

# Preface

## About SunFounder

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

## About the Da Vinci Kit

This Da Vinci kit applies to the Raspberry Pi 4 Model B, 3 Model A+, 3 Model B+, 3 Model B, 2 Model B, 1 Model B+, 1 Model A+, zero W and zero. It includes various components and chips that can help to create various interesting phenomena which you can get via some operation with the guidance of experiment instructions. In this process, you can learn some basic knowledge about programming. Also you can explore more application by yourself. Now go for it!

## Free Support



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



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

# Contents

Component List.....	1
Introduction.....	5
What Do We Need?.....	6
Required Components.....	6
Preparation.....	8
If You Have A Screen.....	8
If You Have No Screen.....	14
Required Components.....	14
Burn System.....	14
Connect the Raspberry Pi to the Internet.....	15
Start SSH.....	17
Get the IP Address.....	17
Use the SSH Remote Control.....	17
For Linux or/Mac OS X Users.....	18
For Windows Users.....	20
Remote Desktop.....	21
VNC.....	21
XRDP.....	26
Libraries.....	29
RPi.GPIO.....	29
WiringPi.....	30
Raspberry Pi GPIO Extension Board.....	31
Download the Code.....	32
1 Output.....	33
1.1 Displays.....	33
1.1.1 Blinking LED.....	33
1.1.2 RGB LED.....	48
1.1.3 LED Bar Graph.....	58
1.1.4 7-segment Display.....	65
1.1.5 4-Digit 7-Segment Display.....	76
1.1.6 LED Dot Matrix.....	91
1.1.7 I2C LCD1602.....	105
1.2 Sound.....	111
1.2.1 Active Buzzer.....	111

1.2.2 Passive Buzzer.....	118
1.3 Drivers.....	127
1.3.1 Motor.....	127
1.3.2 Servo.....	138
1.3.3 Stepper Motor.....	147
2 Input.....	158
2.1 Controllers.....	158
2.1.1 Button.....	158
2.1.2 Slide Switch.....	166
2.1.3 Relay.....	174
2.1.4 Potentiometer.....	182
2.1.5 Keypad.....	191
2.1.6 Joystick.....	199
2.2 Sensors.....	208
2.2.1 Photoresistor.....	208
2.2.2 Thermistor.....	215
2.2.3 DHT-11.....	223
2.2.4 PIR.....	231
2.2.5 Ultrasonic Sensor Module.....	237
2.2.6 MPU6050 Module.....	246
2.2.7 MFRC522 RFID Module.....	254
2.2.8 Tilt Switch.....	272
3 Extension.....	280
3.1 Application.....	280
3.1.1 Reversing Alarm.....	280
3.1.2 Traffic Light.....	292
3.1.3 Password Lock.....	302
3.1.4 Welcome.....	314
Appendix.....	325
I2C Configuration.....	325
SPI Configuration.....	329

# Component List

**Resistor (220R)**

20 pcs



**Resistor (1K)**

10 pcs



**Resistor (10K)**

10 pcs



**1N4007 Diode**

2 pcs



**Zener Diode**

2 pcs



**Green LED**

4 pcs



**Red LED**

10 pcs



**Yellow LED**

4 pcs



**Blue LED**

4 pcs



**RGB LED**

1 pcs



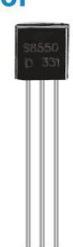
**S8050 Transistor**

4 pcs



**S8550 Transistor**

4 pcs



**Capacitor 0.1 uF**

4 pcs



**Capacitor 10uF**

4 pcs



**Photoresistor**

1 pcs



**Thermistor**

1 pcs



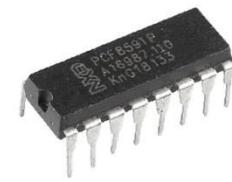
**L293D**

1 pcs



**PCF8591**

1 pcs



**74HC595**

2 pcs



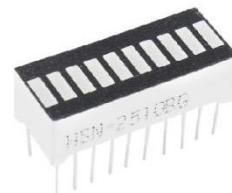
**7-segment Display**

1 pcs



**LED Bar Graph**

1 pcs



**LED Matrix**

1 pcs



**Active Buzzer**

2 pcs



**4-Digit 7-segment Display**

1 pcs



**Tilt Switch**

1 pcs



**Potentiometer**

3 pcs



**Passive Buzzer**

1 pcs



**Button**

4 pcs



**Motor**

1 pcs



**Slide Switch**

2 pcs



**Keypad**

1 pcs



**I2C LCD 1602**

1 pcs



**9G Servo**

1 pcs



**Breadboard Power Module**

1 pcs



**MFRC522 RFID Module**

1 pcs



**Relay**

1 pcs



**Stepping Motor Driver**

1 pcs



**Stepping Motor**

1 pcs



**Ultrasonic Ranging Module**

1 pcs



**Joystick**

1 pcs



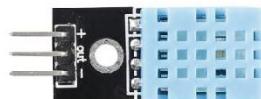
**Infrare Motion Sensor**

1 pcs



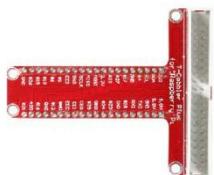
**DHT-11**

1 pcs



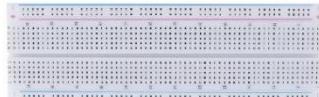
**T-shape Extension Board**

1 pcs



**Breadboard**

1 pcs



**MPU6050 Module**

1 pcs



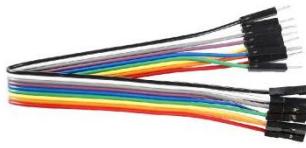
**40 Pin GPIO Cable**

1 pcs



**Jump Wire F/M**

10 pcs



**9V Battery Cable**

1 pcs



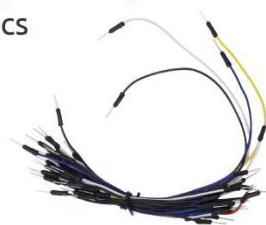
**Jump Wire F/F**

10 pcs



**Jump Wire M/M**

65 pcs



**Fan**

1 pcs



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

# Introduction

Da Vinci Kit is a basic kit suitable to intelligent beginners who have project schedule. It contains 26 commonly used input and output components and modules and a number of basic electronic devices (such as resistors, capacitors) which can provide powerful assistance in your programming learning.

In the light of the kit, you can learn some basic knowledge on Raspberry Pi, including the installation method of Raspberry Pi, knowledge of Bash shell and GPIO. Having understood these knowledge, you can start programming.

If you have no knowledge background of hardware, this document about the Kit provides you with 30 lessons for reference and learning, including 26 basic I/o lessons and 4 simple practical examples. It should be noted that the arrangement of these courses is not based on the degree of difficulty, but on the functions in practice. You can find corresponding courses in accordance with your needs. In other words, even if you haven't finished reading the entire course or mastered the use of the components mentioned, this document will play an important role in guiding you to complete practical projects in the future.

We are looking forward to your projects and hope that you can share your achievements or creation on our forum while reading this document.

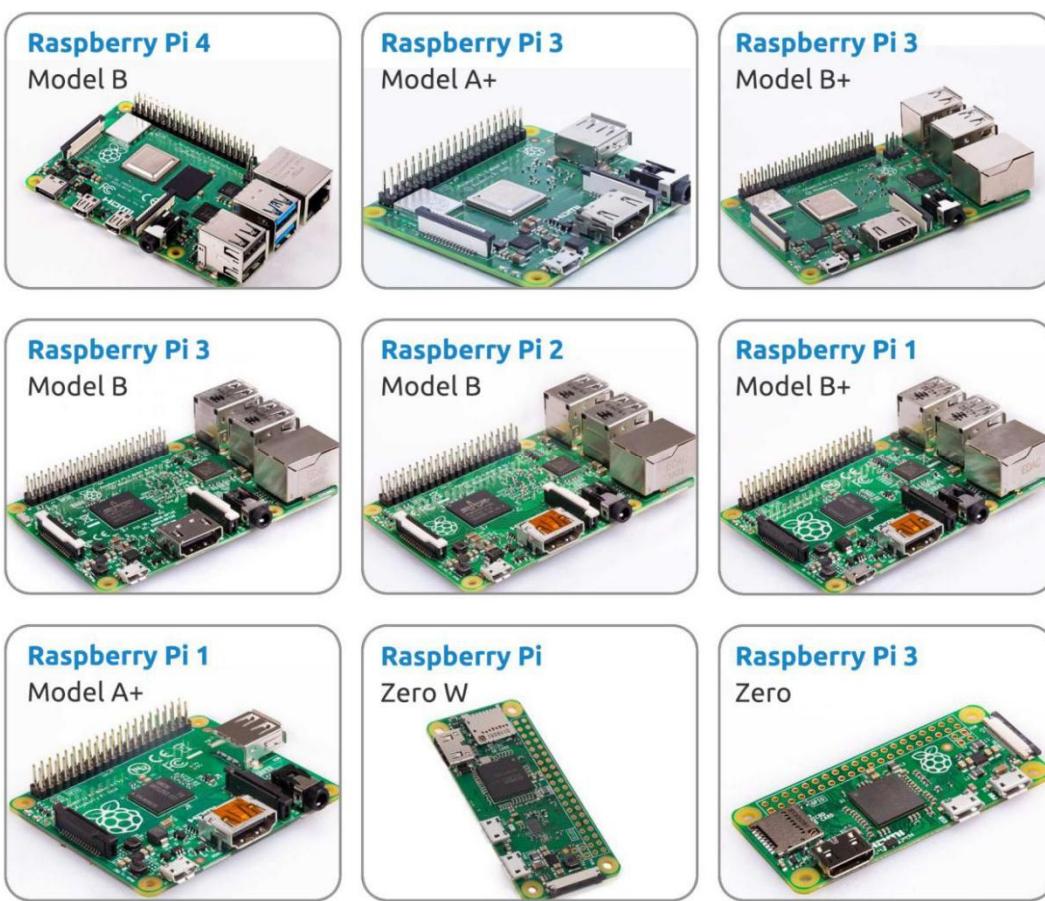
# What Do We Need?

## Required Components

### Raspberry Pi

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

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



### Power Adapter

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

### Micro SD Card

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

## Optional Components

### Screen

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

### Mouse & Keyboard

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

### HDMI

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

### Case

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

### Sound or Earphone

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

# Preparation

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

## If You Have A Screen

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

## Required Components

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

## Procedures

### Step 1

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

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



### Step 2

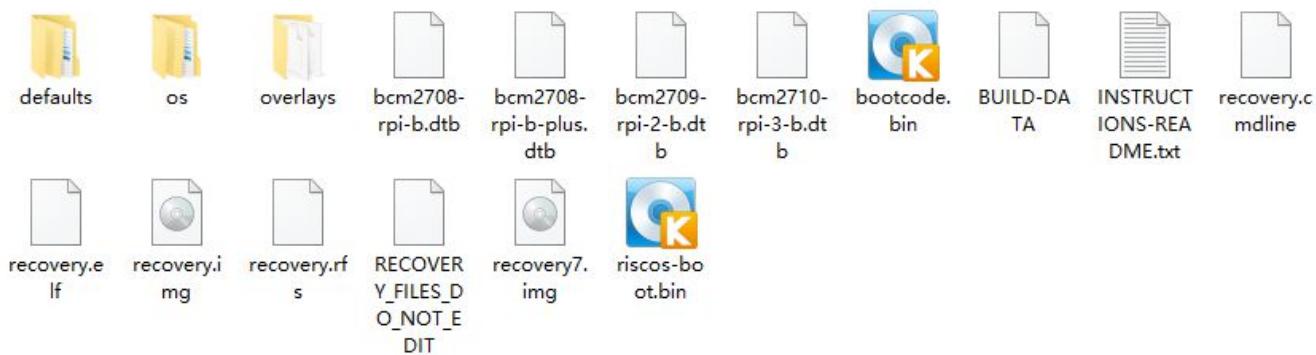
Format the SD card. If there are some important files in the SD card of Raspbian, please backup them first.

## Step 3

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

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

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

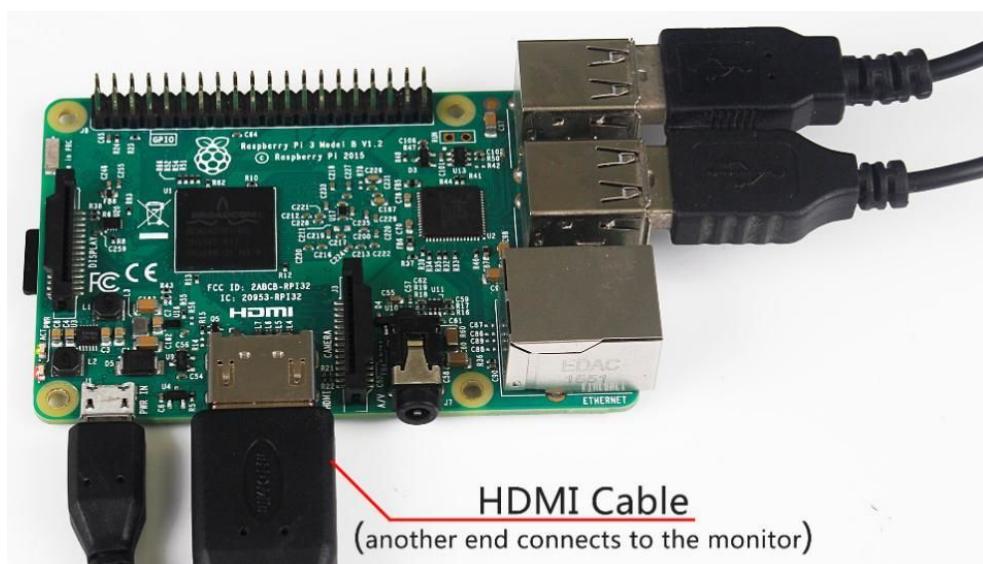


## Step 4

All the files transferred, the SD card pops up.

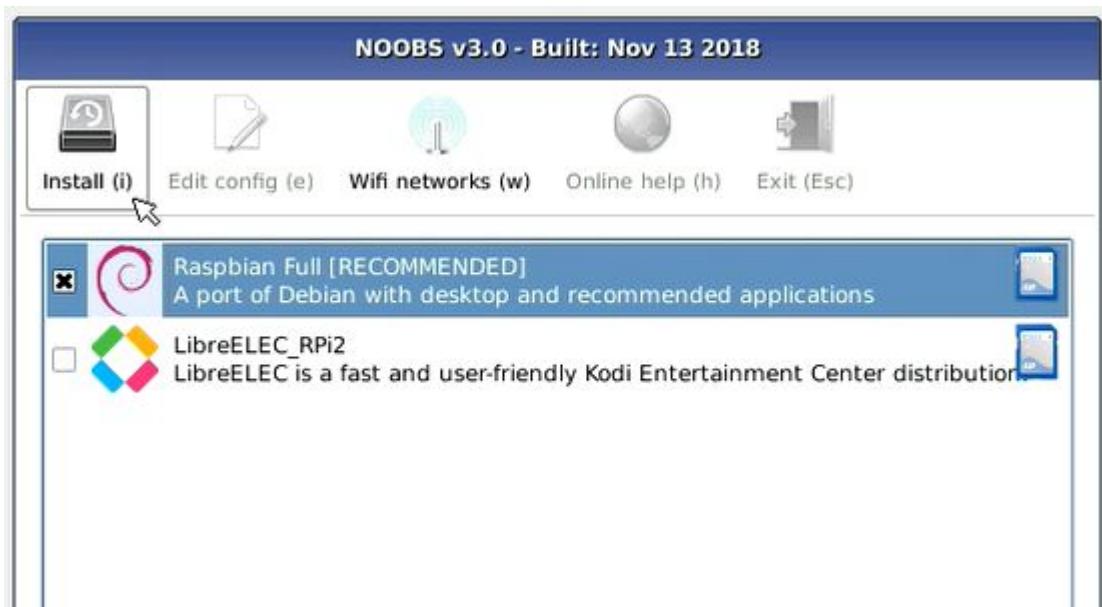
## Step 5

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



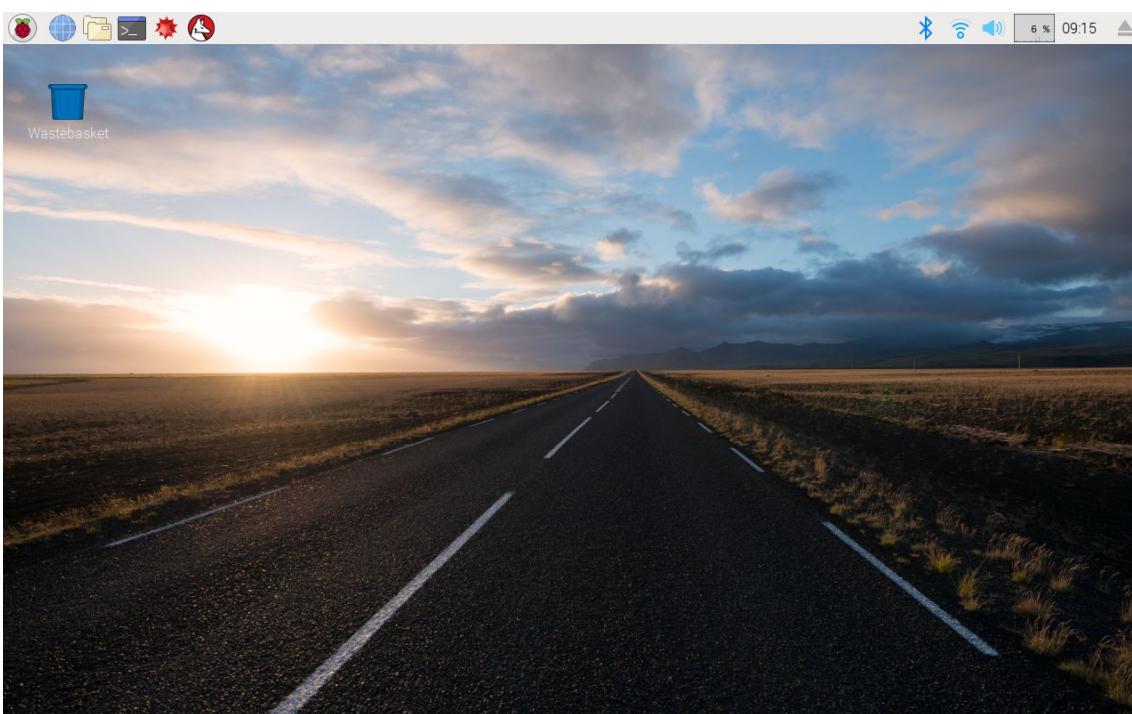
## Step 6

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



## Step 7

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



## Step 8

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



## Step 9

Set country/region, language and time zone, and then click "next" again.



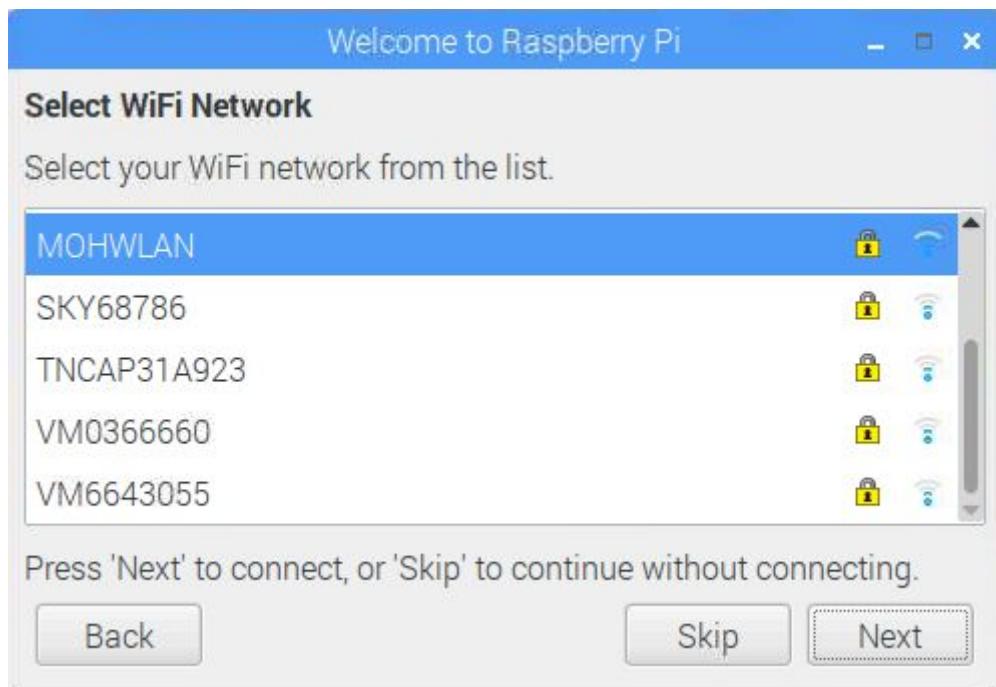
## Step 10

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



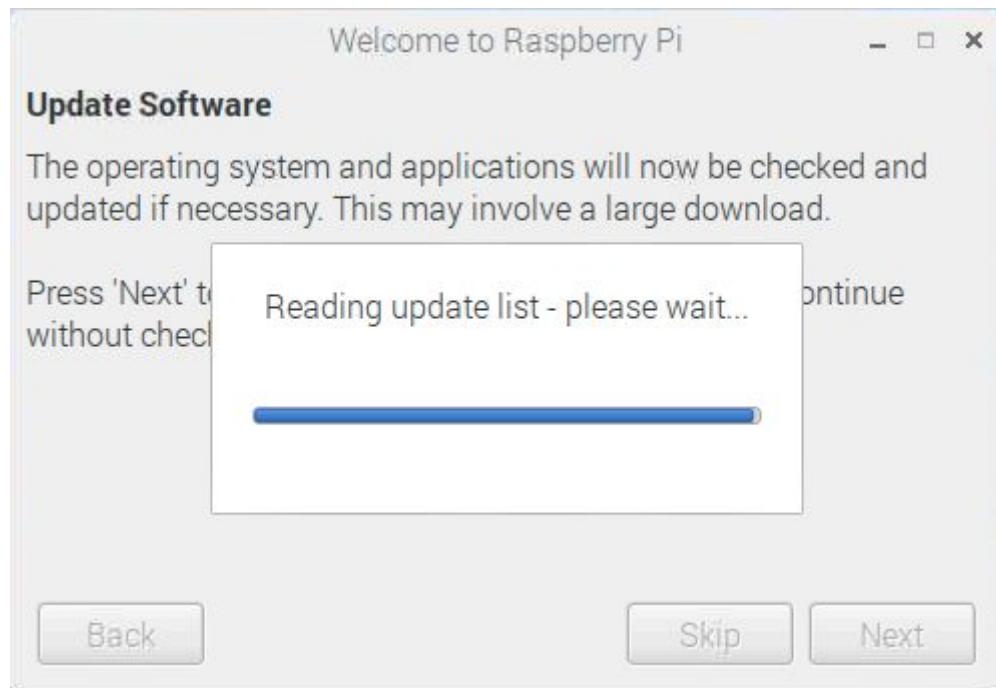
## Step 11

Connect the Raspberry Pi to WIFI and click "Next".



## Step 12

Retrieve update.



## Step 13

Click "Done" to complete the Settings.



Now we can run the Raspberry Pi.

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

## If You Have No Screen

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

## Required Components

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

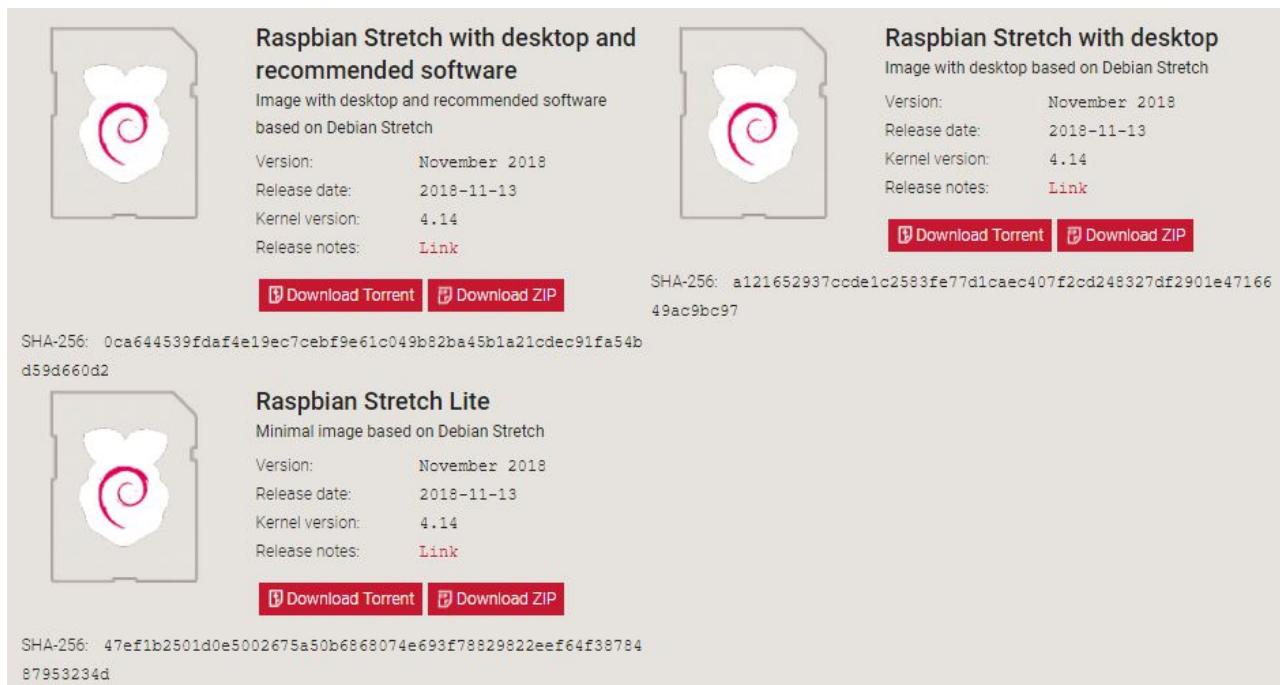
## Burn System

### Step 1

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

### Step 2

Download the complete image on the official website by clicking this link: <https://www.raspberrypi.org/downloads/raspbian/>. There are three different kinds of Raspbian Stretches available, among which the Raspbian Stretch with desktop will be the best choice if you have no other special requirements.



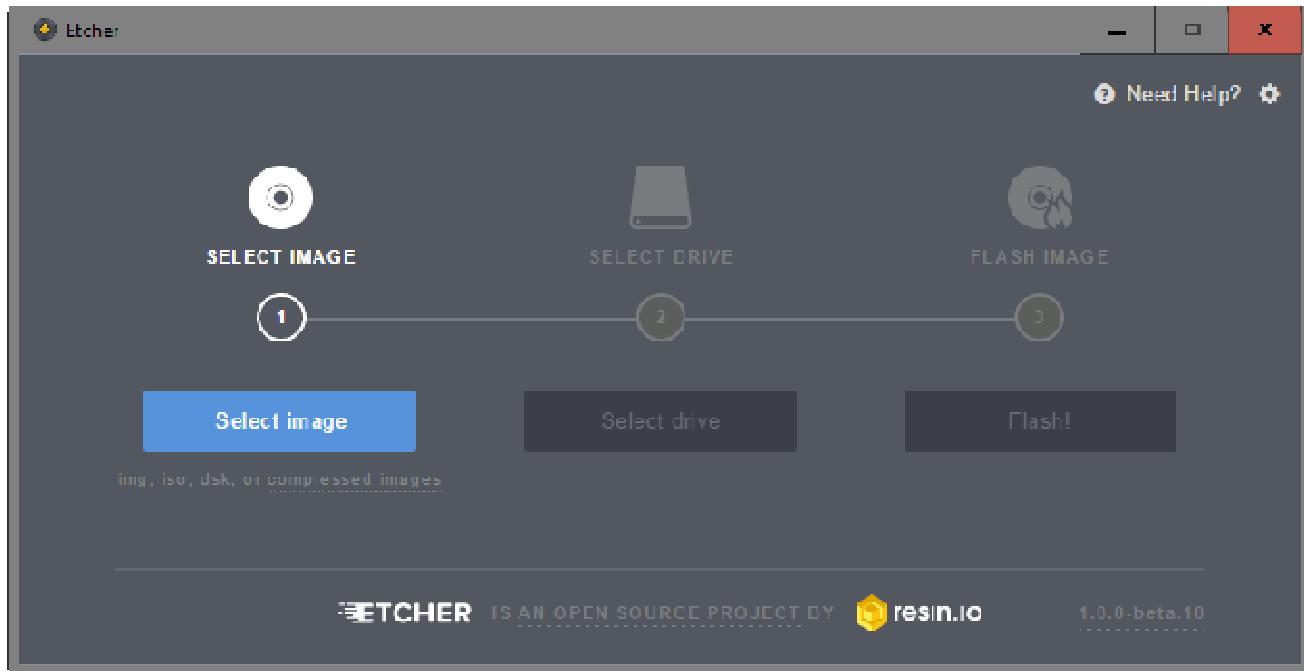
## Step 3

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

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

## Step 4

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



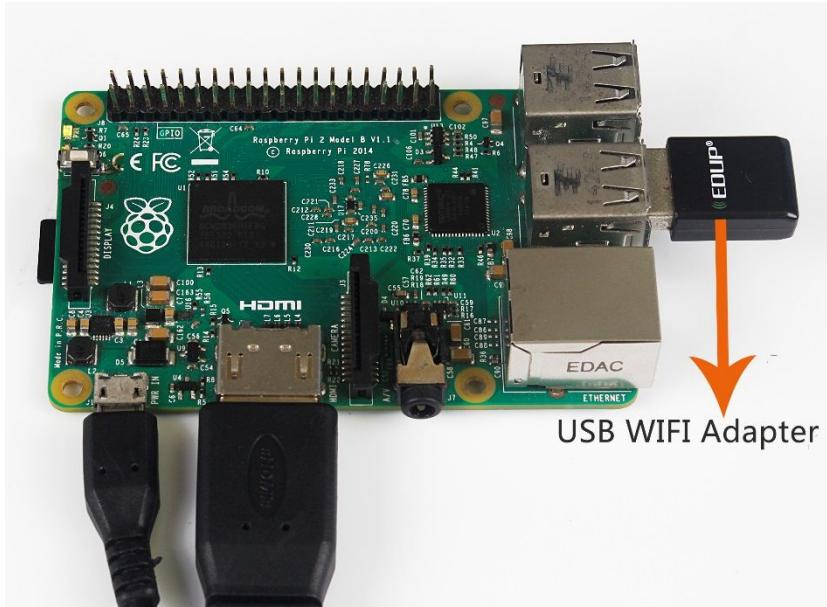
## Step 5

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

## Connect the Raspberry Pi to the Internet

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

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



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

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



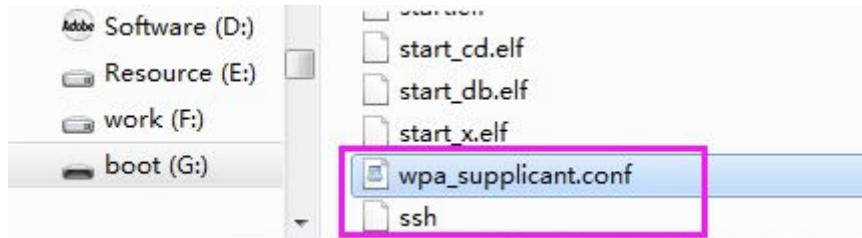
Input the following content in the file.

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

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

## Start SSH

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



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

## Get the IP Address

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

### 1. Checking via the router

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

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

### 2. Network Segment Scanning

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

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

## Use the SSH Remote Control

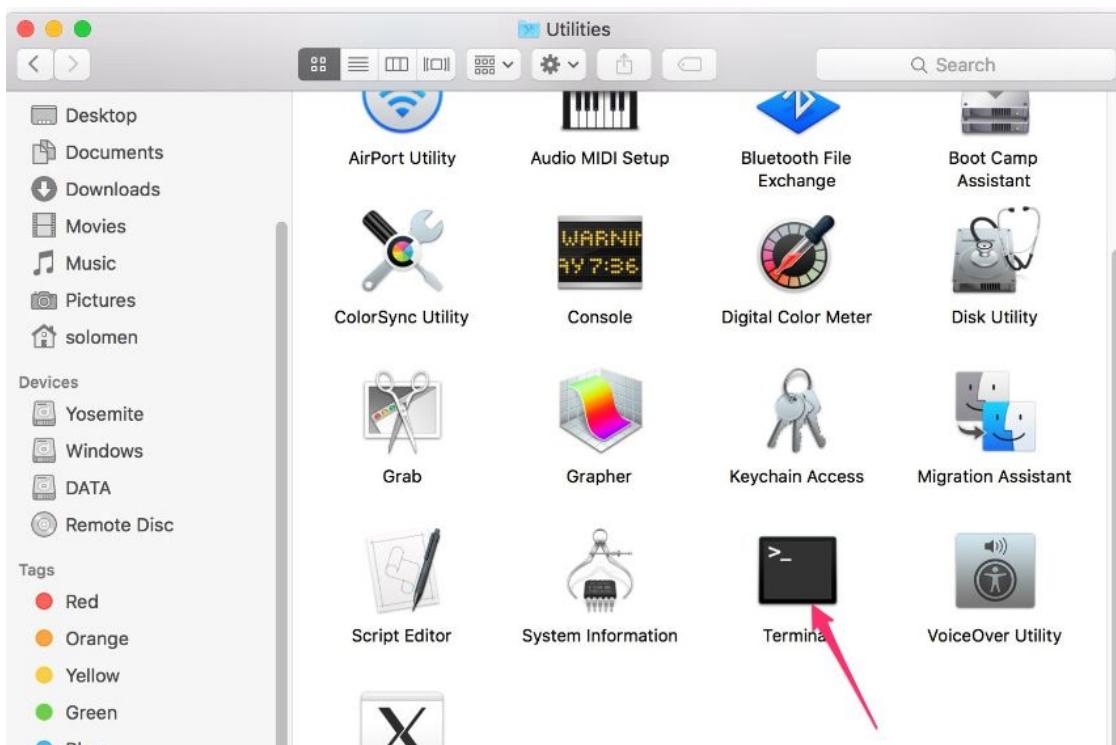
We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge

linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

## For Linux or/Mac OS X Users

### Step 1

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



### Step 2

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

```
ssh pi@192.168.18.197
```

### Step 3

Input"yes".

```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000
# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

## Step 4

Input the passcode and the default password is **raspberry**.

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

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

## Step 5

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

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

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHTQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126

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

pi@raspberrypi:~ $
```

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

## For Windows Users

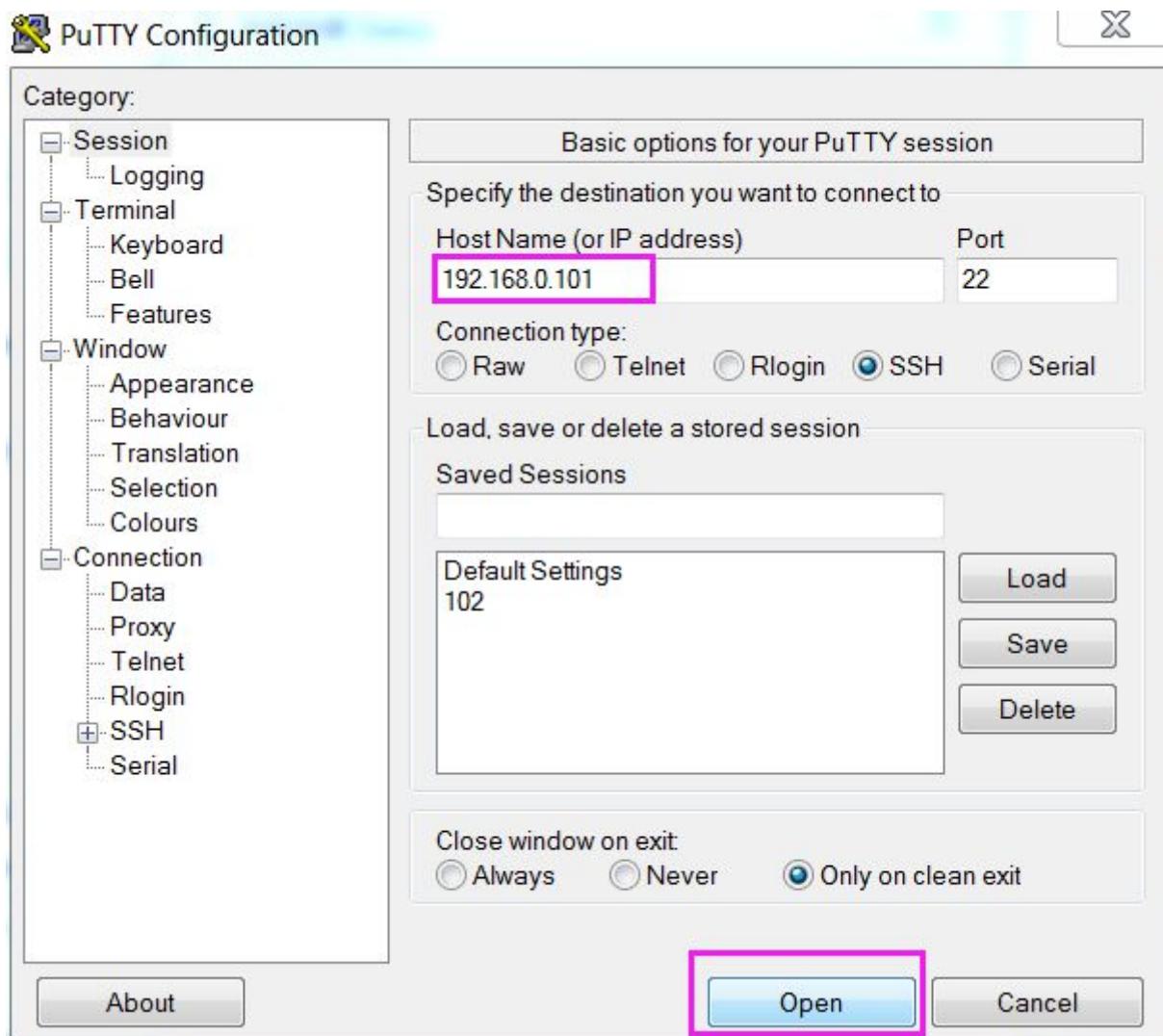
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

### Step 1

Download PuTTY.

### Step 2

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

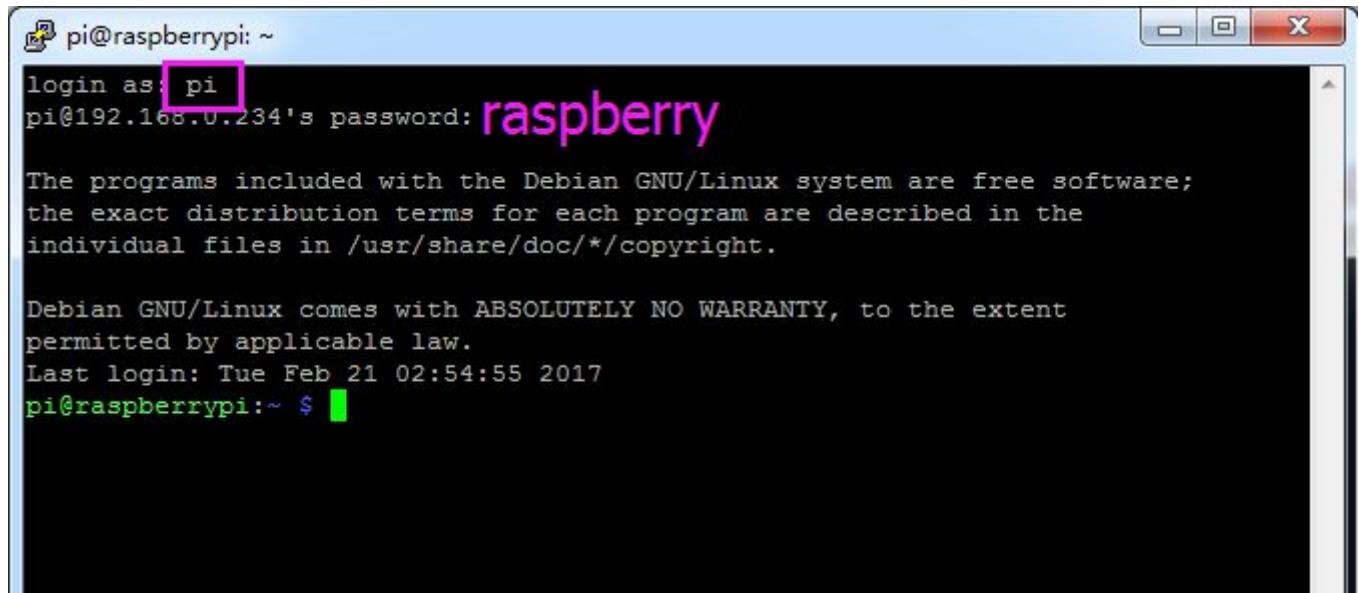


### Step 3

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

## Step 4

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



A screenshot of a PuTTY terminal window. The title bar says "pi@raspberrypi: ~". The window contains the following text:  
login as: pi  
pi@192.168.0.234's password: raspberry  
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: Tue Feb 21 02:54:55 2017  
pi@raspberrypi:~ \$

## Step 5

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

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

## Remote Desktop

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

### VNC

You can use the function of remote desktop through VNC.

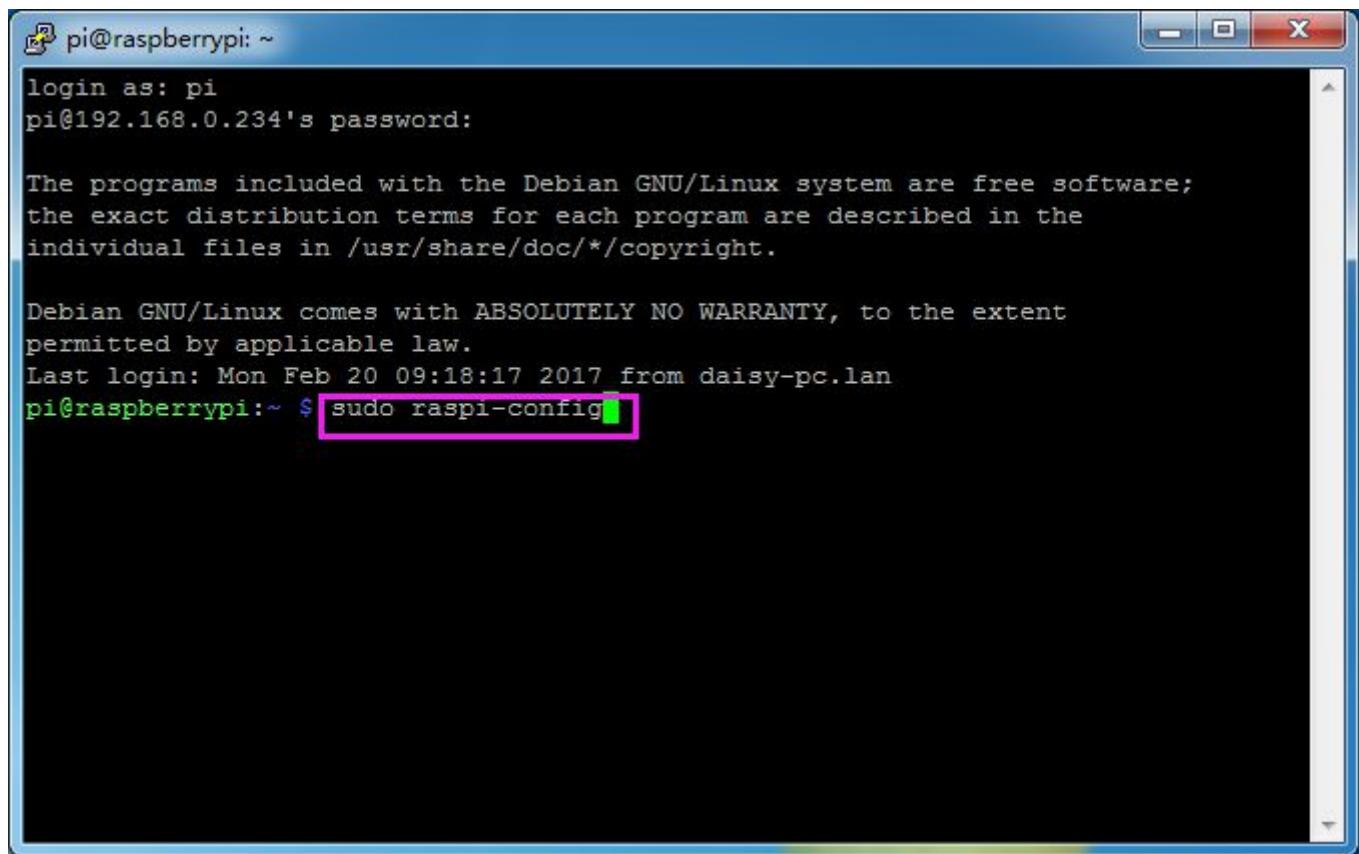
#### Enable VNC service

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

## Step 1

Input the following command:

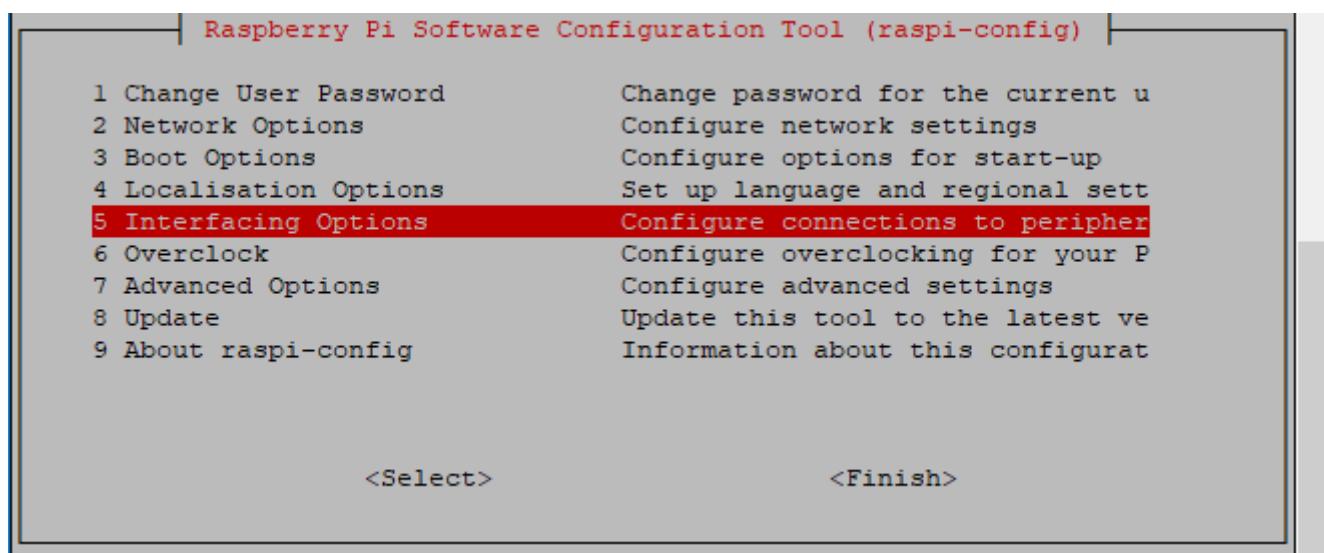
```
sudo raspi-config
```



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows a login message, system information, and a command prompt. The command "sudo raspi-config" is typed at the prompt and highlighted with a red rectangle.

