite(RCLK, 0);
}
while(1){
    for(i=0;i<sizeof(code_H);i++){ // The data of ROW and COL table for the matrix adds
1 each time.
        hc595_in(code_L[i]); // Write to the first data of the Row table
        hc595_in(code_H[i]); // Write to the first data of the COL table, and the ROW
data previously goes to the other HC595.
        hc595_out(); // Update the output of the 74HC595; output the data controlled
by both two HC595, and the dot matrix will show the pattern.
        delay(100);
    }
    for(i=sizeof(code_H);i>=0;i--){ // The data of ROW and COL table for the matrix
decreases by 1 each time.
        hc595_in(code_L[i]); // Write to the first data of the Row table
        hc595_in(code_H[i]); // Write to the first data of the COL table, and the ROW
data previously goes to the other HC595.
        hc595_out(); // Update the output of the 74HC595; output the data controlled
by both two HC595, and the dot matrix will show the pattern.
        delay(100);
    }
}
}

```

### For Python users:

**Step 2:** Get into the folder of code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 15_DotMatrix.py
```

### Code Explanation

```

# We use a Common Anode matrix, so ROW pins are the common anode, and COL, the common
cathode.

# row and column lists. When characters are displayed, an element in row and one in
column are acquired and assigned to the two HC595 chips respectively. Thus a pattern is
shown on the matrix.

```



```

# ROW  ++++
code_H =
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0x
ff,0xff,0xff]

# COL  ----
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,0xef,0x
df,0xbf,0x7f]

def get_matrix(row_buffer, col_buffer, max_row=8, max_col=8): # The functions is to
print the pattern on the matrix by the 2D array on the command line interface (CLI).
    matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)] # Initialize a 2D
array
    print "row_buffer = 0x%02x , col_buffer = 0x%02x"%(row_buffer, col_buffer)
    for row_num in xrange(0,8):
        for col_num in xrange(0,8):
            if (((row_buffer >> row_num) & 0x01) - ((col_buffer >> col_num) & 0x01)): #
for Common Anode type matrix, when row is High and column is low, the LED will light up.
                matrix_msg[row_num][col_num] = 1 # To turn on an LED at a certain row
and column, assign 1 to the corresponding elements in the 2D array
            print_matrix(matrix_msg) # Print the 2D array on the CLI
            matrix_msg = [[0 for i in range(max_row)] for i in range(max_col)] # Reset the array
after one print

def hc595_shift(dat): # Shift the data to 74HC595
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit)) # Write the value of dat bit by bit to pin
SDI of the HC595
        GPIO.output(SRCLK, GPIO.HIGH) # Everytime SRCLK is High, the shift register
shifts one bit
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
        GPIO.output(RCLK, GPIO.HIGH) # Everytime RCLK is high, HC595 updates its output.
        time.sleep(0.001)
        GPIO.output(RCLK, GPIO.LOW)

def main():
    print_msg()
    while True:

```

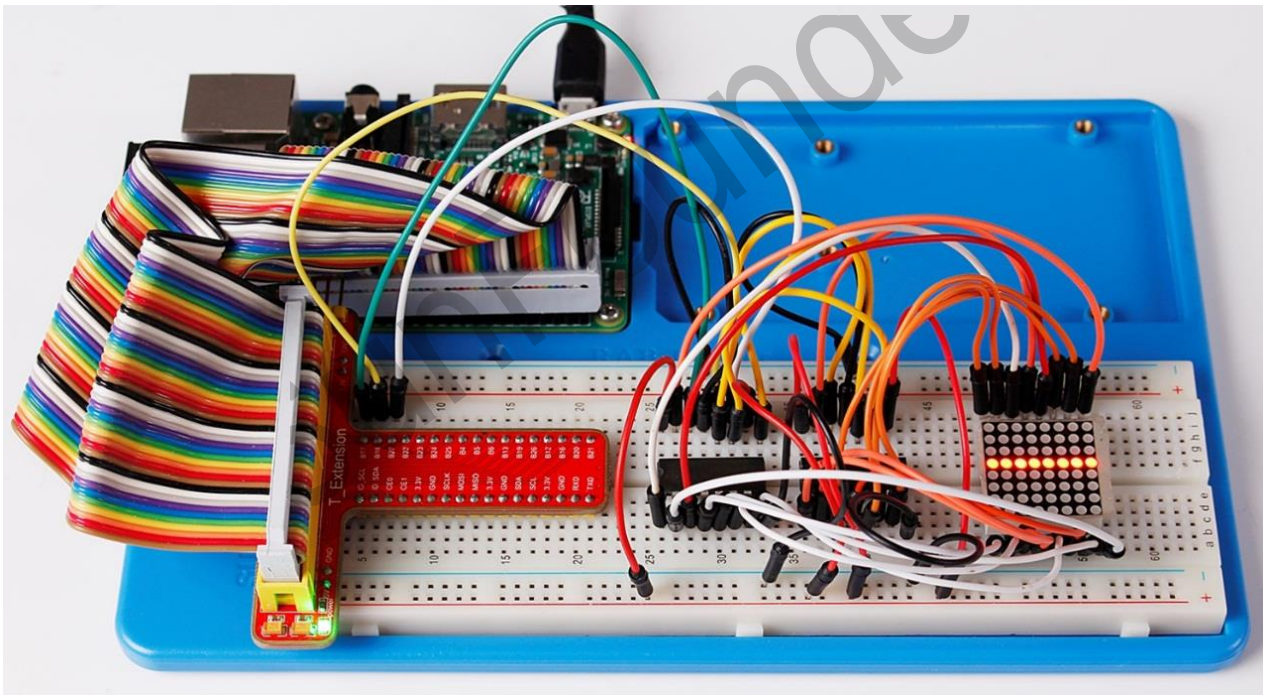
```

for i in range(0, len(code_H)): # Assign elements of the column table in
sequence
    hc595_shift(code_L[i]) # Write to the first data of the Row table
    hc595_shift(code_H[i]) # Write to the first data of the COL table, and the
ROW data previously goes to the other HC595.
    get_matrix(code_L[i], code_H[i]) # Print the 2D array on the CLI
    time.sleep(0.1)

for i in range(len(code_H)-1, -1, -1): # Assign elements of the column table
in inverse order
    hc595_shift(code_L[i])
    hc595_shift(code_H[i])
    get_matrix(code_L[i], code_H[i])
    time.sleep(0.1)