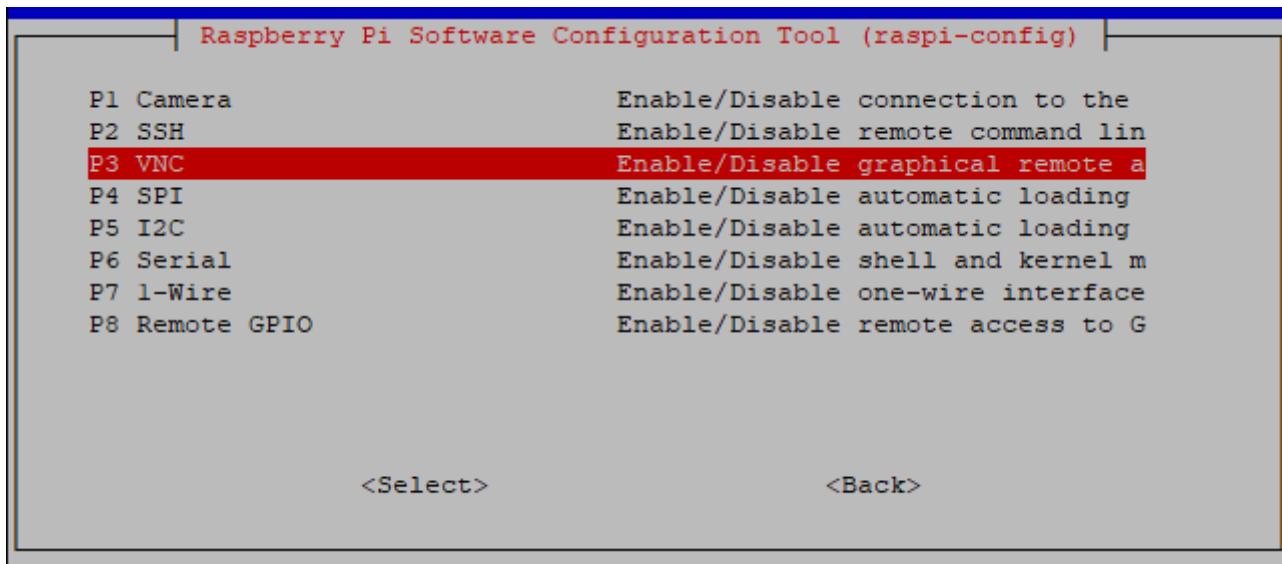
## Step 2

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



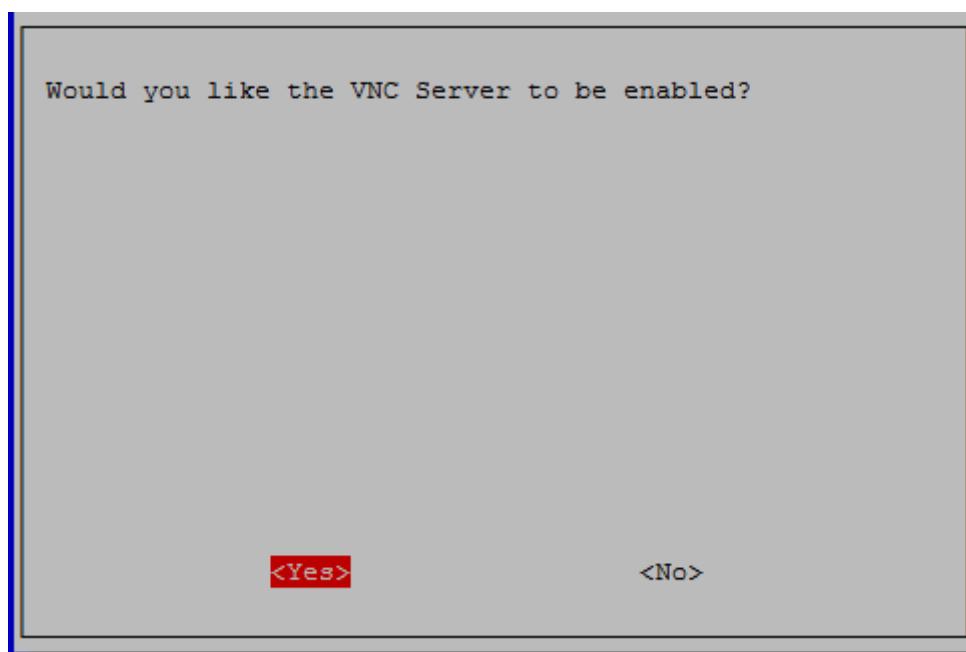
## Step 3

Select **VNC**.



## Step 4

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



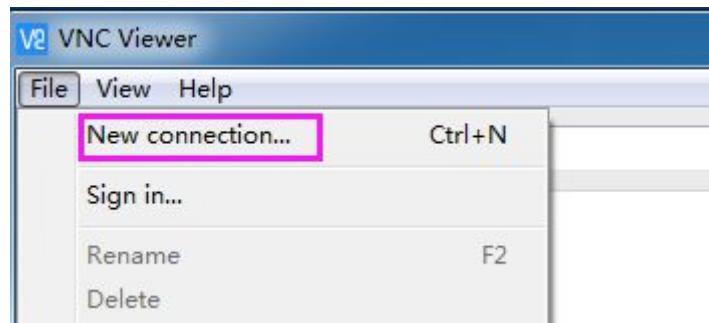
## Login to VNC

### Step 1

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

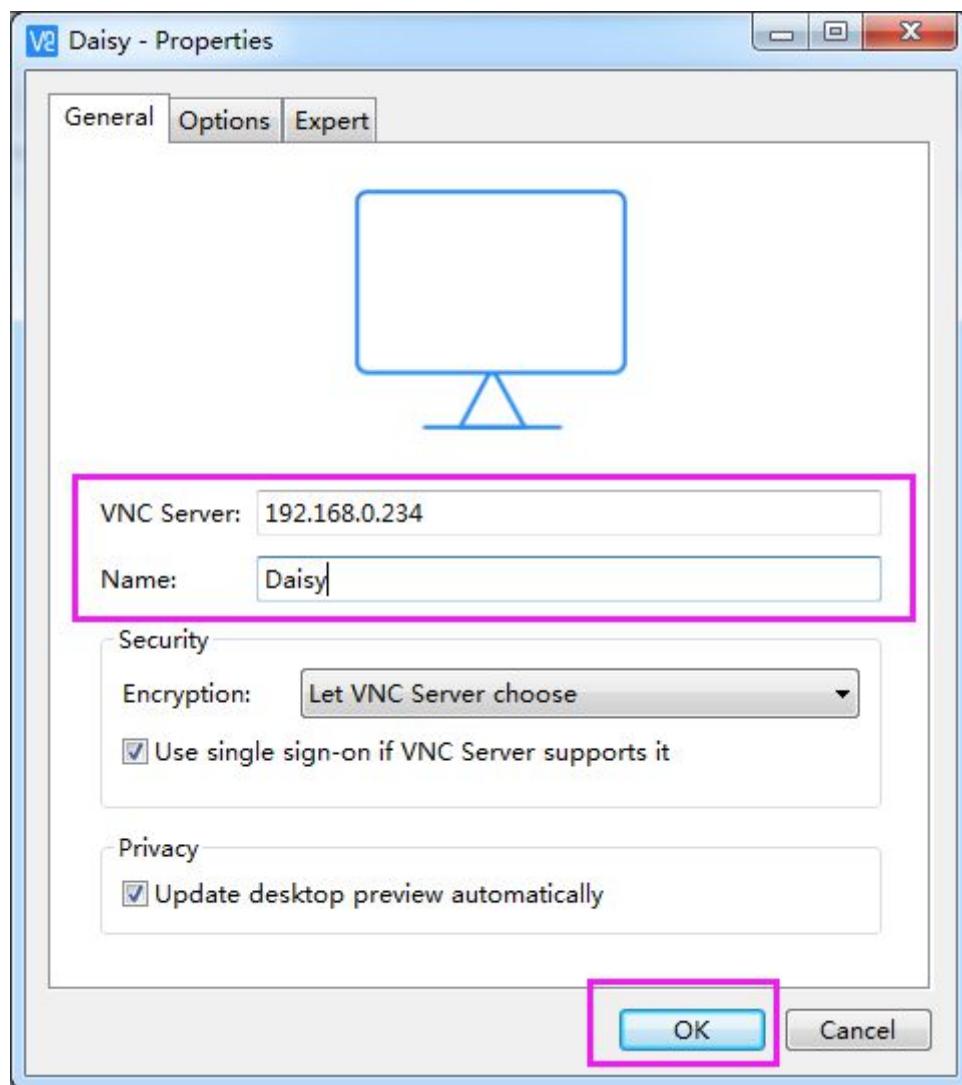
## Step 2

Then select “New connection”.



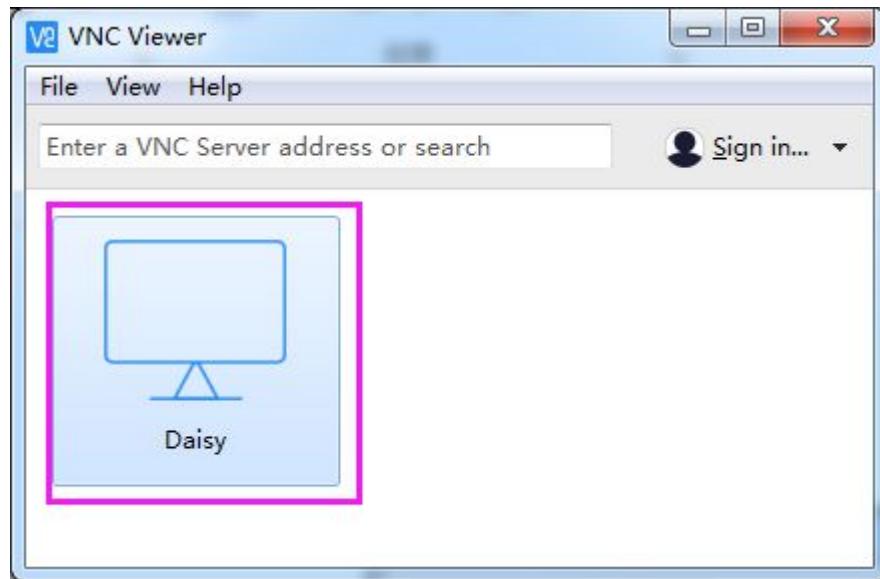
## Step 3

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



## Step 4

Double click the **connection** just created:



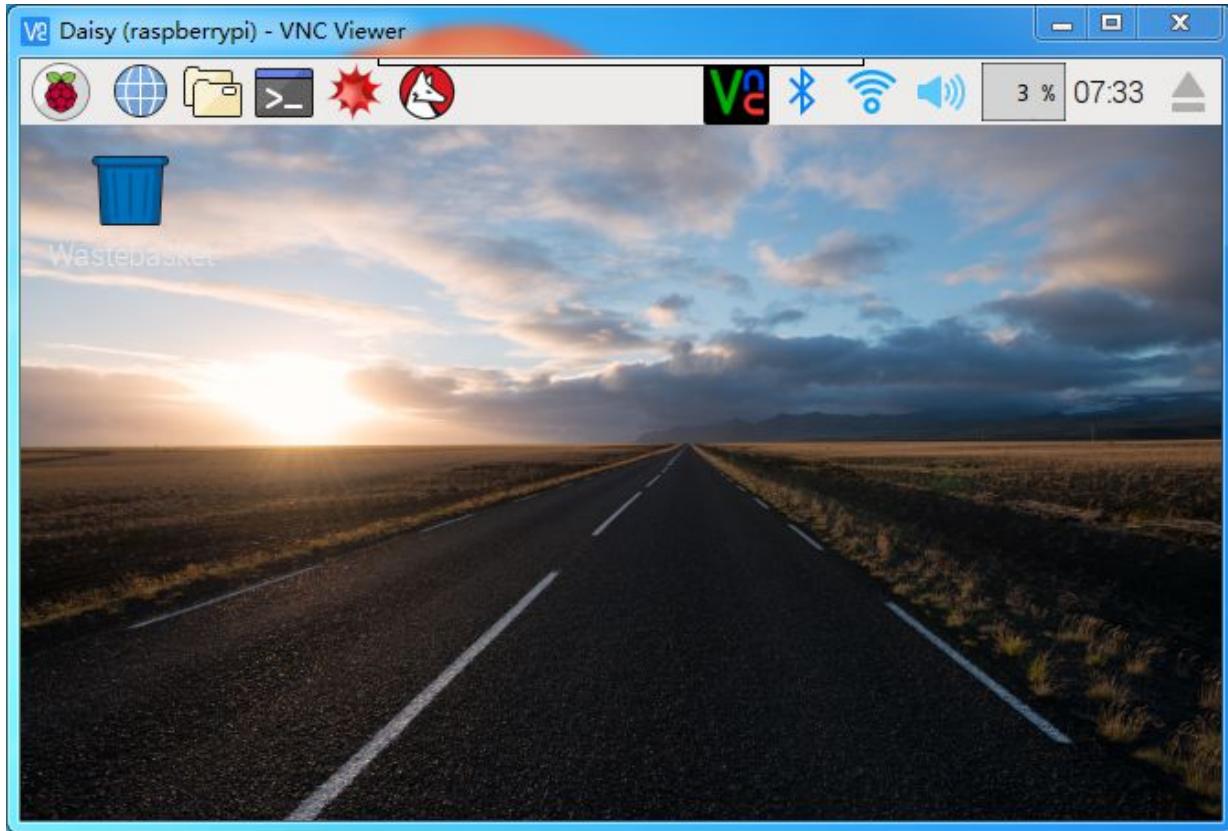
## Step 5

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



## Step 6

Now you can see the desktop of the Raspberry Pi:



## XRDP

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

### Install XRDP

#### Step 1

Login to Raspberry Pi by using SSH.

#### Step 2

Input the following instructions to install XRDP.

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

#### Step 3

Later, the installation starts.

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

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

## Step 4

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

### Login to XRD

#### Step 1

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

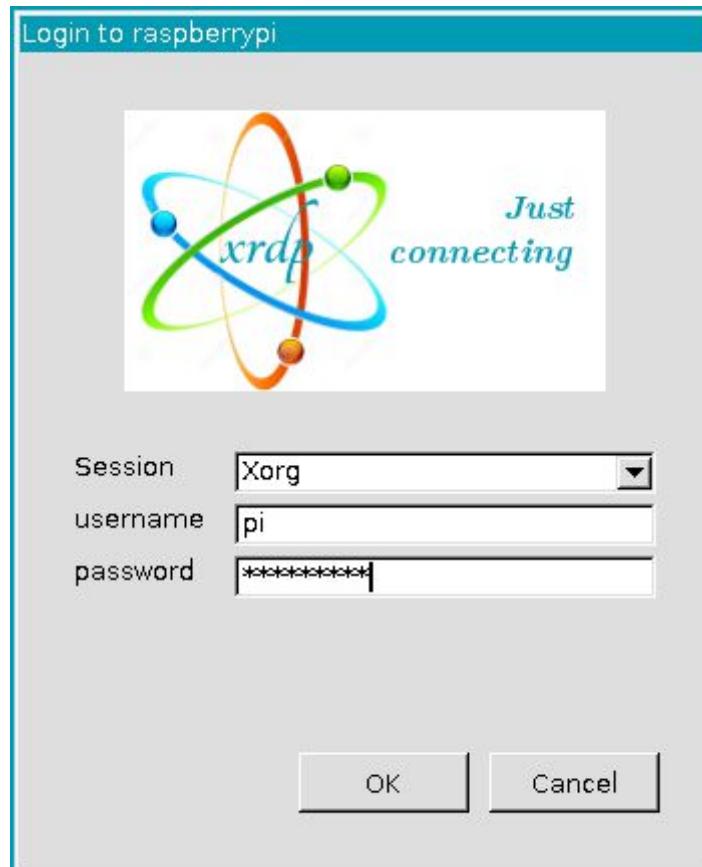
#### Step 2

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



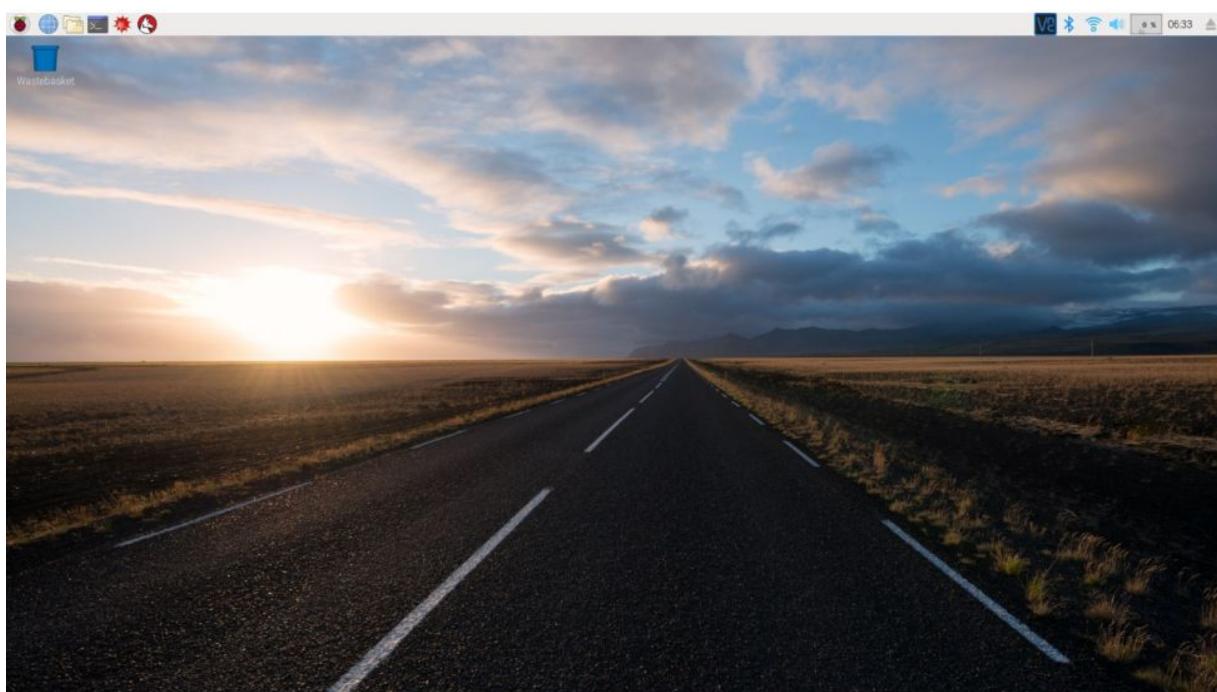
## Step 3

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



## Step 4

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



# Libraries

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

## RPi.GPIO

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

RPi.GPIO is a module to control Raspberry Pi GPIO channels. This package provides a class to control the GPIO on a Raspberry Pi. For examples and documents, visit <http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>

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

```
python
```

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

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

```
import RPi.GPIO
```

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

If you want to quit python CLI, type in:

```
exit()
```

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

# WiringPi

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

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

**gpio -v**

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

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

## gpio readall

```
pi@raspberrypi:~ $ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 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 | IN   | TxD | 15  | 14  |
|      | 0v  |          |      |   | 9       | 10| 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 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|      | 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 | IN   | TxD | 15  | 14  |
|      | 0v  |          |      |   | 9       | 10| 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 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

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

# Raspberry Pi GPIO Extension Board

We apply the GPIO Extension Board to extend the pins of Raspberry Pi to the breadboard and avoid damage caused by frequent plugging and unplugging.

Here, we apply a 40-pin GPIO development board and a 40-pin GPIO cable. In case of the potential risk of short circuit, you must build your circuit in strict accordance with the following picture.



To improve your understanding of the relationship of the marked pins on 40-pin GPIO Extension Board and the pins of Raspberry Pi, we have drawn the following comparison chart.

Name	WiringPi	BCM	GPIO Extention Board		BCM	WiringPi	Name
3.3V	3V3	3V3	3V3	5V0	5.0V	5.0V	5V
SDA	8	2	SDA1	5V0	5.0V	5.0V	5V
SCL	9	3	SCL1	GND	GND	GND	0V
GPIO7	7	4	GPIO4	TXD0	14	15	TXD
0V	GND	GND	GND	RXD0	15	16	RXD
GPIO0	0	17	GPIO17	GPIO18	18	1	GPIO1
GPIO2	2	27	GPIO27	GND	GND	GND	0V
GPIO3	3	22	GPIO22	GPIO23	23	4	GPIO4
3.3V	3.3V	3.3V	3.3V	GPIO24	24	5	GPIO5
MOSI	12	10	SPIMOSI	GND	GND	GND	0V
MISO	13	9	SPIMISO	GPIO25	25	6	GPIO6
SCLK	14	11	SCLK	SPICE0	8	10	CEO
0V	GND	GND	GND	SPICE1	7	11	CE1
IN_SDA	30	0	ID_SD	ID_SC	1	31	ID_SCL
GPIO21	21	5	GPIO5	GND	GND	GND	0V
GPIO22	22	6	GPIO6	GPIO12	12	26	GPIO26
GPIO23	23	13	GPIO13	GND	GND	GND	0V
GPIO24	24	19	GPIO19	GPIO16	16	27	GPIO27
GPIO25	25	26	GPIO26	GPIO20	20	28	GPIO28
0V	GND	GND	GND	GPIO21	21	29	GPIO29

# Download the Code

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

## Method 1: Use git clone (Recommended)

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

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

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

Clone the repository from GitHub

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

## Method 2: Download the code.

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

This screenshot shows the GitHub repository page for the SunFounder Da Vinci Kit for Raspberry Pi. The repository has 5 commits, 1 branch, 0 releases, and 1 contributor. It is licensed under GPL-2.0. The page includes navigation links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Security, and Insights. A green 'Clone or download' button is visible on the right.

This repository is for SunFounder Da Vinci Kit for Raspberry Pi.

This screenshot shows the GitHub repository page for the SunFounder Da Vinci Kit for Raspberry Pi. The repository has 5 commits, 1 branch, 0 releases, and 1 contributor. It is licensed under GPL-2.0. The page includes navigation links for Branch: master, New pull request, Create new file, Upload files, Find File, and Clone or download. A green 'Clone or download' button is visible on the right. The repository contains files: c, python, LICENSE, README.md, and show. The 'Clone with HTTPS' section shows the URL <https://github.com/sunfounder/davinci-kit-for-raspberry-pi>. A red box highlights the 'Download ZIP' button.

# 1 Output

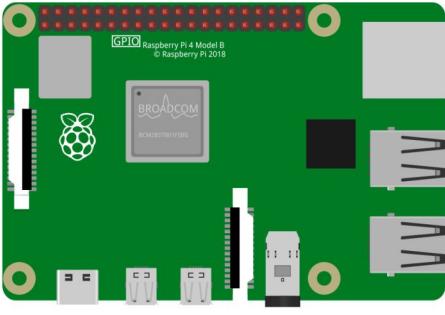
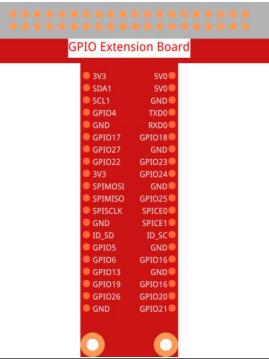
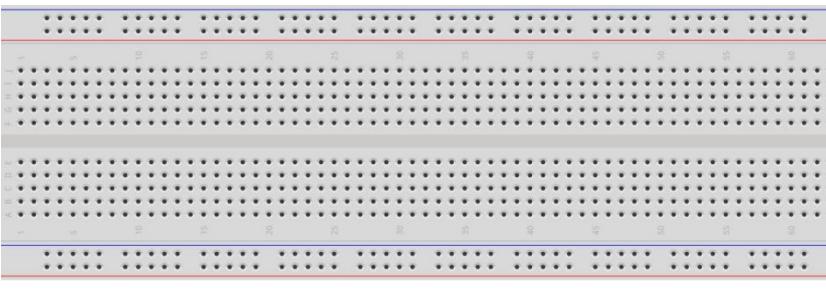
## 1.1 Displays

### 1.1.1 Blinking LED

#### Introduction

In this lesson, we will learn how to make a blinking LED by programming. Through your settings, your LED can produce a series of interesting phenomena. Now, go for it.

#### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * LED
	 GPIO Extension Board Pinout: 3V3 SDA1 GND SCL1 GND TXD0 GND RXD0 GPIO17 GPIO18 GPIO27 GND GPIO22 GPIO23 3V GND GPINOSI GND GPINOSO GPIO25 SPICLK SPICE0 GND SPICE1 ID_SD GND GPIO5 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO26 GPIO20 GND GPIO21	
1 * 40-pin Cable	Several Jumper Wires	
		
1 * Breadboard	1 * Resistor(220Ω)	
		

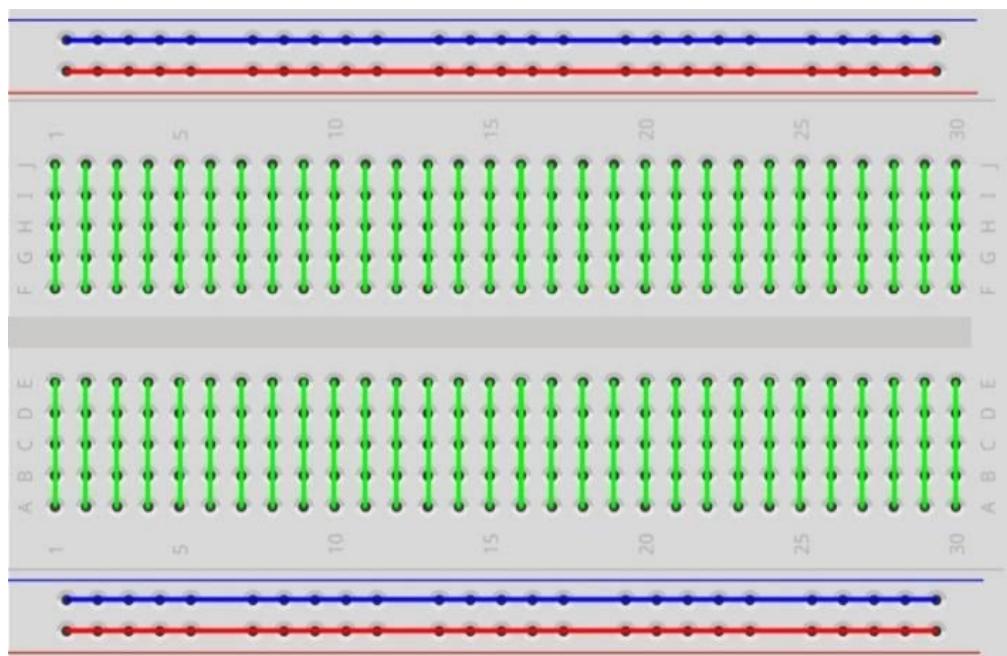
**Note:** In order to proceed smoothly, you need to bring your own Raspberry Pi, TF card and Raspberry Pi power.

## Principle

### Breadboard

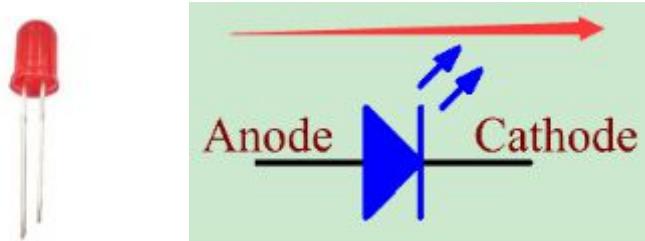
A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you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

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



### LED

LED is a kind of diode. LED will shine only if the long pin of LED is connected to the positive electrode and the short pin is connected to negative electrode.

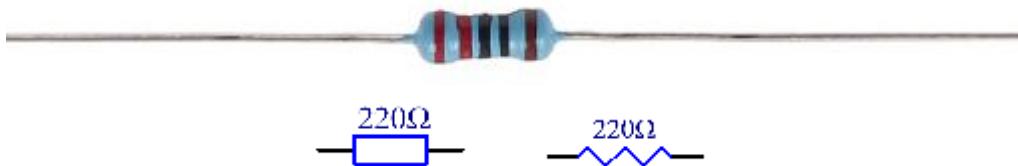


The LED can not be directly connected to power supply, which can damage component. A resistor with  $160\Omega$  or larger (work in 5V) must be connected in series in the circuit of LED.

## Resistor

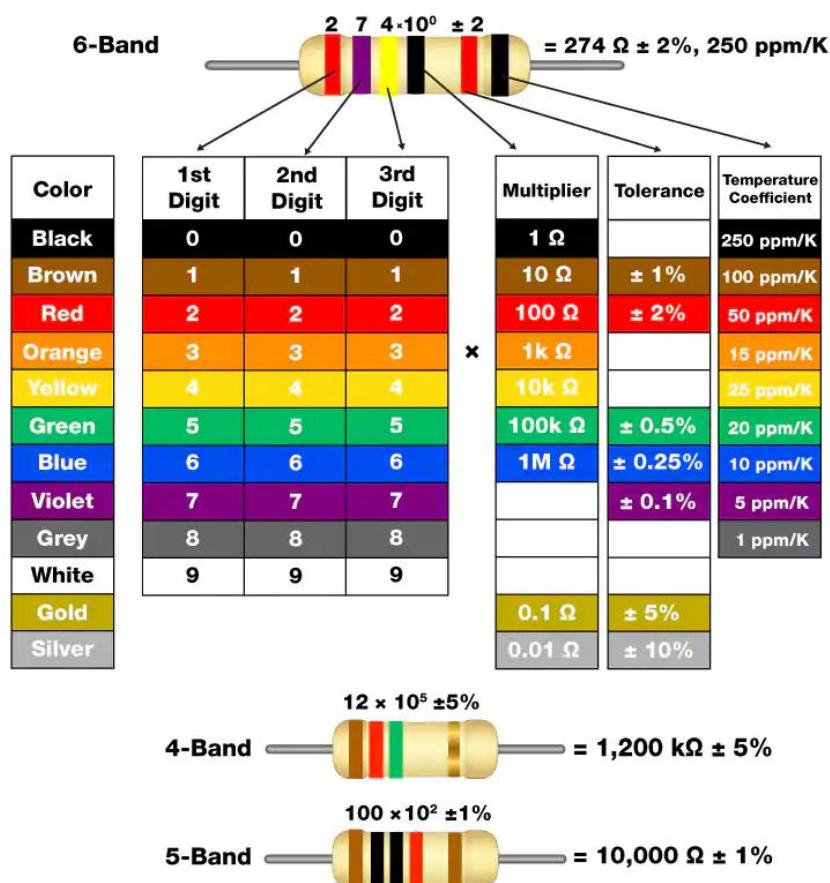
Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

Fixed resistor is applied in this kit. In the circuit, it is essential to protect the connected components. The following pictures show a real object,  $220\Omega$  resistor and two generally used circuit symbols of resistor.  $\Omega$  is the unit of resistance and the larger units include  $K\Omega$ ,  $M\Omega$ , etc. Their relationship can be shown as follows:  $1 M\Omega = 1000 K\Omega$ ,  $1 K\Omega = 1000 \Omega$ . Normally, the value of resistance is marked on it. So if you see these symbols in a circuit, it means that there is a resistor.



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. To measure the value, use multimeter.

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



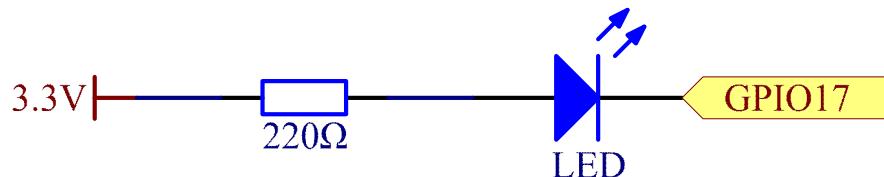
## Schematic Diagram

In this experiment, connect a  $220\Omega$  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 GPIO17 of Raspberry Pi. Therefore, to turn on an LED, we need to make GPIO17 low (0V) level. We can get this phenomenon by programming.

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

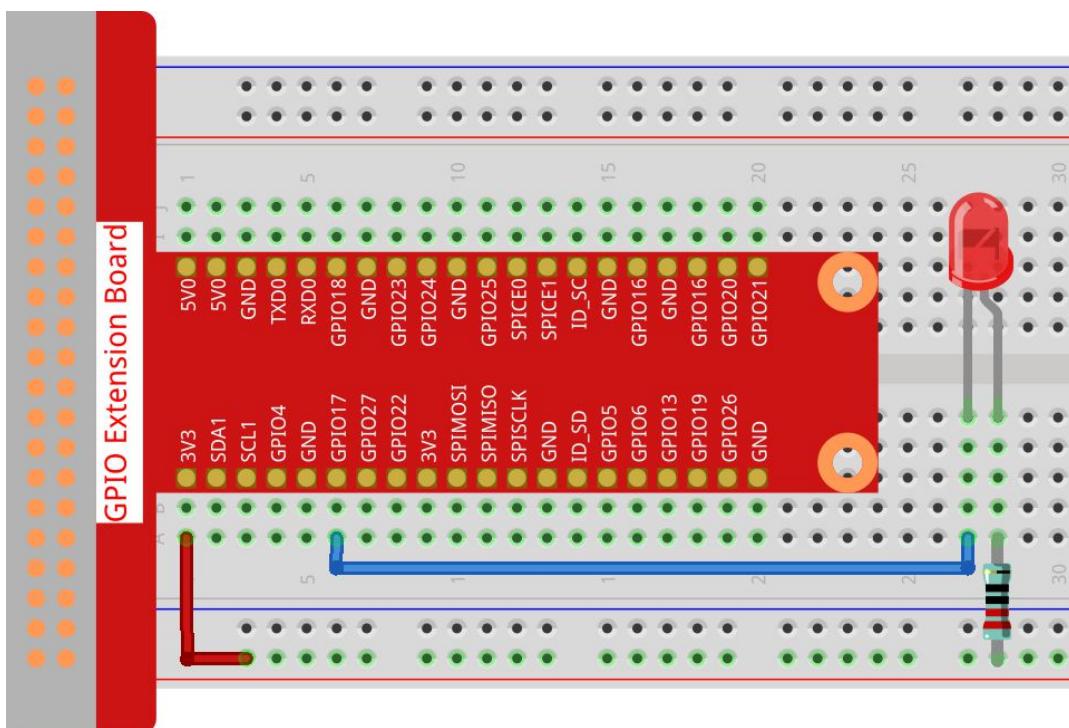
In the C language related content, we make GPIO0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi, Pin 11.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



## Experimental Procedures

**Step 1:** Build the circuit.



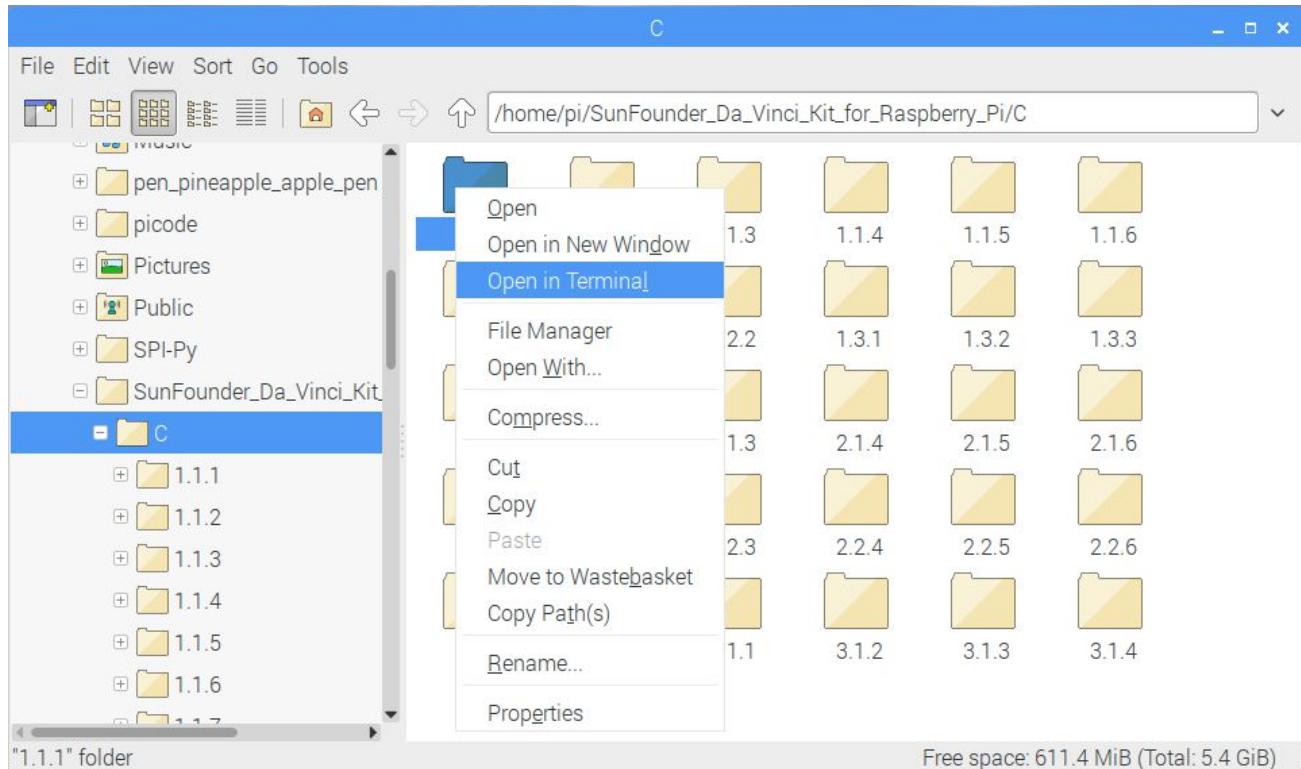
## ➤ For C Language Users

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

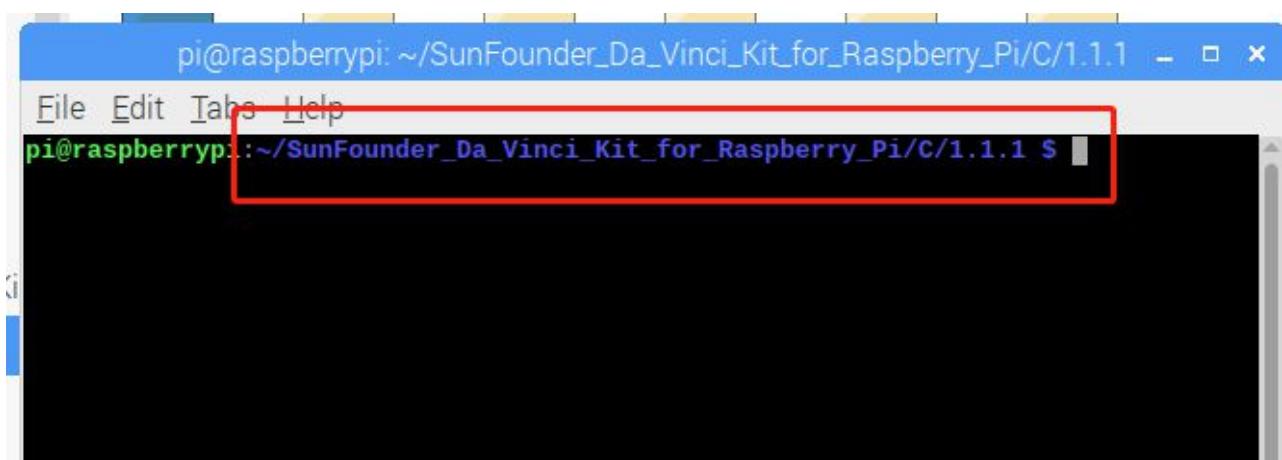
If you use a monitor, you're recommended to take the following steps.

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

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



Then a window will pop up as shown below. So now you've entered the path of the code **1.1.1\_BlinkingLed.c**.

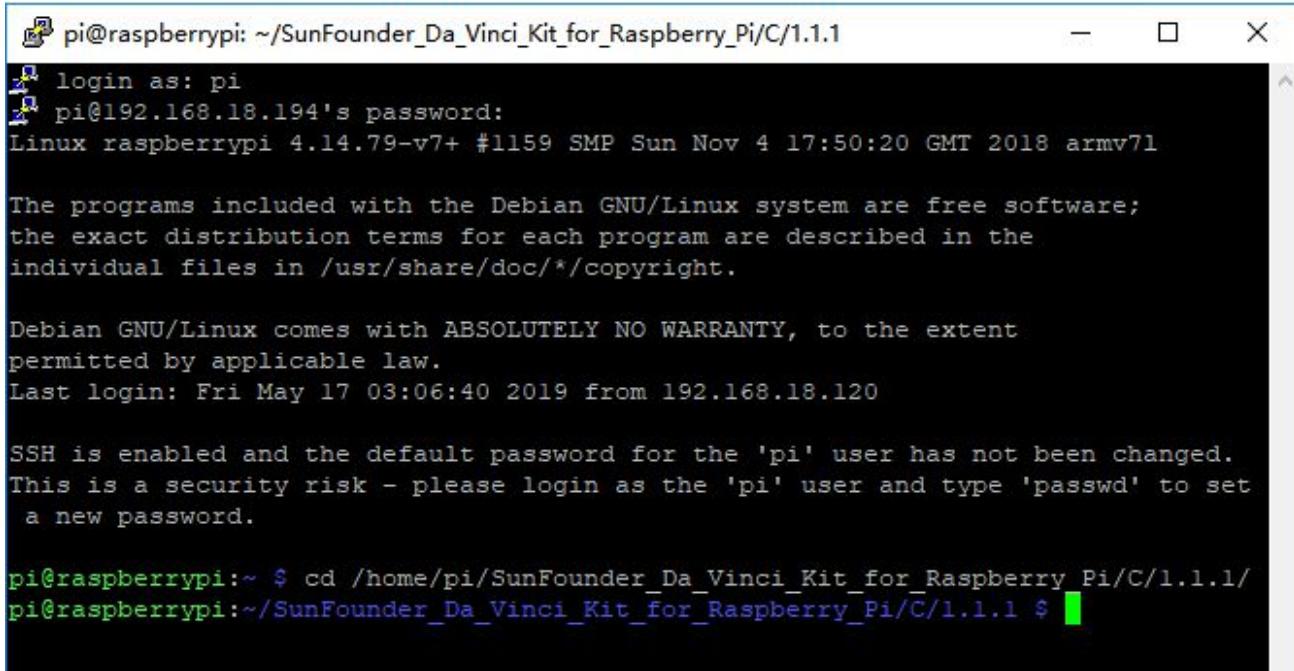


In the following lessons, we will use command to enter the code file instead of right-clicking. But you can choose the method you prefer.

If you log into the Raspberry Pi remotely, use ‘cd’ to change directory:

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.1/
```

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



```
pi@raspberrypi: ~/$ cd /home/pi/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1 $
```

The terminal window shows the user is logged in as 'pi' on a Raspberry Pi. It displays the system's version (Linux raspberrypi 4.14.79-v7+ #1159 SMP Sun Nov 4 17:50:20 GMT 2018 armv7l), the copyright notice for the Debian GNU/Linux system, and the last login information. It also includes a warning about SSH being enabled and the default password for the 'pi' user not being changed. The final command shown is 'cd /home/pi/SunFounder\_Da\_Vinci\_Kit\_for\_Raspberry\_Pi/C/1.1.1'.

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

### Step 3: Compile the code

```
gcc 1.1.1_BlinkingLed.c -o BlinkingLed -lwiringPi
```

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

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

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

```
sudo ./BlinkingLed
```

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

```

pi@raspberrypi: ~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1 $ gcc 1.1.1_BlinkingLed.c
-o BlinkingLed -lwiringPi
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1 $ sudo ./BlinkingLed

=====
|      Blink LED
| -----
|      LED connect to #17
| -----
|      LED will Blink at 500ms
| -----
|      SunFounder
=====

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

```

After the code runs, you will see the LED flashing.

If you want to edit the code file *1.1.1\_BlinkingLed.c*, press **Ctrl + C** to stop running the code. Then type the following command to open it:

**nano 1.1.1\_BlinkingLed.c**

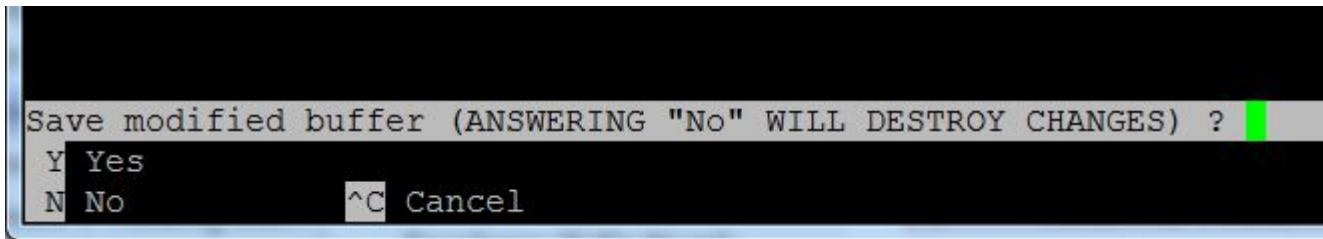
**Note:** nano is a text editor tool. The command is used to open the code file *1.1.1\_BlinkingLed.c* by this tool.

```

pi@raspberrypi: ~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/1.1.1
GNU nano 2.7.4          File: 1.1.1_BlinkingLed.c          Modified
=====
/*
* Filename      : 1.1.1_BlinkingLed.c
* Description   : Make an led blinking.
* Author       : Robot
* E-mail        : support@sunfounder.com
* website      : www.sunfounder.com
* Update       : Cavon    2016/07/01
*/
#include <wiringPi.h>
//The hardware drive library designed for the C language of Raspberry Pi. A$#
#include <stdio.h>
// Standard I/O library. The printf function used for printing the data disp$#
#define LedPin          0
// Pin Bl7 of the T_Extension Board is corresponding to the pin0 in wiringPi$#
int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
}

```

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.



## Code

The program code is shown as follows:

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin    0
int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(LedPin, OUTPUT);// Set LedPin as output to write value to it.
    while(1){
        // LED on
        digitalWrite(LedPin, LOW);
        printf("...LED on\n");
        delay(500);
        // LED off
        digitalWrite(LedPin, HIGH);
        printf("LED off...\n");
        delay(500);
    }
    return 0;
}
```

## Code Explanation

```
#include <wiringPi.h>
```

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

```
#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 GPIO17 of the T\_Extension Board is corresponding to the GPIO0 in wiringPi. Assign GPIO0 to LedPin, LedPin represents GPIO0 in the code later.

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

This initialises wiringPi and assumes that the calling program is going to be using the wiringPi pin numbering scheme.

This function needs to be called with root privileges. When initialize wiring failed, print message to screen. The function "return" is used to jump out of the current function. Using return in main() function will end the program.

```
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.

```
printf("...LED off\n");
```

The printf function is a standard library function and its function prototype is in the header file "stdio.h". The general form of the call is: printf(" format control string ", output table columns). The format control string is used to specify the output format,

which is divided into format string and non-format string. The format string starts with '%' followed by format characters, such as' %d 'for decimal integer output. Unformatted strings are printed as prototypes. What is used here is a non-format string, followed by "\n" that is a newline character, representing automatic line wrapping after printing a string.

delay(500);

Delay (500) keeps the current HIGH or LOW state for 500ms.

This is a function that suspends the program for a period of time. And the speed of the program is determined by our hardware. Here we turn on or off the LED. If there is no delay function, the program will run the whole program very fast and continuously loop. So we need the delay function to help us write and debug the program.

return 0;

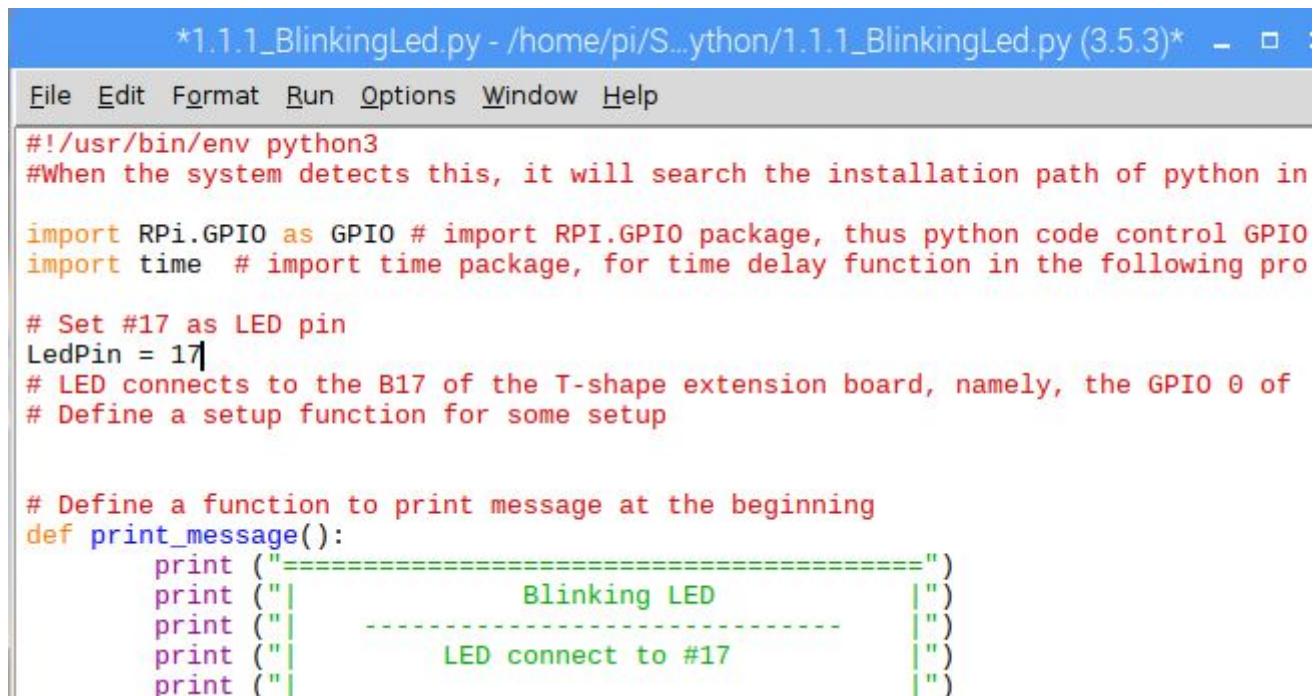
Usually, it is placed behind the main function, indicating that the function returns 0 on successful execution.

## ➤ For Python Language Users

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

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

Find 1.1.1\_BlinkingLed.py and double click it to open. Now you're in the file.



```
*1.1.1_BlinkingLed.py - /home/pi/S...ython/1.1.1_BlinkingLed.py (3.5.3)* - □ : 
File Edit Format Run Options Window Help
#!/usr/bin/env python3
#When the system detects this, it will search the installation path of python in
import RPi.GPIO as GPIO # import RPI.GPIO package, thus python code control GPIO
import time # import time package, for time delay function in the following pro

# Set #17 as LED pin
LedPin = 17
# LED connects to the B17 of the T-shape extension board, namely, the GPIO 0 of
# Define a setup function for some setup

# Define a function to print message at the beginning
def print_message():
    print ("=====")
    print ("|      Blinking LED      |")
    print ("|-----|")
    print ("|      LED connect to #17 |")
    print ("|-----|")
```

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

```

Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: /home/pi/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/python/1.1.1_BlinkingLed.py
=====
        Blinking LED
-----
    LED connect to #17
    LED will Blink at 500ms
    SunFounder
=====

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

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

```

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. If you modify the code, before clicking **Run Module (F5)** you need to save it first. Then you can see the results.

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

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

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

**Step 3:** Run the code

```
sudo python 1.1.1_BlinkingLed.py
```

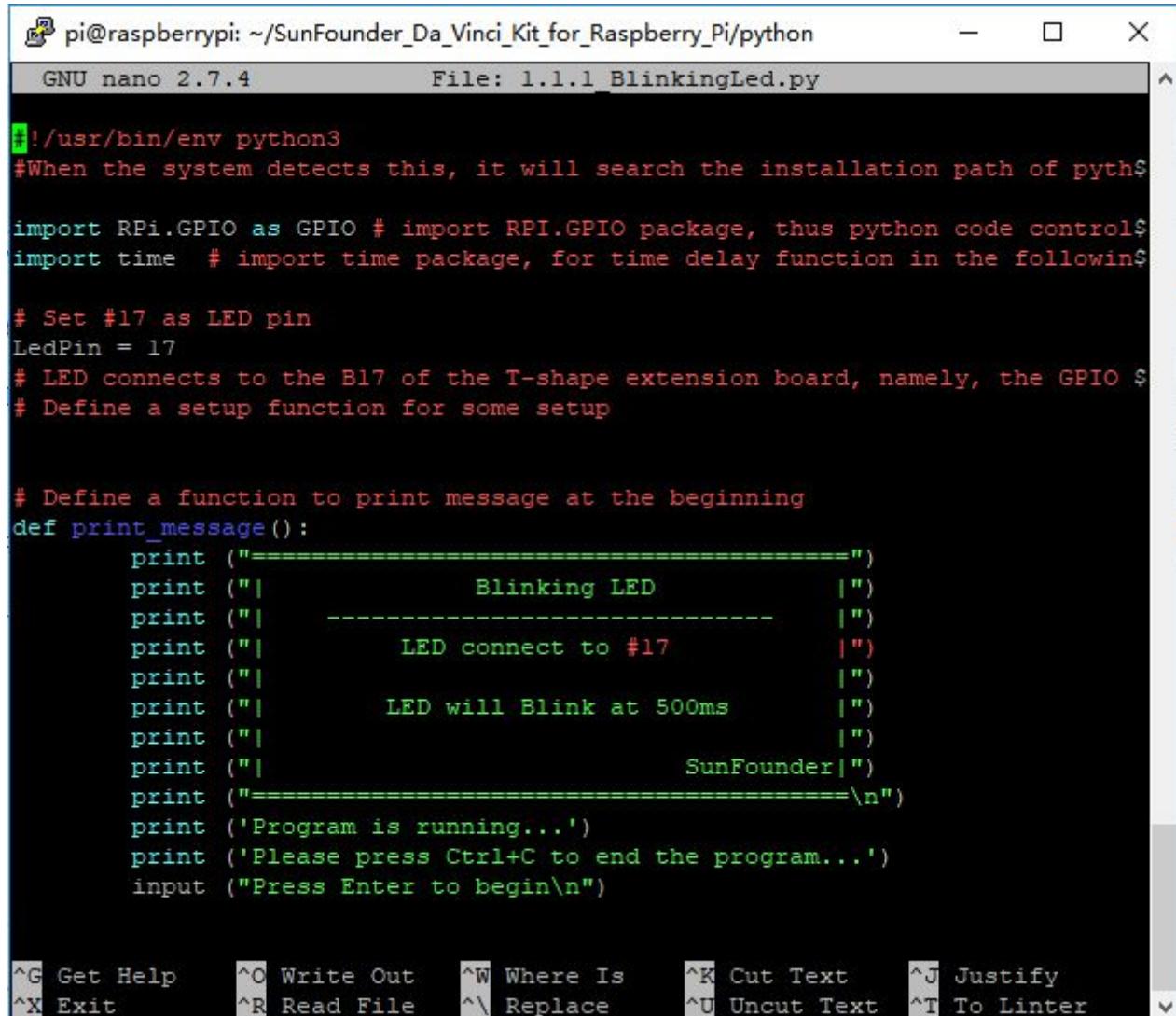
**Note:** Here sudo – superuser do, and python means to run the file by Python.

After the code runs, you will see the LED flashing.

**Step 4:** If you want to edit the code file **1.1.1\_BlinkingLed.py**, press **Ctrl + C** to stop running the code. Then type the following command to open **1.1.1\_BlinkingLed.py**:

```
nano 1.1.1_BlinkingLed.py
```

**Note:** nano is a text editor tool. The command is used to open the code file **1.1.1\_BlinkingLed.py** by this tool.



```

pi@raspberrypi: ~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/python
GNU nano 2.7.4          File: 1.1.1_BlinkingLed.py

#!/usr/bin/env python3
#When the system detects this, it will search the installation path of pyth$


import RPi.GPIO as GPIO # import RPI.GPIO package, thus python code controls$import time # import time package, for time delay function in the followin$

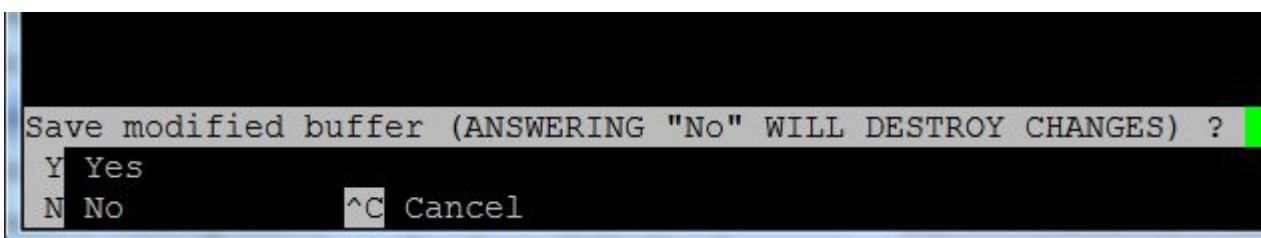

# Set #17 as LED pin
LedPin = 17
# LED connects to the B17 of the T-shape extension board, namely, the GPIO $# Define a setup function for some setup


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

^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit         ^R Read File     ^\ Replace       ^U Uncut Text    ^T To Linter

```

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 1.1.1\_BlinkingLed.py again to see the effect after the change.

## Code

The following is the program code:

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import time
LedPin = 17
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)
# Define a main function for main process
def main():
    while True:
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)
        # Delay for 0.5 second. Here, the statement is similar to delay function in C language,
        the unit is second.
        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 program destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

## Code Explanation

```
#!/usr/bin/env python3
```

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
```

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

```
import time
```

Import time package, for time delay function in the following program.

```
LedPin = 17
```

LED connects to the GPIO17 of the T-shape extension board, namely, BCM 17.

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
```

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

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO: BOARD numbers and BCM numbers. In our lessons, what we use is BCM numbers. You need to set up every channel you are using as an input or an output.

```
GPIO.output(LedPin, GPIO.LOW)
```

Set GPIO17(BCM17) as 0V (low level). Since the cathode of LED is connected to GPIO17, thus the LED will light up.

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

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

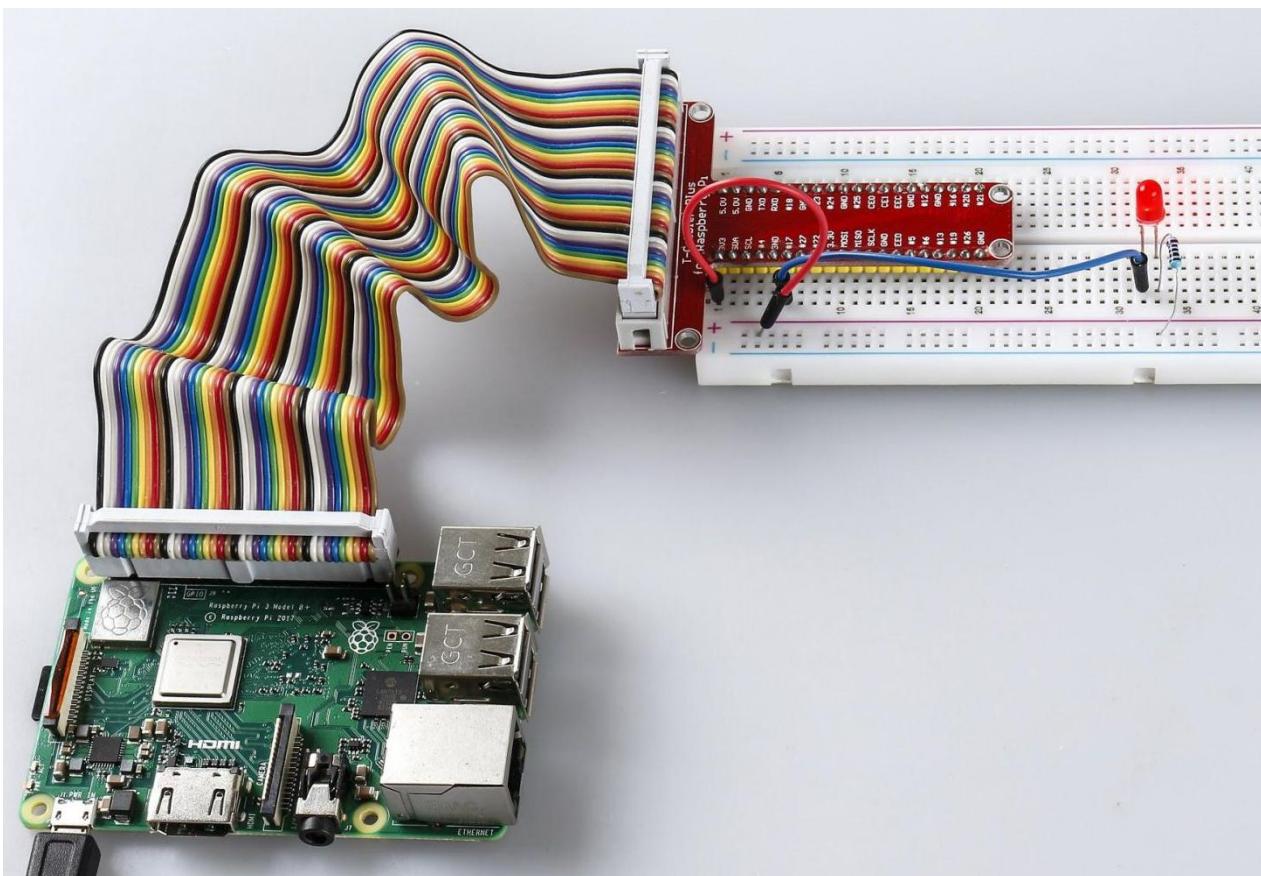
```
def destroy():
    GPIO.cleanup()
```

Define a destroy function for clean up everything after the script finished.

```
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

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

## Phenomenon Picture

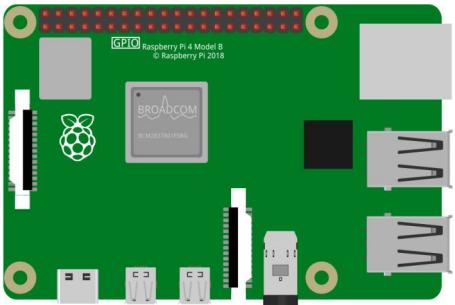
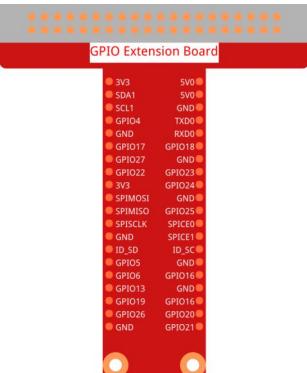
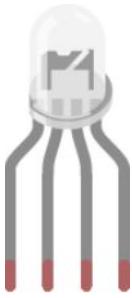
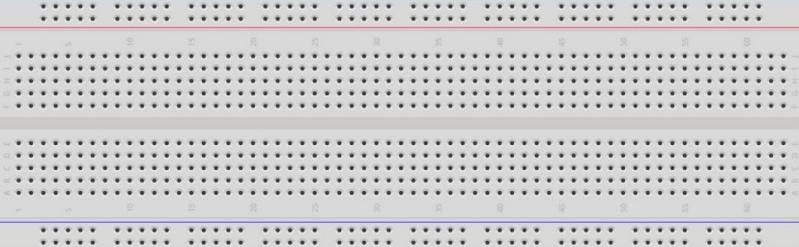


## 1.1.2 RGB LED

### Introduction

In this lesson, we will use it to control an RGB LED to flash various kinds of colors.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * RGB LED																																																																												
	 <table border="1"> <thead> <tr> <th></th> <th>3V3</th> <th>5V</th> <th>GND</th> </tr> </thead> <tbody> <tr><td>SDA1</td><td>SDA0</td><td>SDA0</td><td>GND</td></tr> <tr><td>SCL1</td><td>SCL0</td><td>SCL0</td><td>GND</td></tr> <tr><td>GPIO4</td><td>TXD0</td><td>RXD0</td><td>GND</td></tr> <tr><td>GND</td><td>GND</td><td>GND</td><td>GND</td></tr> <tr><td>GPIO17</td><td>GPIO18</td><td>GPIO18</td><td>GND</td></tr> <tr><td>GPIO27</td><td>GPIO27</td><td>GPIO27</td><td>GND</td></tr> <tr><td>GPIO22</td><td>GPIO23</td><td>GPIO23</td><td>GND</td></tr> <tr><td>3V</td><td>3V0</td><td>3V0</td><td>GND</td></tr> <tr><td>SPI_MOSI</td><td>GND</td><td>GND</td><td>GND</td></tr> <tr><td>SPI_MISO</td><td>GPIO25</td><td>GPIO25</td><td>GND</td></tr> <tr><td>SPI_SCLK</td><td>SPI_CE0</td><td>SPI_CE0</td><td>GND</td></tr> <tr><td>ID_SD</td><td>ID_SD</td><td>ID_SD</td><td>GND</td></tr> <tr><td>GPIO5</td><td>GND</td><td>GND</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td><td>GPIO16</td><td>GND</td></tr> <tr><td>GPIO13</td><td>GPIO16</td><td>GPIO16</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO16</td><td>GPIO16</td><td>GND</td></tr> <tr><td>GPIO26</td><td>GPIO20</td><td>GPIO20</td><td>GND</td></tr> <tr><td>GND</td><td>GPIO21</td><td>GPIO21</td><td>GND</td></tr> </tbody> </table>		3V3	5V	GND	SDA1	SDA0	SDA0	GND	SCL1	SCL0	SCL0	GND	GPIO4	TXD0	RXD0	GND	GND	GND	GND	GND	GPIO17	GPIO18	GPIO18	GND	GPIO27	GPIO27	GPIO27	GND	GPIO22	GPIO23	GPIO23	GND	3V	3V0	3V0	GND	SPI_MOSI	GND	GND	GND	SPI_MISO	GPIO25	GPIO25	GND	SPI_SCLK	SPI_CE0	SPI_CE0	GND	ID_SD	ID_SD	ID_SD	GND	GPIO5	GND	GND	GND	GPIO6	GPIO16	GPIO16	GND	GPIO13	GPIO16	GPIO16	GND	GPIO19	GPIO16	GPIO16	GND	GPIO26	GPIO20	GPIO20	GND	GND	GPIO21	GPIO21	GND	
	3V3	5V	GND																																																																											
SDA1	SDA0	SDA0	GND																																																																											
SCL1	SCL0	SCL0	GND																																																																											
GPIO4	TXD0	RXD0	GND																																																																											
GND	GND	GND	GND																																																																											
GPIO17	GPIO18	GPIO18	GND																																																																											
GPIO27	GPIO27	GPIO27	GND																																																																											
GPIO22	GPIO23	GPIO23	GND																																																																											
3V	3V0	3V0	GND																																																																											
SPI_MOSI	GND	GND	GND																																																																											
SPI_MISO	GPIO25	GPIO25	GND																																																																											
SPI_SCLK	SPI_CE0	SPI_CE0	GND																																																																											
ID_SD	ID_SD	ID_SD	GND																																																																											
GPIO5	GND	GND	GND																																																																											
GPIO6	GPIO16	GPIO16	GND																																																																											
GPIO13	GPIO16	GPIO16	GND																																																																											
GPIO19	GPIO16	GPIO16	GND																																																																											
GPIO26	GPIO20	GPIO20	GND																																																																											
GND	GPIO21	GPIO21	GND																																																																											
1 * 40-pin Cable		Several Jumper Wires																																																																												
																																																																														
1 * Breadboard		3 * Resistor(220Ω)																																																																												
																																																																														

### Principle

#### PWM

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you change, or modulate, that width. If you

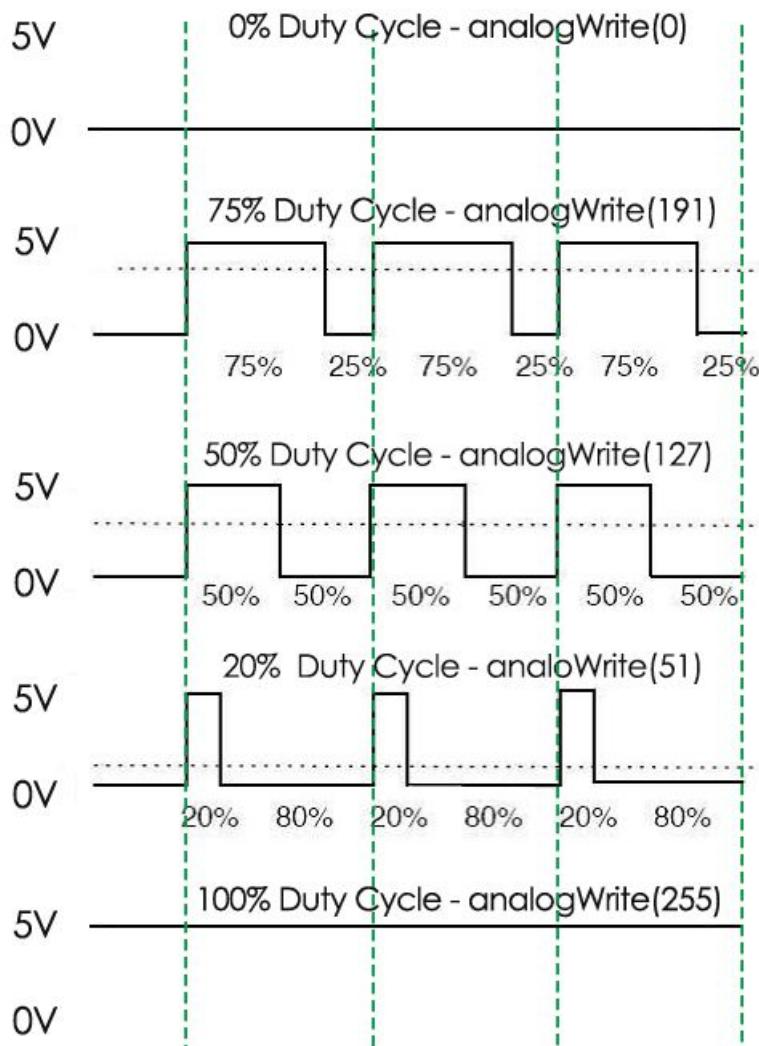
repeat this on-off pattern fast enough with some device, an LED for example, the result would be like this: the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

## Duty Cycle

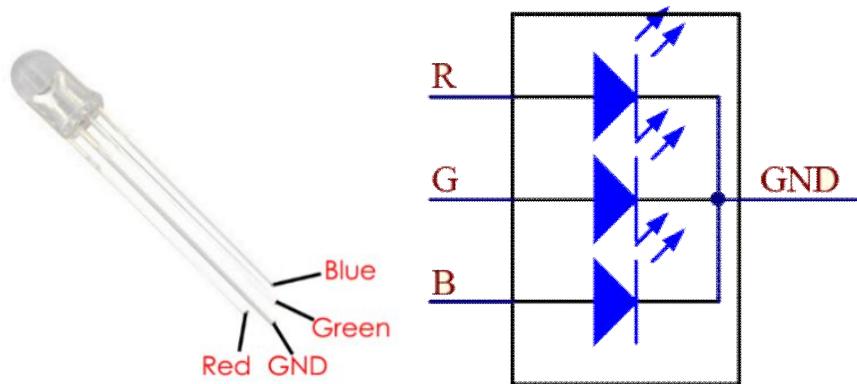
A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

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

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.



## RGB LED

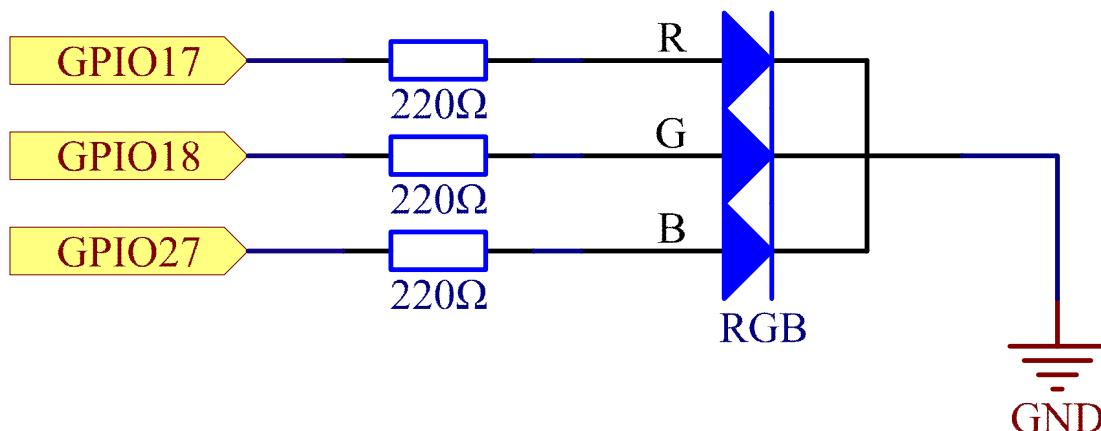


The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with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

### Schematic Diagram

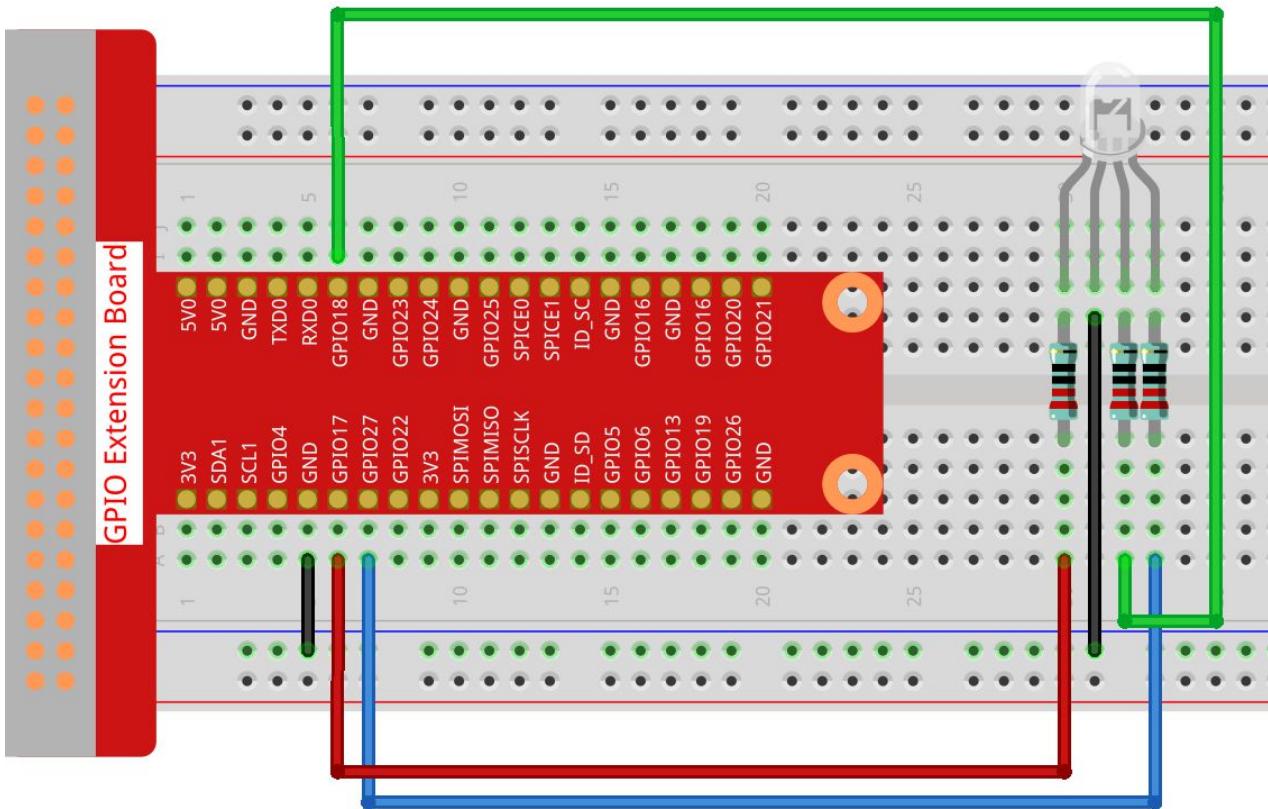
After connecting the pins of R, G, and B to a current limiting resistor, connect them to the GPIO17, GPIO18, and GPIO27 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



## Experimental Procedures

**Step 1:** Build the circuit.



### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.2/
```

**Step 3:** Compile the code.

```
gcc 1.1.2_rgbLed.c -lwiringPi
```

**Note:** When the instruction "gcc" is executed, if "-o " is not called, then the executable file is named "a.out".

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

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

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

## Code

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define uchar unsigned char
#define LedPinRed  0
#define LedPinGreen 1
#define LedPinBlue  2

void ledInit(void){
    softPwmCreate(LedPinRed, 0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
}

void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(LedPinRed,  r_val);
    softPwmWrite(LedPinGreen, g_val);
    softPwmWrite(LedPinBlue,  b_val);
}

int main(void){

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

    ledInit();
    while(1){
        printf("Red\n");
        ledColorSet(0xff,0x00,0x00); //red
        delay(500);
        printf("Green\n");
        ledColorSet(0x00,0xff,0x00); //green
        delay(500);
        printf("Blue\n");
        ledColorSet(0x00,0x00,0xff); //blue
        delay(500);
        printf("Yellow\n");
    }
}
```

```

ledColorSet(0xff,0xff,0x00); //yellow
delay(500);
printf("Purple\n");
ledColorSet(0xff,0x00,0xff); //purple
delay(500);
printf("Cyan\n");
ledColorSet(0xc0,0xff,0x3e); //cyan
delay(500);
}
return 0;
}

```

## Code Explanation

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

Library used for realizing the pwm function of the software.

```

void ledInit(void){
    softPwmCreate(LedPinRed, 0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
}

```

The function is to use software to create a PWM pin, set its period between 0x100us-100x100us.

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

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

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

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

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

```

void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(LedPinRed, r_val);
    softPwmWrite(LedPinGreen, g_val);
    softPwmWrite(LedPinBlue, 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 prototype of the function softPwmWrite(LedPinBlue, b\_val) is as follows:

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

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

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

```
ledColorSet(0xff,0x00,0x00);
```

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

## ➤ For Python Language Users

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

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

**Step 3:** Run.

```
sudo python 1.1.2_rgbLed.py
```

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

## Code

```
import RPi.GPIO as GPIO
import time
# 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}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    for i in pins:
```

```

GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.HIGH)

p_R = GPIO.PWM(pins['Red'], 2000)
p_G = GPIO.PWM(pins['Green'], 2000)
p_B = GPIO.PWM(pins['Blue'], 2000)
p_R.start(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

# Define a function to set up colors
def setColor(color):
    # configures the three LEDs' luminance with the inputted color value.
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0

    # Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

    # Change the colors
    p_R.ChangeDutyCycle(R_val)
    p_G.ChangeDutyCycle(G_val)
    p_B.ChangeDutyCycle(B_val)

    print ("color_msg: R_val = %s, G_val = %s, B_val = %s"%(R_val, G_val, B_val))

def main():
    while True:
        for color in COLOR:
            setColor(color)# change the color of the RGB LED
            time.sleep(0.5)

def destroy():
    # Stop all pwm channel

```

```

p_R.stop()
p_G.stop()
p_B.stop()
# Turn off all LEDs
GPIO.output(pins, GPIO.HIGH)
# Release resource
GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```

p_R = GPIO.PWM(pins['Red'], 2000)
p_G = GPIO.PWM(pins['Green'], 2000)
p_B = GPIO.PWM(pins['Blue'], 2000)

p_R.start(0)
p_G.start(0)
p_B.start(0)

```

Call the GPIO.PWM( )function to define Red, Green and Blue as PWM pins and set the frequency of PWM pins to 2000Hz, then Use the Start() function to set the initial duty cycle to zero.

```

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

```

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

```

def setColor(color):
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0

```

Configures the three LEDs' luminance with the inputted color value, assign the first two values of the hexadecimal to R\_val, the middle two assigned to G\_val, the last two values to B\_val. For instance, if color=0xFF00FF, R\_val= (0xFF00FF & 0x000000) >> 16 = 0xFF, G\_val = 0x00, B\_val=0xFF.

```
R_val = MAP(R_val, 0, 255, 0, 100)
G_val = MAP(G_val, 0, 255, 0, 100)
B_val = MAP(B_val, 0, 255, 0, 100)
```

Use map function to map the R,G,B value among 0~255 into PWM duty cycle range 0-100.

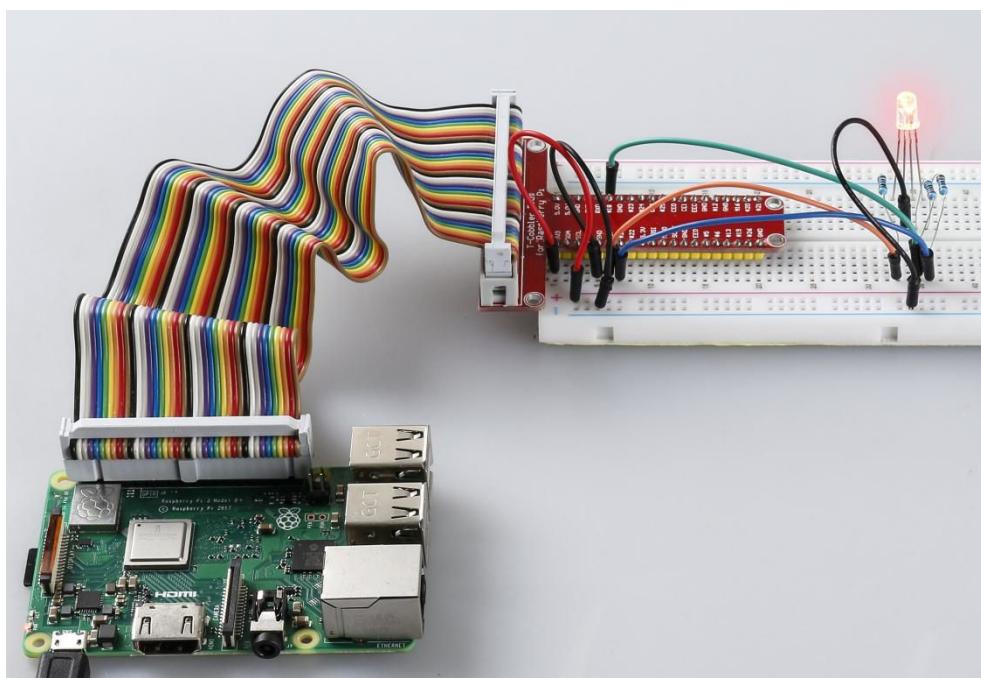
```
p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

Assign the mapped duty cycle value to the corresponding PWM channel to change the luminance.

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

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

## Phenomenon Picture



### 1.1.3 LED Bar Graph

## Introduction

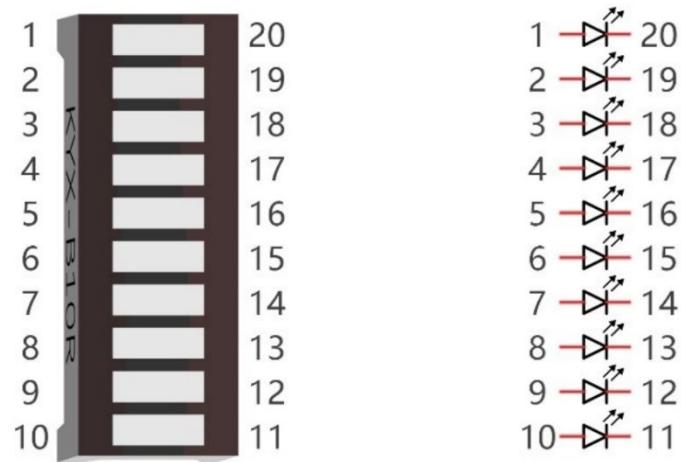
In this project, we sequentially illuminate the lights on the LED Bar Graph.

# Components

# Principle

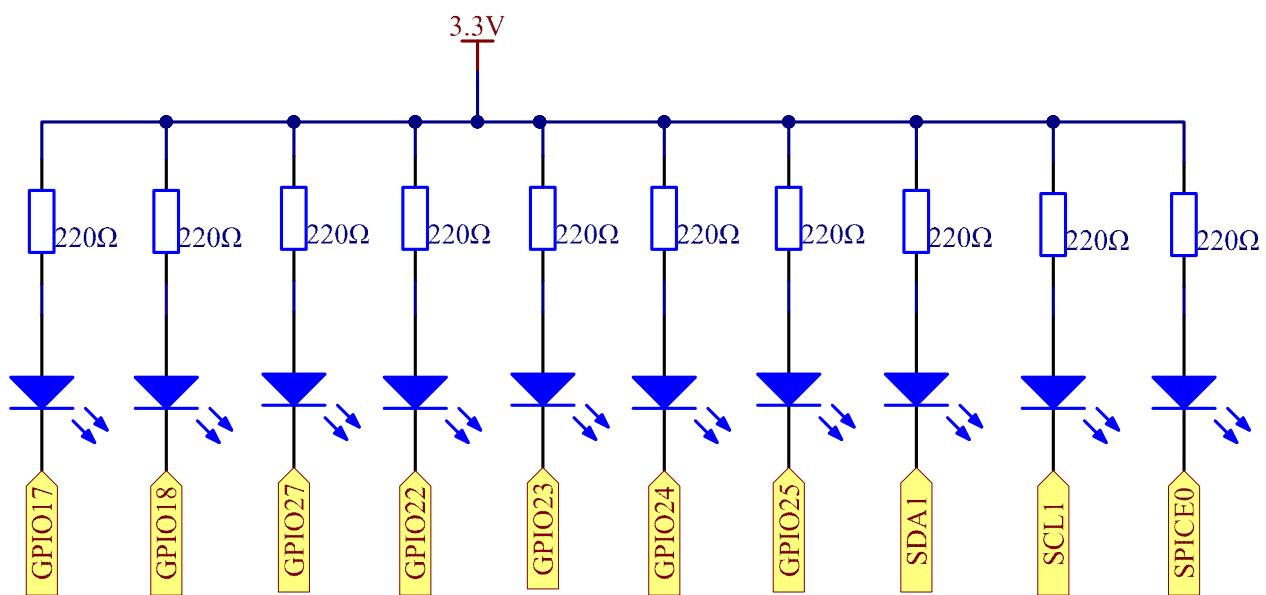
# LED Bar Graph

LED Bar Graph is an LED array, which is used to connect with electronic circuit or microcontroller. It's easy to connect LED bar graph with the circuit like as connecting 10 individual LEDs with 10 output pins. Generally we can use the LED bar graph as a Battery level Indicator, Audio equipments, and Industrial Control panels. There are many other applications of LED bar graphs.



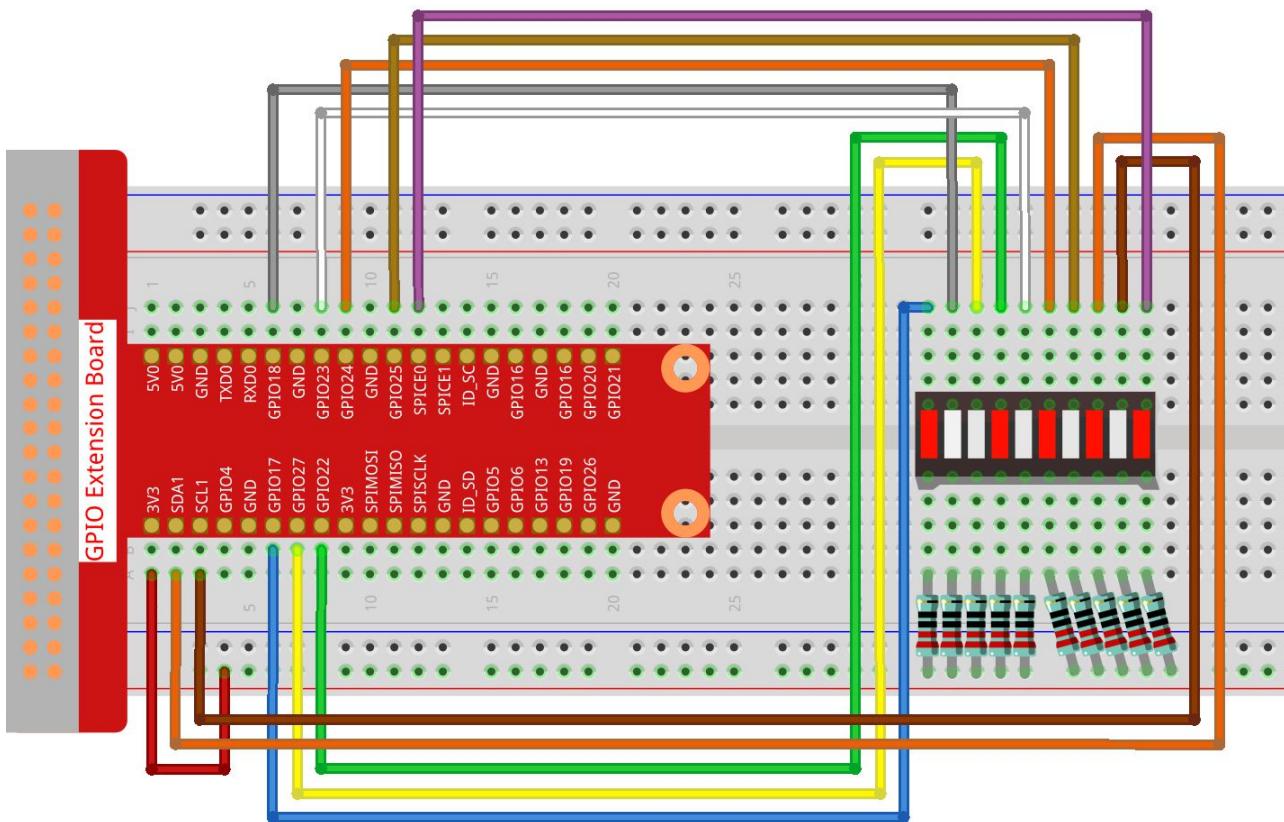
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3	8	2
SCL1	Pin 5	9	3
CE0	Pin 24	10	8



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

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

```
cd ~/davinci-kit-for-raspberry-pi/c/1.1.3_LedBarLed
```

#### Step 3: Compile the code.

```
gcc 1.1.3_LedBarLed.c -lwiringPi
```

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

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

After the code runs, you will see the LEDs on the LED bar lit one by one from left to right, and then lit one by one from right to left.

### Code

```
#include <wiringPi.h>
#include <stdio.h>
int pins[10] = {0,1,2,3,4,5,6,8,9,10};
int main(void)
{
```

```

int i;
if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
    printf("setup wiringPi failed !");
    return 1;
}
for(i=0;i<=10;i++){    //make led pins' mode is output
    pinMode(pins[i], OUTPUT);
}
while(1){
    for(i=0;i<=10;i++){  //make led on from left to right
        if(i==7){continue;} //skip pin 7
        digitalWrite(i, LOW);
        delay(100);
        digitalWrite(i, HIGH);
    }
    for(i=10;i>-1;i--){ //make led on from right to left
        if(i==7){continue;}
        digitalWrite(i, LOW);
        delay(100);
        digitalWrite(i, HIGH);
    }
}
return 0;
}

```

## Code Explanation

```

for(i=0;i<=10;i++){    //make led pins' mode is output
    pinMode(pins[i], OUTPUT);
}

```

Employ a for loop to set the corresponding pins to OUTPUT, and the pin[] array stores the pin numbers that are connected to the LED bar graph.

```

for(i=0;i<=10;i++){  //make led on from left to right
    if(i==7){continue;} //skip pin 7
    digitalWrite(i, LOW);
    delay(100);
    digitalWrite(i, HIGH);
}

```

A for loop is employed to turn the lights on and off once on the LED bar graph, from left to right.

```
if(i==7){continue;}
```

In this case, we don't use pin7, instead we use continue; let's skip the case where i is 7.

**Note:** The **break** statement terminates the loop, whereas **continue** statement forces the next iteration of the loop.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 1.1.3_LedBarLed.py
```

After the code runs, you will see the LEDs on the LED bar lit one by one from left to right, and then lit one by one from right to left.

## Code

```
import RPi.GPIO as GPIO
import time

ledPins = [11, 12, 13, 15, 16, 18, 22, 3, 5, 24]

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    for pin in ledPins:
        GPIO.setup(pin, GPIO.OUT)  # Set all ledPins' mode is output
        GPIO.output(pin, GPIO.HIGH) # Set all ledPins to high(+3.3V) to off led

def loop():
    while True:
        for pin in ledPins:    #make led on from left to right
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)
        for pin in ledPins[10:0:-1]:   #make led on from right to left
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
```

```

GPIO.output(pin, GPIO.HIGH)

def destroy():
    for pin in ledPins:
        GPIO.output(pin, GPIO.HIGH)    # turn off all leds
    GPIO.cleanup()                  # Release resource

if __name__ == '__main__':    # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    destroy()

```

## Code Explanation

```

def setup():
    GPIO.setmode(GPIO.BOARD)
    for pin in ledPins:
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.HIGH)

```

Set the numbering mode of GPIO to BOARD, and employ a for loop to set the corresponding pins to "output". By the way, the array ledPins stores the numbering that is connected to the pins of LED bar graph.

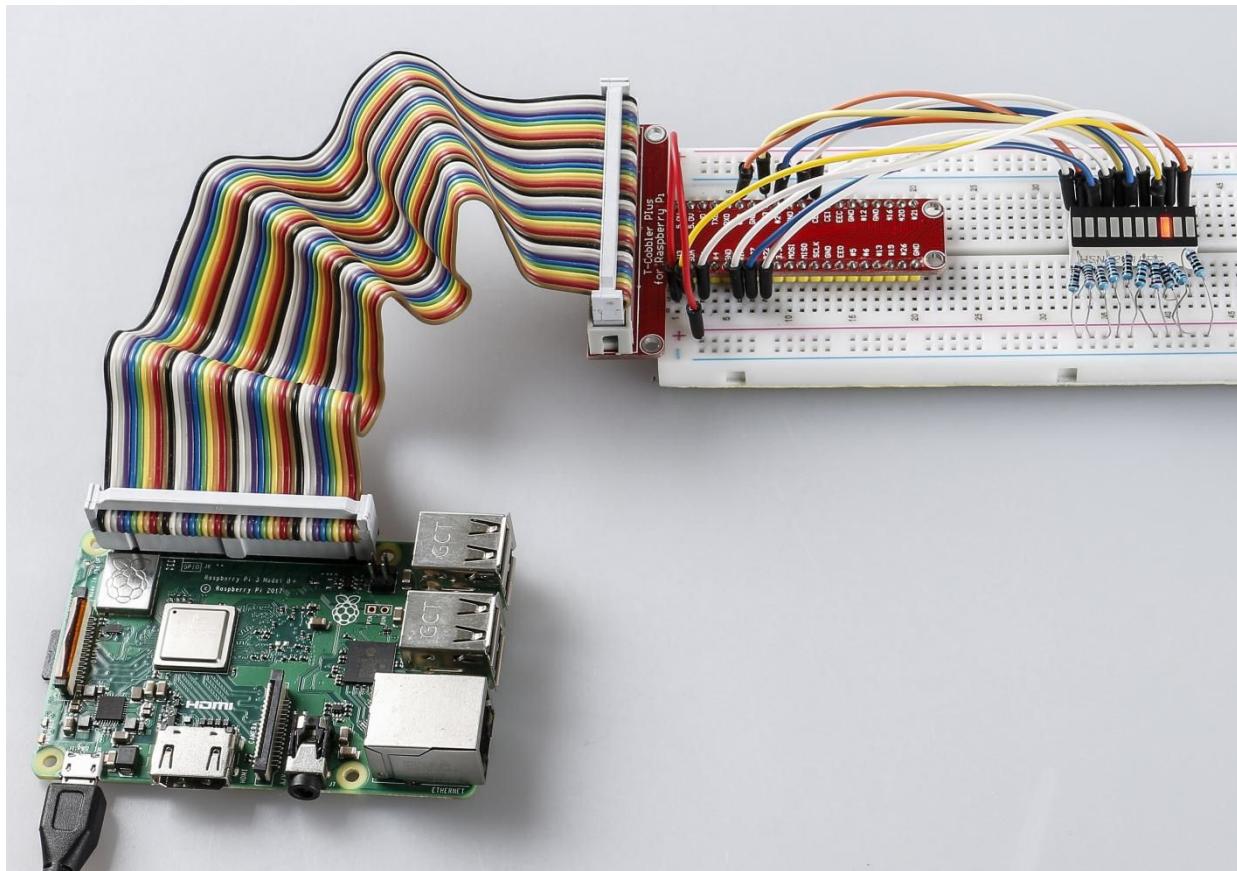
```