```

You should see LEDs light up as you control.



## Summary

Through this lesson, you have got the basic principle of LED dot matrix and how to program the Raspberry Pi to drive an LED dot matrix based on 74HC595 cascade. With the knowledge learnt, try more fascinating creations!

## Further Exploration

If you want to display characters on the matrix, please refer to a python code: [https://github.com/sunfounder/SunFounder\\_Dot\\_Matrix](https://github.com/sunfounder/SunFounder_Dot_Matrix).

# Lesson 16 LCD1602

## Introduction

In this lesson, we will learn how to use LCD1602 to display characters and strings.



## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* LCD1602
- 1 \* Potentiometer
- Jumper wires

## Principle

### LCD1602

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 SunFounder Uno board 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.

LCD1602 uses the standard 16-pin port, among which:

**Pin 1 (GND):** connected to Ground

**Pin 2 (Vcc):** connected to 5V power supply

**Pin 3 (Vo):** used to adjust the contrast of LCD1602; the level is lowest when it's connected to a positive power supply, and highest when connected to ground (you can connect a 10K potentiometer to adjust its contrast when using LCD1602)

**Pin 4 (RS):** register select pin, controlling 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, where the LCD's controller looks for instructions on what to do next.

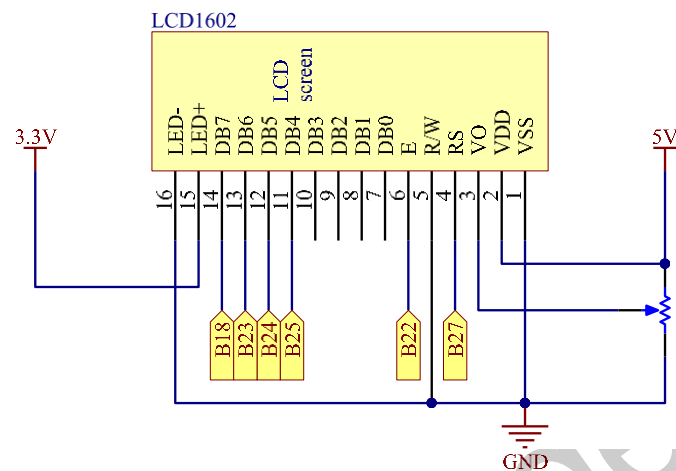
**Pin 5 (R/W):** to read/write signals; it reads signals when supplied with high level (1), and writes when low level (0) (in this experiment, you only need to write data to LCD1602, so just connect this pin to ground)

**Pin 6 (E):** An enable pin that, when low-level energy is supplied, causes the LCD module to execute relevant instructions

**Pin 7 (D0-D7):** pins that read and write data

**A and K:** controlling LCD backlight; K connects to GND, and A to 3.3V. Turn the backlight on and you can see the characters displayed clear in a dim environment