for pin in ledPins:
    GPIO.output(pin, GPIO.LOW)
    time.sleep(0.1)
    GPIO.output(pin, GPIO.HIGH)

```

A for loop is used to turn the lights on and off once on the LED bar graph, from left to right.

## Phenomenon Picture



## 1.1.4 7-segment Display

### Introduction

Let's try to drive a 7-segment display to show a figure from 0 to 9 and A to F.

### Components

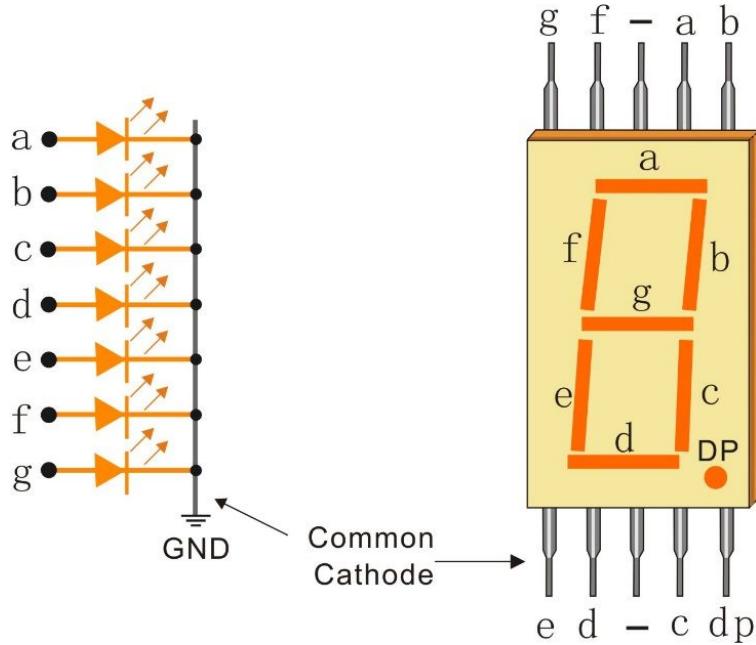
1 * Raspberry Pi	1 * T-Extension Board	1 * 7-segment display
	 GPIO Extension Board Pinout: 3V3 SVO SDA1 SVO SCL1 GND GPIOD4 TXD0 GND RXD0 GPIOD17 GPIO18 GPIOD27 GND GPIOD22 GPIO23 GND GPIO24 SPIMOSI GND SPIMISO GPIO25 SPISCLK SPICE0 GND ID_SD GPIOD5 ID_SC GPIOD6 GND GPIOD13 GPIO16 GPIOD19 GPIO16 GPIOD26 GPIO20 GND GPIO21	
1 * 40-pin Cable	1 * Resistor(220Ω)	Several Jumper Wires
1 * Breadboard	1 * 74HC595	

### Principle

#### 7-Segment Display

A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral to be displayed.

There are two types of pin connection: Common Cathode (CC) and Common Anode (CA). As the name suggests, a CC display has all the cathodes of the 7 LEDs connected when a CA display has all the anodes of the 7 segments connected. In this kit, we use the former.



Each of the LEDs in the display is given a positional segment with one of its connection pins led out from the rectangular plastic package. These LED pins are labeled from "a" through to "g" representing each individual LED. The other LED pins are connected together forming a common pin. So by forward biasing the appropriate pins of the LED segments in a particular order, some segments will brighten and others stay dim, thus showing the corresponding character on the display.

## Display Codes

To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

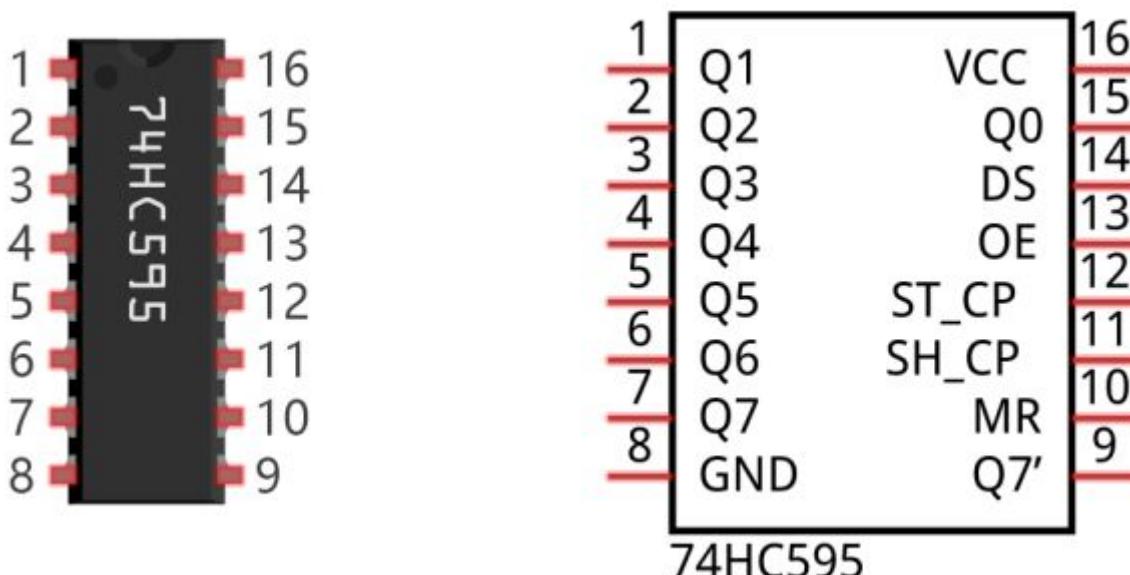
Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCB A	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39

3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d	.	10000000	0x80
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

## 74HC595

The 74HC595 consists of an 8 – bit shift register and a storage register with three – state parallel outputs. It converts serial input into parallel output so you can save IO ports of an MCU.

When MR (pin10) is high level and OE (pin13) is low level, data is input in the rising edge of SHcp and goes to the memory register through the rising edge of SHcp. If the two clocks are connected together, the shift register is always one pulse earlier than the memory register. There is a serial shift input pin (Ds), a serial output pin (Q) and an asynchronous reset button (low level) in the memory register. The memory register outputs a Bus with a parallel 8-bit and in three states. When OE is enabled (low level), the data in memory register is output to the bus.



## Pins of 74HC595 and their functions:

**Q0-Q7**: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.

**Q7'**: Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series

**MR**: Reset pin, active at low level;

**SHcp**: Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

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

**CE**: Output enable pin, active at low level.

**DS**: Serial data input pin

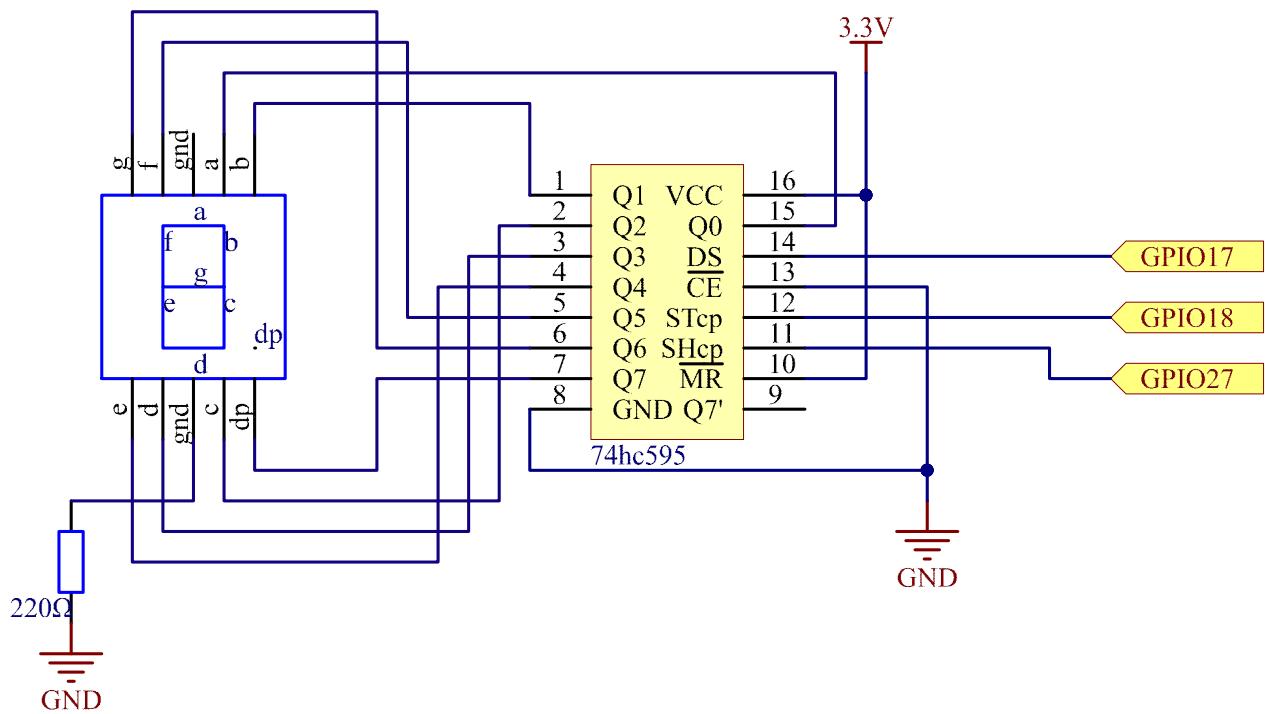
**VCC**: Positive supply voltage

**GND**: Ground

## Schematic Diagram

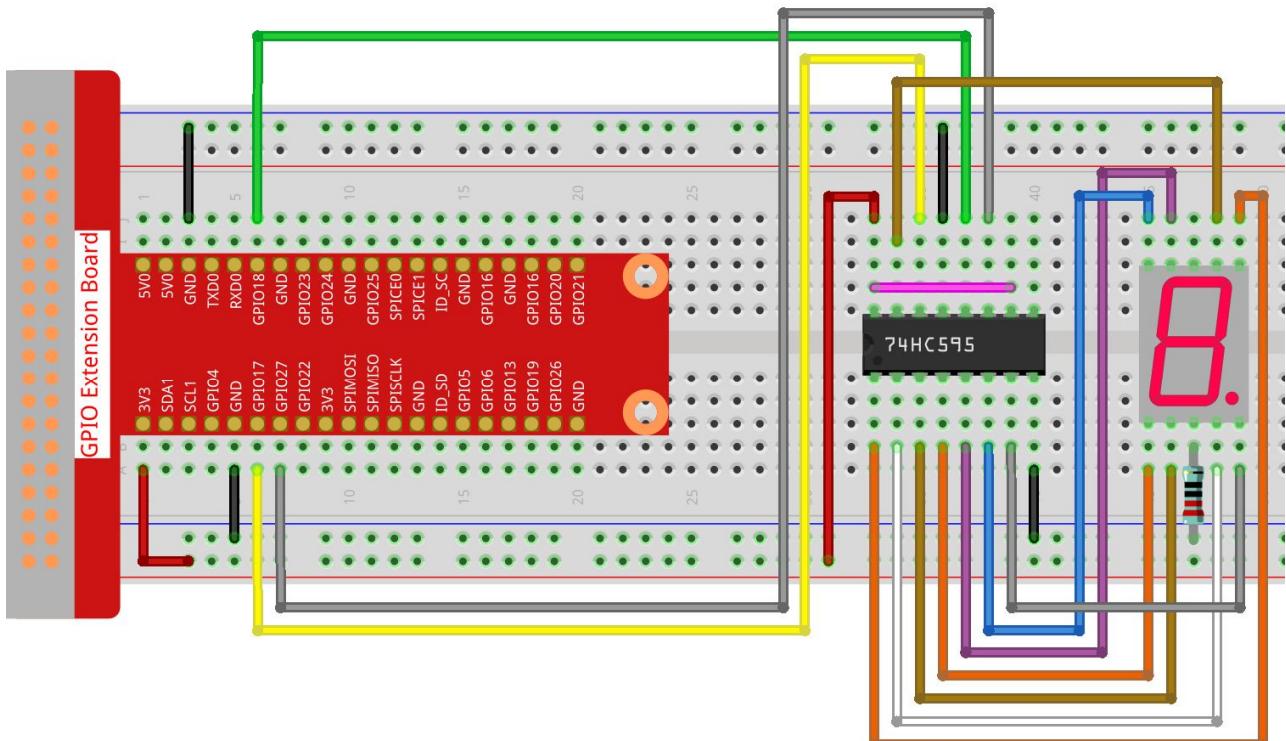
Connect pin ST\_CP of 74HC595 to Raspberry Pi GPIO18, SH\_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH\_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST\_CP (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.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.4
```

**Step 3:** Compile.

```
gcc 1.1.4_7-Segment.c -lwiringPi
```

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

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

After the code runs, you'll see the 7-segment display display 0-9, A-F.

## Code

```
#include <wiringPi.h>
#include <stdio.h>

#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)
unsigned char SegCode[17] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80};

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

void hc595_shift(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
    }
    digitalWrite(RCLK, 1);
    delay(1);
}
```

```

    digitalWrite(RCLK, 0);
}

int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<17;i++){
            printf("Print 0x%1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
    return 0;
}

```

## Code Explanation

```

unsigned char SegCode[17] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80};

```

A segment code array from 0 to F in Hexadecimal (Common cathode).

```

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

```

Set ds, st\_cp, sh\_cp three pins to OUTPUT, and the initial state as 0.

```
void hc595_shift(unsigned char dat){}
```

To assign 8 bit value to 74HC595's shift register.

```
digitalWrite(SDI, 0x80 & (dat << i));
```

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when i=2, 0x3f will shift left(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true.

```
digitalWrite(SRCLK, 1);
```

SRCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge pulse, then shift the DS date to shift register.

```
digitalWrite(RCLK, 1);
```

RCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge, then shift data from shift register to storage register.

```
while(1){
    for(i=0;i<17;i++){
        printf("Print 0x%1X on Segment\n", i); // %X means hex output
        hc595_shift(SegCode[i]);
        delay(500);
    }
}
```

In this for loop, we use "0x%1X" to output i as a hexadecimal number. Apply i to find the corresponding segment code in the SegCode[] array, and employ hc595\_shift() to pass the SegCode into 74HC595's shift register.

## ➤ For Python Language Users

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

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

**Step 3:** Run.

```
sudo python 1.1.4_7-Segment.py
```

After the code runs, you'll see the 7-segment display display 0-9, A-F.

## Code

```
import RPi.GPIO as GPIO
import time

# Set up pins
```

```

SDI = 17
RCLK = 18
SRCLK = 27

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

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)

# 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)

def main():
    while True:
        # Shift the code one by one from segCode list
        for code in segCode:
            hc595_shift(code)
            print ("segCode[%s]: 0x%02X"%(segCode.index(code), code)) # %02X means double
digit HEX to print
            time.sleep(0.5)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        main()
    
```

```
except KeyboardInterrupt:  
    destroy()
```

## Code Explanation

```
segCode = [0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]
```

A segment code array from 0 to F in Hexadecimal (Common cathode).

```
def setup():  
    GPIO.setmode(GPIO.BCM)  
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)  
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)  
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
```

Set ds, st\_cp, sh\_cp three pins to output and the initial state as low level.

```
GPIO.output(SDI, 0x80 & (dat << bit))
```

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when bit=2, 0x3f will shift right(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true.

```
GPIO.output(SRCLK, GPIO.HIGH)
```

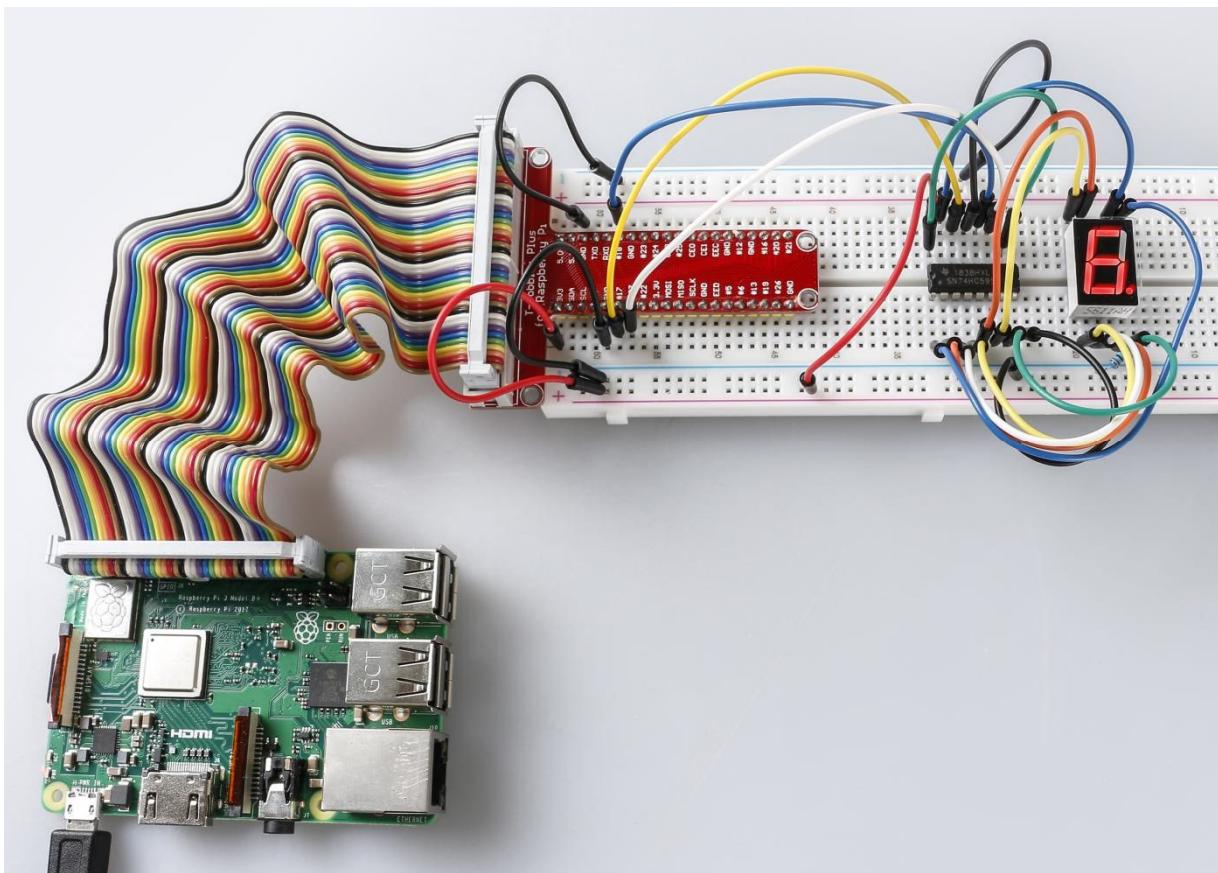
SRCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge pulse, then shift the DS date to shift register.

```
GPIO.output(RCLK, GPIO.HIGH)
```

RCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge, then shift data from shift register to storage register.

**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)

## Phenomenon Picture

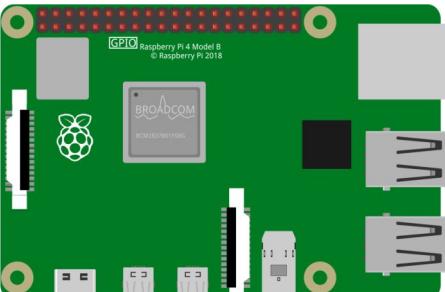
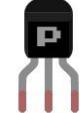
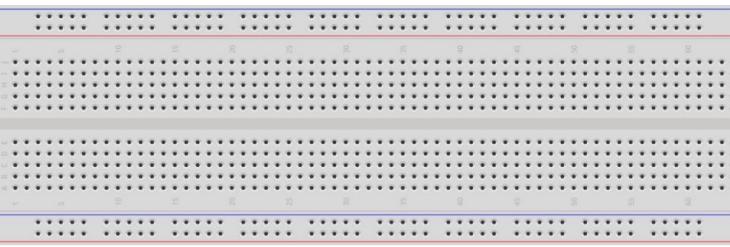


## 1.1.5 4-Digit 7-Segment Display

### Introduction

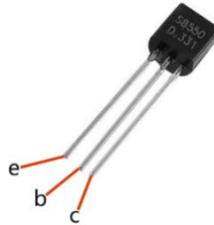
Now, let's try to control 4-digit 7-segment display.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * 4-Digit 7-Segment Display																																								
	 <p>GPIO Extension Board</p> <table border="1"> <tbody> <tr><td>3V3</td><td>5V0</td></tr> <tr><td>SDA1</td><td>GND</td></tr> <tr><td>SCL1</td><td>TX00</td></tr> <tr><td>GPIO4</td><td>RX00</td></tr> <tr><td>GND</td><td>GPIO18</td></tr> <tr><td>GPIO17</td><td>GND</td></tr> <tr><td>GPIO27</td><td>GPIO22</td></tr> <tr><td>GPIO22</td><td>GPIO23</td></tr> <tr><td>3V3</td><td>GPIO24</td></tr> <tr><td>SPIIMOSI</td><td>GND</td></tr> <tr><td>SPIIMOSO</td><td>GPIO25</td></tr> <tr><td>SPIISCLK</td><td>SPICE0</td></tr> <tr><td>GND</td><td>SPICE1</td></tr> <tr><td>ID_SD</td><td>ID_SC</td></tr> <tr><td>GPIO5</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td></tr> <tr><td>GPIO13</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO16</td></tr> <tr><td>GPIO26</td><td>GPIO20</td></tr> <tr><td>GND</td><td>GPIO21</td></tr> </tbody> </table>	3V3	5V0	SDA1	GND	SCL1	TX00	GPIO4	RX00	GND	GPIO18	GPIO17	GND	GPIO27	GPIO22	GPIO22	GPIO23	3V3	GPIO24	SPIIMOSI	GND	SPIIMOSO	GPIO25	SPIISCLK	SPICE0	GND	SPICE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	
3V3	5V0																																									
SDA1	GND																																									
SCL1	TX00																																									
GPIO4	RX00																																									
GND	GPIO18																																									
GPIO17	GND																																									
GPIO27	GPIO22																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPIIMOSI	GND																																									
SPIIMOSO	GPIO25																																									
SPIISCLK	SPICE0																																									
GND	SPICE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
1 * 40-pin Cable		4 * PNP Transistor																																								
																																										
1 * Breadboard		1 * 74HC595																																								
																																										
		Several Jumper Wires																																								
																																										
		8 * Resistor(220Ω)																																								
																																										
		4 * Resistor 1KΩ																																								
																																										

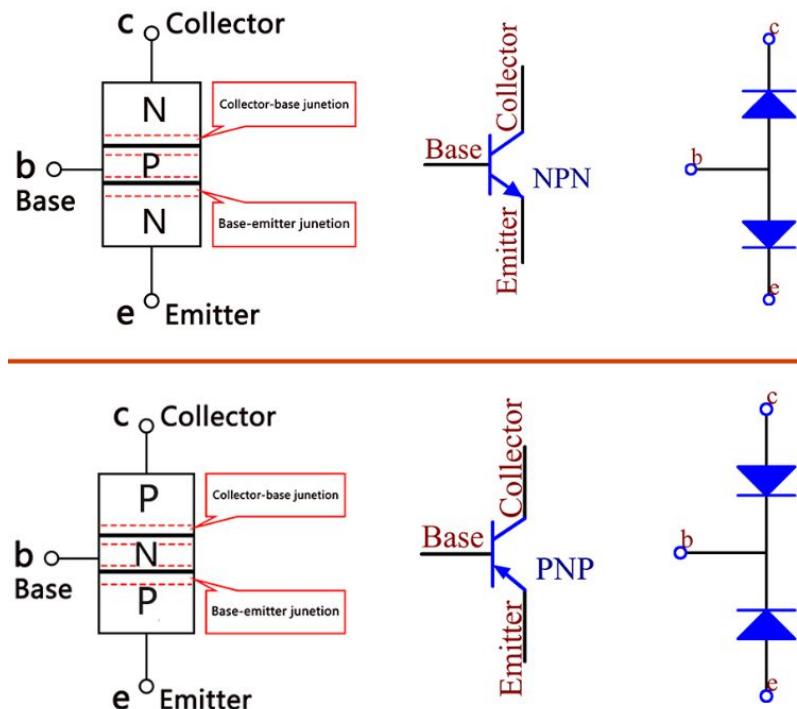
## Principle

### 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.

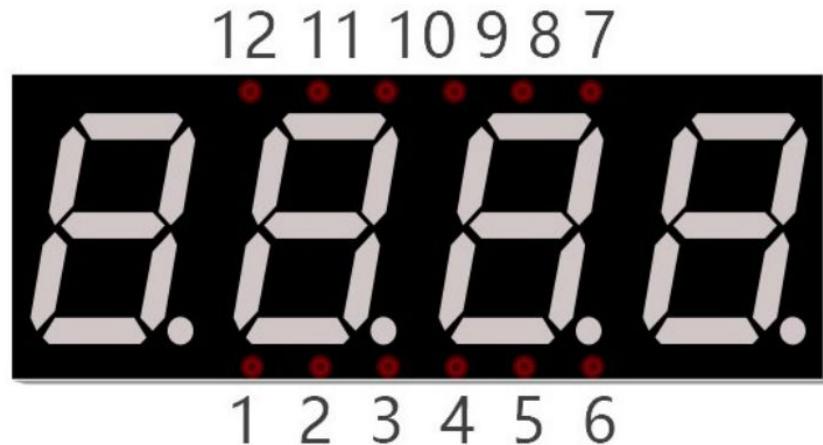
From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The arrow in the circuit symbol indicates the direction of emitter junction. Transistors can be divided 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.

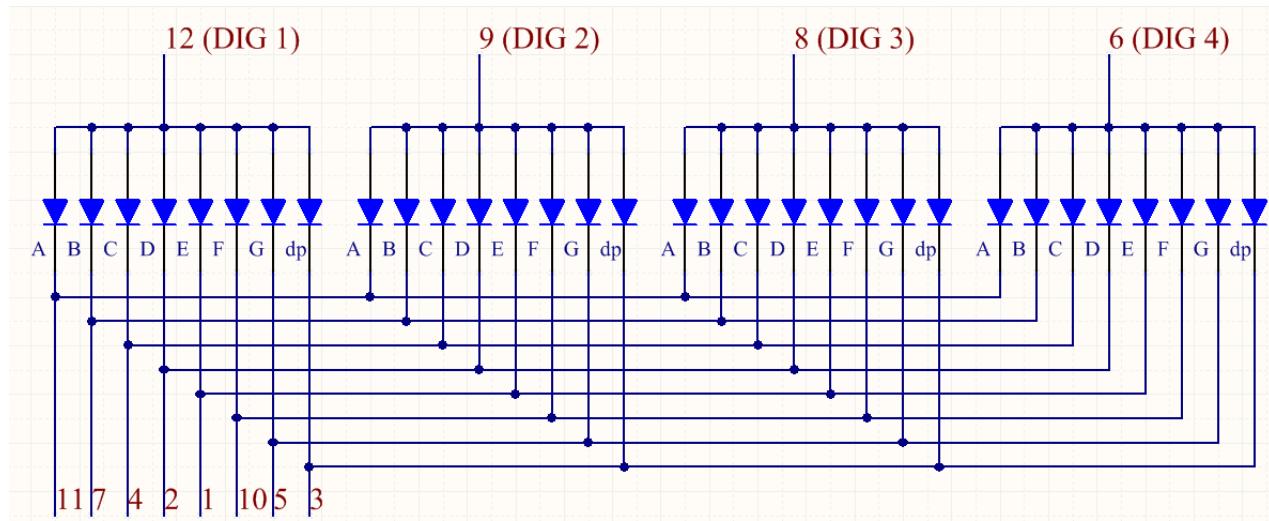
## 4-Digit 7-Segment Display

4-Digit 7-segment display consists of four 7- segment displays working together.



The 4-digital 7-segment display works independently. It uses the principle of human visual persistence to quickly display the characters of each 7-segment in a loop to form continuous strings.

For example, when "1234" is displayed on the display, "1" is displayed on the first 7-segment, and "234" is not displayed. After a period of time, the second 7-segment shows "2", the 1st 3th 4th of 7-segment does not show, and so on, the four digital display show in turn. This process is very short (typically 5ms), and because of the optical afterglow effect and the principle of visual residue, we can see four characters at the same time.



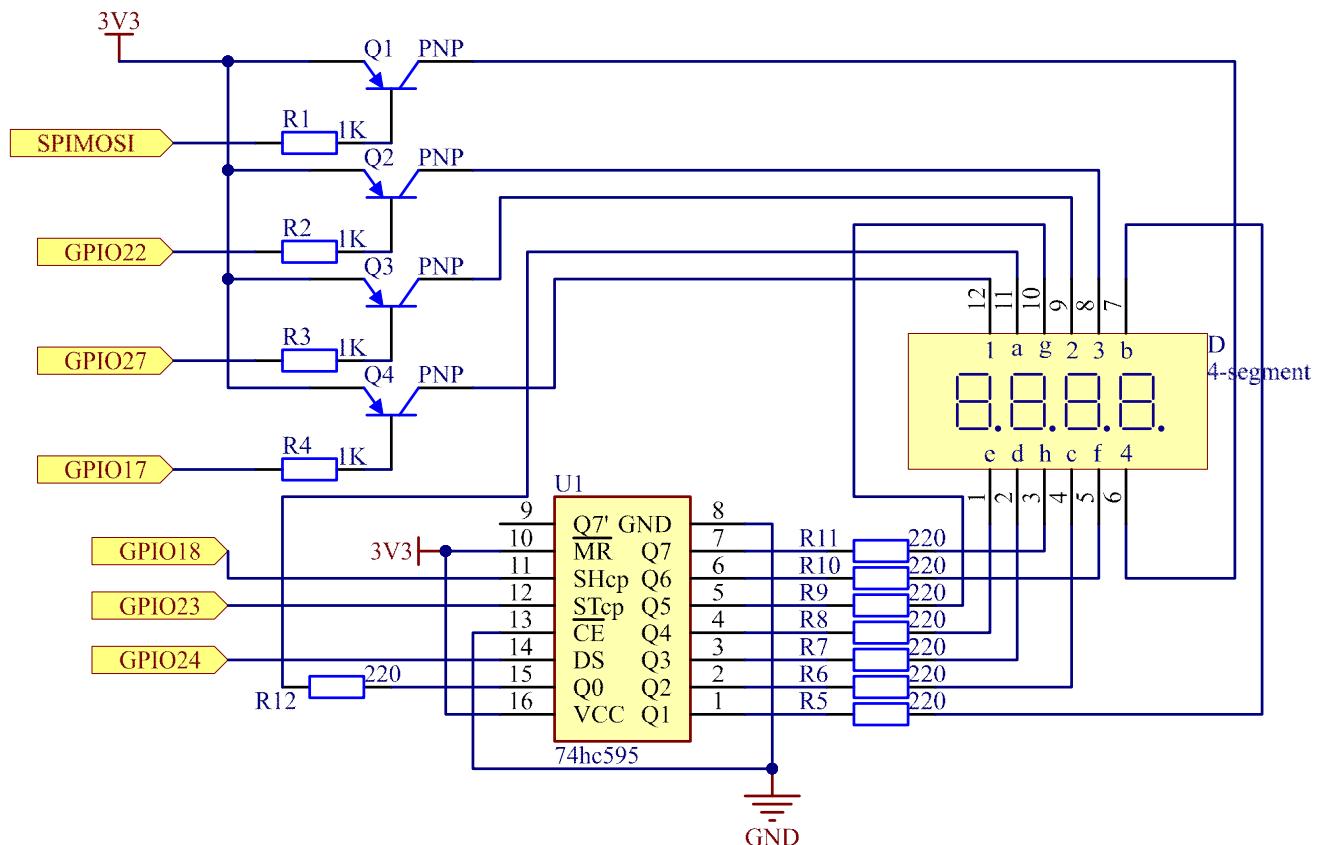
## Display Codes

To help you get to know how 7-segment displays(Common Anode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 11000000 means that DP and G are set to 1, while others are set to 0. Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Numbers	Common Anode		Numbers	Common Anode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCB A	Hex Code
0	11000000	0xc0	A	10001000	0x88
1	11111001	0xf9	B	10000011	0x83
2	10100100	0xa4	C	11000110	0xc6
3	10110000	0xb0	D	10100001	0xa1
4	10011001	0x99	E	10000110	0x86
5	10010010	0x92	F	10001110	0x8e
6	10000010	0x82	.	01111111	0x7f
7	11111000	0xf8			
8	10000000	0x80			
9	10010000	0x90			

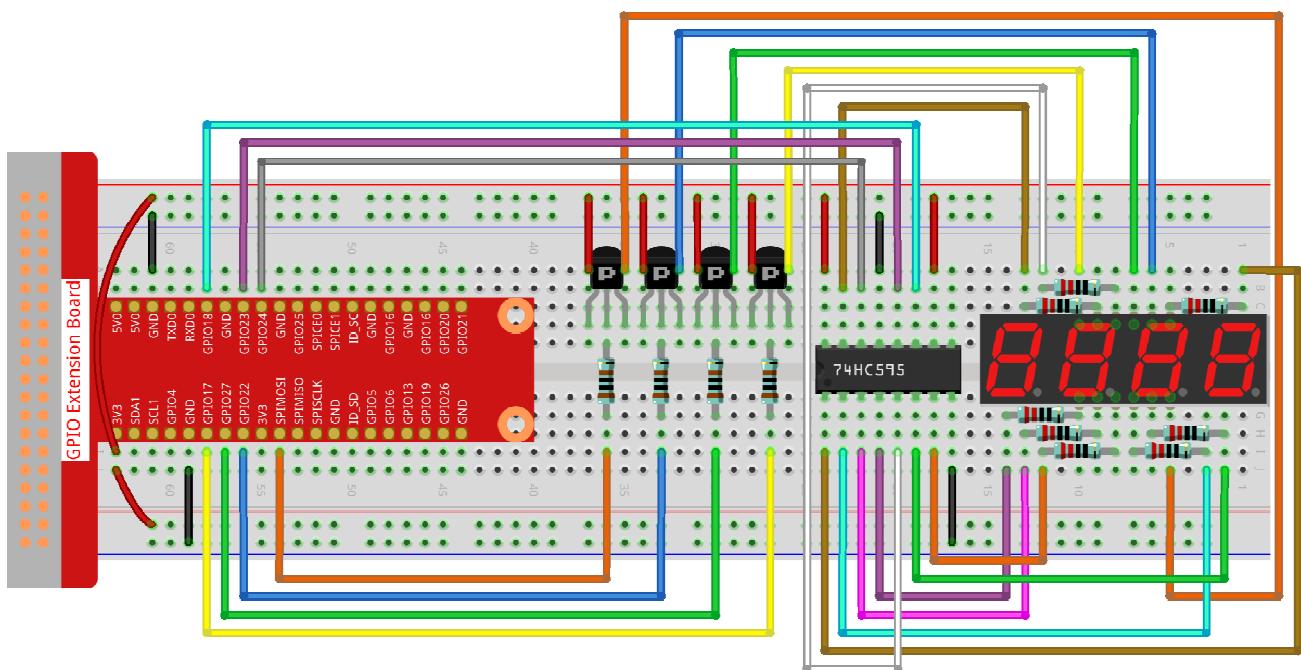
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.5/
```

**Step 3:** Compile the code.

```
gcc 1.1.5_4-Digit.c -lwiringPi
```

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

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

After the code runs, the program takes a count, increasing by 1 per second, and the 4-digit 7-segment display displays the count.

## Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define SDI 5 //serial data input(DS)
#define RCLK 4 //memory clock input(STCP)
#define SRCLK 1 //shift register clock input(SHCP)
const int placePin[]={0,2,3,12}; // Define 4 digit's common pin.

// A segment code array from 0 to 9 in Hexadecimal (Common anode)
unsigned char number[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

int counter = 0;//the number will be displayed.

//select the place of the value, this is only one place that will be enable each time.
//the parameter digit is optional for 0001,0010,0100,1000
void selectPlace(int digit){
    digitalWrite(placePin[0],!(digit&0x08));
    digitalWrite(placePin[1],!(digit&0x04));
    digitalWrite(placePin[2],!(digit&0x02));
    digitalWrite(placePin[3],!(digit&0x01));
}

void hc595_shift(int8_t data){//To assign 8 bit value to 74HC595's shift register
```

```

int i;
for(i = 0; i < 8; i++){
    digitalWrite(SDI, 0x80 & (dat << i));
    digitalWrite(SRCLK,1);
    delay(1);
    digitalWrite(SRCLK,0);
}
digitalWrite(RCLK,1);
delay(1);
digitalWrite(RCLK,0);

}

void display(int counter){ //display the number
hc595_shift(0xff);
selectPlace(0x01);
hc595_shift(number[counter%10]);
//display the number on the single digit of the value
delay(1);

hc595_shift(0xff);
selectPlace(0x02);
hc595_shift(number[counter%100/10]);
delay(1);

hc595_shift(0xff);
selectPlace(0x04);
hc595_shift(number[counter%1000/100]);
delay(1);

hc595_shift(0xff);
selectPlace(0x08);
hc595_shift(number[counter%10000/1000]);
delay(1);
}

void timer(int sig){ //Timer function
if(sig == SIGALRM){ //If the signal is SIGALRM, the value of counter plus 1, and update the
    number displayed by 7-segment display
    counter++;
}

```

```
alarm(1);      //set the next timer time
printf("counter : %d \n",counter);
}

}

int main(void)
{
if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
    printf("setup wiringPi failed !");
    return 1;
}
pinMode(SDI,OUTPUT);      //set the pin connected to 74HC595 for output mode
pinMode(RCLK,OUTPUT);
pinMode(SRCLK,OUTPUT);
//set the pin connected to 7-segment display common end to output mode
int i;
for(i=0;i<4;i++){
    pinMode(placePin[i],OUTPUT);
    digitalWrite(placePin[i],HIGH);
}
signal(SIGALRM,timer); //configure the timer
alarm(1);      //set the time of timer to 1s
while(1){
    display(counter); //display the number counter
}
return 0;
}
```

## Code Explanation

```
const int placePin[]={0,2,3,12};
```

These four pins control the common anode pins of the four-digit 7-segment displays.

```
unsigned char number[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
```

A segment code array from 0 to 9 in Hexadecimal (Common anode).

```
void selectPlace(int digit){
    digitalWrite(placePin[0],!(digit&0x08));
    digitalWrite(placePin[1],!(digit&0x04));
    digitalWrite(placePin[2],!(digit&0x02));
    digitalWrite(placePin[3],!(digit&0x01));
}
```

Select the place of the value. there is only one place that should be enable each time. The enabled place will be written low. For example, if digit=0x01:

$!(0x01 \& 0x08) = 1$ ,

$!(0x01 \& 0x04) = 1$ ,

$!(0x01 \& 0x02) = 1$ ,

$!(0x01 \& 0x01) = 0$ .

Write 0 into **placePin[3](GPIO12)**, while the corresponding pins of placePin[] are connected to a PNP transistor,

Then the fourth common anode pin **DIG4** of the 4-digit segment display corresponding to placePin[3](GPIO12) will get a high level to light up the 4th 7-segment display.

Similarly, placePin[0]~[2] are written to 1. After the signal passes through the PNP transistor, these three corresponding common anode pins will get a low level and then the three 7-segment displays will not light up.

```
void display(int counter){ //display the number
    hc595_shift(0xff);
    selectPlace(0x01);
    hc595_shift(number[counter%10]);
    delay(1);
```

The function `display()` is used to set the number displayed on the 4-digit 7-segment display

**hc595\_shift(0xff)** : write in 11111111 to turn off these eight LEDs on 7-segment display so as to clear the displayed content.

**selectPlace(0x01)**: Light up the fourth 7-segment display.

**hc595\_shift(number[counter%10])** : the number in the single digit of counter will display on the forth segment.

```
signal(SIGALRM,timer);
```

This is a system-provided function, the prototype of code is:

```
sig_t signal(int signum,sig_t handler);
```

After executing the `signal()`, once the process receives the corresponding signum (in this case SIGALRM), it immediately pauses the existing task and processes the set function (in this case `timer(sig)`).

```
alarm(1);
```

This is also a system-provided function. The code prototype is

```
unsigned int alarm (unsigned int seconds);
```

It generates a SIGALRM signal after a certain number of seconds.

```
void timer(int sig){  
    if(sig == SIGALRM){  
        counter ++;  
        alarm(1);  
    }  
}
```

We use the functions above to implement the timer function.

After the `alarm()` generates the SIGALRM signal, the `timer` function is called. Add 1 to counter, and the function, `alarm(1)` will be repeatedly called after 1 second.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 1.1.5_4-Digit.py
```

After the code runs, the program takes a count, increasing by 1 per second, and the 4 digit display displays the count.

## Code

```
import RPi.GPIO as GPIO
import time
import threading

#define the pins connect to 74HC595
SDI = 18    #serial data input(DS)
RCLK = 16    #memory clock input(STCP)
SRCLK = 12    #shift register clock input(SHCP)
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
placePin = (11,13,15,19)
counter = 0      # the number will be displayed
t = 0            # define the Timer object

def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(SDI, GPIO.OUT)    # Set pin mode to output
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    for pin in placePin:
        GPIO.setup(pin,GPIO.OUT)

def hc595_shift(dat):    #shift the data to 74HC595
    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)

def selectPlace(digit): # select the place of the value
    GPIO.output(placePin[0],GPIO.LOW if (digit&0x08) else GPIO.HIGH)
    GPIO.output(placePin[1],GPIO.LOW if (digit&0x04) else GPIO.HIGH)
    GPIO.output(placePin[2],GPIO.LOW if (digit&0x02) else GPIO.HIGH)
    GPIO.output(placePin[3],GPIO.LOW if (digit&0x01) else GPIO.HIGH)

def display(counter): #display the number
    hc595_shift(0xff) #clean up the display
    selectPlace(0x01) #Select place
    hc595_shift(number[counter%10]) #display the number on the single digit of the value
    time.sleep(0.003)
    hc595_shift(0xff)
    selectPlace(0x02)
    hc595_shift(number[counter%100//10])
    time.sleep(0.003)
    hc595_shift(0xff)
    selectPlace(0x04)
    hc595_shift(number[counter%1000//100])
    time.sleep(0.003)
    hc595_shift(0xff)
    selectPlace(0x08)
    hc595_shift(number[counter%10000//1000])
    time.sleep(0.003)

def timer(): #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer) #set the time again
    t.start() #Start timing
    counter+=1
    print ("counter : %d"%counter)

def loop():
    global t
    global counter
    t = threading.Timer(1.0,timer) #set the timer
    t.start() # Start timing

```

```

while True:
    display(counter)          # display the number counter

def destroy(): # When "Ctrl+C" is pressed, the function is executed.
    global t
    GPIO.cleanup()
    t.cancel()   #cancel the timer

if __name__ == '__main__': # Program starting from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
```

A segment code array from 0 to 9 in Hexadecimal (common anode).

```
placePin = (11,13,15,19) # Define the pin of 7-segment display common end
```

These four pins control the common anode of the 4-digit 7-segment display.

```

def selectPlace(digit): # select the place of the value
    GPIO.output(placePin[0],GPIO.LOW if (digit&0x08) else GPIO.HIGH)
    GPIO.output(placePin[1],GPIO.LOW if (digit&0x04) else GPIO.HIGH)
    GPIO.output(placePin[2],GPIO.LOW if (digit&0x02) else GPIO.HIGH)
    GPIO.output(placePin[3],GPIO.LOW if (digit&0x01) else GPIO.HIGH)

```

Select the place of the value. there is only one place that should be enable each time. The enabled place will be written low.

**GPIO.LOW if (digit&0x08) else GPIO.HIGH:** If digit&0x08 is equal to 1, it is GPIO.LOW; otherwise, it's GPIO.HIGH.

For instance, if digit=0x01 and the selectPlace(digital) function is run, only placePin[3](physical pin19) can get a low level that will let the PNP transistor activated, and then the fourth 7-segment display will get a high level which is to light up the fourth digit 7-segment display.

Similarly, placePin[0]~[2] are written to 1. After the signal passes through the PNP transistor, these three corresponding common anode pins will get a low level and then the three 7-segment displays will not light up.

```
def display(counter): #display the number
    hc595_shift(0xff) #clean up the display
    selectPlace(0x01) #Select place
    hc595_shift(number[counter%10]) #display the number on the single digit of the value
    time.sleep(0.003)
```

The display() function is used to set the number displayed on the 4-digit 7-segment Dispaly.

**hc595\_shift(0xff):** write in 0xff ( binary system: 1111111) so that the eight LEDs on the 7-segment Dispaly will turn off so as to clear the displayed content.

**selectPlace(0x01):** Light up the fourth 7-segment display .

**hc595\_shift(number[counter%10]):** the number in the single digit of counter will display on the forth segment.

```
t = threading.Timer(1.0,timer)
t.start()
```

The module, threading is the common threading module in Python, and Timer is the subclass of it.

The prototype of code is:

```
class threading.Timer(interval, function, args=[], kwargs={})
```

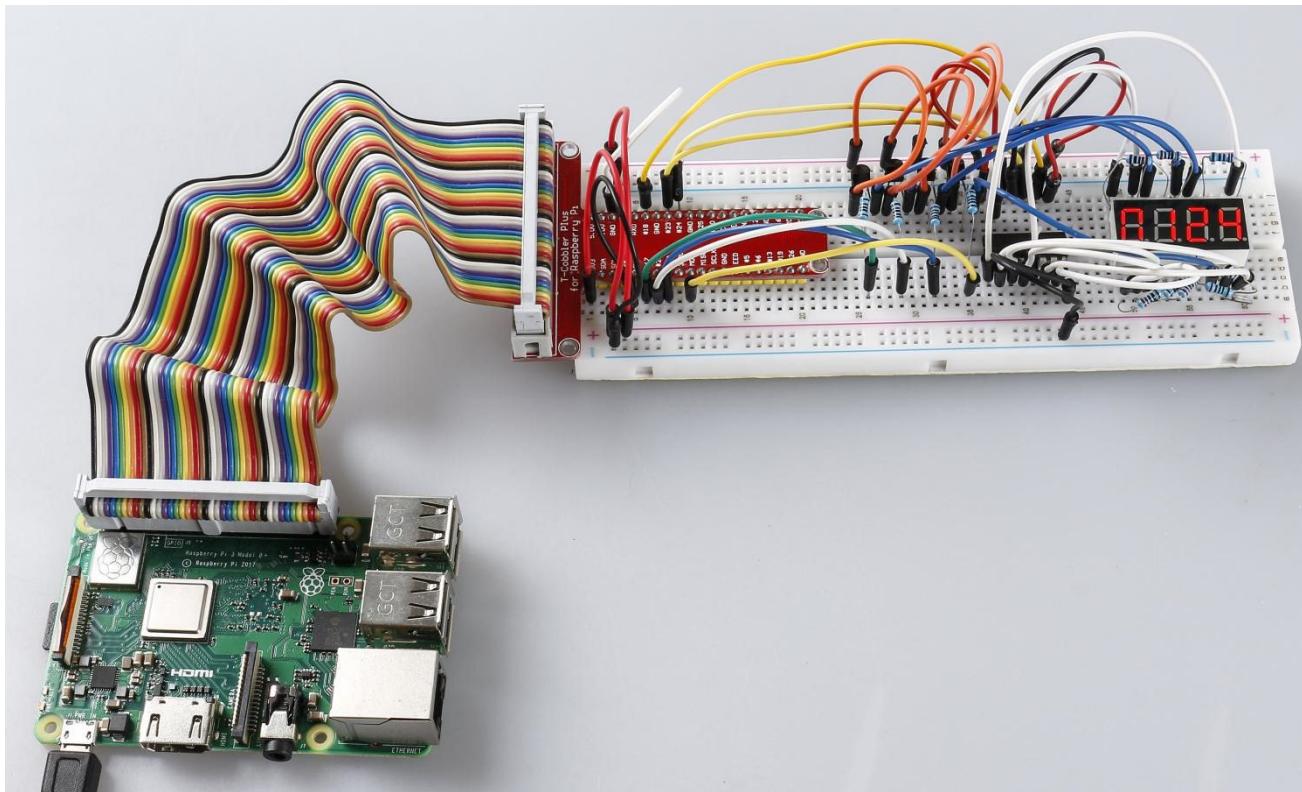
After the interval, the function will be run. Here, the interval is 1.0 , and the function is timer().

t.start () means the Timer will start at this point.

```
def timer(): #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer) #set the time again
    t.start() #Start timing
    counter+=1
```

After Timer reaches 1.0s, the Timer function is called; add 1 to counter, and the Timer is used again to execute itself repeatedly every second.

## Phenomenon Picture



## 1.1.6 LED Dot Matrix

## 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

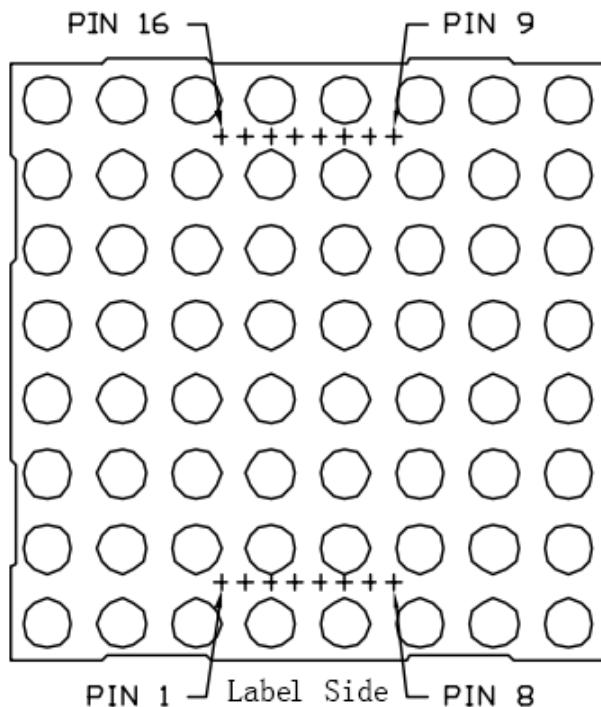
## Principle

### LED Dot Matrix

Generally, LED 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.

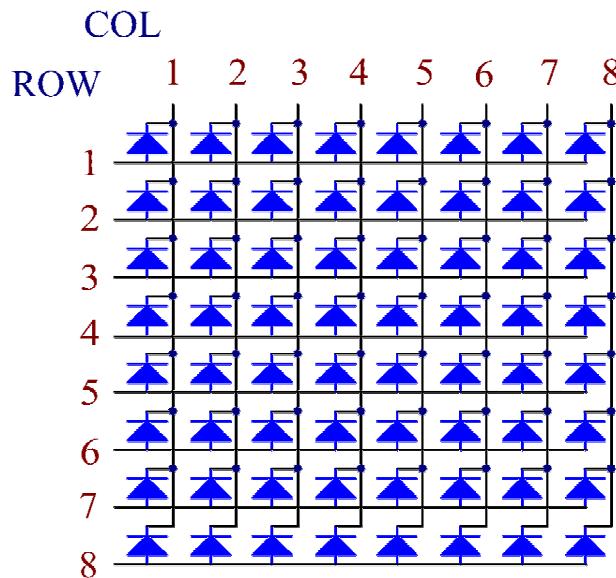
See the figure below. The pins are arranged at the two ends at the back. Take the label side for reference: pins on this end are pin 1-8, and oh the other are pin 9-16.

The external view:



Below the figures show their internal structure. You can see in a CA LED dot 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 LED dot 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:



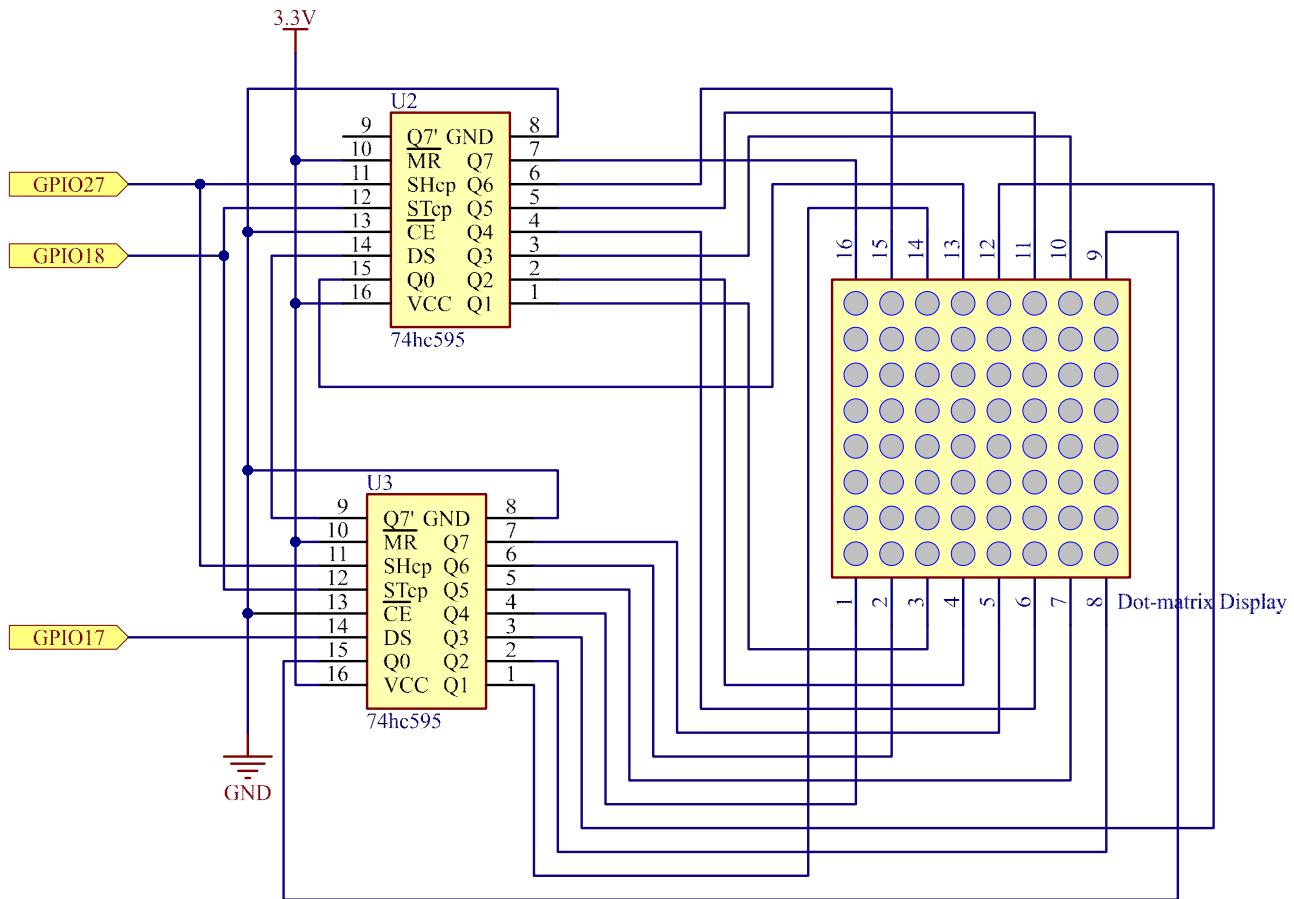
Pin numbering corresponding to the above rows and columns:

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 addition, two 74HC595 chips are used here. One is to control the rows of the LED dot matrix while the other, the columns.

## Schematic Diagram

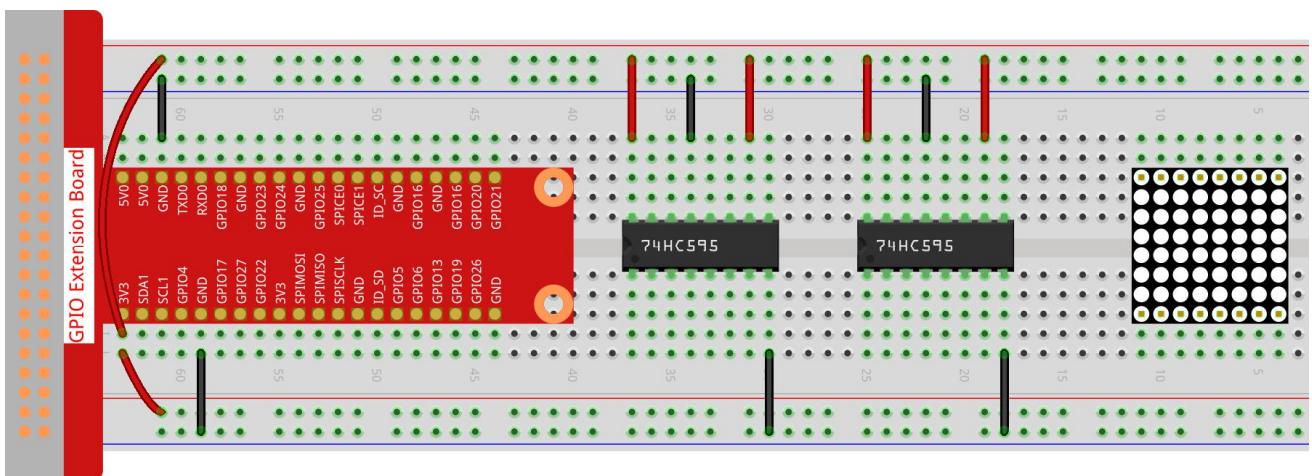
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPI MOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



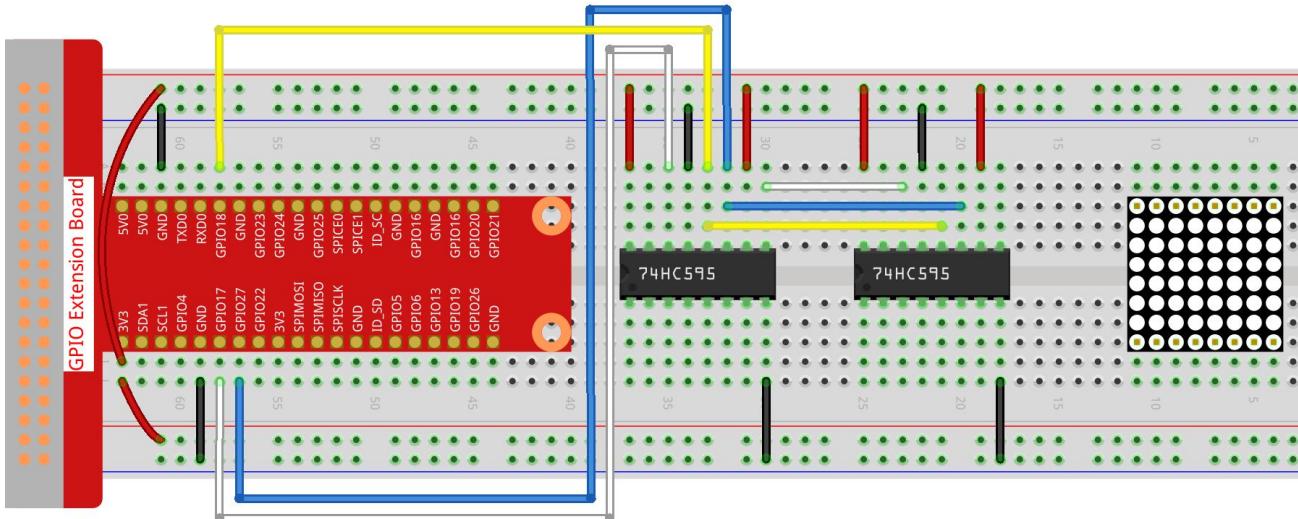
## Experimental Procedures

**Step 1:** Build the circuit. Since the wiring is complicated, let's make it step by step. First, insert the T-Cobbler, the LED dot matrix and two 74HC595 chips into breadboard. Connect the 3.3V 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, pin 13 and pin 8 to GND.

**Note:** In the Fritzing image above, the side with label is at the bottom.

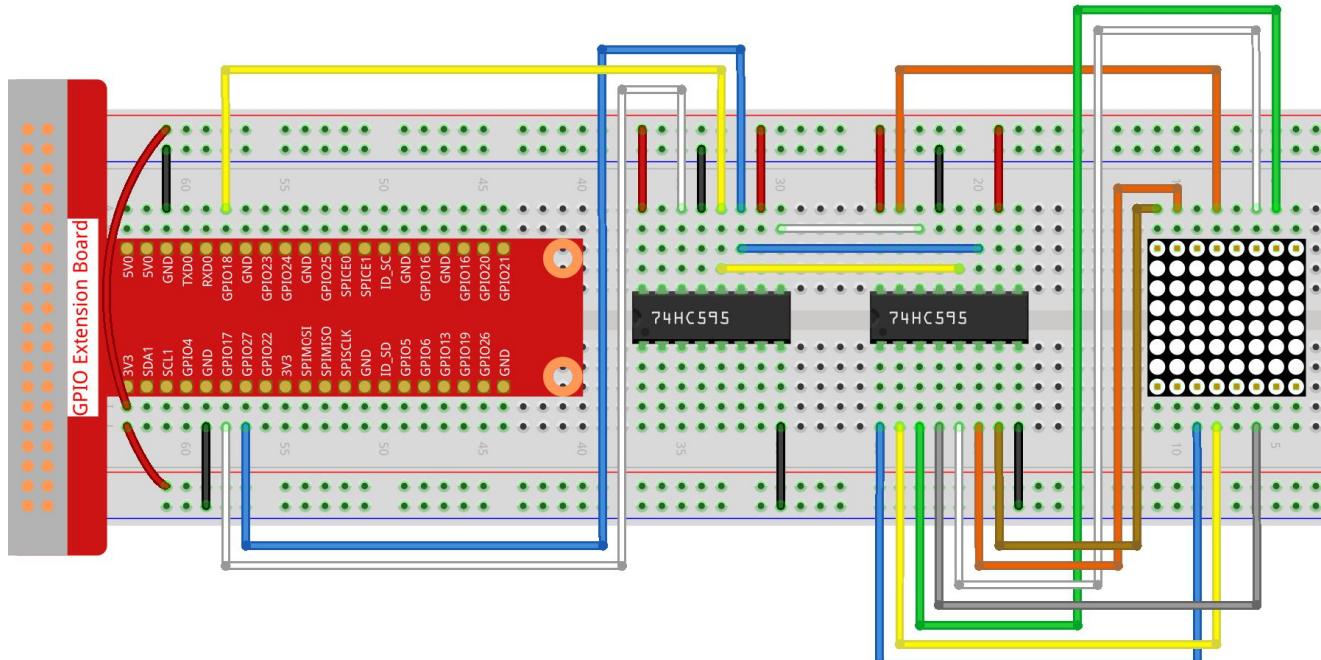


**Step 2:** Connect pin 11 of the two 74HC595 together, and then to GPIO27; then pin 12 of the two chips, and to GPIO18; next, pin 14 of the 74HC595 on the left side to GPIO17 and pin 9 to pin 14 of the second 74HC595.



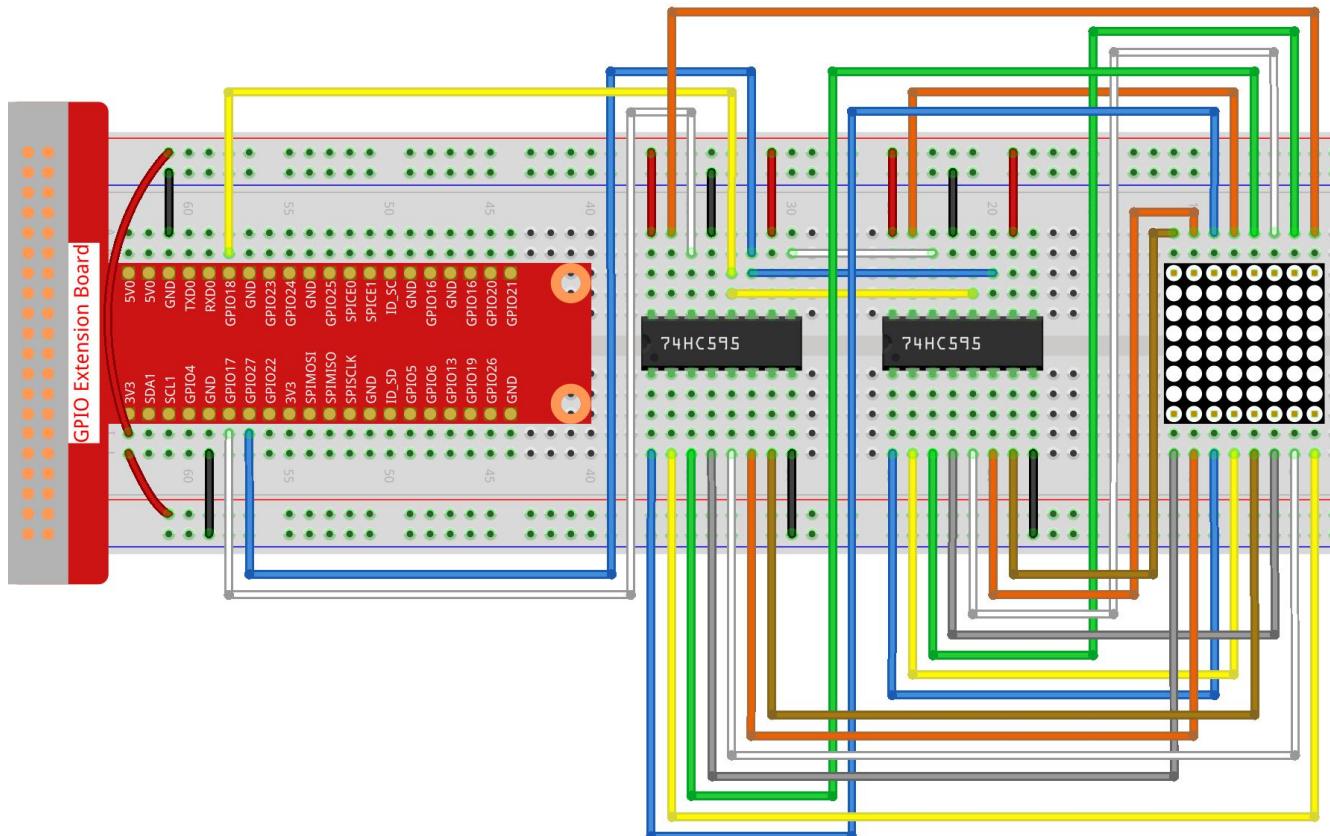
**Step 3:** The 74HC595 on the right side is to control columns of the LED dot matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	13	3	4	10	6	11	15	16



**Step 4:** Now connect the ROWs of the LED dot matrix. The 74HC595 on the left controls ROW of the LED dot 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.

74HC595	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LED Dot Matrix	9	14	8	12	1	7	2	5



## ➤ For C Language Users

**Step 5:** Go to the folder of code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.6/
```

**Step 6:** Compile.

```
gcc 1.1.6_LedMatrix.c -lwiringPi
```

**Step 7:** Run.

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

After the code runs, the LED dot matrix lights up and out row by row and column by column.

## Code

```
#include <wiringPi.h>
#include <stdio.h>

#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)

unsigned char code_H[20] =
{0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0xff};

unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);

    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}

void hc595_in(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
    }
}

void hc595_out(){
    digitalWrite(RCLK, 1);
    delay(1);
    digitalWrite(RCLK, 0);
}
```

```
int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<sizeof(code_H);i++){
            hc595_in(code_L[i]);
            hc595_in(code_H[i]);
            hc595_out();
            delay(100);
        }

        for(i[sizeof(code_H)];i>=0;i--){
            hc595_in(code_L[i]);
            hc595_in(code_H[i]);
            hc595_out();
            delay(100);
        }
    }

    return 0;
}
```

## Code Explanation

```
unsigned char code_H[20] =
{0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0xff};

unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x7f};
```

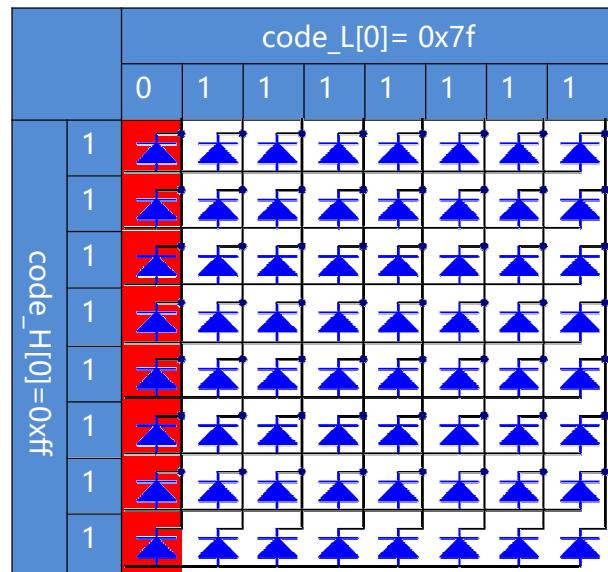
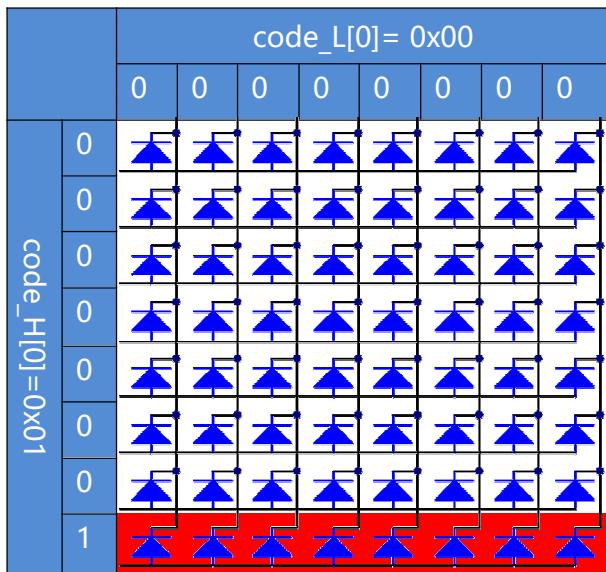
The array code\_H represents the elements of the LED dot matrix row, and the array code\_L refers to the elements of the column. When characters are displayed, an element in row and one in column are acquired and assigned to the two HC595 chips respectively. Thus a pattern is shown on the LED dot matrix.

Take the first number of code\_H, 0x01 and the first number of code\_L, 0x00 as examples.

0x01 converted to binary becomes 00000001; 0x00 converted to binary becomes 0000 0000.

In this kit, common anode LED dot matrix display is applied, so only the eight LEDs in the eighth row light up.

When the conditions that code\_H is 0xff and code\_L is 0x7f are met simultaneously, these 8 LEDs in the first column are lit.



```
void hc595_in(unsigned char dat){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, 0x80 & (dat << i));
        digitalWrite(SRCLK, 1);
        delay(1);
        digitalWrite(SRCLK, 0);
```

Write the value of dat to pin SDI of the HC595 bit by bit. SRCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge pulse, then shift the pinSDI(DS) date to shift register.

```
void hc595_out(){
    digitalWrite(RCLK, 1);
    delay(1);
    digitalWrite(RCLK, 0);
```

RCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge, then shift data from shift register to storage register.

```
while(1){
    for(i=0;i<sizeof(code_H);i++){
        hc595_in(code_L[i]);
        hc595_in(code_H[i]);
        hc595_out();
        delay(100);
    }
}
```

In this loop, these 20 elements in the two arrays, code\_L and code\_H will be uploaded to the two 74HC595 chip one by one. Then call the function, hc595\_out() to shift data from shift register to storage register.

## ➤ For Python Language Users

## **Step 5:** Get into the folder of code.

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

## **Step 6: Run.**

**sudo python 1.1.6\_LedMatrix.py**

After the code runs, the LED dot matrix lights up and out row by row and column by column.

# Code

```
import RPi.GPIO as GPIO
import time

SDI = 17
RCLK = 18
SRCLK = 27

# we use BX matrix, ROW for anode, and COL for cathode
# ROW +++
code_H =
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0x
ff]
# COL ----
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x
bf,0x7f]

def setup():
    GPIO.setmode(GPIO.BCM) # Number GPIOs by its BCM location
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    GPIO.output(SDI, GPIO.LOW)
    GPIO.output(RCLK, GPIO.LOW)
    GPIO.output(SRCLK, GPIO.LOW)

# 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)

def main():
    while True:
        for i in range(0, len(code_H)):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            time.sleep(0.1)

        for i in range(len(code_H)-1, -1, -1):
            hc595_shift(code_L[i])
            hc595_shift(code_H[i])
            time.sleep(0.1)

    def destroy():
        GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

## Code Explanation

```
code_H =
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xff,0xff]
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00]
```

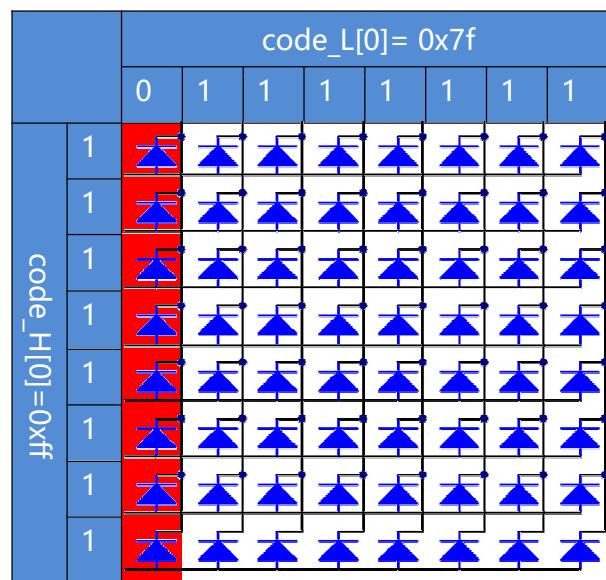
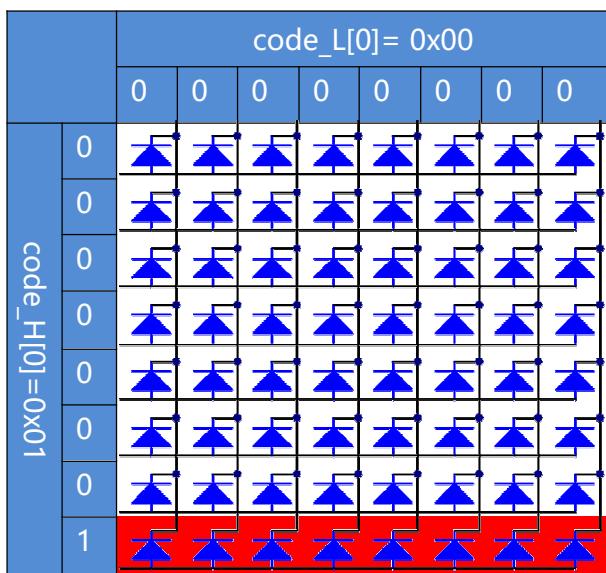
The array code\_H represents the elements of the matrix row, and the array code\_L refers to the elements of the column. When characters are displayed, an element in row and one in column are acquired and assigned to the two HC595 chips respectively. Thus a pattern is shown on the LED dot matrix.

Take the first number of code\_H, 0x01 and the first number of code\_L, 0x00 as examples.

0x01 converted to binary becomes 00000001; 0x00 converted to binary becomes 0000 0000.

In this kit, common anode LED dot matrix is applied, so only the eight LEDs in the eighth row light up.

When the conditions that code\_H is 0xff and code\_L is 0x7f are met simultaneously, these 8 LEDs in the first column are lit.

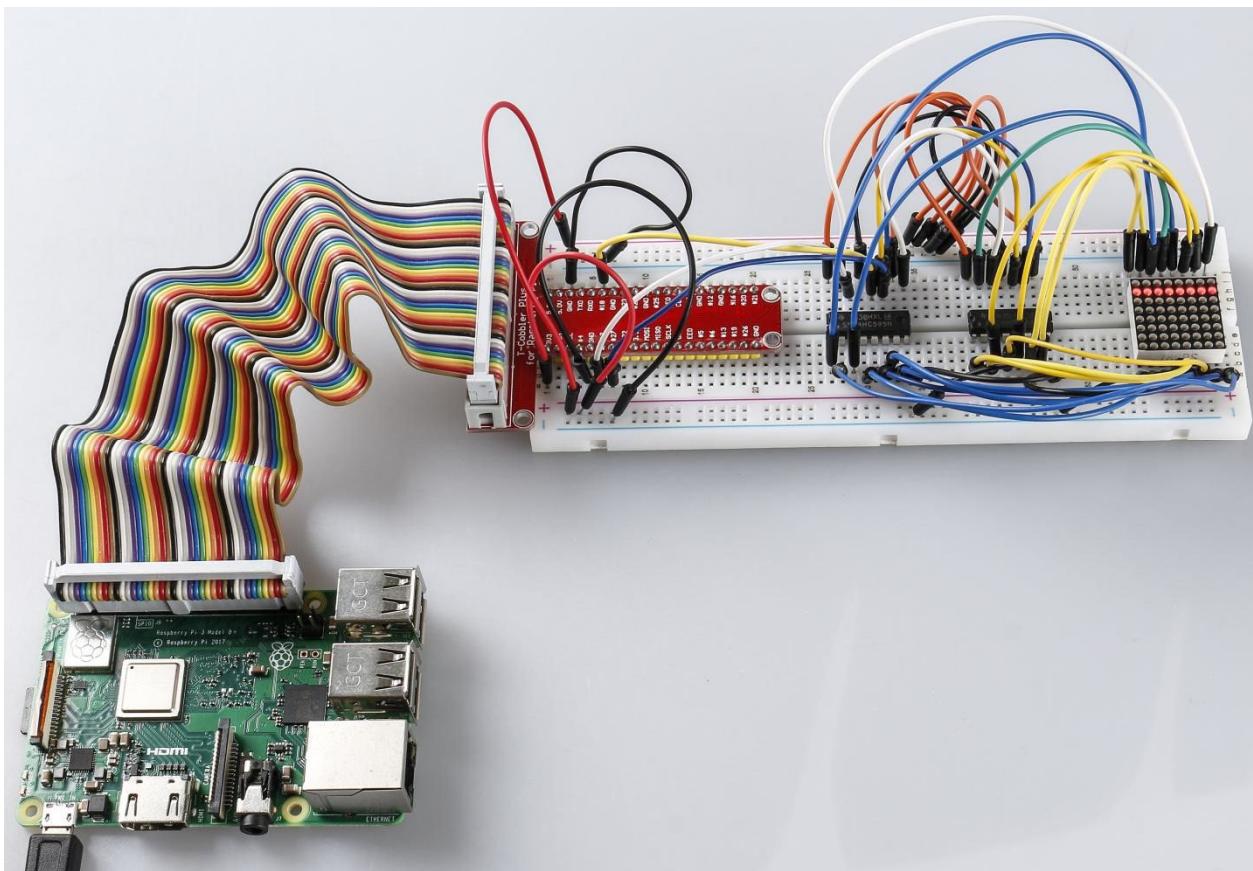


```
for i in range(0, len(code_H)):  
    hc595_shift(code_L[i])  
    hc595_shift(code_H[i])
```

In this loop, these 20 elements in the two arrays, code\_L and code\_H will be uploaded to the HC595 chip one by one.

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

## Phenomenon Picture

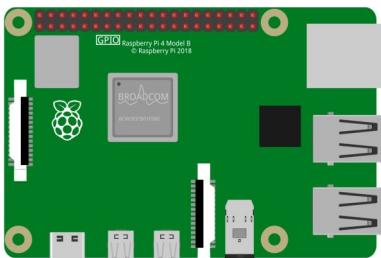
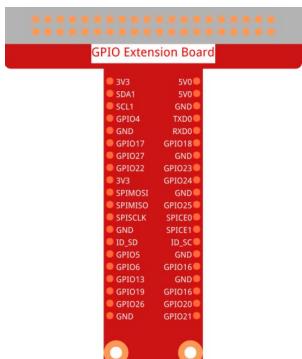
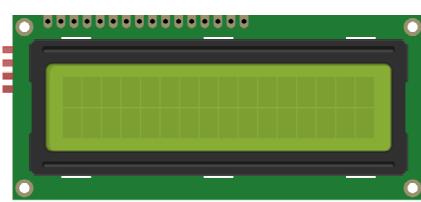
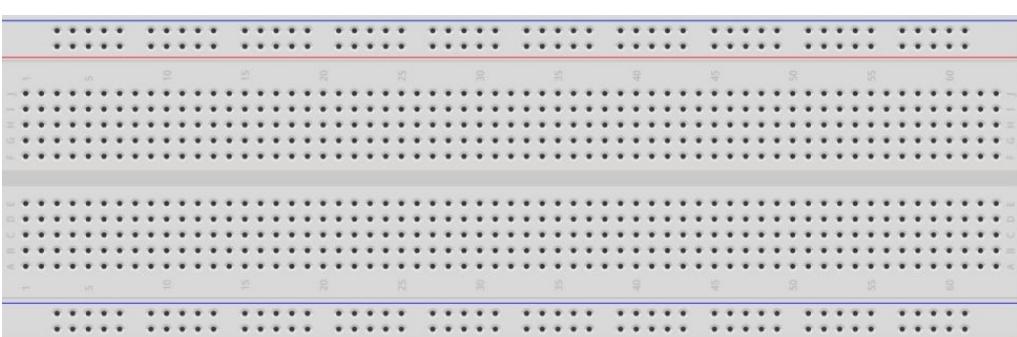


## 1.1.7 I2C LCD1602

### Introduction

LCD1602 is a character type liquid crystal display, which can display 32 (16\*2) characters at the same time.

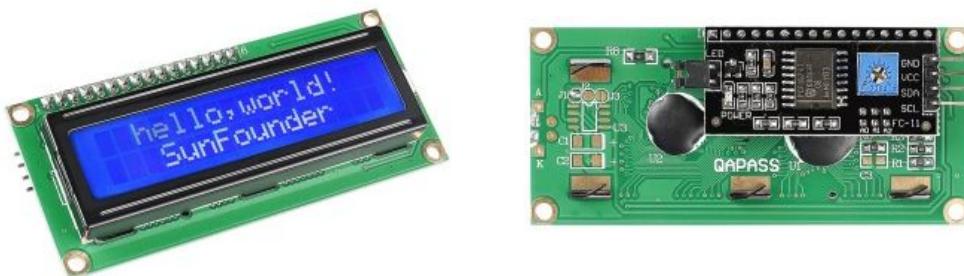
### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * I2C LCD1602
		
1 * 40-pin Cable		Several Jumper Wires
		
1 * Breadboard		

## Principle

### I2C LCD1602

As we all know, though LCD and some other displays greatly enrich the man-machine interaction, they share a common weakness. When they are connected to a controller, multiple IOs will be occupied of the controller which has no so many outer ports. Also it restricts other functions of the controller. Therefore, LCD1602 with an I2C bus is developed to solve the problem.



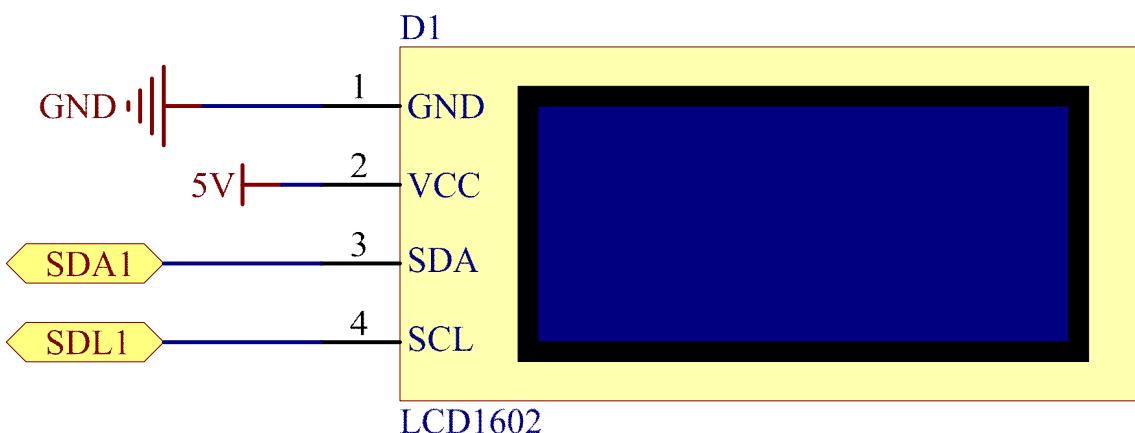
### I2C communication

I2C(Inter-Integrated Circuit) bus is a very popular and powerful bus for communication between a master device (or master devices) and a single or multiple slave devices.

I2C main controller can be used to control IO expander, various sensors, EEPROM, ADC/DAC and so on. All of these are controlled only by the two pins of host, the serial data (SDA1) line and the serial clock line(SCL1).

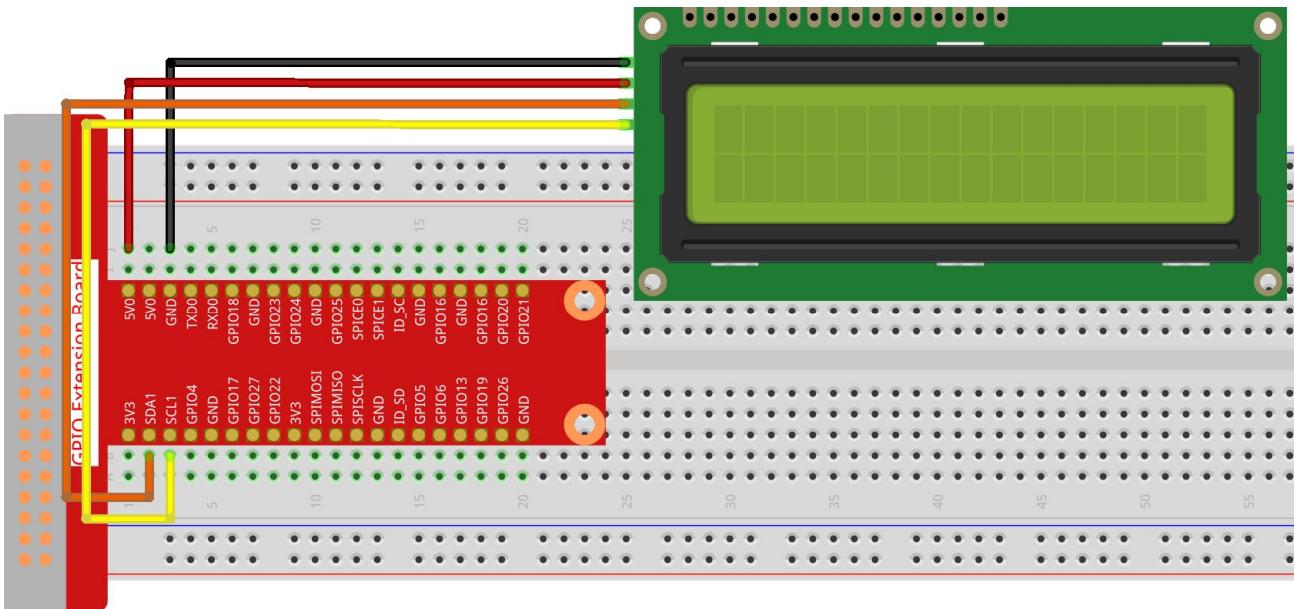
### Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



## Experimental Procedures

### Step 1: Build the circuit.



**Step 2:** Setup I2C (see Appendix. If you have set I2C, skip this step.)

### ➤ For C Language Users

**Step 3:** Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.1.7/
```

**Step 4:** Compile.

```
gcc 1.1.7_Lcd1602.c -lwiringPi
```

**Step 5:** Run.

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

After the code runs, you can see "Greetings", "From SunFounder" displaying on the LCD.

### Code

**Note:** None of the following functions with ellipses is complete. You can view the complete code by using the command, nano 1.1.7 \_lcd1602.c in the Bash interface.

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <string.h>
```

```

int LCDAddr = 0x27;
int BLEN = 1;
int fd;

void write_word(int data){.....}
void send_command(int comm){.....}
void send_data(int data){.....}
void init(){.....}
void clear(){.....}
void write(int x, int y, char data[]){.....}

void main(){
    fd = wiringPil2CSetup(LCDAddr);
    init();
    write(0, 0, "Greetings!");
    write(1, 1, "From SunFounder");
}

```

## Code Explanation

```

void write_word(int data){.....}
void send_command(int comm){.....}
void send_data(int data){.....}
void init(){.....}
void clear(){.....}
void write(int x, int y, char data[]){.....}

```

These functions are used to control I2C LCD1602 open source code. They allow us to easily use I2C LCD1602.

Among these functions, init() is used for initialization, clear() is used to clear the screen, write() is used to write what is displayed, and other functions support the above functions.

```
fd = wiringPil2CSetup(LCDAddr);
```

This function initializes the I2C system with the specified device symbol. The prototype of the function:

```
int wiringPil2CSetup(int devId);
```

Parameters devId is the address of the I2C device, it can be found through the i2cdetect command(see Appendix) and the devId of I2C LCD1602 is generally 0x27.

```
void write(int x, int y, char data[]){}
```

In this function, data[] is the character to be printed on the LCD, and the parameters x and y determine the printing position (line y+1, column x+1 is the starting position of the character to be printed).

## ➤ For Python Language Users

**Step 3:** Change directory.

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

**Step 4:** Run.

```
sudo python 1.1.7_Lcd1602.py
```

After the code runs, you can see "Greetings", "From SunFounder" displaying on the LCD.

## Code

```
import LCD1602
import time

def setup():
    LCD1602.init(0x27, 1) # init(slave address, background light)
    LCD1602.write(0, 0, 'Greetings!!')
    LCD1602.write(1, 1, 'from SunFounder')
    time.sleep(2)

def destroy():
    LCD1602.clear()

if __name__ == "__main__":
    try:
        setup()
    except KeyboardInterrupt:
        destroy()
```

## Code Explanation

```
import LCD1602
```

This file is an open source file for controlling I2C LCD1602. It allows us to easily use I2C LCD1602.

```
LCD1602.init(0x27, 1)
```

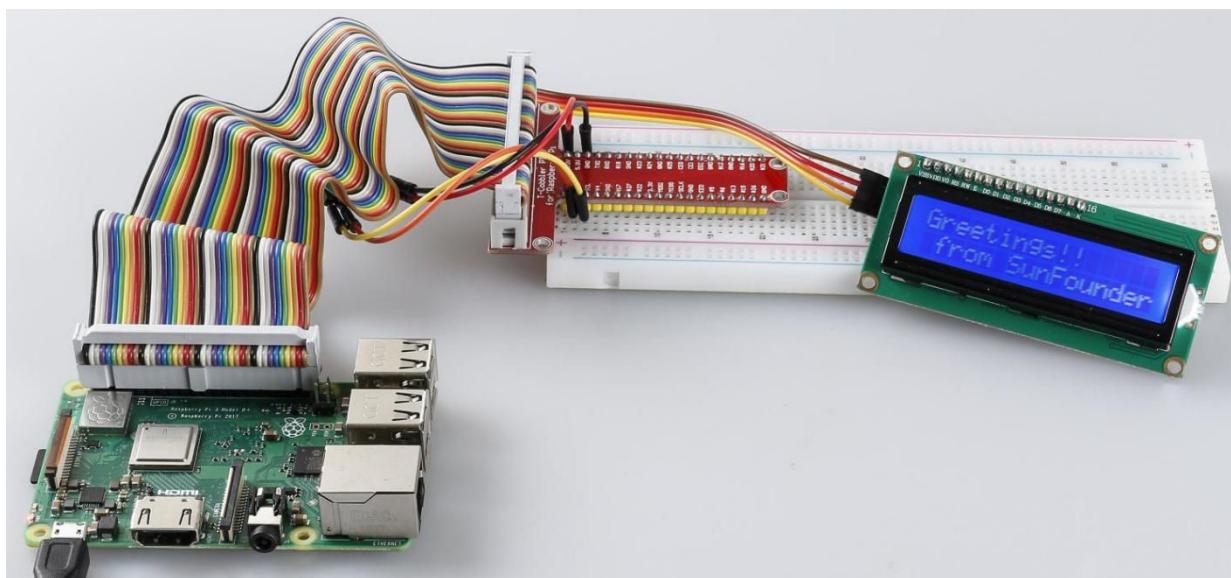
The function initializes the I2C system with the designated device symbol. The first parameter is the address of the I2C device, which can be detected through the i2cdetect command (see Appendix for details). The address of I2C LCD1602 is generally 0x27.

```
LCD1602.write(0, 0, 'Greetings!!')
```

Within this function, 'Greetings!!' is the character to be printed on the Row 0+1, column 0+1 on LCD.

Now you can see "Greetings! From SunFounder" displayed on the LCD.

## Phenomenon Picture



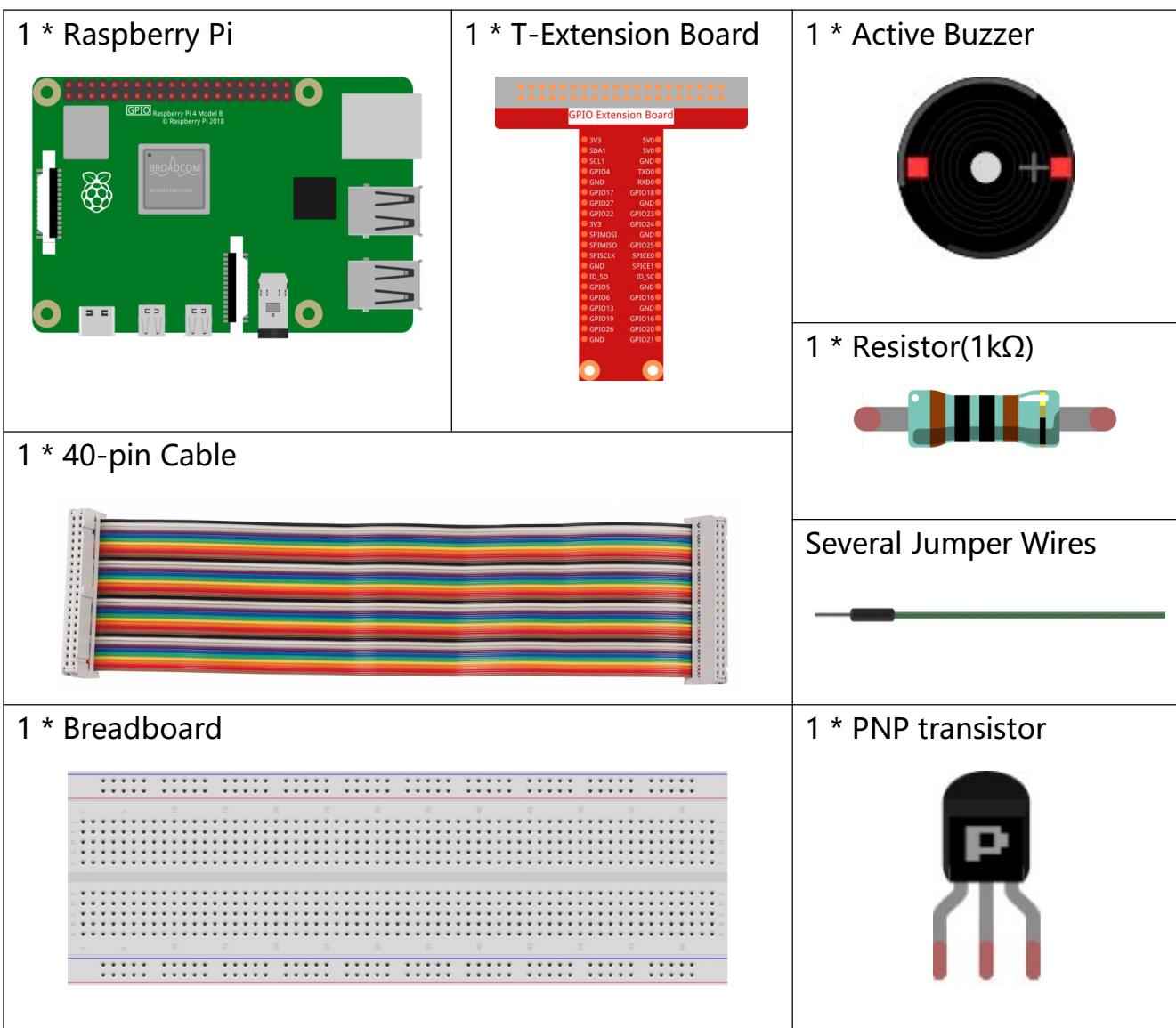
## 1.2 Sound

### 1.2.1 Active Buzzer

# Introduction

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

# Components



## Principle

### Buzzer

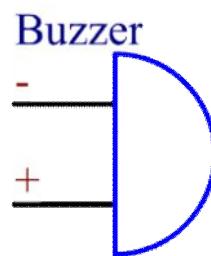
As a type of electronic buzzer with an integrated structure, buzzers, which are supplied by DC power, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products or voice devices. Buzzers can be categorized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

The difference between an active buzzer and a passive buzzer:



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.

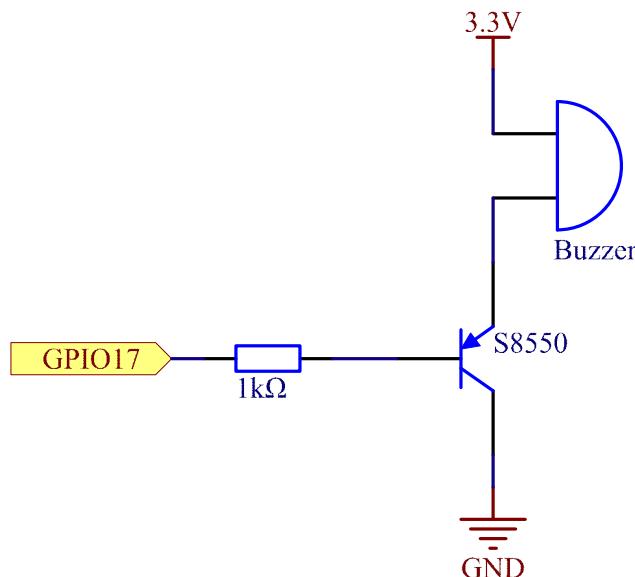


You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

## Schematic Diagram

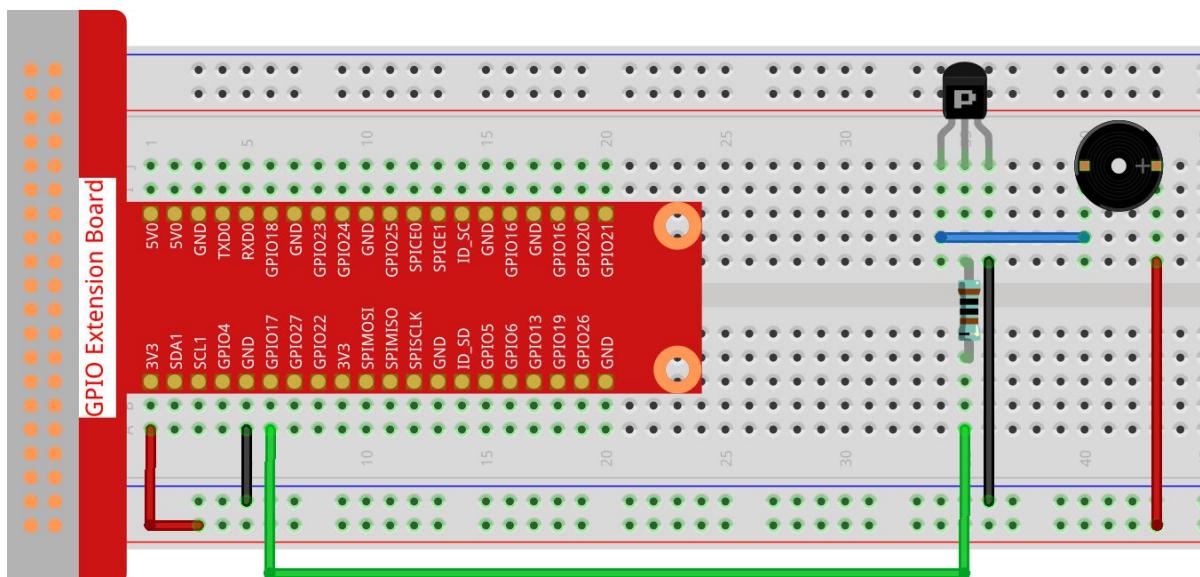
In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and 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.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



## 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/davinci-kit-for-raspberry-pi/c/1.2.1/
```

**Step 3:** Compile the code.

```
gcc 1.2.1_ActiveBuzzer -lwiringPi
```

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

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

The code run, the buzzer beeps.

### Code

```
#include <wiringPi.h>
#include <stdio.h>

#define BeepPin 0
int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(BeepPin, OUTPUT); //set GPIO0 output
    while(1){
        //beep on
        printf("Buzzer on\n");
        digitalWrite(BeepPin, LOW);
        delay(100);
        printf("Buzzer off\n");
        //beep off
        digitalWrite(BeepPin, HIGH);
        delay(100);
    }
    return 0;
}
```

## 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(3.3V), thus the transistor is not energized and the buzzer doesn't beep.

### ➤ For Python Language Users

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

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

**Step 3:** Run.

```
sudo python 1.2.1_ActiveBuzzer.py
```

The code run, the buzzer beeps.