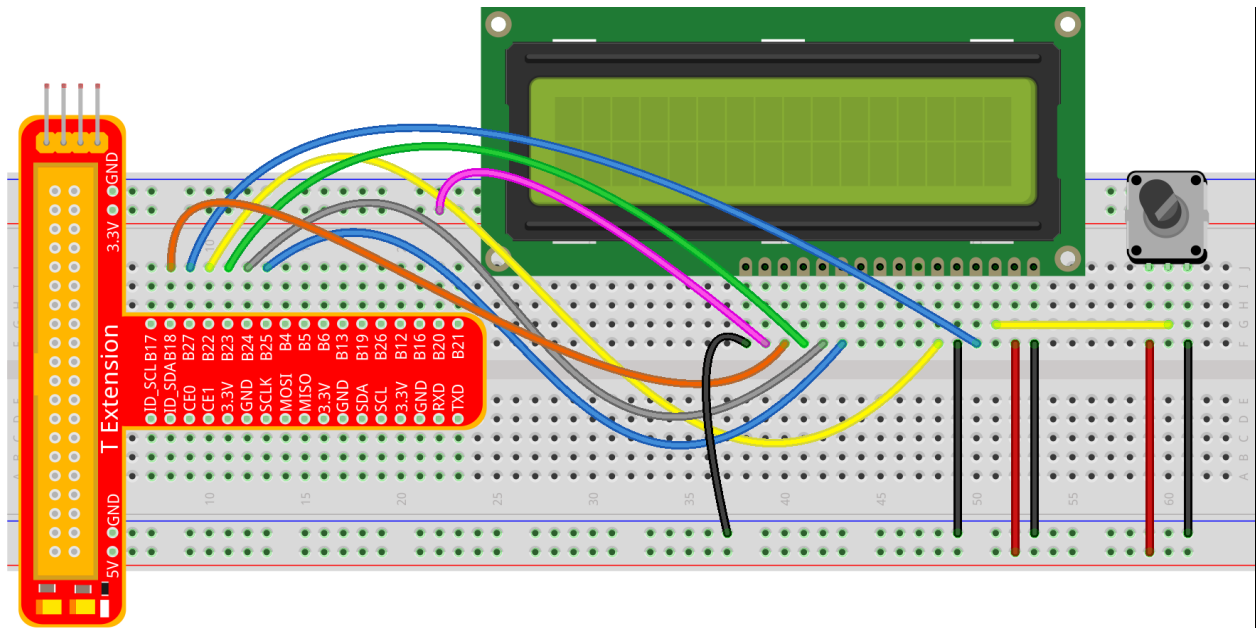
LCD1602 has two operation modes: 4-bit and 8-bit. When the IOs of the MCU are insufficient, you can choose the 4-bit mode, under which only pins D4~D7 are used. After connecting the circuit, you can operate LCD1602 by the Raspberry Pi.



### Experimental Procedures

**Step 1:** Build the circuit (please be sure the pins are connected correctly. Otherwise, characters will not be displayed properly):

LCD1602	T-Extension Board
VDD	5V
VSS	GND
OV	Connect to the middle pin of potentiometer
RS	B27
R/W	GND
E	B22
D0-D3	Not connected
D4	B25
D5	B24
D6	B23
D7	B18
A	3.3V
K	GND



**Note:** After you run the code, characters may not appear on the LCD1602. You need to adjust the contrast of the screen (the gradual change from black to white) by spinning the potentiometer clockwise or anticlockwise, until the screen displays characters clearly.

### For C language users:

**Step 2:** Get into the folder of code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

**Step 3:** Compile

```
make 16_lcd1602
```

**Step 4:** Run

```
sudo ./16_lcd1602
```

### Code Explanation

```
#include <lcd.h> // includes the lcd library, containing some functions for the LCD1602
display for convenient use

const unsigned char Buf[] = "---SUNFOUNDER---"; // An array to store the characters to
be displayed on the LCD1602

const unsigned char myBuf[] = " sunfounder.com"; // Another array to store the
characters

fd = lcdInit(2,16,4, 2,3, 6,5,4,1,0,0,0,0); // Initialize the LCD display, see
/usr/local/include/lcd.h

// lcdInit(rows, cols, bits, rs, strb, d0, d1, d2, d3, d4, d5, d6, d7) - LCD1602 shows
2 rows and 16 columns. If the initialization succeeds, it will return True.

lcdClear(fd); // Clear the screen
```

```

lcdPosition(fd, 0, 0); // Locate the position of the cursor at Row 0 and Col 0 (in fact
it's the first line and first column)
lcdPuts(fd, "Welcom To--->"); // Display the character "Welcom To--->" on the LCD1602
lcdPosition(fd, 0, 1); // Place the cursor at Col 0, Row 0.
lcdPuts(fd, " sunfounder.com");
while(1){
    lcdClear(fd);
    for(i=0; i<16; i++){ // i adds one in the loop. i means the number of columns, so i
adds to 16 at most.
        lcdPosition(fd, i, 0); // Place the cursor at the first row, and moves left to
right from the first character
        lcdPutchar(fd, *(myBuf+i)); // *(myBuf+i) is a pointer that points to contents in
the myBuf[] array, and output the pointed data to lcd
        delay(100);
    }
    for(i=0; i<sizeof(Buf)-1; i++){
        lcdPosition(fd, i, 1); // Place the cursor at the second row, moves from the
first character
        lcdPutchar(fd, *(Buf+i)); // A pointer that points to data in the Buf[] array;
output it to lcd
        delay(200);
    }
    sleep(0.5);
}