## Code

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as buzzer pin
BeepPin = 17

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.HIGH)

def main():
    while True:
        # Buzzer on (Beep)
        print ('Buzzer On')
        GPIO.output(BeepPin, GPIO.LOW)
        time.sleep(0.1)
        # Buzzer off
        print ('Buzzer Off')
        GPIO.output(BeepPin, GPIO.HIGH)
```

```
time.sleep(0.1)

def destroy():
    # Turn off buzzer
    GPIO.output(BeepPin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()
```

## Code Explanation

```
GPIO.output(BeepPin, GPIO.LOW)
```

Set the buzzer pin as low level to make the buzzer beep.

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

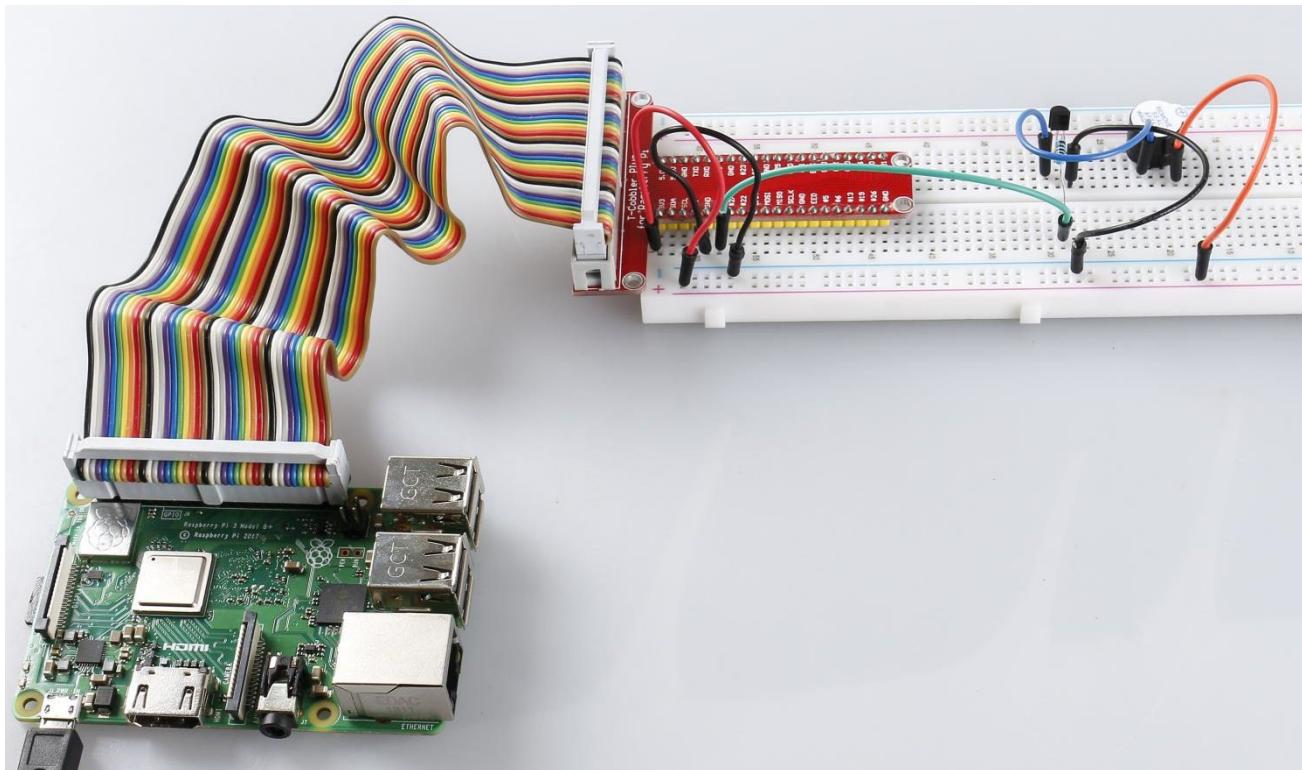
Wait for 0.1 second. Change the switching frequency by changing this parameter.

**Note:** Not the sound frequency. Active Buzzer cannot change sound frequency.

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

close the buzzer.

## Phenomenon Picture



## 1.2.2 Passive Buzzer

## Introduction

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

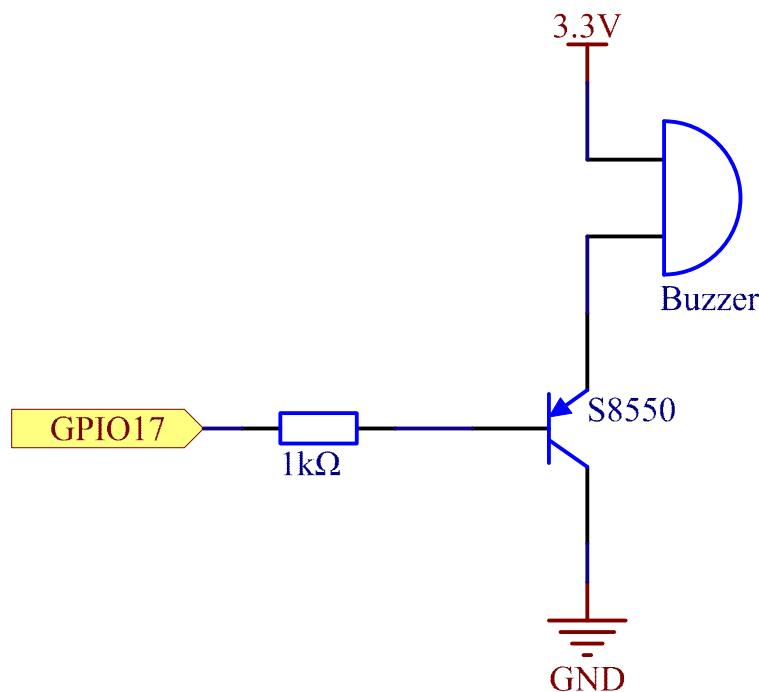
# Components

## Schematic Diagram

In this experiment, a passive buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor.

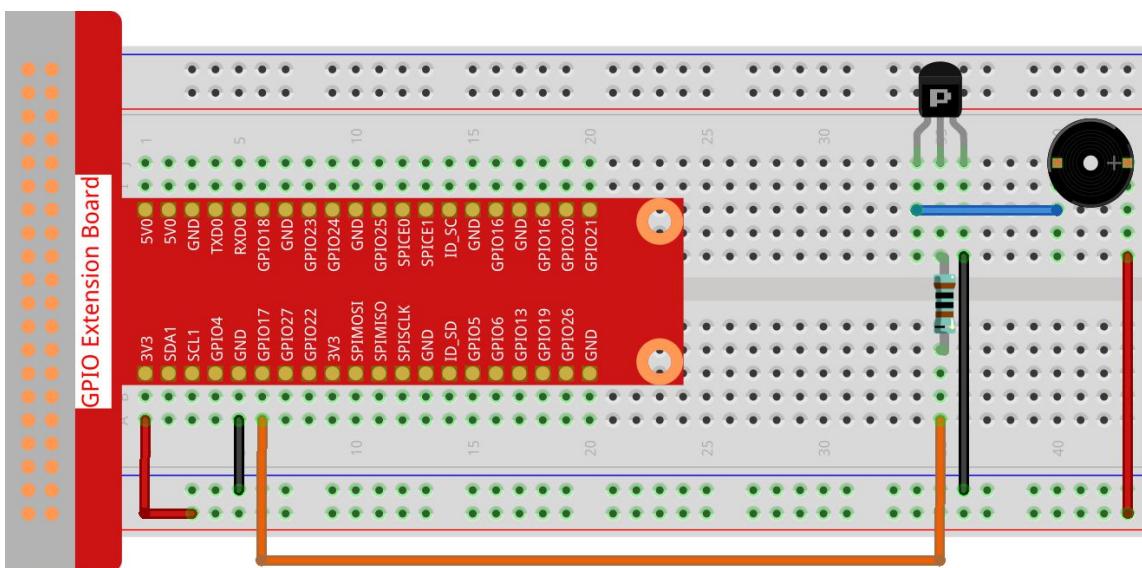
When GPIO17 is given different frequencies, the passive buzzer will emit different sounds; in this way, the buzzer plays music.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

**Step 2:** Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.2.2/
```

**Step 3:** Compile.

```
gcc 1.2.2_PassiveBuzzer.c -lwiringPi
```

**Step 4:** Run.

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

The code run, the buzzer plays a piece of music.

## Code

```
#include <wiringPi.h>
#include <softTone.h>
#include <stdio.h>

#define BuzPin 0

#define CL1 131
#define CL2 147
#define CL3 165
#define CL4 175
#define CL5 196
#define CL6 221
#define CL7 248

#define CM1 262
#define CM2 294
#define CM3 330
#define CM4 350
#define CM5 393
#define CM6 441
#define CM7 495

#define CH1 525
#define CH2 589
#define CH3 661
#define CH4 700
#define CH5 786
```

```
#define CH6 882
#define CH7 990

int song_1[] = {CM3,CM5,CM6,CM3,CM2,CM3,CM5,CM6,CH1,CM6,CM5,CM1,CM3,CM2,
    CM2,CM3,CM5,CM2,CM3,CM3,CL6,CL6,CM1,CM2,CM3,CM2,CL7,
    CL6,CM1,CL5};

int beat_1[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,1,2,1,1,
    1,1,1,1,1,3};

int song_2[] = {CM1,CM1,CM1,CL5,CM3,CM3,CM1,CM1,CM3,CM5,CM5,CM4,CM3,CM2,
    CM2,CM3,CM4,CM4,CM3,CM2,CM3,CM1,CM1,CM3,CM2,CL5,CL7,CM2,CM1
};

int beat_2[] = {1,1,1,3,1,1,1,3,1,1,1,1,1,1,1,3,1,1,1,2,1,1,1,3,1,1,1,3,2,3};

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

    if(softToneCreate(BuzPin) == -1){
        printf("setup softTone failed !");
        return 1;
    }

    while(1){
        printf("music is being played...\n");

        for(i=0;i<sizeof(song_1)/4;i++){
            softToneWrite(BuzPin, song_1[i]);
            delay(beat_1[i] * 500);
        }

        for(i=0;i<sizeof(song_2)/4;i++){
            softToneWrite(BuzPin, song_2[i]);
        }
    }
}
```

```

        delay(beat_2[i] * 500);
    }
}

return 0;
}

```

## Code Explanation

```

#define CL1 131
#define CL2 147
#define CL3 165
#define CL4 175
#define CL5 196
#define CL6 221
#define CL7 248

#define CM1 262
#define CM2 294
...

```

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

```

int song_1[] = {CM3,CM5,CM6,CM3,CM2,CM3,CM5,CM6,CH1,CM6,CM5,CM1,CM3,CM2,
                CM2,CM3,CM5,CM2,CM3,CM3,CL6,CL6,CL6,CM1,CM2,CM3,CM2,CL7,
                CL6,CM1,CL5};
int beat_1[] = {1,1,3,1,1,3,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,1,2,1,1,
                1,1,1,1,1,3};

```

The array, song\_1[] stores a musical score of a song in which beat\_1[] refers to the beat of each note in the song (0.5s for each beat).

```

if(softToneCreate(BuzPin) == -1){
    printf("setup softTone failed !");
    return 1;
}

```

This creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the wiringPiSetup() function you used. The return value is 0 for success. Anything else and you should check the global errno variable to see what went wrong.

```
for(i=0;i<sizeof(song_1)/4;i++){
    softToneWrite(BuzPin, song_1[i]);
    delay(beat_1[i] * 500);
}
```

Employ a for statement to play song\_1.

In the judgment condition, **i<sizeof(song\_1)/4**, “devide by 4” is used because the array song\_1[] is an array of the data type of integer, and each element takes up four bytes.

The number of elements in song\_1 (the number of musical notes) is gotten by deviding sizeof(song\_4) by 4.

To enable each note to play for beat \* 500ms, the function delay(beat\_1[i] \* 500) is called.

The prototype of softToneWrite(BuzPin, song\_1[i]):

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

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

## ➤ For Python Language Users

### Step 2: Change directory.

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

### Step 3: Run.

```
sudo python 1.2.2_PassiveBuzzer.py
```

The code run, the buzzer plays a piece of music.

## Code

```
import RPi.GPIO as GPIO
import time

Buzzer = 11

CL = [0, 131, 147, 165, 175, 196, 211, 248] # Frequency of Bass tone in C major
CM = [0, 262, 294, 330, 350, 393, 441, 495] # Frequency of Midrange tone in C major
CH = [0, 525, 589, 661, 700, 786, 882, 990] # Frequency of Treble tone in C major
```

```

song_1 = [ CM[3], CM[5], CM[6], CM[3], CM[2], CM[3], CM[5], CM[6], # Notes of song1
          CH[1], CM[6], CM[5], CM[1], CM[3], CM[2], CM[2], CM[3],
          CM[5], CM[2], CM[3], CM[3], CL[6], CL[6], CL[6], CM[1],
          CM[2], CM[3], CM[2], CL[7], CL[6], CM[1], CL[5] ]

beat_1 = [ 1, 1, 3, 1, 1, 3, 1, 1,           # Beats of song 1, 1 means 1/8 beat
            1, 1, 1, 1, 1, 1, 3, 1,
            1, 3, 1, 1, 1, 1, 1, 1,
            1, 2, 1, 1, 1, 1, 1, 1,
            1, 1, 3 ]

song_2 = [ CM[1], CM[1], CM[1], CL[5], CM[3], CM[3], CM[3], CM[1], # Notes of song2
          CM[1], CM[3], CM[5], CM[5], CM[4], CM[3], CM[2], CM[2],
          CM[3], CM[4], CM[4], CM[3], CM[2], CM[3], CM[1], CM[1],
          CM[3], CM[2], CL[5], CL[7], CM[2], CM[1]   ]

beat_2 = [ 1, 1, 2, 2, 1, 1, 2, 2,           # Beats of song 2, 1 means 1/8 beat
            1, 1, 2, 2, 1, 1, 3, 1,
            1, 2, 2, 1, 1, 2, 2, 1,
            1, 2, 2, 1, 1, 3 ]

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(Buzzer, GPIO.OUT)   # Set pins' mode is output
    global Buzz                  # Assign a global variable to replace GPIO.PWM
    Buzz = GPIO.PWM(Buzzer, 440)  # 440 is initial frequency.
    Buzz.start(50)               # Start Buzzer pin with 50% duty cycle

def loop():
    while True:
        print ('\n  Playing song 1...')
        for i in range(1, len(song_1)):  # Play song 1
            Buzz.ChangeFrequency(song_1[i]) # Change the frequency along the song note
            time.sleep(beat_1[i] * 0.5)    # delay a note for beat * 0.5s
            time.sleep(1)                 # Wait a second for next song.

        print ('\n\n  Playing song 2...')
        for i in range(1, len(song_2)):  # Play song 1
            Buzz.ChangeFrequency(song_2[i]) # Change the frequency along the song note
            time.sleep(beat_2[i] * 0.5)    # delay a note for beat * 0.5s

```

```

def destory():
    Buzz.stop()          # Stop the buzzer
    GPIO.output(Buzzer, 1) # Set Buzzer pin to High
    GPIO.cleanup()        # Release resource

if __name__ == '__main__': # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
        executed.
    destory()

```

## Code Explanation

```

CL = [0, 131, 147, 165, 175, 196, 211, 248] # Frequency of Bass tone in C major
CM = [0, 262, 294, 330, 350, 393, 441, 495] # Frequency of Midrange tone in C major
CH = [0, 525, 589, 661, 700, 786, 882, 990] # Frequency of Treble tone in C major

```

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

```

song_1 = [ CM[3], CM[5], CM[6], CM[3], CM[2], CM[3], CM[5], CM[6],
           CH[1], CM[6], CM[5], CM[1], CM[3], CM[2], CM[2], CM[3],
           CM[5], CM[2], CM[3], CM[3], CL[6], CL[6], CL[6], CM[1],
           CM[2], CM[3], CM[2], CL[7], CL[6], CM[1], CL[5] ]

```

These arrays are the notes of a song.

```

beat_1 = [ 1, 1, 3, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1,
           1, 3, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 3 ]

```

Every sound beat (each number) represents the  $\frac{1}{8}$  beat, or 0.5s

```

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

```

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

```

for i in range(1, len(song_1)):

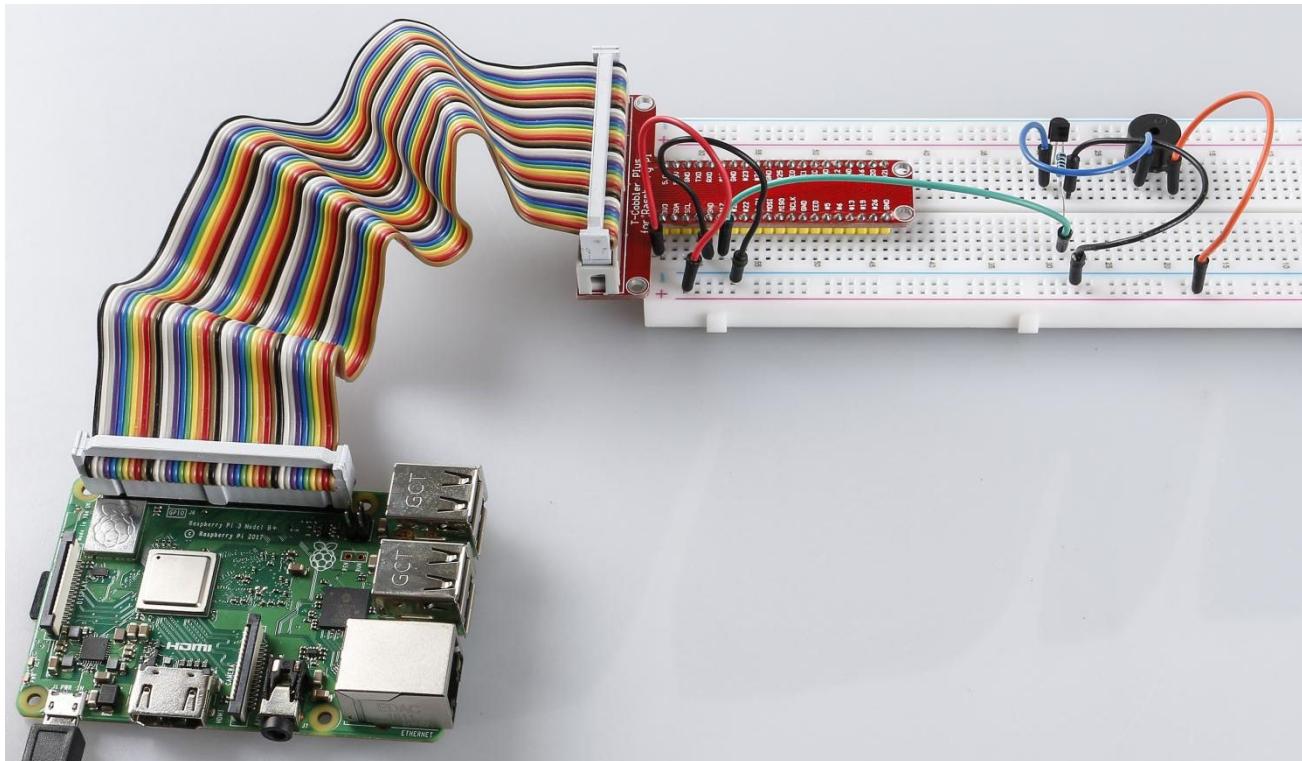
```

```
Buzz.ChangeFrequency(song_1[i])  
time.sleep(beat_1[i] * 0.5)
```

Run a for loop, then the buzzer will play the notes in the array song\_1[] with the beats in the beat\_1[] array.

Now you can hear the passive buzzer playing music.

## Phenomenon Picture



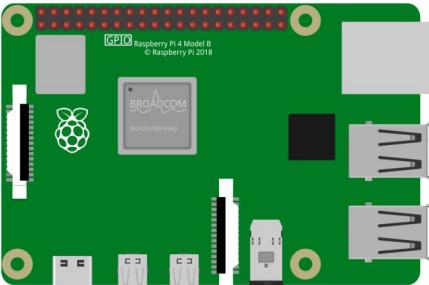
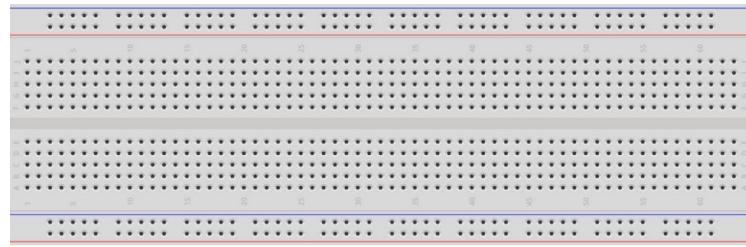
## 1.3 Drivers

### 1.3.1 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

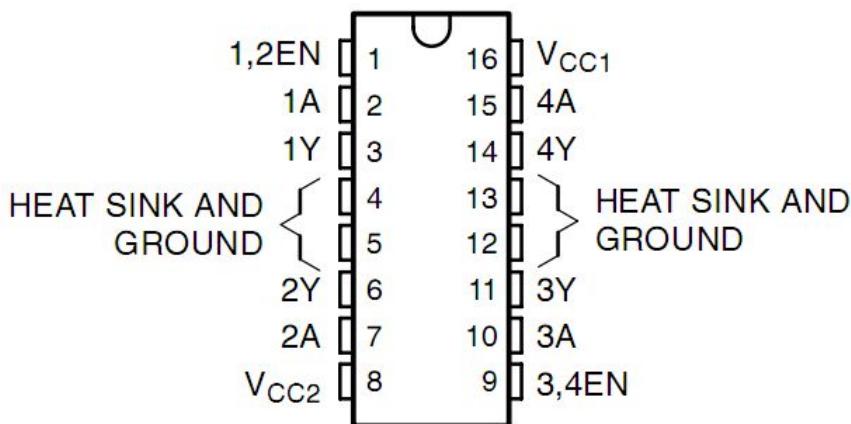
1 * Raspberry Pi	1 * T-Extension Board	1 * Power Module (with 9V battery and buckle)
	 GPIO Extension Board Pinout: 3V3 SDA1 SVO GND GND GPIO4 TXD0 GND RXD0 GPIO17 GPIO18 GND GND GPIO22 GPIO23 3V3 GPIO24 SPI MOSI GND SPI MISO GPIO25 SPI SCLK SPI CE0 GND SPI CE1 ID_SD ID_SD GPIO5 GND GPIO6 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO26 GPIO20 GND GPIO21	
1 * 40-pin Cable		1 * L293D
		 Several Jumper Wires 
1 * Breadboard		1 * DC Motor
		

## Principle

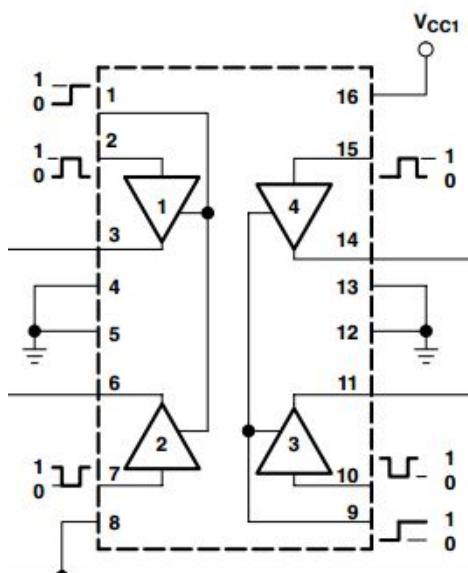
### L293D

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, Stepper 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.



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
A	EN	Y
H	H	H
L	H	L
X	L	Z

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

## 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

Operation Voltage: 1-6V

Free-run current (3V): 70mA

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

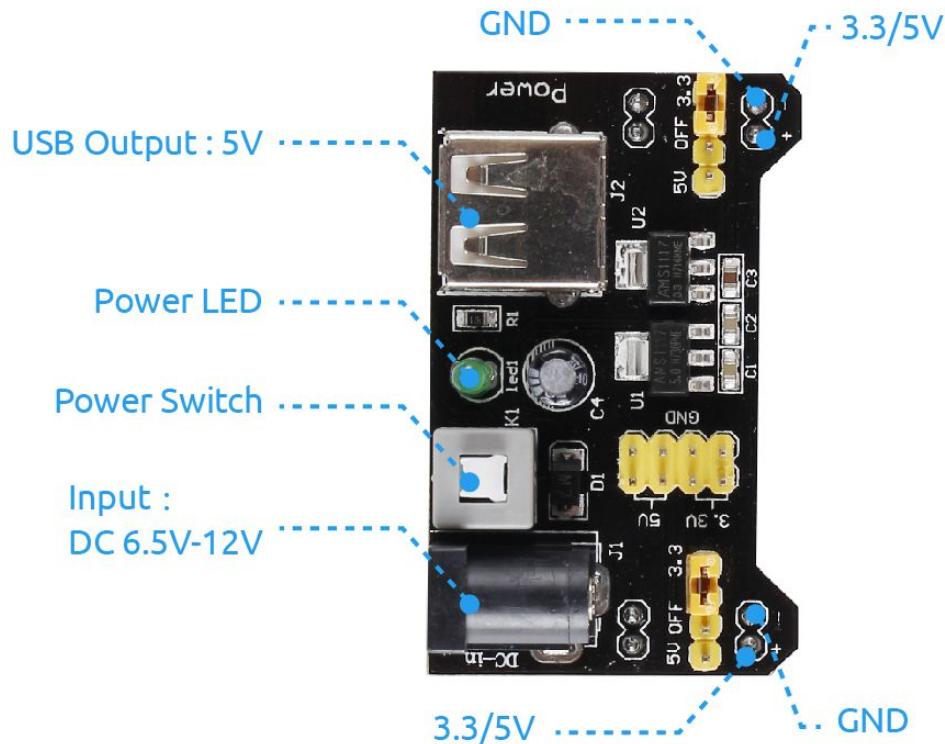
Stall current (3V): 800mA

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.

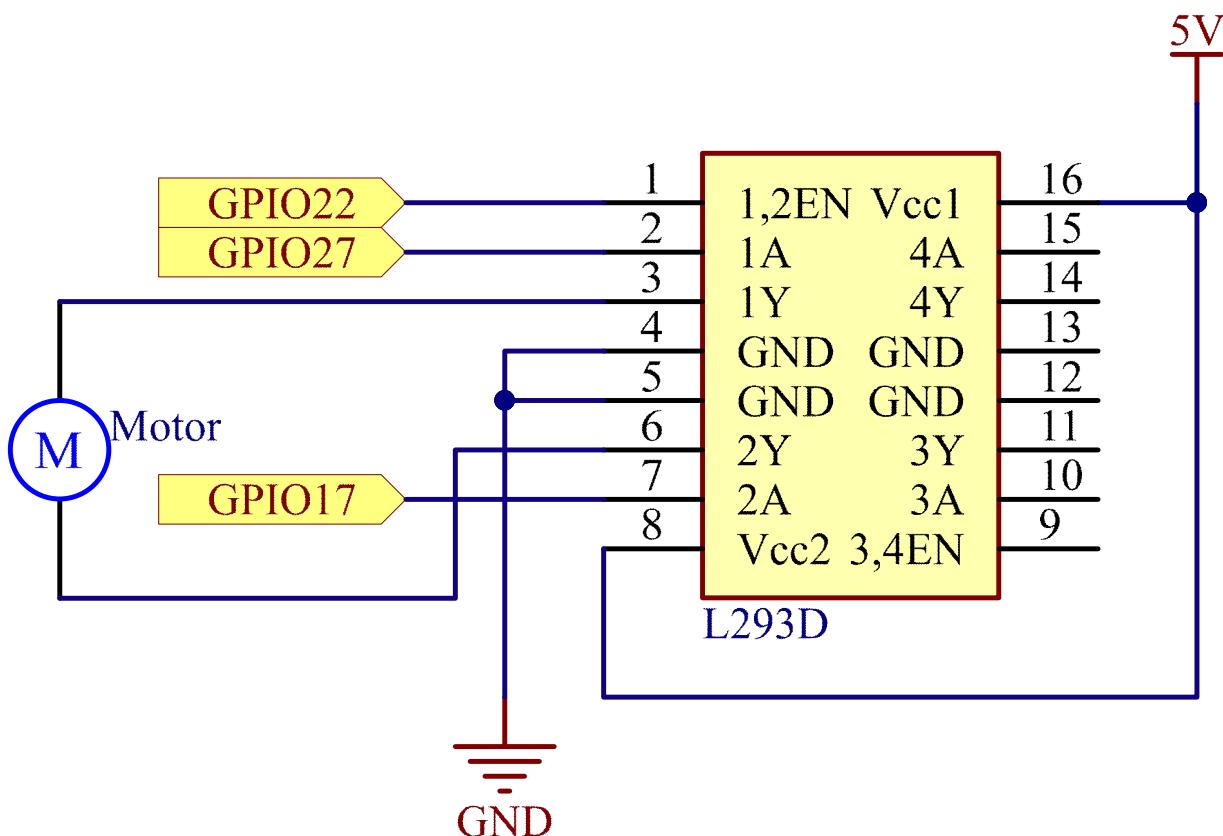
You can just plug it in the breadboard to supply power. It provides a voltage of 3.3V and 5V, and you can connect either via a jumper cap included.



## Schematic Diagram

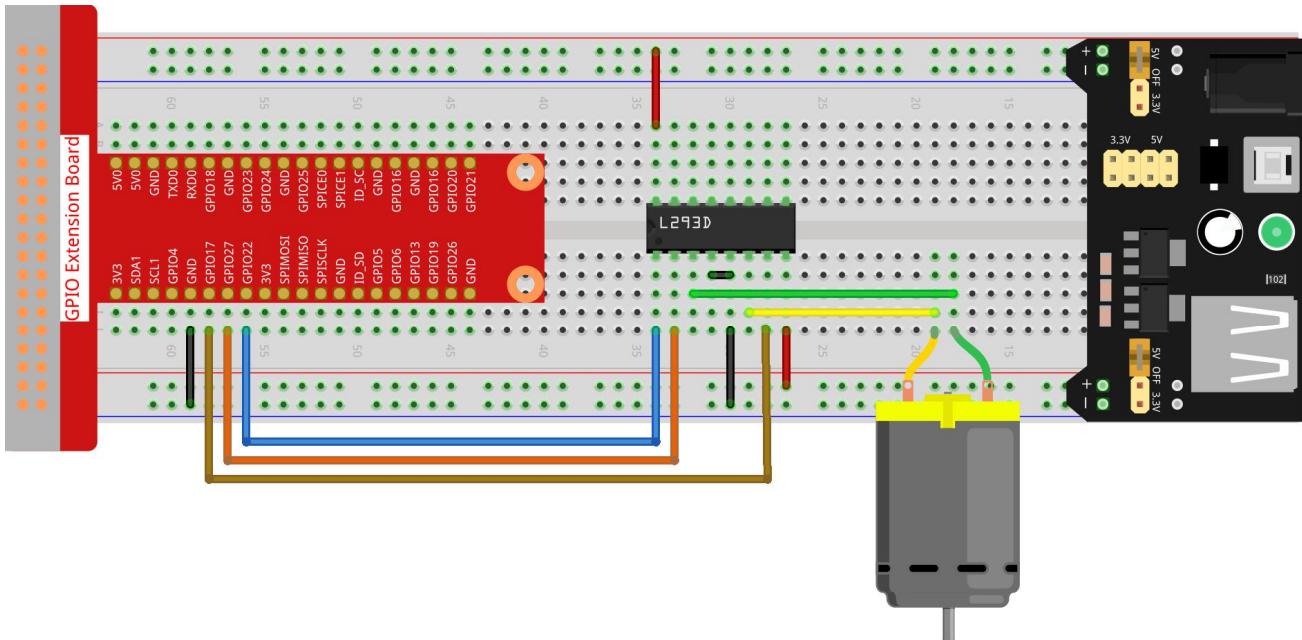
Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to GPIO22, and set it as high level. Connect pin2 to GPIO27, and pin7 to GPIO17, then set one pin high, while the other low. Thus you can change the motor's rotation direction.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



## Experimental Procedures

### Step 1: Build the circuit.



**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/davinci-kit-for-raspberry-pi/c/1.3.1/
```

#### Step 3: Compile.

```
gcc 1.3.1_Motor -lwiringPi
```

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

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

As the code runs, the motor first rotates clockwise for 5s then stops for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

## Code

```
#include <wiringPi.h>
#include <stdio.h>

#define MotorPin1    0
#define MotorPin2    2
#define MotorEnable   3

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

    pinMode(MotorPin1, OUTPUT);
    pinMode(MotorPin2, OUTPUT);
    pinMode(MotorEnable, OUTPUT);

    while(1){
        printf("Clockwise\n");
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, HIGH);
        digitalWrite(MotorPin2, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Stop\n");
        digitalWrite(MotorEnable, LOW);
        for(i=0;i<3;i++){
            delay(1000);
        }

        printf("Anti-clockwise\n");
        digitalWrite(MotorEnable, HIGH);
        digitalWrite(MotorPin1, LOW);
        digitalWrite(MotorPin2, HIGH);
        for(i=0;i<3;i++){
            delay(1000);
        }
    }
}
```

```
printf("Stop\n");
digitalWrite(MotorEnable, LOW);
for(i=0;i<3;i++){
    delay(1000);
}
return 0;
}
```

## Code Explanation

```
digitalWrite(MotorEnable, HIGH);
```

Enable the L239D.

```
digitalWrite(MotorPin1, HIGH);
digitalWrite(MotorPin2, LOW);
```

Set a high level for 2A(pin 7); since 1,2EN(pin 1) is in high level, 2Y will output high level.

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 Language Users

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

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

**Step 3:** Run.

```
sudo python 1.3.1_Motor.py
```

As the code runs, the motor first rotates clockwise for 5s, then stops for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

## Code

```
import RPi.GPIO as GPIO
import time

# Set up pins
MotorPin1 = 17
MotorPin2 = 27
MotorEnable = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set pins to output
    GPIO.setup(MotorPin1, GPIO.OUT)
    GPIO.setup(MotorPin2, GPIO.OUT)
    GPIO.setup(MotorEnable, GPIO.OUT, initial=GPIO.LOW)

    # Define a motor function to spin the motor
    # direction should be
    # 1(clockwise), 0(stop), -1(counterclockwise)
    def motor(direction):
        # Clockwise
        if direction == 1:
            # Set direction
            GPIO.output(MotorPin1, GPIO.HIGH)
            GPIO.output(MotorPin2, GPIO.LOW)
            # Enable the motor
            GPIO.output(MotorEnable, GPIO.HIGH)
            print ("Clockwise")
```

```

# Counterclockwise
if direction == -1:
    # Set direction
    GPIO.output(MotorPin1, GPIO.LOW)
    GPIO.output(MotorPin2, GPIO.HIGH)
    # Enable the motor
    GPIO.output(MotorEnable, GPIO.HIGH)
    print ("Counterclockwise")

# Stop
if direction == 0:
    # Disable the motor
    GPIO.output(MotorEnable, GPIO.LOW)
    print ("Stop")

def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}
    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)
        # Anticlockwise
        motor(directions['CCW'])
        time.sleep(5)
        # Stop
        motor(directions['STOP'])
        time.sleep(5)

def destroy():
    # Stop the motor
    GPIO.output(MotorEnable, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':

```

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

## Code Explanation

```
def motor(direction):
    # Clockwise
    if direction == 1:
        # Set direction
        GPIO.output(MotorPin1, GPIO.HIGH)
        GPIO.output(MotorPin2, GPIO.LOW)
        # Enable the motor
        GPIO.output(MotorEnable, GPIO.HIGH)
        print ("Clockwise")
    ...

```

Create a function, **motor()** whose variable is direction. As the condition that direction=1 is met, the motor rotates clockwise; when direction=-1, the motor rotates anticlockwise; and under the condition that direction=0, it stops rotating.

```
def main():
    # Define a dictionary to make the script more readable
    # CW as clockwise, CCW as counterclockwise, STOP as stop
    directions = {'CW': 1, 'CCW': -1, 'STOP': 0}

    while True:
        # Clockwise
        motor(directions['CW'])
        time.sleep(5)

        # Stop
        motor(directions['STOP'])
        time.sleep(5)

        # Anticlockwise
        motor(directions['CCW'])
        time.sleep(5)

        # Stop
        motor(directions['STOP'])
```

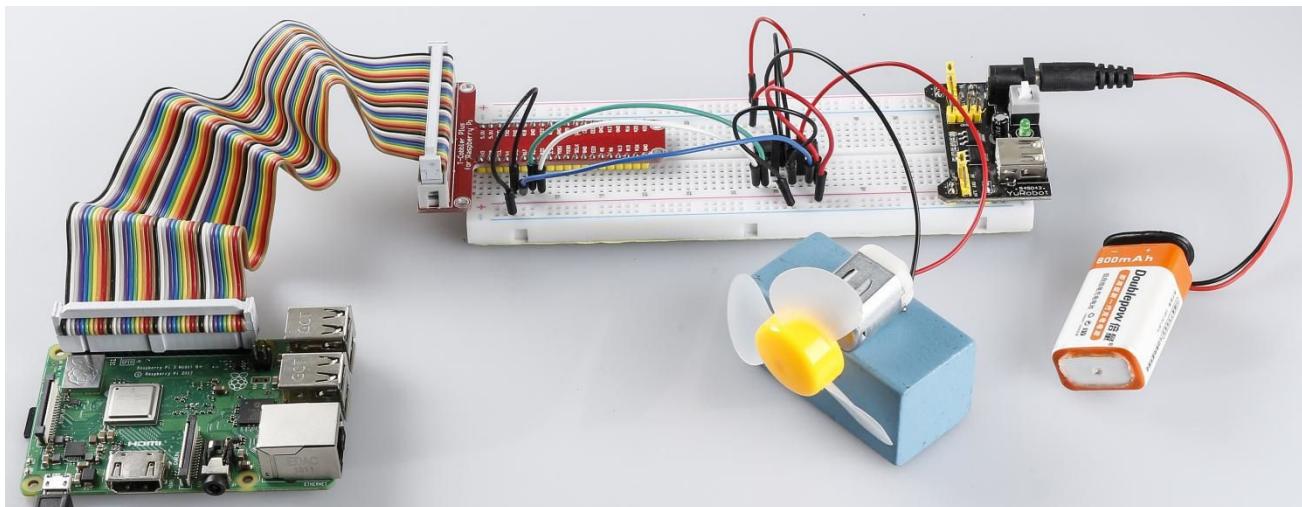
```
time.sleep(5)
```

In the main () function, create an array, directions[], in which CW is equal to 1, the value of CCW is -1, and the number 0 refers to Stop.

As the code runs, the motor first rotates clockwise for 5s then stop for 5s, after that, it rotates anticlockwise for 5s; subsequently, the motor stops for 5s. This series of actions will be executed repeatedly.

Now, you should see the motor blade rotating.

## Phenomenon Picture

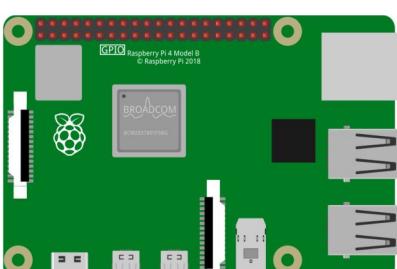
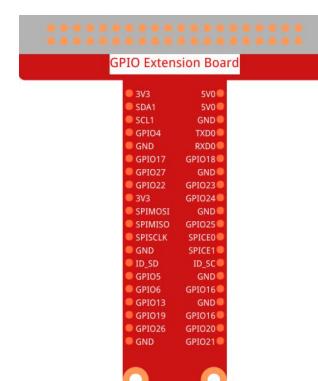
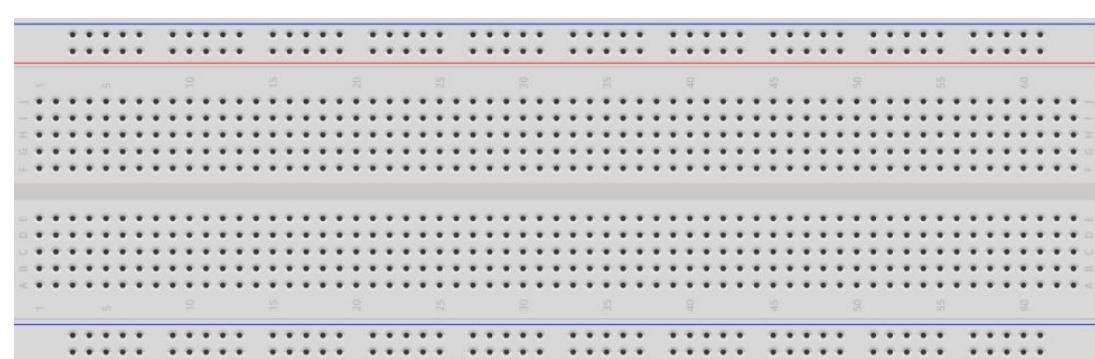


## 1.3.2 Servo

### Introduction

In this lesson, we will learn how to make the servo rotate.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Servo																																								
	 GPIO Extension Board <table border="1"> <tr><td>3V3</td><td>5V0</td></tr> <tr><td>SDA1</td><td>5V0</td></tr> <tr><td>SCL1</td><td>GND</td></tr> <tr><td>GPIO4</td><td>TXD0</td></tr> <tr><td>GND</td><td>RXD0</td></tr> <tr><td>GPIO17</td><td>GPIO18</td></tr> <tr><td>GPIO27</td><td>GND</td></tr> <tr><td>GPIO22</td><td>GPIO23</td></tr> <tr><td>3V3</td><td>GPIO24</td></tr> <tr><td>SPI_MOSI</td><td>GND</td></tr> <tr><td>SPI_MISO</td><td>GPIO25</td></tr> <tr><td>SPI_SCLK</td><td>SPI_CE0</td></tr> <tr><td>GND</td><td>SPI_CE1</td></tr> <tr><td>ID_SD</td><td>ID_SC</td></tr> <tr><td>GPIO5</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td></tr> <tr><td>GPIO13</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO16</td></tr> <tr><td>GPIO26</td><td>GPIO20</td></tr> <tr><td>GND</td><td>GPIO21</td></tr> </table>	3V3	5V0	SDA1	5V0	SCL1	GND	GPIO4	TXD0	GND	RXD0	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPI_MOSI	GND	SPI_MISO	GPIO25	SPI_SCLK	SPI_CE0	GND	SPI_CE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	 <b>SERVO</b> Several Jumper Wires
3V3	5V0																																									
SDA1	5V0																																									
SCL1	GND																																									
GPIO4	TXD0																																									
GND	RXD0																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPI_MOSI	GND																																									
SPI_MISO	GPIO25																																									
SPI_SCLK	SPI_CE0																																									
GND	SPI_CE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
1 * 40-pin Cable																																										
1 * Breadboard																																										

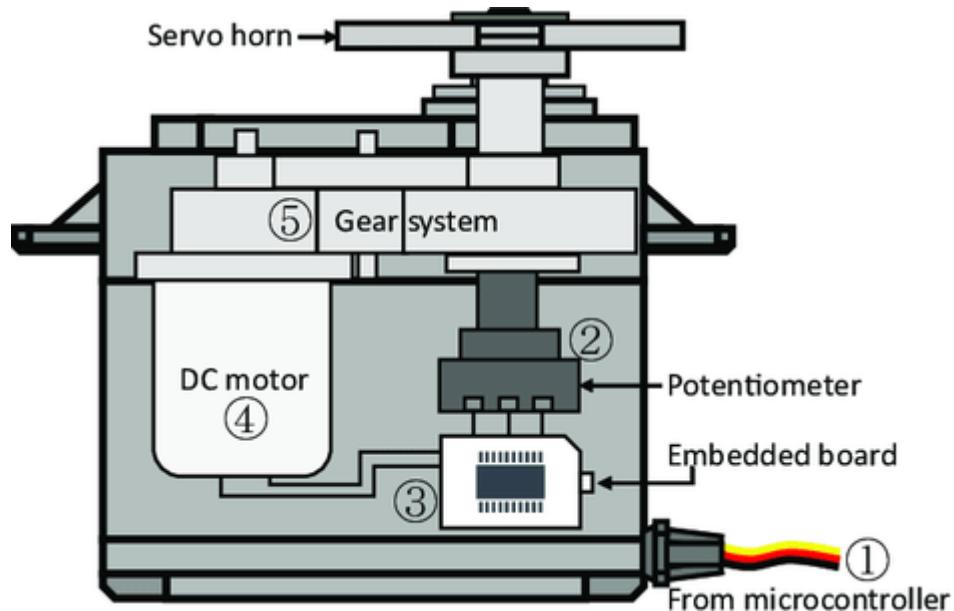
## Principle

### Servo

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



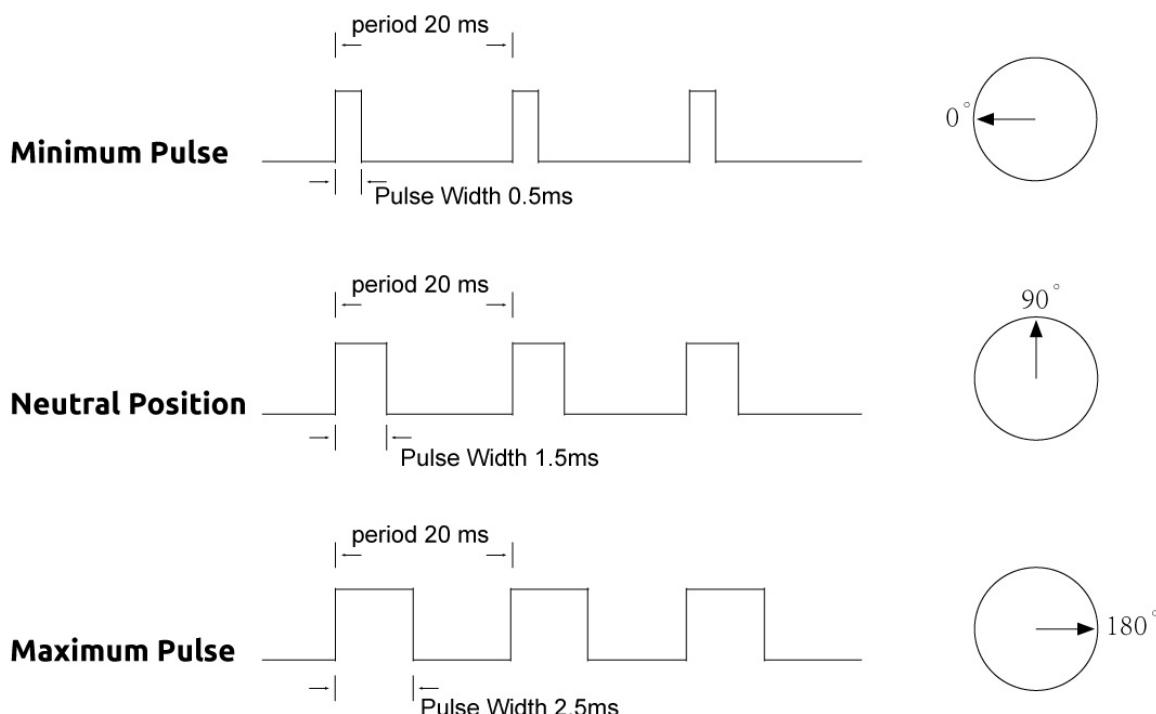
It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft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



The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms.

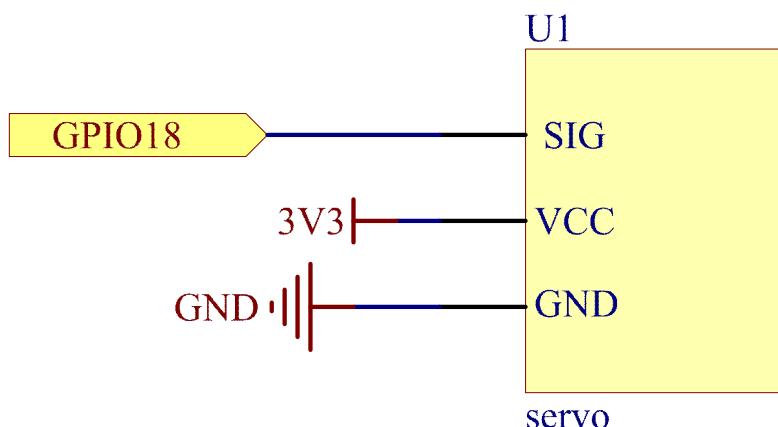
The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position).

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



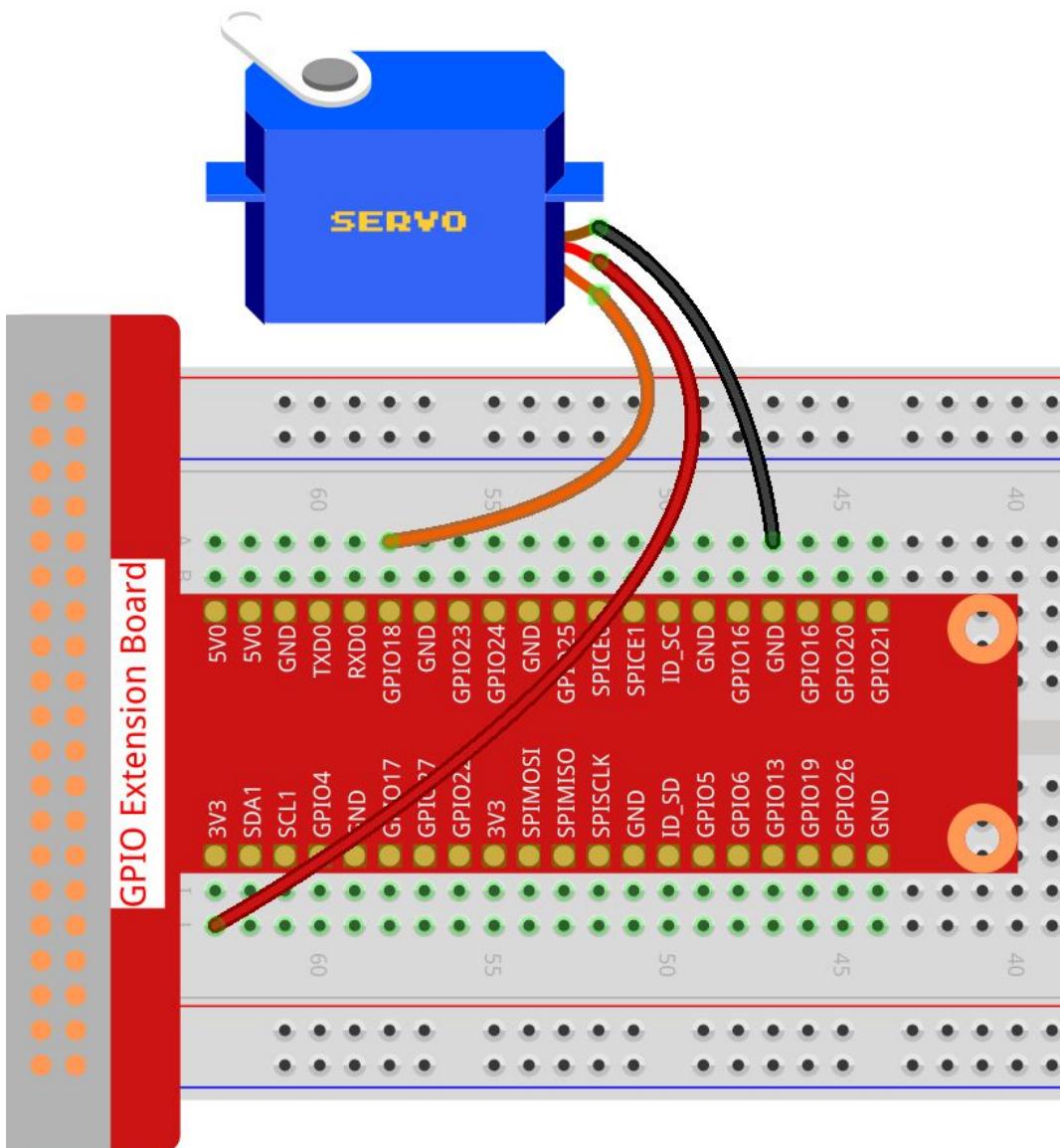
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18



## Experimental Procedures

**Step 1:** Build the circuit.



### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.2
```

**Step 3:** Compile the code.

```
gcc 1.3.2_Servo.c -lwiringPi
```

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

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

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

## Code

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>

#define servoPin 1 //define the GPIO number connected to servo
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void servoWrite(int pin, int angle){ //Create a function, servoWrite() to control the rotate
angle of the servo.
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,map(angle,0,180,5,25));
}
int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(servoPin, 0, 200); //initialize PMW pin of servo
    while(1){
        for(i=0;i<181;i++){ //make servo rotate from minimum angle to maximum angle
            servoWrite(servoPin,i);
            delay(1);
        }
        delay(500);
        for(i=181;i>-1;i--){ //make servo rotate from maximum angle to minimum angle
            servoWrite(servoPin,i);
            delay(1);
        }
        delay(500);
    }
    return 0;
}
```

## Code Explanation

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

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

```
void servoWrite(int pin, int angle){ //Create a funtion, servoWrite() to control the rotate  
angle of the servo.  
    if(angle < 0)  
        angle = 0;  
    if(angle > 180)  
        angle = 180;  
    softPwmWrite(pin,map(angle,0,180,5,25));  
}
```

Create a funtion, servoWrite() to write angle to the servo.

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

This function can change the duty cycle of the PWM.

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

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

The function is to use softwares to create a PWM pin, servoPin, then the initial pulse widths of them are set to 0, and the period of PWM is  $200 \times 100\text{us}$ .

The prototype of this function is shown below.

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

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

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

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

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 1.3.2_Servo.py
```

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

## Code

```
import RPi.GPIO as GPIO
import time

servoPin = 12

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

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(servoPin, GPIO.OUT)  # Set servoPin's mode is output
    GPIO.output(servoPin, GPIO.LOW) # Set servoPin to low

    p = GPIO.PWM(servoPin, 50)    # set Frequency to 50Hz
    p.start(0)                   # Duty Cycle = 0

def servoWrite(angle):    # make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))#map the angle to duty cycle and output it

def loop():
    while True:
        for i in range(0, 181, 1): #make servo rotate from 0 to 180 deg
            servoWrite(i)      # Write to servo
            time.sleep(0.001)
```

```

time.sleep(0.5)
for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
    servoWrite(i)
    time.sleep(0.001)
    time.sleep(0.5)

def destroy():
    p.stop()
    GPIO.cleanup()

if __name__ == '__main__': #Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    destroy()

```

## Code Explanation

```

p = GPIO.PWM(servoPin, 50) # set Frequency to 50Hz
p.start(0) # Duty Cycle = 0

```

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

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

```

def servoWrite(angle): # make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))#map the angle to duty cycle and output it

```

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

```

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

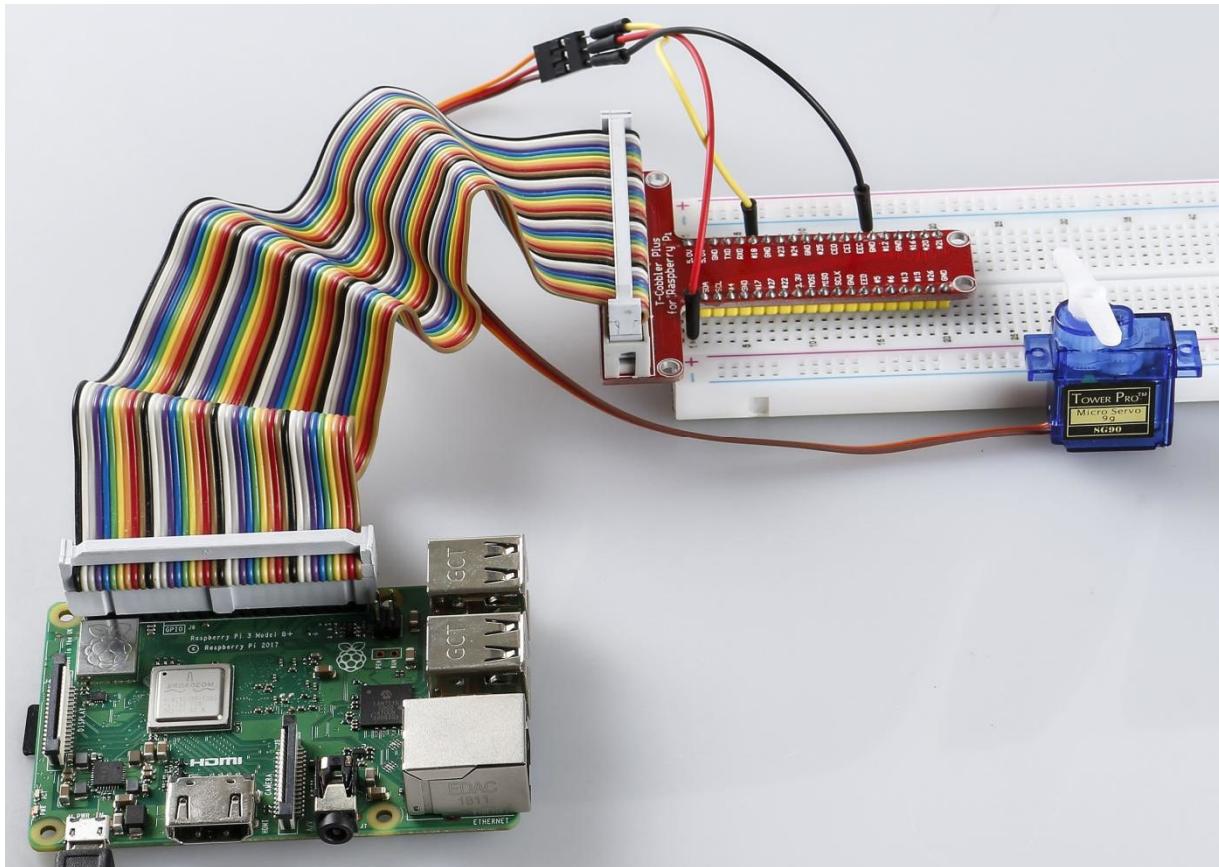
```

This function can change the duty cycle of the PWM.

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

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

## Phenomenon Picture

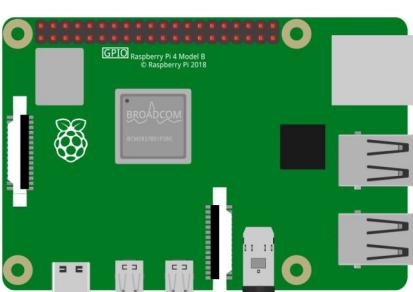
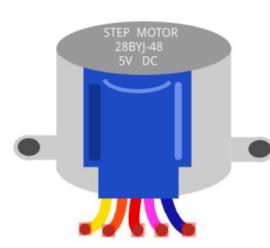
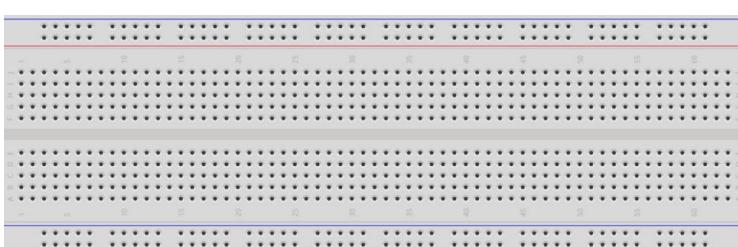
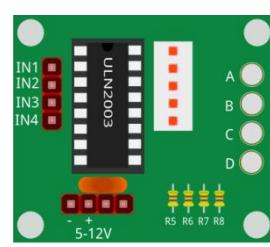


### 1.3.3 Stepper Motor

#### Introduction

Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely moving it forward or backward in small "steps".

#### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Stepper Motor																																										
	 <table border="1"> <thead> <tr> <th colspan="2">GPIO Extension Board</th> </tr> </thead> <tbody> <tr><td>3V3</td><td>SVS#</td></tr> <tr><td>SDA1</td><td>SVD#</td></tr> <tr><td>SCL1</td><td>GND#</td></tr> <tr><td>GPIO4</td><td>TXD#</td></tr> <tr><td>GPIO17</td><td>RXD#</td></tr> <tr><td>GPIO18</td><td>GPIO18#</td></tr> <tr><td>GPIO27</td><td>GND#</td></tr> <tr><td>GPIO23</td><td>GPIO23#</td></tr> <tr><td>3V3</td><td>GPIO24#</td></tr> <tr><td>SPIMOSI</td><td>GND#</td></tr> <tr><td>SPIMISO</td><td>GPIO25#</td></tr> <tr><td>SPISCLK</td><td>SPICED#</td></tr> <tr><td>GND</td><td>SPIGND#</td></tr> <tr><td>ID_SD</td><td>ID_SD#</td></tr> <tr><td>GPIO5</td><td>GND#</td></tr> <tr><td>GPIO6</td><td>GPIO16#</td></tr> <tr><td>GPIO13</td><td>GND#</td></tr> <tr><td>GPIO19</td><td>GPIO18#</td></tr> <tr><td>GPIO26</td><td>GPIO23#</td></tr> <tr><td>GND</td><td>GPIO21#</td></tr> </tbody> </table>	GPIO Extension Board		3V3	SVS#	SDA1	SVD#	SCL1	GND#	GPIO4	TXD#	GPIO17	RXD#	GPIO18	GPIO18#	GPIO27	GND#	GPIO23	GPIO23#	3V3	GPIO24#	SPIMOSI	GND#	SPIMISO	GPIO25#	SPISCLK	SPICED#	GND	SPIGND#	ID_SD	ID_SD#	GPIO5	GND#	GPIO6	GPIO16#	GPIO13	GND#	GPIO19	GPIO18#	GPIO26	GPIO23#	GND	GPIO21#	
GPIO Extension Board																																												
3V3	SVS#																																											
SDA1	SVD#																																											
SCL1	GND#																																											
GPIO4	TXD#																																											
GPIO17	RXD#																																											
GPIO18	GPIO18#																																											
GPIO27	GND#																																											
GPIO23	GPIO23#																																											
3V3	GPIO24#																																											
SPIMOSI	GND#																																											
SPIMISO	GPIO25#																																											
SPISCLK	SPICED#																																											
GND	SPIGND#																																											
ID_SD	ID_SD#																																											
GPIO5	GND#																																											
GPIO6	GPIO16#																																											
GPIO13	GND#																																											
GPIO19	GPIO18#																																											
GPIO26	GPIO23#																																											
GND	GPIO21#																																											
1 * 40-pin Cable	Several Jumper Wires																																											
																																												
1 * Breadboard	1 * ULN2003																																											
																																												

## Principle

### Stepper Motor

There are two types of steppers, unipolars and bipolars, and it is very important to know which type you are working with. In this experiment, we will use a unipolar stepper.

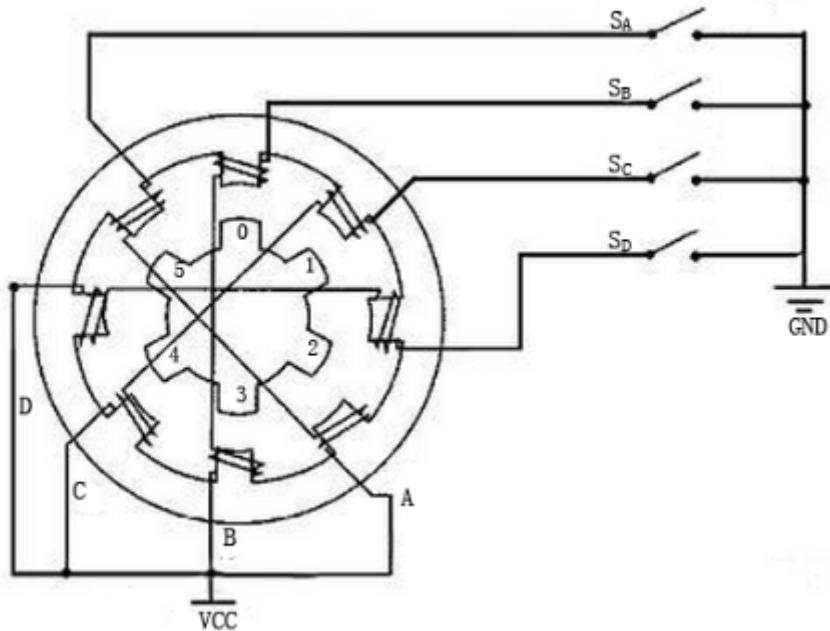
The stepper motor is a four-phase one, which uses a unipolarity DC power supply. As long as you electrify all phase windings of the motor by an appropriate timing sequence, you can make it rotate step by step. The schematic diagram of a four-phase reactive stepper motor:



In the figure, in the middle of the motor is a rotor – a gear-shaped permanent magnet. Around the rotor, 0 to 5 are teeth. Then more outside, there are 8 magnetic poles, with each two opposite ones connected by coil winding. So they form four pairs from A to D, which is called a phase. It has four lead wires to be connected with switches SA, SB, SC, and SD. Therefore, the four phases are in parallel in the circuit, and the two magnetic poles in one phase are in series.

#### Here's how a 4-phase stepper motor works:

When switch SB is power on, switch SA, SC, and SD is power off, and B-phase magnetic poles align with tooth 0 and 3 of the rotor. At the same time, tooth 1 and 4 generate staggered teeth with C- and D-phase poles. Tooth 2 and 5 generate staggered teeth with D- and A-phase poles. When switch SC is power on, switch SB, SA, and SD is power off, the rotor rotates under magnetic field of C-phase winding and that between tooth 1 and 4. Then tooth 1 and 4 align with the magnetic poles of C-phase winding. While tooth 0 and 3 generate staggered teeth with A- and B-phase poles, and tooth 2 and 5 generate staggered teeth with the magnetic poles of A- and D-phase poles. The similar situation goes on and on. Energize the A, B, C and D phases in turn, and the rotor will rotate in the order of A, B, C and D.



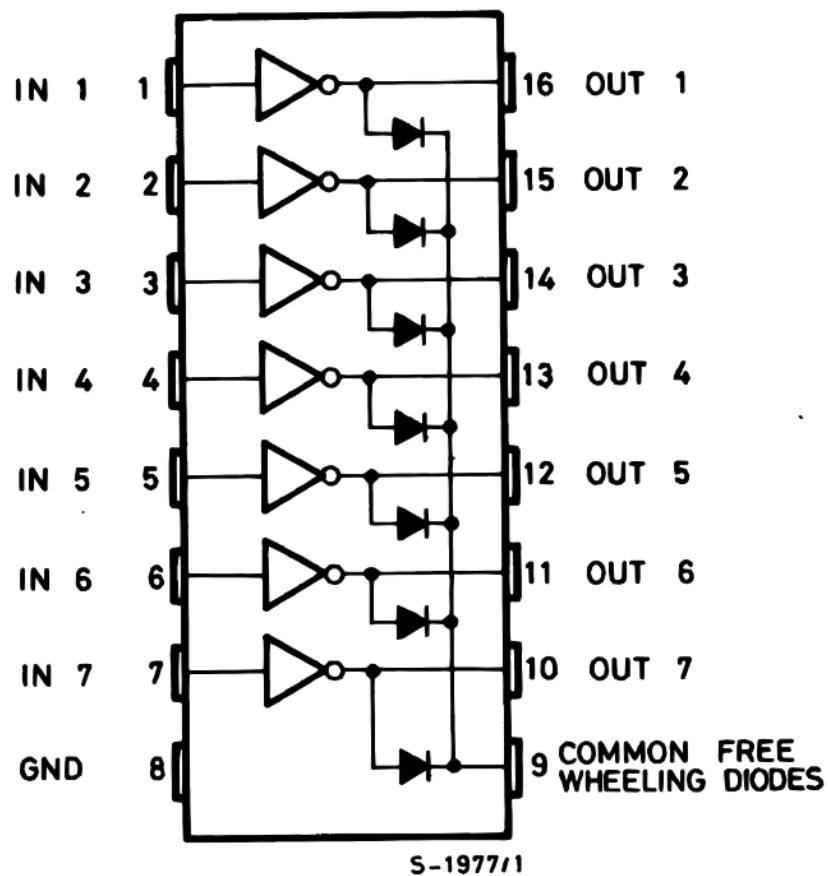
The four-phase stepper motor has three operating modes: single four-step, double four-step, and eight-step. The step angle for the single four-step and double four-step are the same, but the driving torque for the single four-step is smaller. The step angle of the eight-step is half that of the single four-step and double four-step. Thus, the eight-step operating mode can keep high driving torque and improve control accuracy.

The stator of Stepper Motor we use has 32 magnetic poles, so a circle needs 32 steps. The output shaft of the Stepper Motor is connected with a reduction gear set, and the reduction ratio is 1/64. **So the final output shaft rotates a circle requiring a  $32 \times 64 = 2048$  step.**

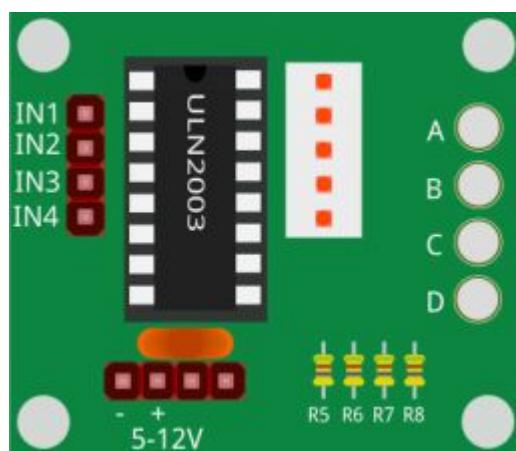
## ULN2003

To apply the motor in the circuit, a driver board needs to be used. Stepper Motor Driver-ULN2003 is a 7-channel inverter circuit. That is, when the input pin is at high level, the output pin of ULN2003 is at low level, and vice versa. If we supply high level to IN1, and low level to IN2, IN3 and IN4, then the output end OUT1 is at low level, and all the other output ends are at high level.

The internal structure of the chip is shown as below.

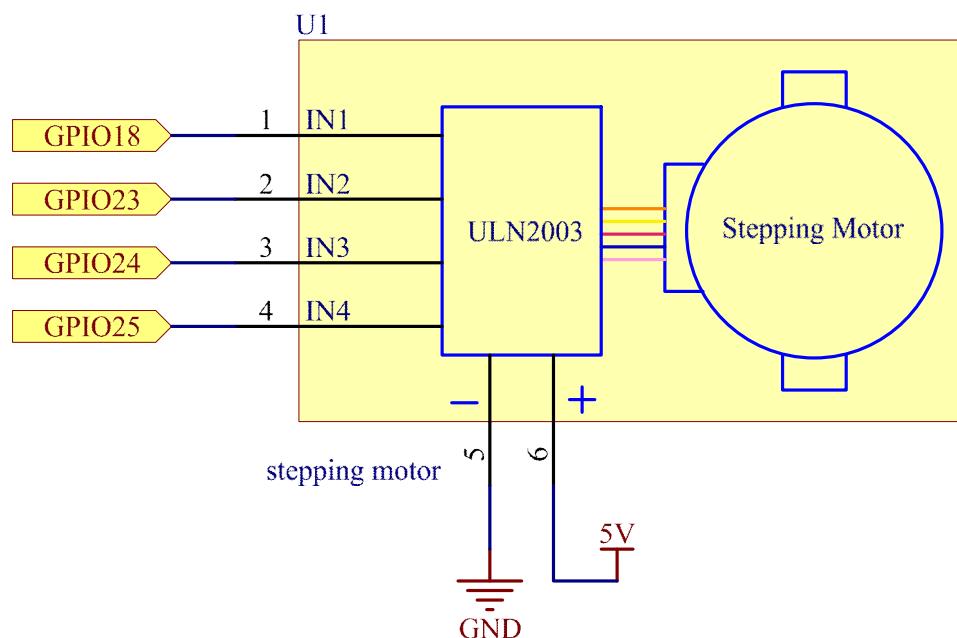


The stepper motor driver constituted by ULN2003 chip and 4 LEDs is shown as follows. On the board, IN1,IN2,IN3 and IN4 work as input and the four LEDs, A, B, C, D are the indicators of input pin. In addition, OUT1,OUT2, OUT3 and OUT4 are connected to SA, SB, SC and SD on the stepper motor driver. When the value of IN1 is set to a high level, A lights up; switch SA is power on, and the stepper motor rotates one step. The similar case repeats on and on. Therefore, just give the stepper motor a specific timing sequence, it will rotate step by step. The ULN2003 here is used to provide particular timing sequences for the stepper motor.



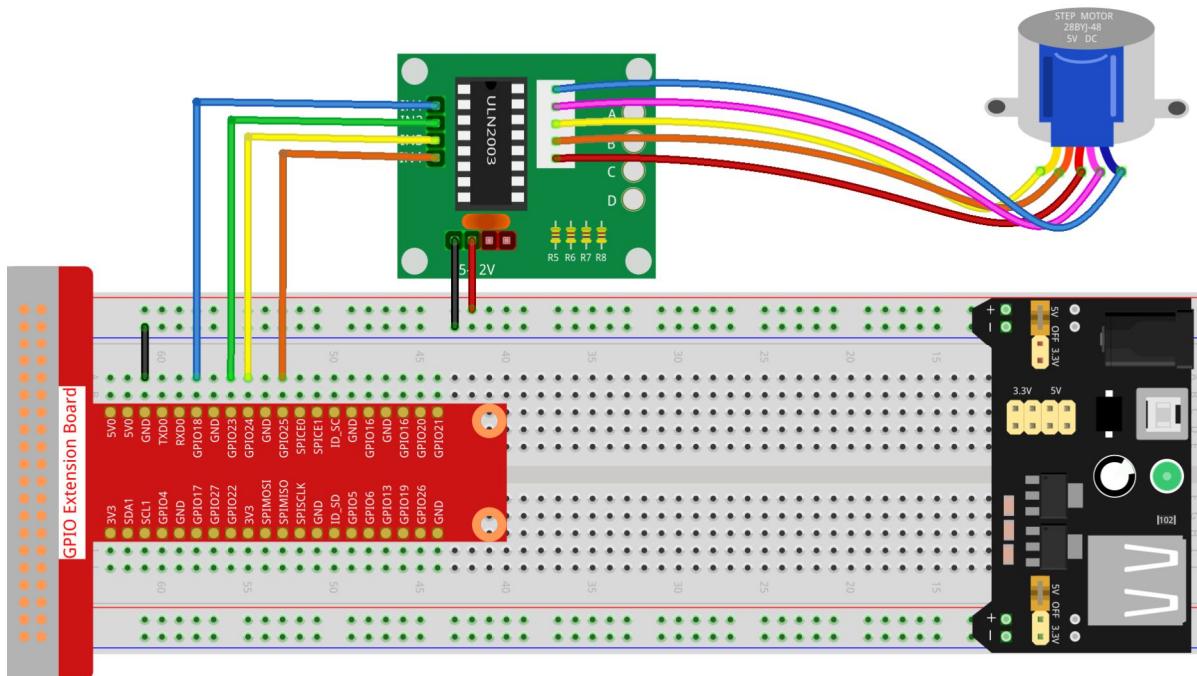
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	22
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/1.3.3/
```

**Step 3:** Compile the code.

```
gcc 1.3.3_SteppingMotor.c -lwiringPi
```

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

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

As the code runs, the stepper motor first turns clockwise for a round then stop for a second, after that, it rotates anticlockwise for a round; subsequently, the motor stops for a second. This series of actions will be executed repeatedly.

## Code

```
#include <stdio.h>
#include <wiringPi.h>

const int motorPins[]={1,4,5,6}; //pins connected to four phase
const int antiClk[]={0x01,0x02,0x04,0x08};
const int clk[]={0x08,0x04,0x02,0x01};

void moveSteps(int direction, int steps){
    int step;
    int i=0,j=0;
    for(step=0;step<steps;step++){
        for (j=0;j<4;j++){ //cycle according to power supply order
            for (i=0;i<4;i++){ //assign to each pin, a total of 4 pins
                if(direction == 1) //clockwise
                    digitalWrite(motorPins[i],antiClk[j] == (1<<i));
                else //anticlockwise
                    digitalWrite(motorPins[i],clk[j] == (1<<i));
            }
            delay(3);
        }
    }
}

void motorStop(){ //function used to stop rotating
    int i;
    for(i=0;i<4;i++){
}
```

```

digitalWrite(motorPins[i],LOW);
}
}

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

for(i=0;i<4;i++){
pinMode(motorPins[i],OUTPUT);
}

while(1){
moveSteps(1,512);
delay(1000);
moveSteps(0,512);
delay(1000);
}
return 0;
}

```

## Code Explanation

```

const int antiClk[]={0x01,0x02,0x04,0x08};
const int clk[]={0x08,0x04,0x02,0x01};

```

These two values are used to indicate the rotation of the Stepper Motor, and the values of them are 0001,0010, 0100, and 1000 respectively that correspond to the electrification condition of four-phase motor when it rotates.

```

for (j=0;j<4;j++){ //cycle according to power supply order
    for (i=0;i<4;i++){ //assign to each pin
        digitalWrite(motorPins[i],antiClk[j] == (1 << i));
    }
}

```

This two-layer cycle means that the stepper motor runs for a complete cycle (four steps of the four-phase stepper motor is one cycle).

```
void moveSteps(int direction, int steps){
    int step;
    int i=0,j=0;
    for(step=0;step<steps;step++){
        for (j=0;j<4;j++){ //cycle according to power supply order
            for (i=0;i<4;i++){ //assign to each pin, a total of 4 pins
                if(direction == 1) //clockwise
                    digitalWrite(motorPins[i],antiClk[j] == (1<<i));
                else //anticlockwise
                    digitalWrite(motorPins[i],clk[j] == (1<<i));
            }
            delay(3);
        }
    }
}
```

This function is used to control the work of the stepper motor, the variable "direction" is used to determine the rotation direction, and the variable "steps" is used to determine the times of the work of stepper motor.

```
moveSteps(1,512);
```

It takes 2048 steps for the output shaft to turn a circle, and  $2048/4=512$  times for the stepper motor to work.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 1.3.3_SteppingMotor.py
```

As the code runs, the stepper motor first turns clockwise for a round then stop for a second, after that, it rotates anticlockwise for a round; subsequently, the motor stops for a second. This series of actions will be executed repeatedly.

## Code

```
import RPi.GPIO as GPIO
import time

motorPins = (12, 16, 18, 22) #pins connected to four phase
antiClk = (0x01,0x02,0x04,0x08)
```

```

clk = (0x08,0x04,0x02,0x01)

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    for pin in motorPins:
        GPIO.setup(pin,GPIO.OUT)

def moveSteps(direction, steps):
    for step in range(steps):
        for j in range(0,4,1): #cycle according to power supply order
            for i in range(0,4,1): #assign to each pin
                if (direction == 1):#clockwise
                    GPIO.output(motorPins[i],((antiClk[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
                else :           #anticlockwise
                    GPIO.output(motorPins[i],((clk[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        time.sleep(0.003)  #the delay can not be less than 3ms

#function used to stop rotating
def motorStop():
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)

def loop():
    while True:
        moveSteps(1,512)
        time.sleep(1)
        moveSteps(0,512)
        time.sleep(1)

def destroy():
    GPIO.cleanup()          # Release resource

if __name__ == '__main__':  # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    destroy()

```

## Code Explanation

```
antiClk = (0x01,0x02,0x04,0x08)
clk = (0x08,0x04,0x02,0x01)
```

These two values are used to indicate the rotation of the Stepper Motor, and the values of them are 0001,0010, 0100, and 1000 respectively that correspond to the electrification condition of four-phase motor when it rotates.

```
for j in range(0,4,1): #cycle according to power supply order
    for i in range(0,4,1): #assign to each pin
        GPIO.output(motorPins[i],((antiClk[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
```

This two-layer cycle means that the stepper motor runs for a complete cycle (four steps of the four-phase stepper motor is one cycle).

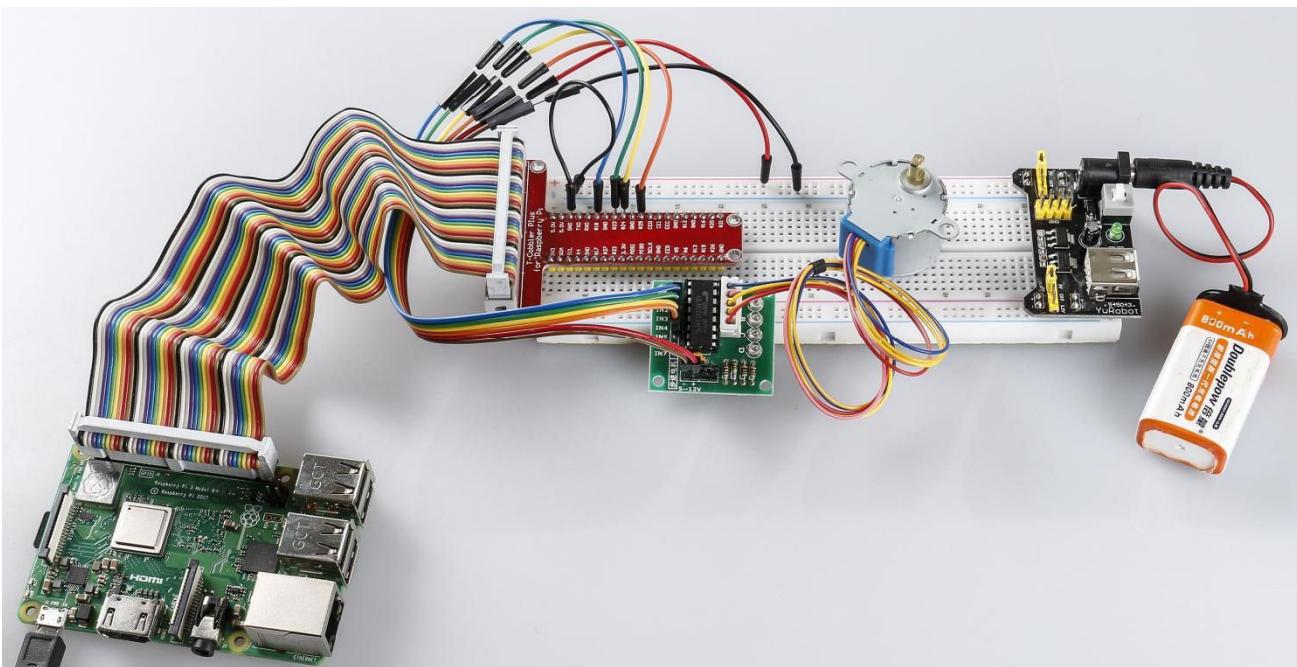
```
def moveSteps(direction, steps):
    for step in range(steps):
        for j in range(0,4,1): #cycle according to power supply order
            for i in range(0,4,1): #assign to each pin
                if (direction == 1):#clockwise
                    GPIO.output(motorPins[i],((antiClk[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
                else :           #anticlockwise
                    GPIO.output(motorPins[i],((clk[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        time.sleep(0.003)  #the delay can not be less than 3ms
```

This function is used to control the work of the stepper motor, in which direction is used to determine the rotation direction, and steps are used to determine the times of the work of stepper motor.

```
moveSteps(1,512)
```

It takes 2048 steps for the output shaft to turn a circle, and  $2048/4=512$  times for the stepper motor to work.

## Phenomenon Picture



# 2 Input

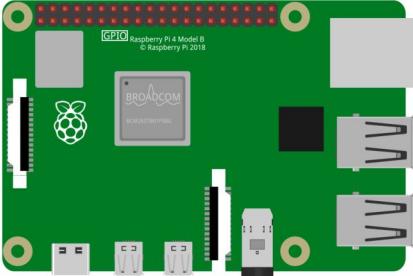
## 2.1 Controllers

### 2.1.1 Button

#### Introduction

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

#### Components

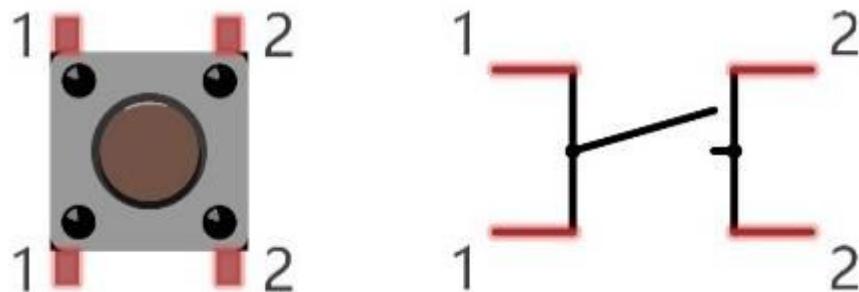
1 * Raspberry Pi	1 * T-Extension Board	1 * LED
	 GPIO Extension Board Pinout: 3V3 SVO SDA1 SVO SCL1 GND I2C SDA2 I2C GND RXD0 GPIO17 GPIO18 GPIO27 GND GPIO22 GPIO23 3V GND SPI MOSI GND SPI MISO GPIO15 SPI SCLK SPICE0 GND SPICE1 JD4 GND JD5 GND JD6 GPIO16 JD7 GPIO13 JD8 GND JD9 GPIO19 JD10 GPIO16 JD11 GND JD12 GPIO20 JD13 GND JD14 GPIO21	
1 * 40-pin Cable		1 * Resistor(220Ω)  Several Jumper Wires 
1 * Breadboard		1 * Button 

## Principle

### Button

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

Two pins on the left are connected, and the one on the right is similar to the left, which is shown below:



The symbol shown as below is usually used to represent a button in circuits.



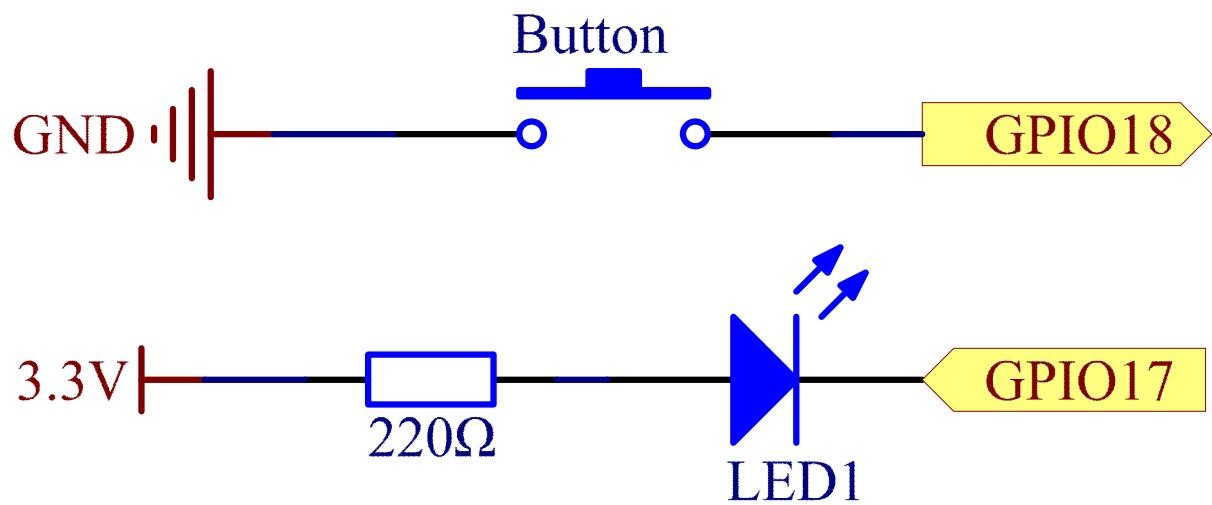
When the button is pressed, the 4 pins are connected, thus closing the circuit.

### Schematic Diagram

Use a normally open button as the input of Raspberry Pi, the connection is shown in the schematic diagram below. When the button is pressed, the GPIO18 will turn into low level (0V). We can detect the state of the GPIO18 through programming. That is, if the GPIO18 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.

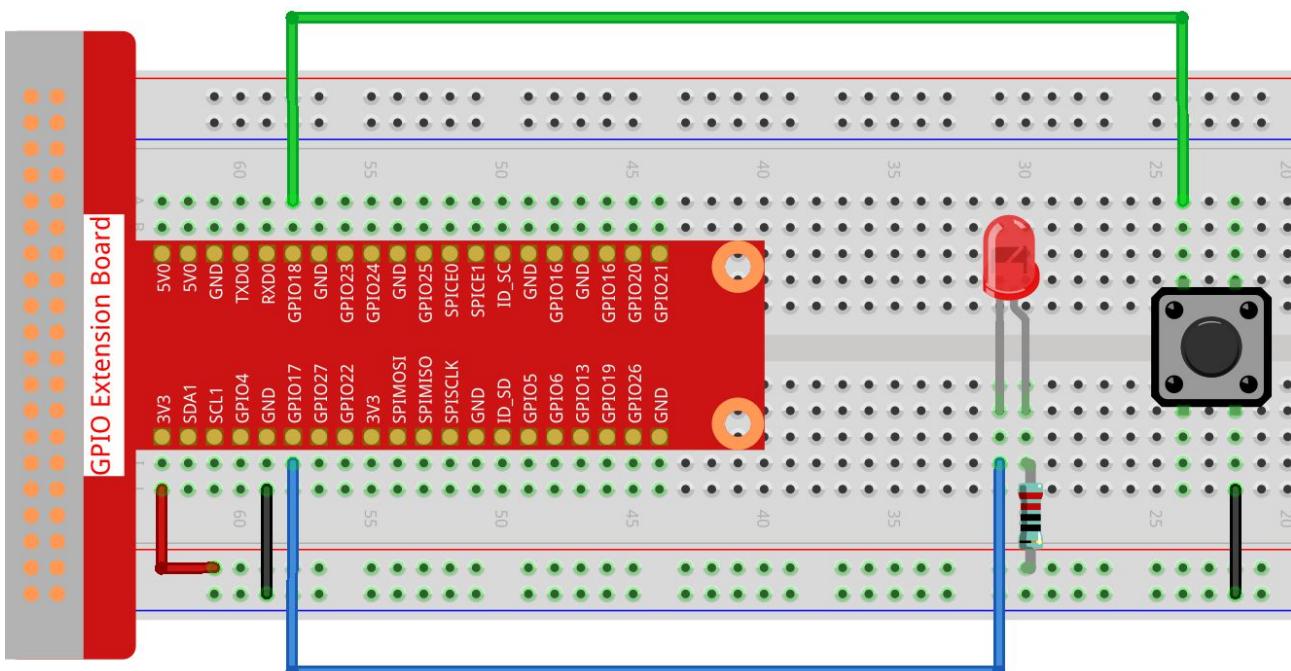
**Note:** The longer pin of the LED is the anode and the shorter one is the cathode.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18



## Experimental Procedures

**Step 1:** Build the circuit.



➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.1/
```

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

**Step 3:** Compile the code.

```
gcc 2.1.1_Button.c -lwiringPi
```

## Step 4: Run the executable file.

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

After the code runs, press the button, the LED lights up; otherwise, turns off.

## Code

```
#include <wiringPi.h>
#include <stdio.h>

#define LedPin    0
#define ButtonPin  1

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

    pinMode(LedPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    // Pull up to 3.3V,make GPIO1 a stable level
    pullUpDnControl(ButtonPin, PUD_UP);
    digitalWrite(LedPin, HIGH);

    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");
        }
    }
    return 0;
}
```

## Code Explanation

```
#define LedPin 0
```

Pin GPIO17 in the T\_ Extension Board is equal to the GPIO0 in the wiringPi.

```
#define ButtonPin 1
```

ButtonPin is connected to GPIO1.

```
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 (OV). 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");  
    }  
}
```

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

### ➤ For Python Language Users

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

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

### Step 3: Run the code.

```
sudo python 2.1.1_Button.py
```

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.

### Code

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as LED pin
LedPin = 17
# Set GPIO18 as button pin
BtnPin = 18

# Set Led status to True(OFF)
Led_status = True

# 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)
    # Set BtnPin's mode to input,
    # and pull up to high (3.3V)
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    # 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
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'

# Define a main function for main process
def main():
    while True:
        # Don't do anything.
        time.sleep(1)

# 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 program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

LedPin = 17

Set GPIO17 as LED pin

BtnPin = 18

Set GPIO18 as button pin

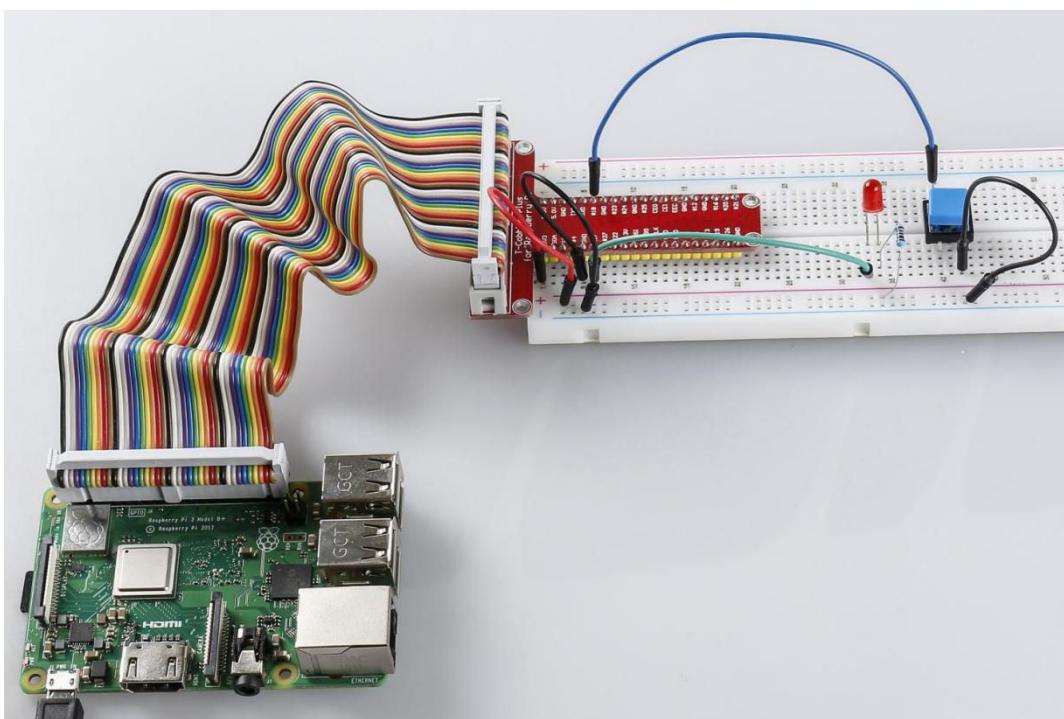
GPIO.add\_event\_detect(BtnPin, GPIO.FALLING, callback=swLed)

Set up a falling detect on BtnPin, and then when the value of BtnPin changes from a high level to a low level, it means that the button is pressed. The next step is calling the function, swled.

```
def swLed(ev=None):
    global Led_status
    # Switch led status(on-->off; off-->on)
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
```

Define a callback function as button callback. When the button is pressed at the first time , and the condition, not Led\_status is false, GPIO.output() function is called to light up the LED. As the button is pressed once again, the state of LED will be converted from false to true, thus the LED will turn off.

## Phenomenon Picture

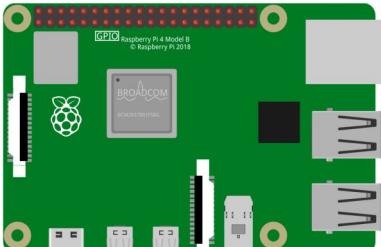
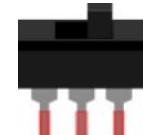
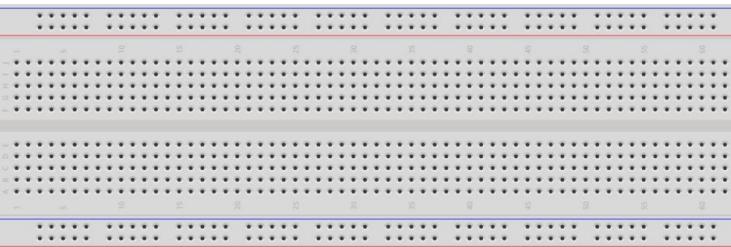


## 2.1.2 Slide Switch

### Introduction

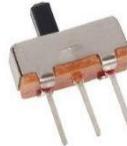
In this lesson, we will learn how to use a slide switch. Usually, the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Slide Switch
	 GPIO Extension Board Pinout: <ul style="list-style-type: none"><li>● 3V3 5V(B)</li><li>● SDA1 5V(B)</li><li>● SCL1 GND(B)</li><li>● GND RXD0(B)</li><li>● GPIO4 TXD0(B)</li><li>● GND RXD1(B)</li><li>● GPIO17 GPIO18(B)</li><li>● GND GND(B)</li><li>● GPIO27 GPIO22(B)</li><li>● 3V3 GPIO23(B)</li><li>● SPI(MOSI) GND(B)</li><li>● SPI(MISO) GPIO25(B)</li><li>● SPI(SCLK) GPIO24(B)</li><li>● GND SPI(CS)(B)</li><li>● ID_SD ID(SC)(B)</li><li>● GPIO5 GND(B)</li><li>● GPIO13 GND(B)</li><li>● GPIO19 GPIO16(B)</li><li>● GPIO26 GPIO20(B)</li><li>● GND GPIO21(B)</li></ul>	
1 * 40-pin Cable		2 * LED
		1 * 104 Capacitor
		Several Jumper Wires
		2 * Resistor(220Ω)
		
		1 * Resistor 10KΩ
		

## Principle

### Slide Switch

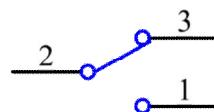


A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The slide switch is commonly used in low-voltage circuit. It has the features of flexibility and stability, and applies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

How it works: Set the middle pin as the fixed one. When you pull the slide to the left, the two pins on the left are connected; when you pull it to the right, the two pins on the right are connected. Thus, it works as a switch connecting or disconnecting circuits. See the figure below:



The circuit symbol of the slide switch is shown as below. The pin2 in the figure refers to the middle pin.



## Capacitor

The capacitor is a component that has the capacity to store energy in the form of electrical charge or to produce a potential difference (Static Voltage) between its plates, much like a small rechargeable battery.

### Standard Units of Capacitance

Microfarad ( $\mu\text{F}$ )  $1\mu\text{F} = 1/1,000,000 = 0.000001 = 10^{-6} \text{ F}$

Nanofarad ( $\text{nF}$ )  $1\text{nF} = 1/1,000,000,000 = 0.000000001 = 10^{-9}\text{F}$

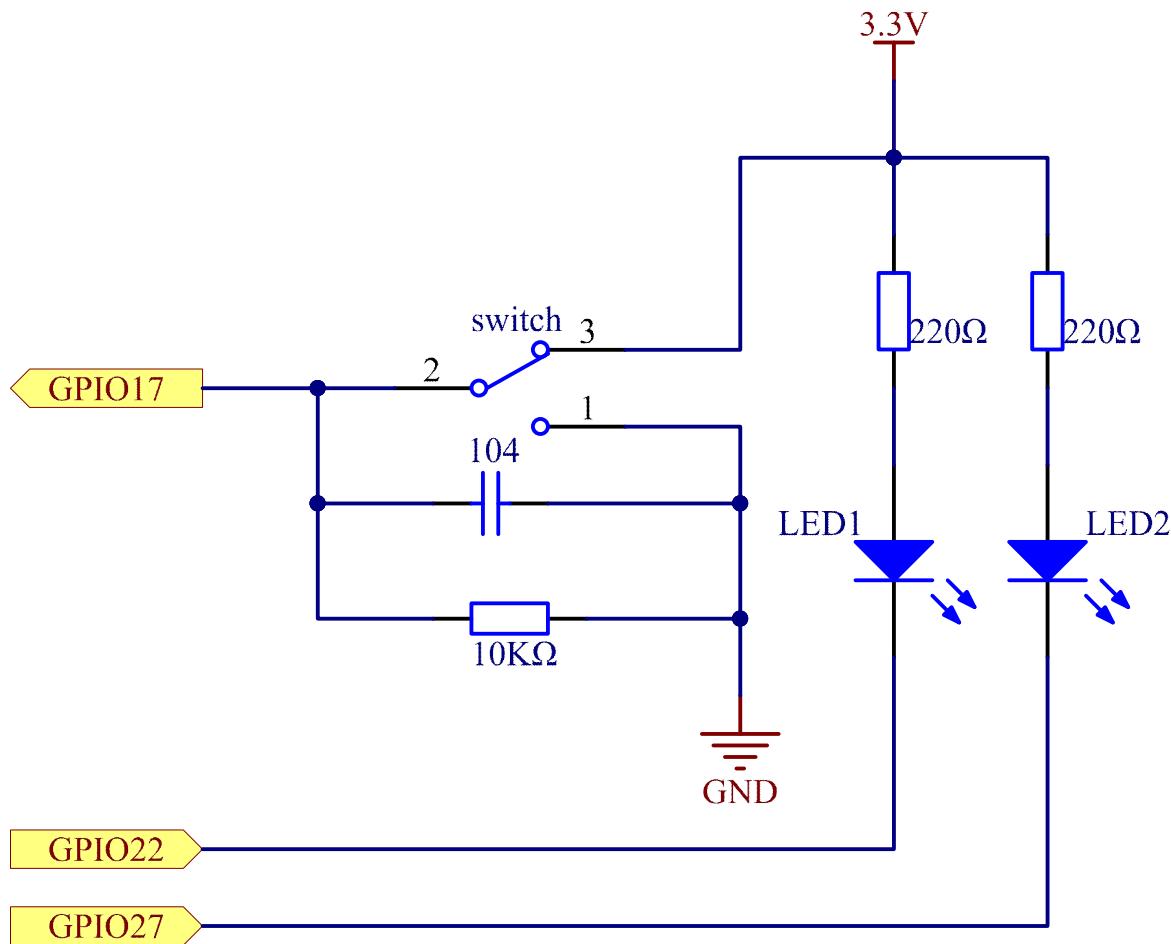
Picofarad ( $\text{pF}$ )  $1\text{pF} = 1/1,000,000,000,000 = 0.000000000001 = 10^{-12}\text{F}$

**Note:** Here we use **104 capacitor( $10 \times 10^4 \text{ pF}$ )**. Just like the ring of resistors, the numbers on the capacitors help to read the values once assembled onto the board. The first two digits represent the value and the last digit of the number means the multiplier. Thus 104 represents a power of  $10 \times 10$  to 4 (in pF) equal to 100 nF.

## Schematic Diagram

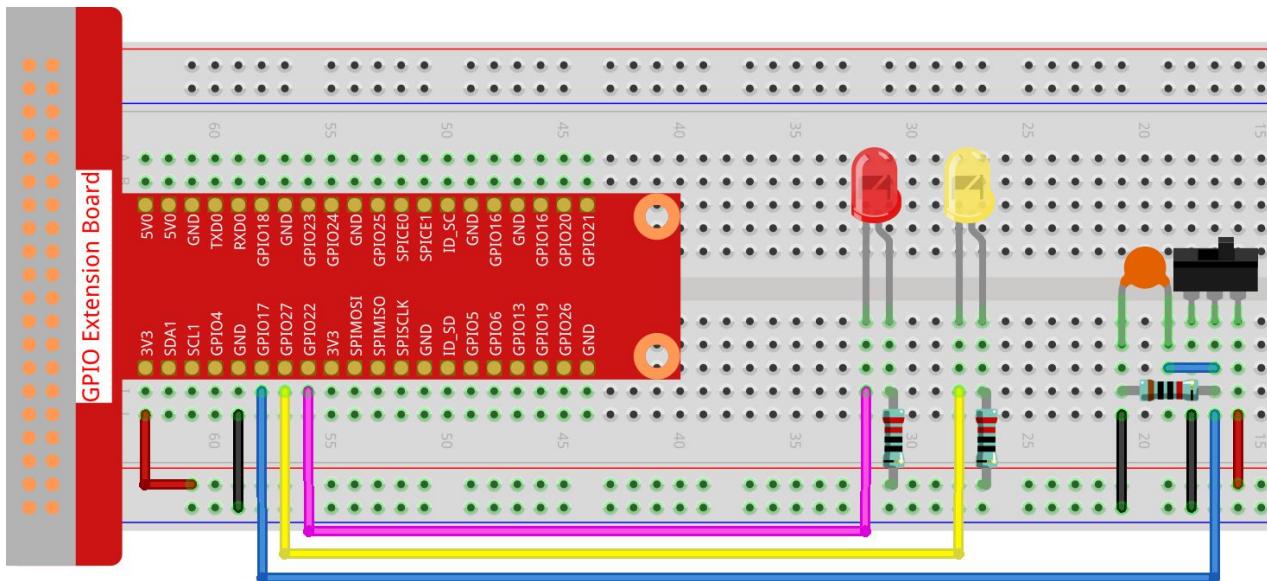
Connect the middle pin of the Slide Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.2
```

#### Step 3: Compile.

```
gcc 2.1.2_Slider.c -lwiringPi
```

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

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

While the code is running, get the switch connected to the left, then the yellow LED lights up; to the right, the red light turns on.