```

## For Python users

**Step 2:** Get into the folder of code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 16_lcd1602.py
```

## Code Explanation

```

class LCD: # Write an LCD class
    def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO = None):
# Initialization function for the class, run when an object is created of the class. A
parameter needs to be transferred to the object when it's created; otherwise, the
default value in __init__ will be assigned.

```



```

        self.used_gpio = self.pins_db[:] # Note down the used gpio to easily clear
IO setting after the stop. pins_db[:] writes all in the pins_db list to the used_gpio
list; if here use used_gpio = self.pins_db, it means used_gpio call pins_db, in other
words, any change of pins_db will affect used_gpio.

```

```

        self.used_gpio.append(pin_e)
        self.used_gpio.append(pin_rs)
        self.write4bits(0x33) # initialization
        self.write4bits(0x32) # initialization
        self.write4bits(0x28) # 2 line 5x7 matrix
        self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor
        self.write4bits(0x06) # shift cursor right
        """ Initialize to default text direction (for romance languages) """
        self.displaymode = self.LCD_ENTRYLEFT | self.LCD_ENTRYSHIFTDECREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode) # Set the entry
mode

```

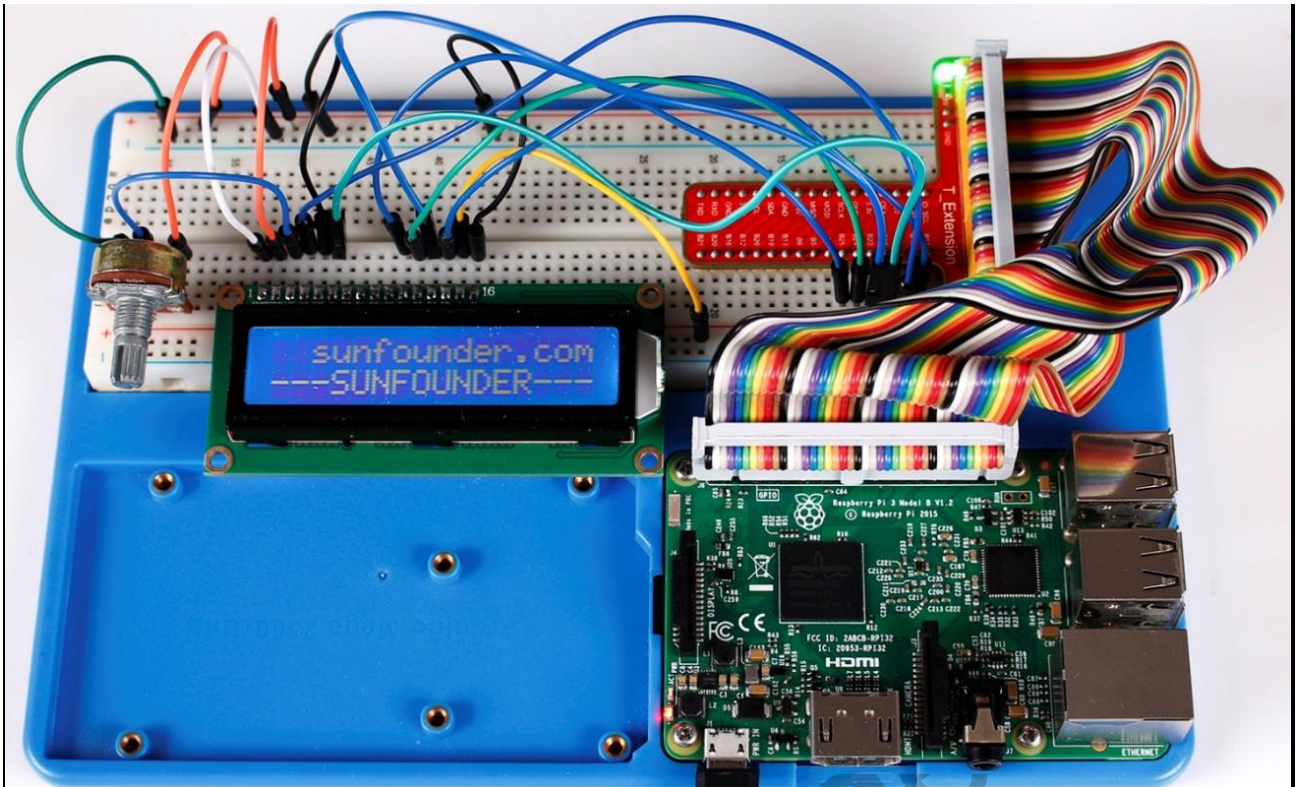
```

def begin(self, cols, lines): # Start the LCD
def setCursor(self, col, row): # Set the cursor location
def message(self, text): # Send strings to the LCD. The new line wraps to the
second line
def destroy(self): # Clean up the used gpio

lcd = LCD(0, 2) # Create an lcd object
lcd.clear() # Clear the LCD display
for i in range(0, len(line0)): # i adds 1 each time within the length of the
character line0
    lcd.setCursor(i, 0) # Locate the cursor at character No. i, Row 0
    lcd.message(line0[i]) # Display the character on the screen
    sleep(0.1)
for i in range(0, len(line1)): # i adds 1 each time within the length of the
character line0
    lcd.setCursor(i, 1) # Locate the cursor at character No. i, Row 1
    lcd.message(line1[i]) # Display the character on the LCD

```

You should see two lines of characters displayed on the LCD1602: " **Welcome to ---> "** , "**sunfounder.com** " and "**---SUNFOUNDER---** ".



### Further Exploration

In this experiment, the LCD1602 is driven in the 4-bit mode. You can try programming by yourself to drive it in the 8-bit mode.

# Lesson 17 ADXL345

## Introduction

In this lesson, we will learn how to use the acceleration sensor ADXL345.

## Components

- 1 \* Raspberry Pi
- 1 \* Breadboard
- 1 \* ADXL345 module
- Jumper wires

## Principle

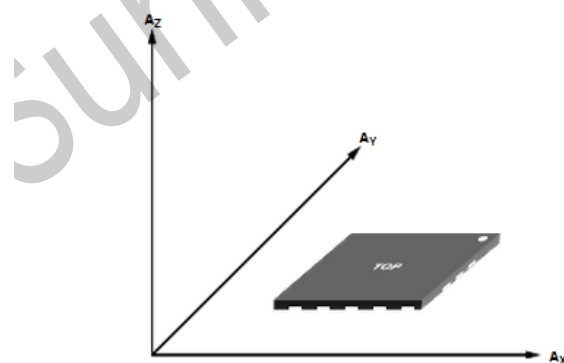
### ADXL345

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to  $\pm 16$  g. Digital output data is formatted as 16-bit two's complement and is accessible through either an SPI (3- or 4-wire) or I2C digital interface.

The ADXL345 is well suited to measure the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables the inclination change measurement by less than  $1.0^\circ$ . And the excellent sensitivity (3.9mg/LSB @2g) provides a high-precision output of up to  $\pm 16$ g.

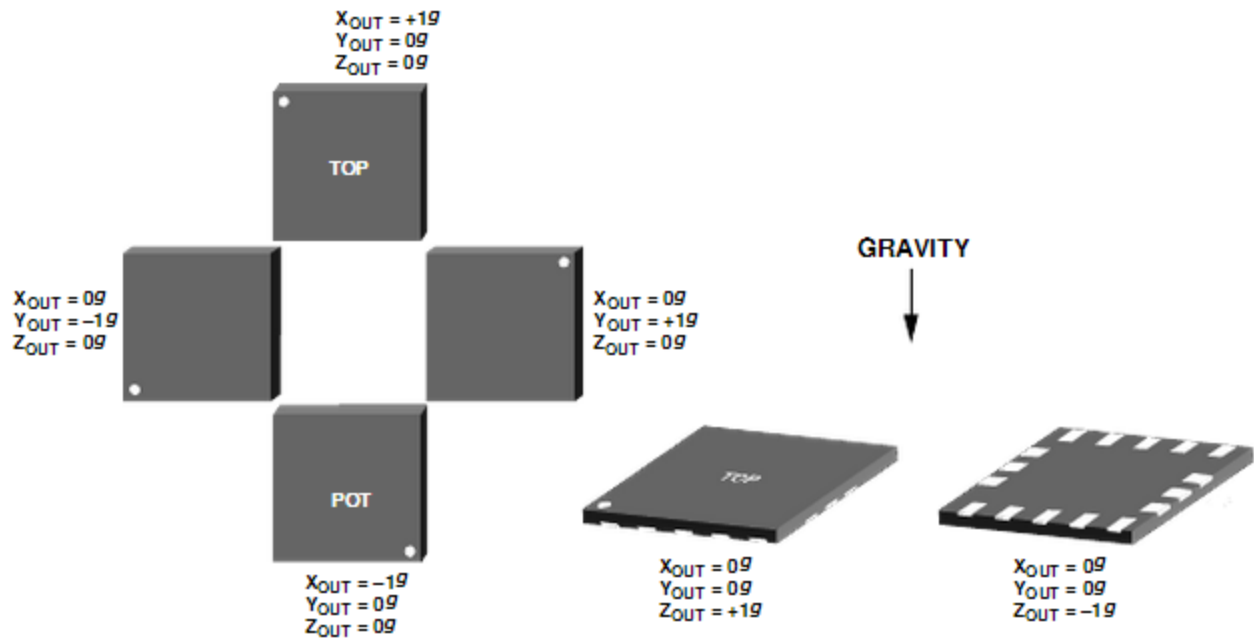
In this experiment, I2C digital interface is used.

ADXL345 works like this:



Axes of detection by ADXL345

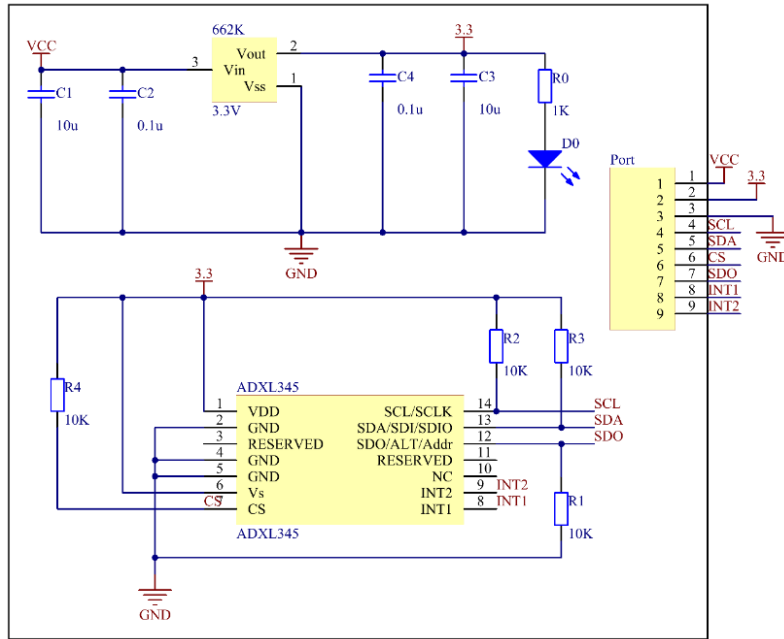
When you place the module face up, **Z\_OUT** is at the maximum which is +1g; face down, **Z\_OUT** is at the minimum. No matter of face, as long as it's placed on a level surface, **X\_OUT** increases along the **Ax** axis direction, so does **Y\_OUT** along the **Ay** axis. See the picture below. Thus, when you rotate the module, you can see the changes of **X\_OUT**, **Y\_OUT**, and **Z\_OUT**.



Relationship between output and gravity direction

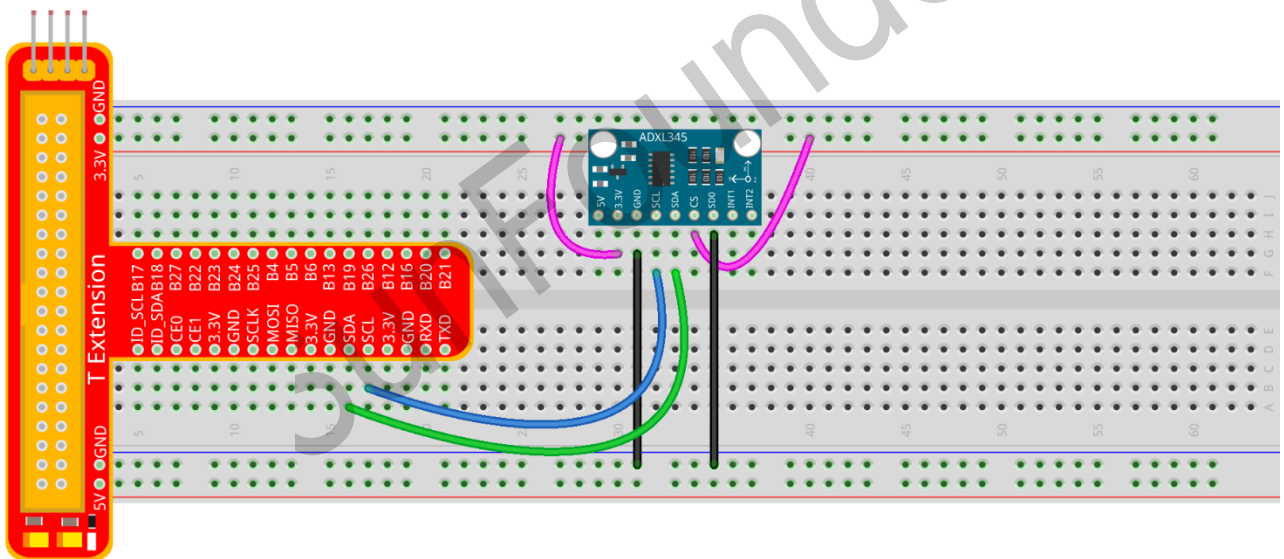
Pin Function of ADXL345 Module:

Name	Description
VS	Supply Voltage
CS	Chip Select; I2C mode is enabled if it's tie-high to VDD I/O (VDD I/O = 1.8V).
SDO	Serial Data Out, alternate I2C address select
INT1	Interrupt 1 Output
INT2	Interrupt 2 Output
3.3V	3.3V
SDA	Serial Data (I2C), Serial Data In (SPI 4-Wire), Serial Data In/Out (SPI 3-Wire)
SCL	Serial Communications Clock
GND	GND



## Experimental Procedures

### Step 1: Build the circuit



The I2C interface is used in the following program. Before running the program, please make sure the I2C driver module of Raspberry Pi has loaded normally.

### For C language users:

#### Step 2: Get into the folder of code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/C
```

#### Step 3: Compile the Code

```
make 17_adxl345
```

#### Step 4: Run

```
sudo ./17_adxl345
```

## Code Explanation

```
#include <wiringPiI2C.h> // Include functions and method for the IIC protocol
#define DevAddr 0x53 // device address
struct acc_dat{ // a struct variable to store the value of x, y, and z
    int x;
    int y;
    int z;
};
fd = wiringPiI2CSetup(DevAddr); // This initialises the I2C system with your given
device identifier
void adxl345_init(int fd){ // Initialize the device by i2c
    wiringPiI2CWriteReg8(fd, 0x31, 0x0b); // These write an 8-bit data value into the
device register indicated.
    wiringPiI2CWriteReg8(fd, 0x2d, 0x08); // Write 0x08 to the address (0x21) of the i2c
device
    ... ..
}
struct acc_dat adxl345_read_xyz(int fd){ // a struct function, returning a struct value
    char x0, y0, z0, x1, y1, z1;
    struct acc_dat acc_xyz;
    x0 = 0xff - wiringPiI2CReadReg8(fd, 0x32); // These read an 8- or 16-bit value from
the device register indicated.
    x1 = 0xff - wiringPiI2CReadReg8(fd, 0x33); // Read an 8-bit data from the 0x33
register of the I2C device fd, assign to x1
    y0 = 0xff - wiringPiI2CReadReg8(fd, 0x34);
    y1 = 0xff - wiringPiI2CReadReg8(fd, 0x35);
    z0 = 0xff - wiringPiI2CReadReg8(fd, 0x36);
    z1 = 0xff - wiringPiI2CReadReg8(fd, 0x37);
    printf(" x0 = %d ",x0);printf("x1 = %d \n",x1);
    printf(" y0 = %d ",y0);printf("y1 = %d \n",y1);
    printf(" z0 = %d ",z0);printf("z1 = %d \n",z1);
    acc_xyz.x = (int)(x1 << 8) + (int)x0; // Assign values to members of the struct; the
value of x consists of x1 (high 8 bits) and x0 (low 8 bits).
    acc_xyz.y = (int)(y1 << 8) + (int)y0;
    acc_xyz.z = (int)(z1 << 8) + (int)z0;
    if(acc_xyz.x > 32767){ // Set the value of x as no more than 0x7FFF
        acc_xyz.x -= 65536;
    }
}
```



```

if(acc_xyz.y > 32767){ // Set the value of y as no more than 0x7FFF
    acc_xyz.y -= 65536;
}
if(acc_xyz.z > 32767){
    acc_xyz.z -= 65536;
}
return acc_xyz; // The function ends, return to the acc_xyz struct
}
acc_xyz = adxl345_read_xyz(fd); // Call the function to read the data collected by the
accelerometer module
printf("x: %05d y: %05d z: %05d\n", acc_xyz.x, acc_xyz.y, acc_xyz.z); // Print the
data collected by the accelerometer; %05d means the printed data is a 5-bit one, and the
empty bit will be replaced by 0.

```

### For Python users:

**Step 2:** Get into the folder of the code

```
cd /home/pi/SunFounder_Super_Kit_V3.0_for_Raspberry_Pi/Python
```

**Step 3:** Run

```
sudo python 17_ADXL345.py
```

### Code Explanation

```

class ADXL345(I2C):    # Define a class ADXL345, and the class inheritance is I2C
    def __init__(self, busnum=-1, debug=False): # The initialize function of the
class, which is run when an instance is created of the class
    def setRange(self, range): # Read the data format register to preserve bits.
Update the data rate, make sure that the FULL-RES bit is enabled for range scaling
    def getRange(self): # Read an 8-bit data from the device register
    def setDataRate(self, dataRate): # Note: The LOW_POWER bits are currently
ignored; we always keep the device in 'normal' mode
    def getDataRate(self): # get the rate from the register
    def read(self): # Read data from the accelerometer
        raw = self.accel.readList(self.ADXL345_REG_DATA0, 6) # Read 6 values from
the register, respectively equal to the high and low bits of the x, y, and z value
        print raw
        res = []
        for i in range(0, 6, 2):
            g = raw[i] | (raw[i+1] << 8) # Combine the high 8 bits and low 8 bits
and obtain a measurement value

```

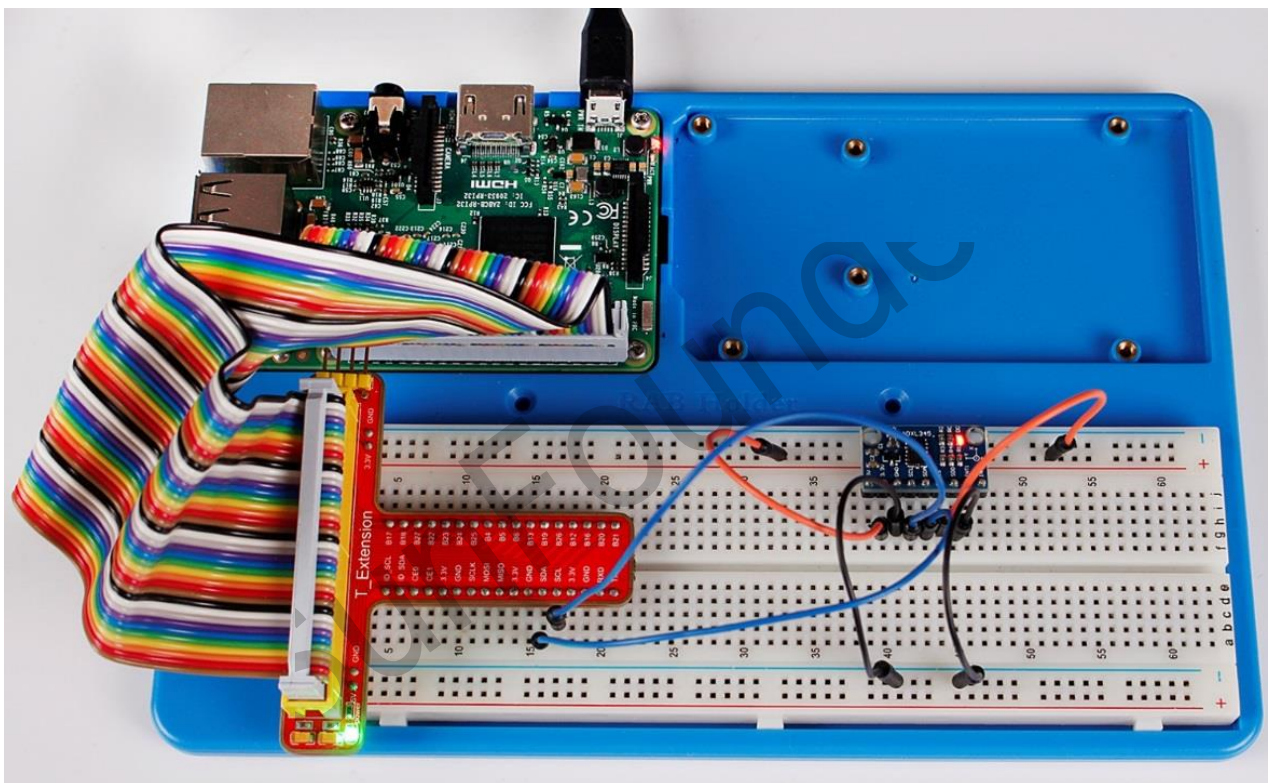
```

        if g > 32767:
            g -= 65535
        res.append(g)
    return res

accel = ADXL345() # Create an instance accel of class ADXL345
x, y, z = accel.read() # accel calls itself to measure x, y, and z and store
                        # them in a list. Then assign the values measured to x, y, and z.

```

Now, rotate the acceleration sensor, and you should see the values printed on the screen change.



## 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.