### Code

```
#include <wiringPi.h>
#include <stdio.h>

#define slidePin      0
#define led1          3
#define led2          2

int main(void)
{
    // When initialize wiring failed, print message to screen
    if(wiringPiSetup() == -1){
```

```

printf("setup wiringPi failed !");
return 1;
}
pinMode(slidePin, INPUT);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
while(1){
    // slide switch high, led1 on
    if(digitalRead(slidePin) == 1){
        digitalWrite(led1, LOW);
        digitalWrite(led2, HIGH);
        printf("LED1 on\n");
    }
    // slide switch low, led2 on
    if(digitalRead(slidePin) == 0){
        digitalWrite(led2, LOW);
        digitalWrite(led1, HIGH);
        printf(".....LED2 on\n");
    }
}
return 0;
}

```

## Code Explanation

```

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 high level at the middle pin, so the LED1 is on and LED2 off

```

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

```

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

## ➤ For Python Language Users

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

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

**Step 3:** Run.

```
sudo python 2.1.2_slider.py
```

While the code is running, get the switch connected to the left, then the yellow LED lights up; to the right, the red light turns on.

## Code

```
import RPi.GPIO as GPIO
import time

# Set GPIO17 as slide switch pin, GPIO22 as led1 pin, GPIO27 as led2 pin
slidePin = 17
led1Pin = 22
led2Pin = 27

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set slidePin input
    # Set ledPin output,
    # and initial level to High(3.3v)
    GPIO.setup(slidePin, GPIO.IN)
    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.HIGH)
    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
def main():
    while True:
        # slide switch high, led1 on
        if GPIO.input(slidePin) == 1:
            print('  LED1 ON  ')
            GPIO.output(led1Pin, GPIO.LOW)
        else:
            print('  LED2 ON  ')
            GPIO.output(led2Pin, GPIO.LOW)
```

```

GPIO.output(led2Pin, GPIO.HIGH)

# slide switch low, led2 on
if GPIO.input(slidePin) == 0:
    print ('  LED2 ON  ')
    GPIO.output(led2Pin, GPIO.LOW)
    GPIO.output(led1Pin, 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(led1Pin, GPIO.HIGH)
    GPIO.output(led2Pin, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

# If run this script directly, do:
if __name__ == '__main__':
    setup()
    try:
        main()
    # When 'Ctrl+C' is pressed, the program
    # destroy() will be executed.
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```

if GPIO.input(slidePin) == 1:
    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 high level at the middle pin, so the LED1 is on and LED2 off.

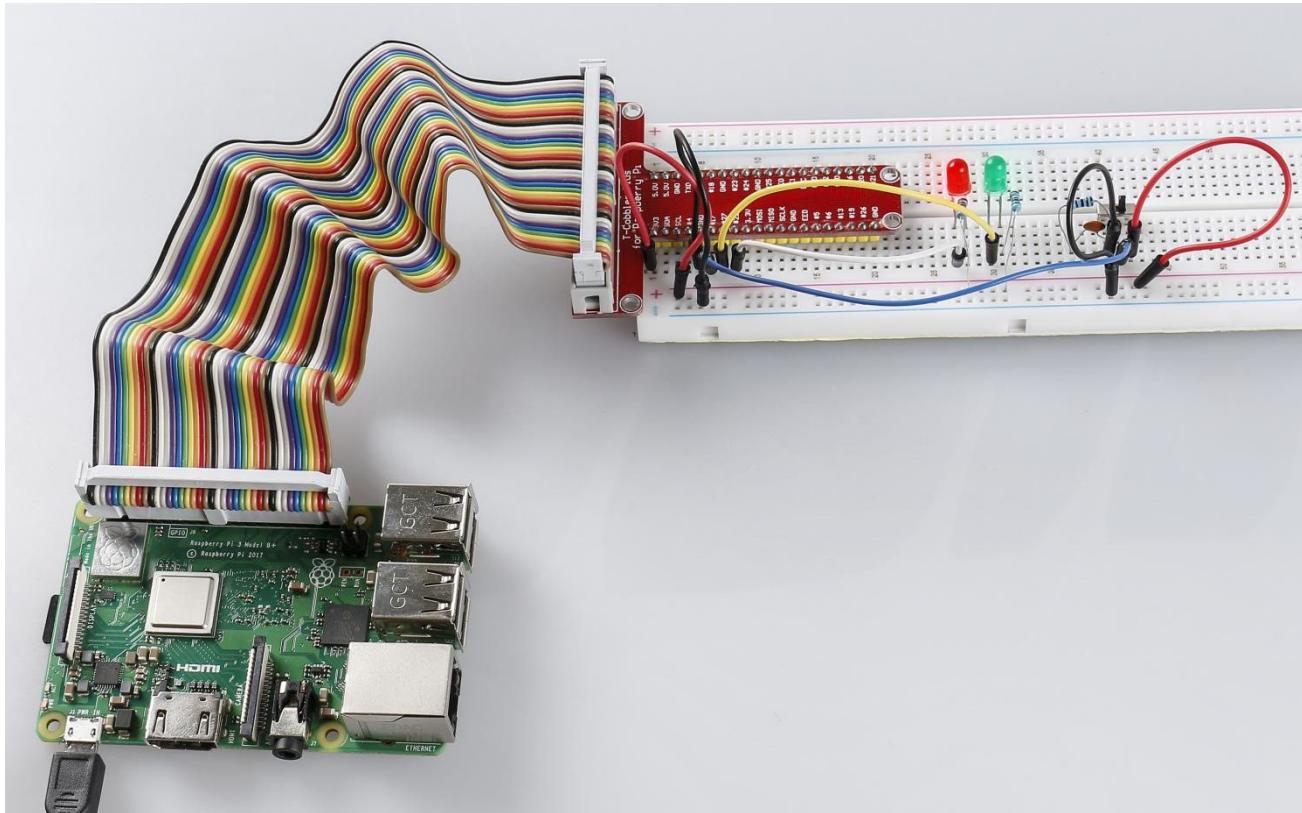
```

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

```

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

## Phenomenon Picture

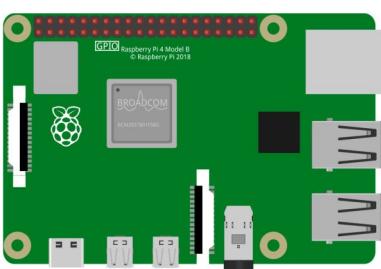
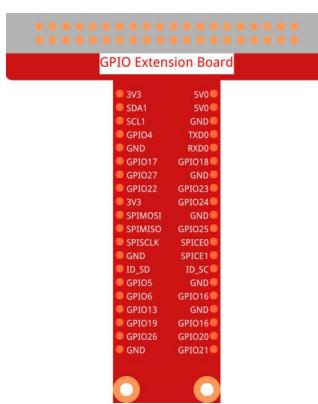
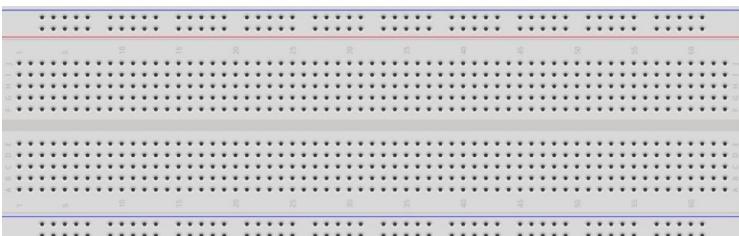
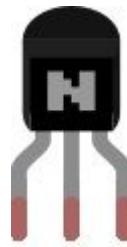


## 2.1.3 Relay

### Introduction

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

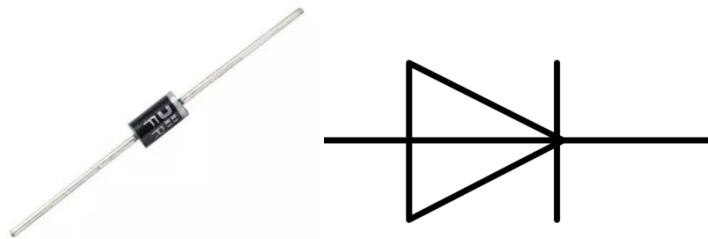
### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Relay
	 GPIO Extension Board Pinout: 3V3, SDA1, SCL1, GND, GPIO4, TXD0, RXD0, GND, GPIO17, GPIO18, GND, GPIO27, GPIO23, GND, GPIO22, GPIO24, GND, SPI_MOSI, SPI_MISO, SPI_SCLK, GND, ID_SD, SPI_CE1, ID_SC, GND, GPIO5, GND, GPIO6, GPIO16, GND, GPIO13, GND, GPIO19, GPIO16, GND, GPIO26, GPIO20, GND, GND, GPIO21, GND	
1 * 40-pin Cable		1 * 1N4007 Diode
		
1 * Breadboard	1 * LED	1 * NPN Transistor
		
Several Jumper Wires		
1 * Resistor(220Ω)		
1 * Resistor 1KΩ		

## Principle

### Diode

A diode is a two-terminal component in electronics with a unidirectional flow of current. It offers low resistance in the direction of current flow and offers high resistance in the opposite direction. Diodes are mostly used to prevent damage to components, especially due to electromotive force in circuits which are usually polarized.



The two terminals of a diode are polarized, with the positive end called anode and the negative end called cathode. The cathode is usually made of silver or has a color band. Controlling the direction of current flow is one of the key features of diodes — the current in a diode flows from anode to cathode. The behavior of a diode is similar to the behavior of a check valve. One of the most important characteristics of a diode is the non-linear current voltage. If higher voltage is connected to the anode, then current flows from anode to cathode, and the process is known as forward bias. However, if the higher voltage is connected to the cathode, then the diode does not conduct electricity, and the process is called reverse bias.

### Relay

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

There are 5 parts in every relay:

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

to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

3. **Spring** – When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

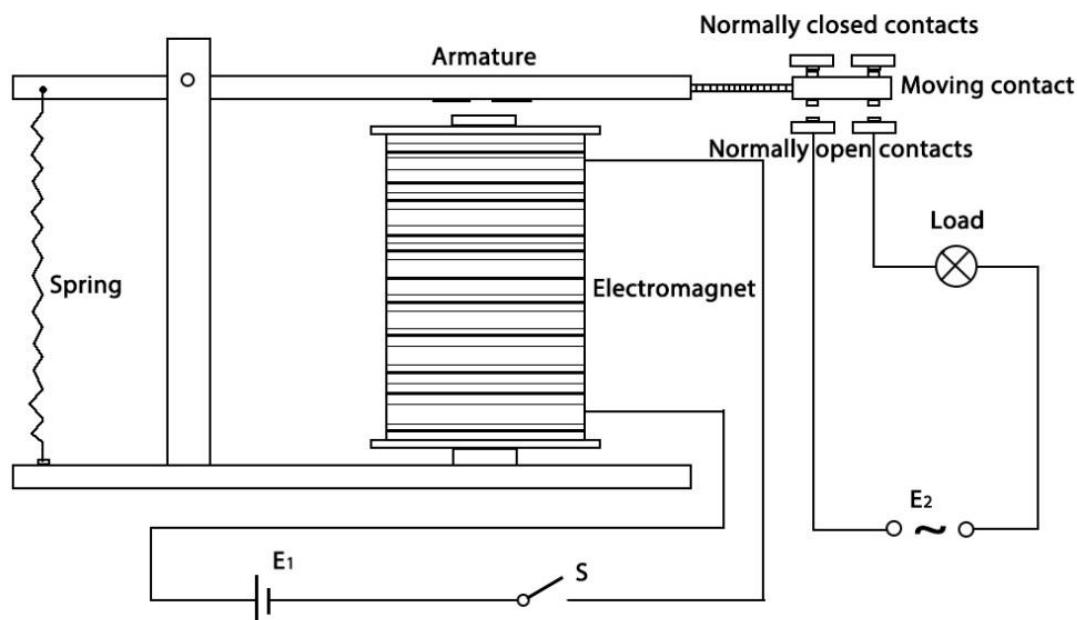
4. Set of electrical **contacts** – There are two contact points:

- Normally open - connected when the relay is activated, and disconnected when it is inactive.
- Normally close – not connected when the relay is activated, and connected when it is inactive.

5. Molded frame – Relays are covered with plastic for protection.

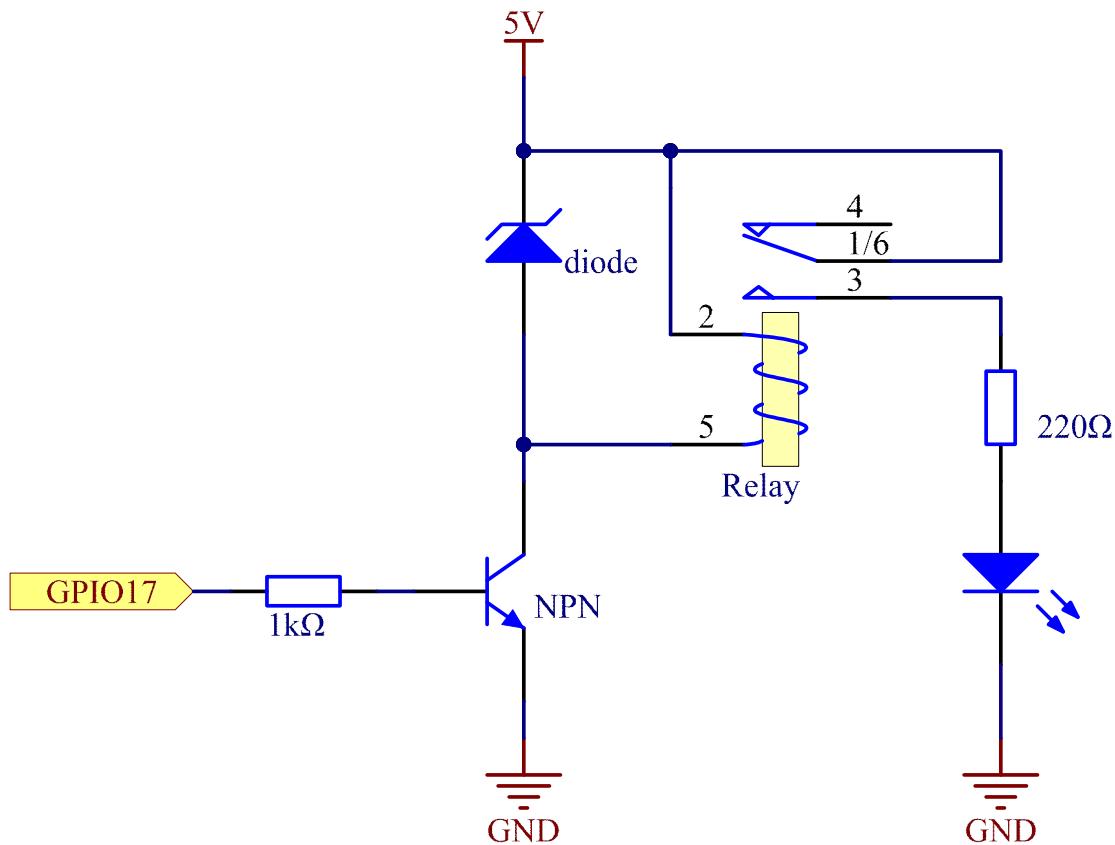
## Working of Relay

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



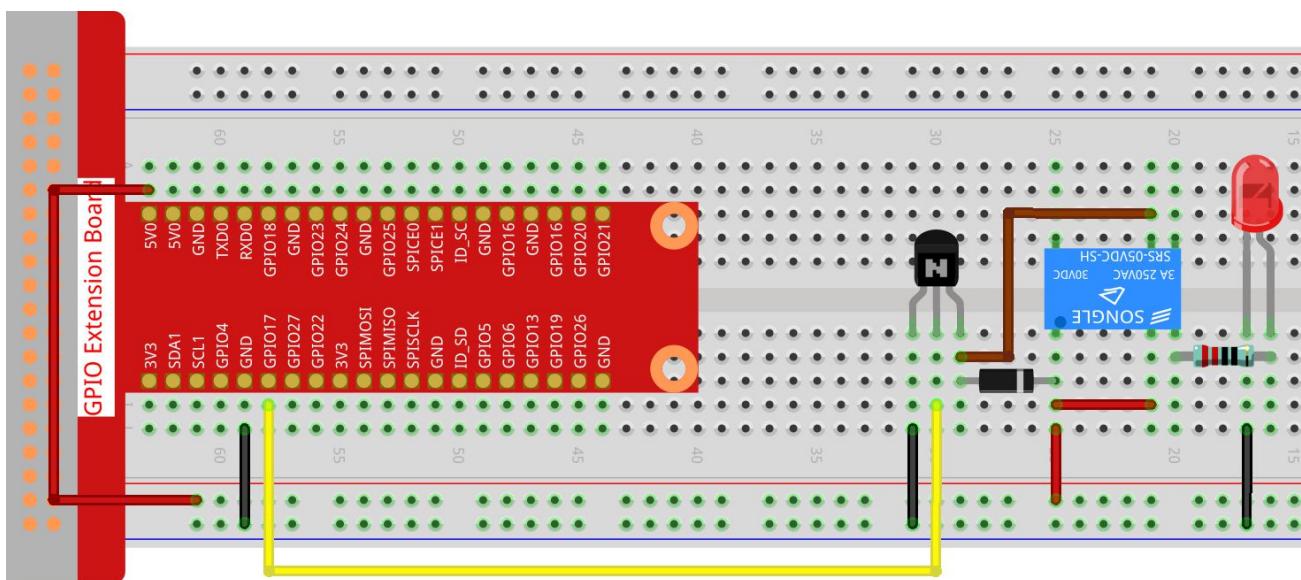
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.3
```

**Step 3:** Compile the code.

```
gcc 2.1.3_Relay -lwiringPi
```

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

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

After the code runs, the LED will light up. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.

## Code

```
#include <wiringPi.h>
#include <stdio.h>
#define RelayPin 0

int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(RelayPin, OUTPUT); //set GPIO17(GPIO0) output
    while(1){
        // Tick
        printf("Relay Open.....\n");
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
        printf(".....Relay Close\n");
        digitalWrite(RelayPin, HIGH);
        delay(1000);
    }
    return 0;
}
```

## Code Explanation

```
digitalWrite(RelayPin, LOW);
```

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

```
digitalWrite(RelayPin, HIGH);
```

set the I/O port as high level (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes, LED lights up.

### ➤ For Python Users:

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

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

**Step 3:** Run.

```
sudo python 2.1.3_relay.py
```

While the code is running, the LED lights up. In addition, you can hear a ticktock caused by breaking normally close contact and closing normally open contact.

## Code

```
import RPi.GPIO as GPIO
import time

# GPIO0 connect to relay's control pin
# led connect to relay's NormalOpen pin
# 5v connect to relay's com pin
# Set #17 as contral pin
relayPin = 17

# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set relayPin's mode to output,
    # and initial level to High(3.3v)
    GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.HIGH)

# Define a main function for main process
```

```
def main():
    while True:
        print ('Relay open...')
        # Tick
        GPIO.output(relayPin, GPIO.LOW)
        time.sleep(1)
        print ('...Relay close')
        # Tock
        GPIO.output(relayPin, GPIO.HIGH)
        time.sleep(1)

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

## Code Explanation

GPIO.output(relayPin, GPIO.LOW)

Set the pins of transistor as low level to let the relay open, LED does not turn on.

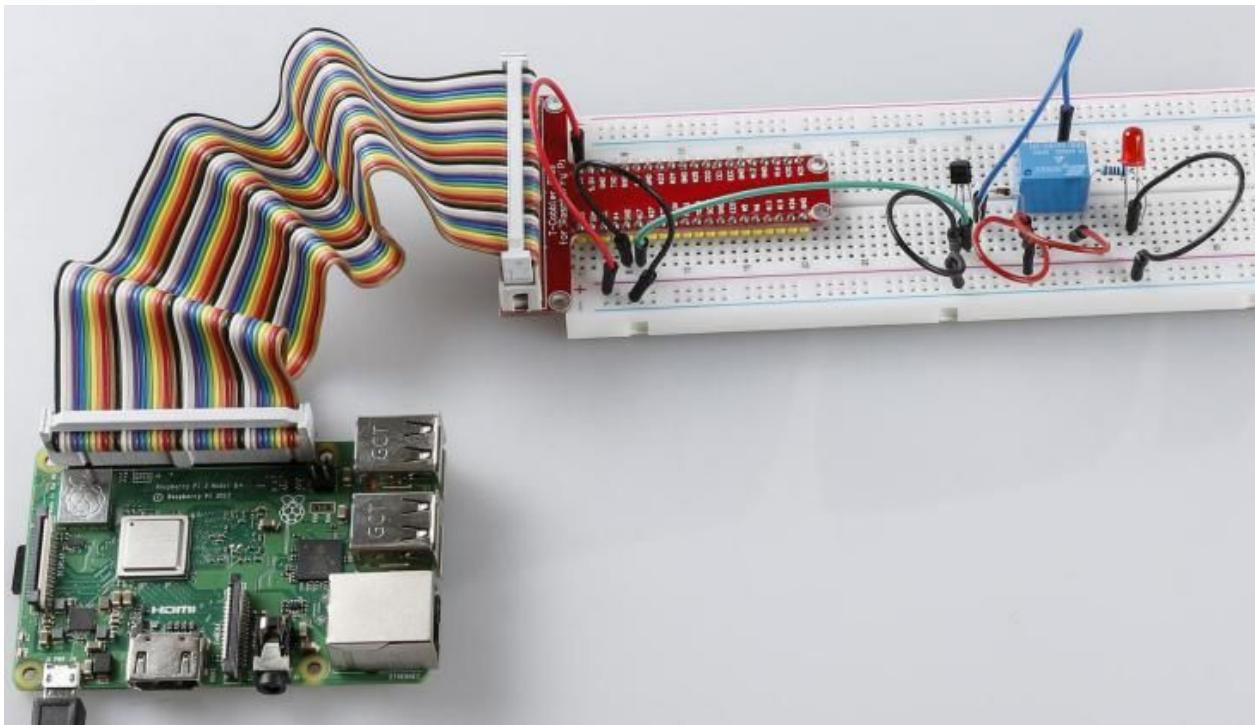
time.sleep(1)

wait for 1 second.

GPIO.output(relayPin, GPIO.HIGH)

Set the pins of the transistor as low level to actuate the relay, LED lights up.

## Phenomenon Picture

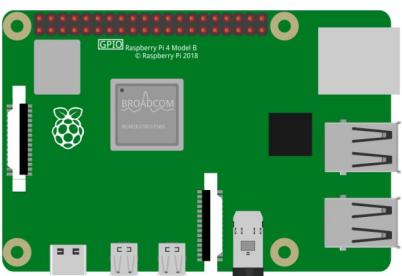
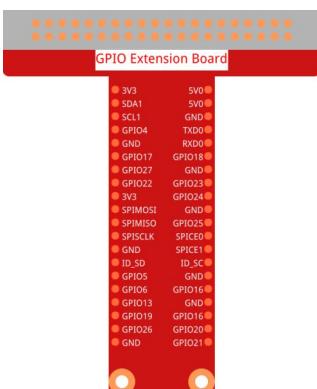
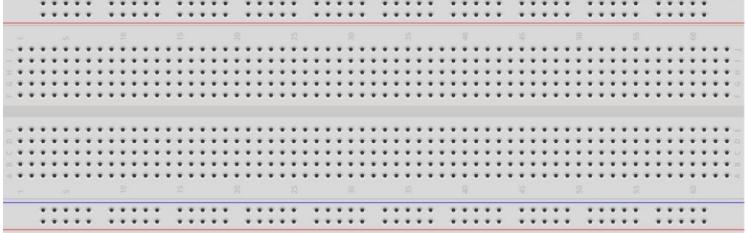


## 2.1.4 Potentiometer

### Introduction

The ADC function can be used to convert analog signals to digital signals, and in this experiment, PCF8591 is used to get the function involving ADC. Here, we implement this process by using potentiometer. Potentiometer changes the physical quantity -- voltage, which is converted by the ADC function and is programmed to change the LED brightness accordingly.

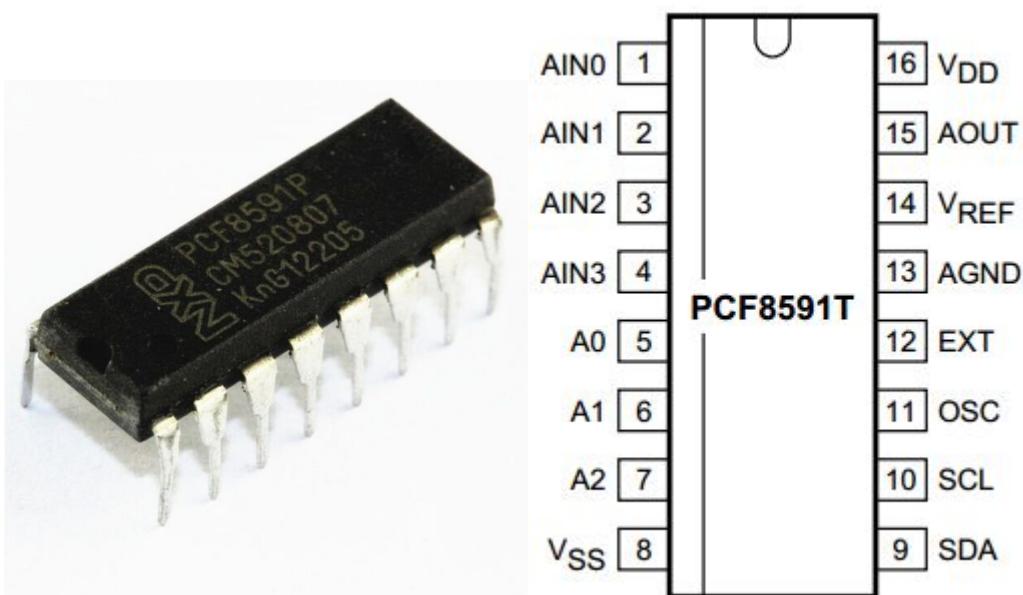
### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Potentiometer
	 GPIO Extension Board Pinout: 3V3 SDA1 SDA1 SCL1 SCL1 GND GND GPIO4 TXD0 TXD0 GND RXD0 RXD0 GPIO17 GPIO18 GND GND GPIO27 GPIO23 GPIO24 3V3 GND GND SPIMOSI SPIMOSI GND GND SPIMISO SPIMISO GND GND SPISCLK SPISCLK SPICE0 GND SPICE1 ID_SD ID_SD ID_SC ID_SC GPIO5 GPIO5 GND GND GPIO6 GPIO16 GND GND GPIO13 GND GND GPIO19 GPIO16 GND GND GPIO26 GPIO20 GND GND GND GND GND GND	
1 * 40-pin Cable		1 * PCF8591
		
1 * Breadboard		1 * LED
		
Several Jumper Wires		
1 * Resistor(220Ω)		
2 * Resistor 10KΩ		

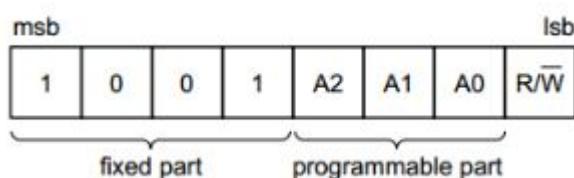
## Principle

### PCF8591

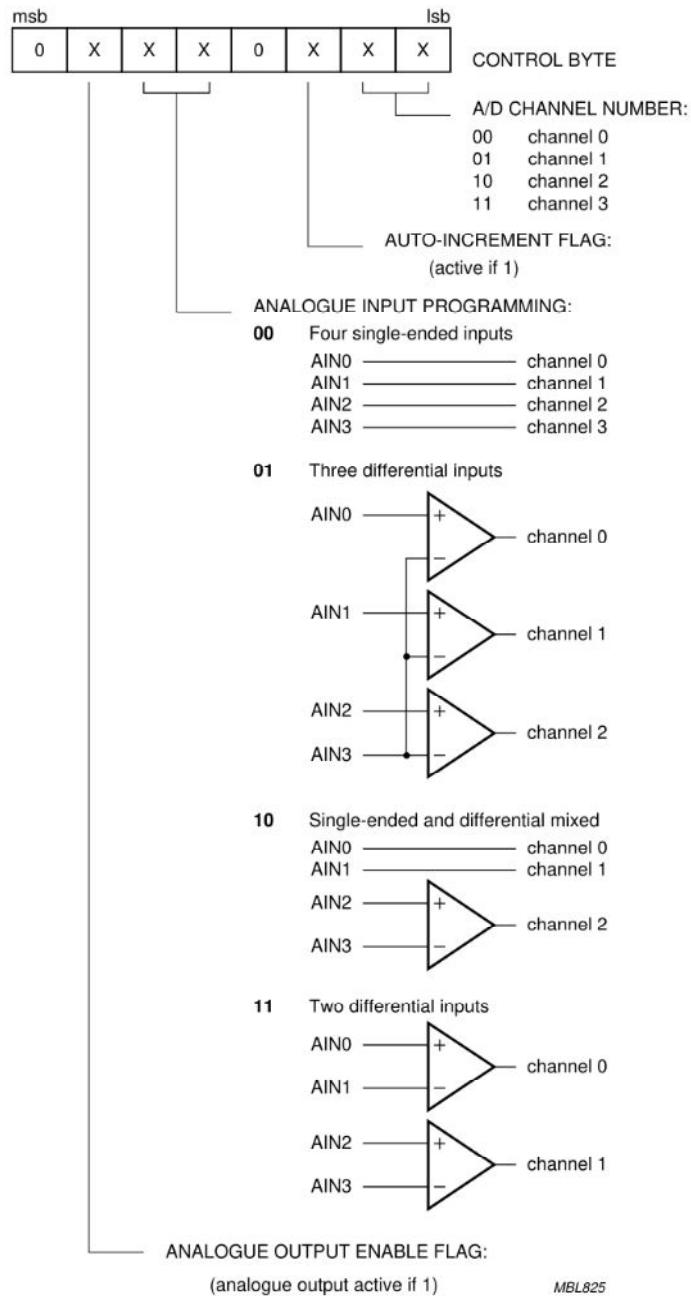
The PCF8591 is a single-chip, single-supply low-power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. The functions of the device include analog input multiplexing, on-chip track and hold function, 8-bit **analog-to-digital conversion** and an 8-bit **digital-to-analog conversion**.



Each PCF8591 device in an I2C-bus system is activated by sending a valid address to the device. The address consists of a fixed part and a programmable part. The programmable part must be set according to the address pins A0, A1 and A2. The address always has to be sent as the first byte after the start condition in the I2C-bus protocol. The last bit of the address byte is the read/write-bit which sets the direction of the following data transfer (see as below).



The second byte sent to a PCF8591 device will be stored in its control register and is required to control the device function. The upper nibble of the control register is used for enabling the analog output, and for programming the analog inputs as single-ended or differential inputs. The lower nibble selects one of the analog input channels defined by the upper nibble. If the auto-increment flag is set, the channel number is incremented automatically after each A/D conversion. See the figure below.



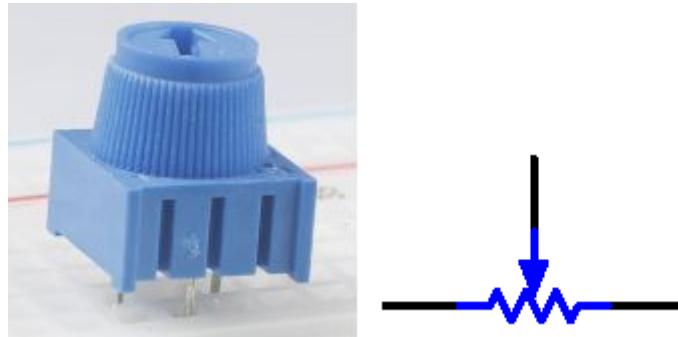
**Note:** In this experiment, the AIN0 (Analog Input 0) port is used to receive analog signals from the potentiometer, and AOUT (Analog Output) is used to output analog signals to the LED so as to change the luminance of the LED.

## ADC

Analog-to-Digital converters (ADC) translate analog signals, real world signals like temperature, pressure, voltage, current, distance, or light intensity, into a digital representation of that signal. This digital representation can then be processed, manipulated, computed, transmitted or stored.

## Potentiometer

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



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

### 1. Serving as a voltage divider

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

### 2. Serving as a rheostat

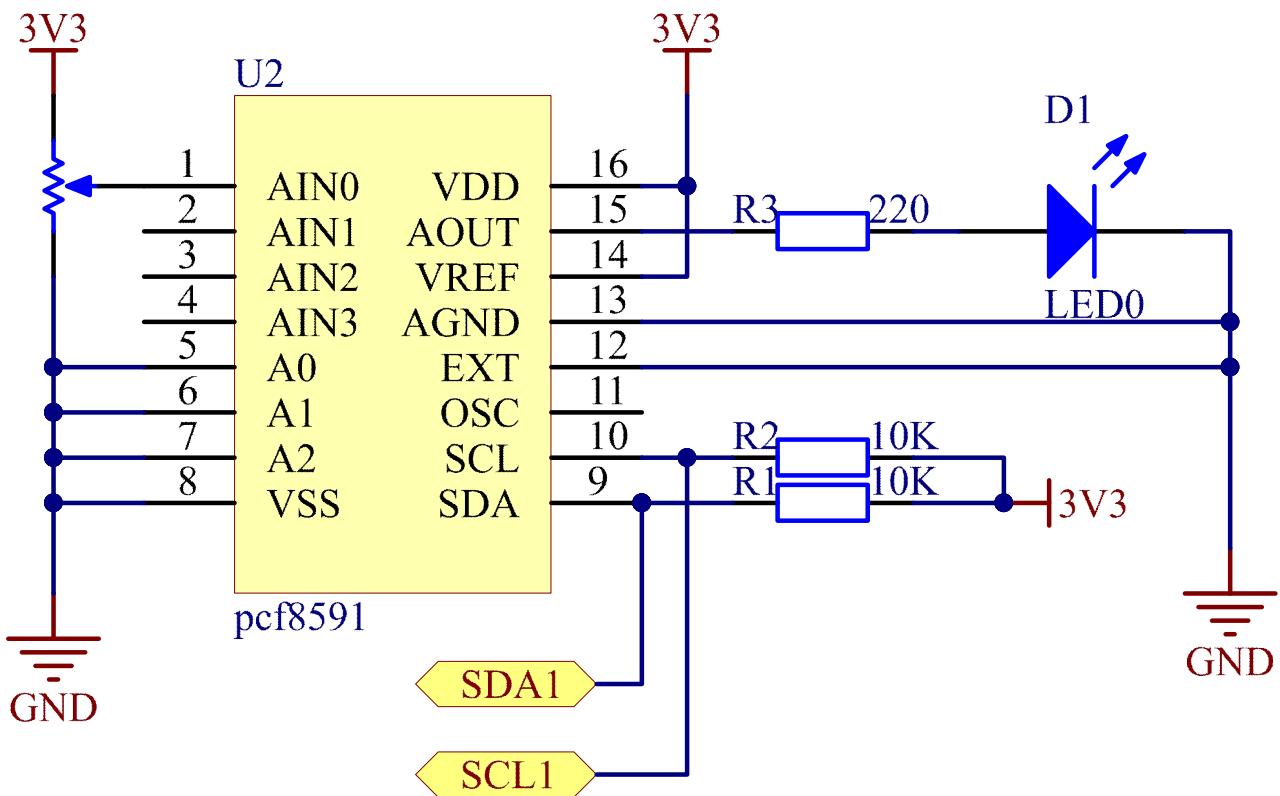
When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value cused by moving contact.

### 3. Serving as a current controller

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

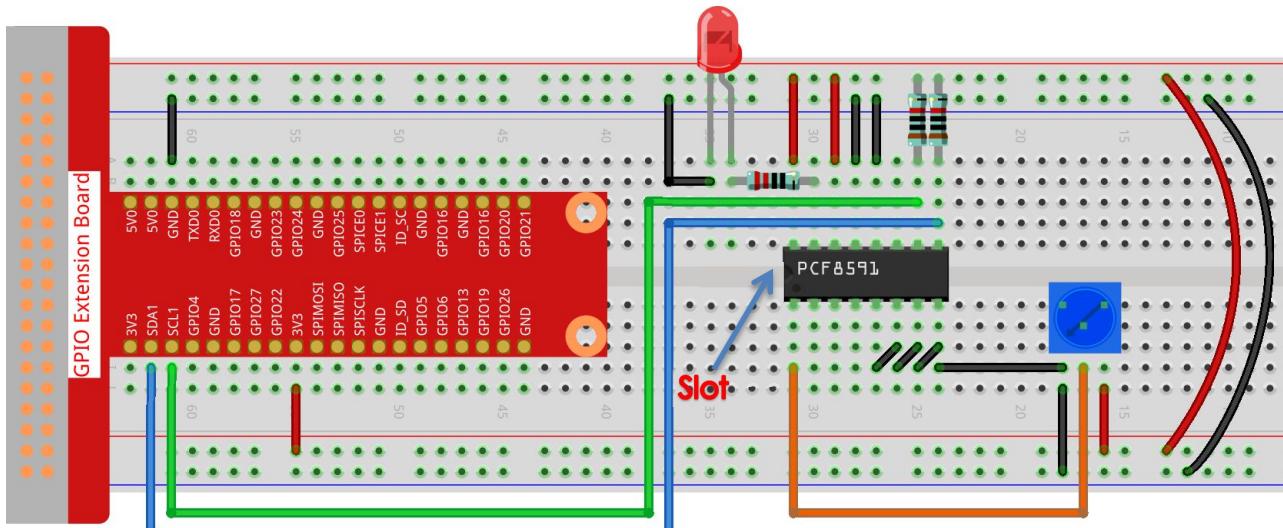
## Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



## Experimental Procedures

**Step 1:** Build the circuit.



Note: Please place the chip by referring to the corresponding position depicted in the picture. Note that the grooves on the chip should be on the left when it is placed.

**Step 2:** Setup I2C (see Appendix. If you have set I2C, skip this step.)

## ➤ For C Language Users

**Step 3:** Open the code file.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.4/
```

**Step 4:** Compile the code.

```
gcc 2.1.4_Potentiometer.c -lwiringPi
```

**Step 5:** Run.

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

After the code runs, rotate the knob on the potentiometer , the intensity of LED will change accordingly.

## Code

```
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>

#define PCF    120

int main (void)
{
    int value ;
    wiringPiSetup () ;
    // Setup pcf8591 on base pin 120, and address 0x48
    pcf8591Setup (PCF, 0x48) ;
    while(1) // loop forever
    {
        value = analogRead (PCF + 0) ;
        printf("Value: %d\n", value);
        analogWrite (PCF + 0, value) ;
        delay (200) ;
    }
    return 0 ;
}
```

## Code Explanation

```
#include <pcf8591.h>
```

Extend wiringPi with the PCF8591 I2C GPIO Analog expander chip. The chip has 1 8-bit DAC and 4 x 8-bit ADCs

```
#define PCF 120
```

PCF is the basic pin of the device when you include it in the wiringPi device system. wiringPi is a pin based GPIO system and pin 0 ~ 63 are considered to be on the Pi, so you should pick a pin number when you add new devices into it. You can use any number you like from 64 to  $2^{31}-1$ . In this case, we picked 120. Then you have PCF + 0 which is the first channel, PCF + 1, the second channel, and so on.

```
pcf8591Setup (PCF, 0x48) ;
```

The 0x48 is the I2C device address of the PCF8591. Each I2C device has its own address - some can be changed some are fixed. The PCF8591 can have 8 different I2C addresses, but the default is 0x48. To check the I2C address, see Appendix-I2C Configuration.

```
value = analogRead (PCF + 0) ;
```

PCF+0 refers to the first input channel of PCF8591. Here we get a potentiometer connected. Then, read the value of potentiometer, and assign it to the variable, value.

```
analogWrite (PCF + 0, value) ;
```

Here, PCF+0 refers to the first output channel of PCF8591 which is connected to an LED. Then, assign the variable, value into PCF+0 channel.

### ➤ For Python Users

**Step 3:** Install a smbus module.

```
sudo apt-get install python-smbus
```

**Step 4:** Open the code file

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

**Step 5:** Run.

```
sudo python 2.1.4_Potentiometer.py
```

After the code runs, rotate the knob on the potentiometer , the intensity of LED will change accordingly.

## Code

```
import PCF8591 as ADC
import time
import RPi.GPIO as GPIO

def setup():
    ADC.setup(0x48)

def loop():
    status = 1
    while True:
        print ('Value:', ADC.read(0))
        Value = ADC.read(0)
        outvalue = map(Value,0,255,120,255)
        ADC.write(outvalue)
        time.sleep(0.2)
def destroy():
    ADC.write(0)

def map(x, in_min, in_max, out_min, out_max):
    '''To map the value from arange to another'''
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

if __name__ == '__main__':
    try:
        setup()
        loop()
    except KeyboardInterrupt:
        destroy()
```

## Code Explanation

```
import PCF8591 as ADC
```

import PCF8591 library, then create a variable, ADC to replace PCF8591 in the following code.

```
ADC.setup(0x48)
```

The default I2C address of PCF8591 is 0x48. To check the I2C address, see Appendix-I2C Configuration.

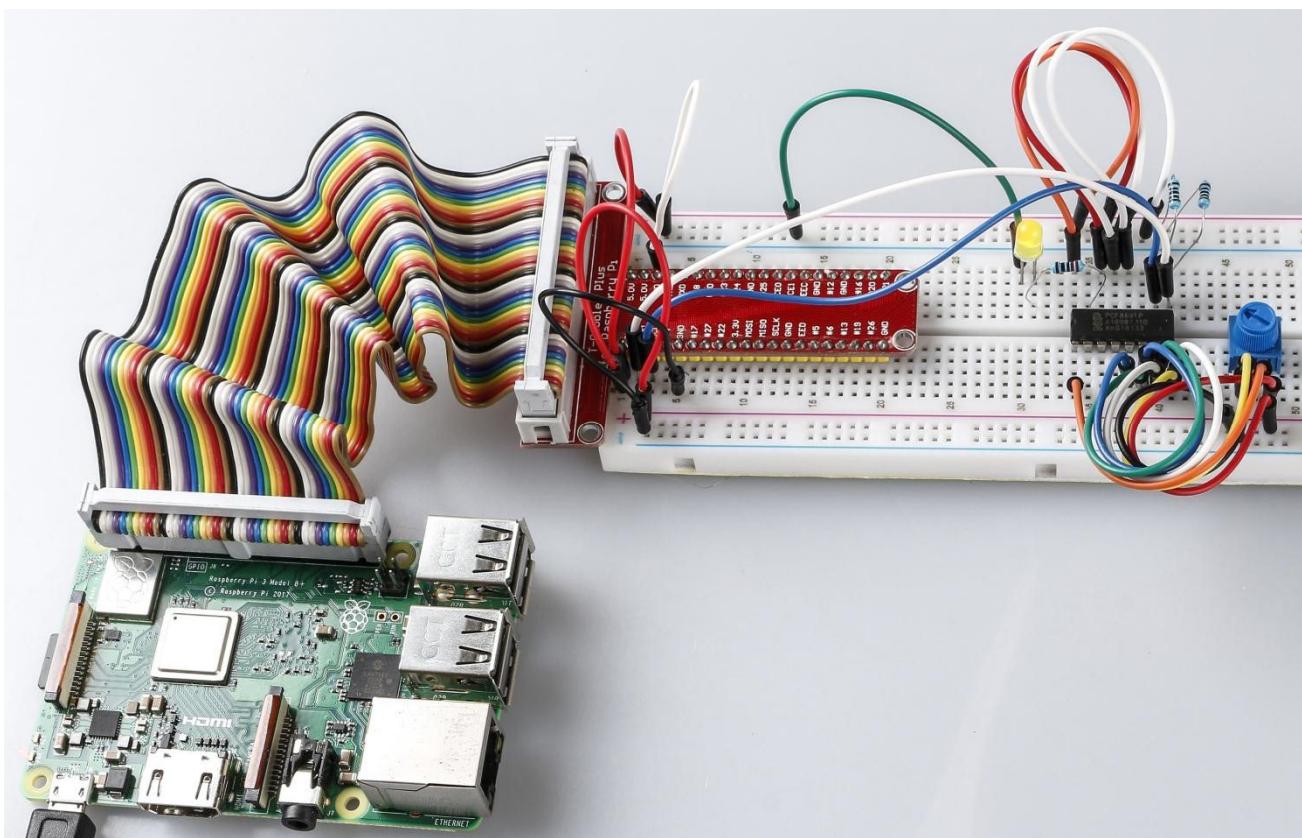
**while True:**

```
print ("Value:", ADC.read(0))
Value = ADC.read(0)
outvalue = map(Value,0,255,120,255)
ADC.write(outvalue)
time.sleep(0.2)
```

Read the value of AIN0(potentiometer), then store it in the variable, Value. After that, call a map function to map Value, 0-255 to 120-255. Finally, write the value into LED with write function.

You will see the intensity of LED changing with the rotation of the knob on the potentiometer.

## Phenomenon Picture



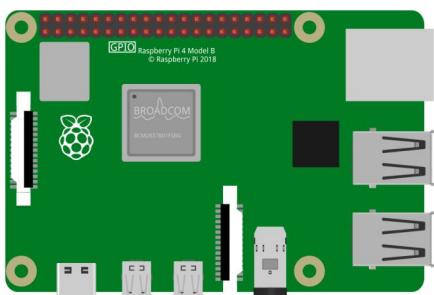
## 2.1.5 Keypad

### Introduction

A keypad is a rectangular array of buttons. In this project, We will use it input characters.

### Components

1 \* Raspberry Pi



1 \* T-Extension Board

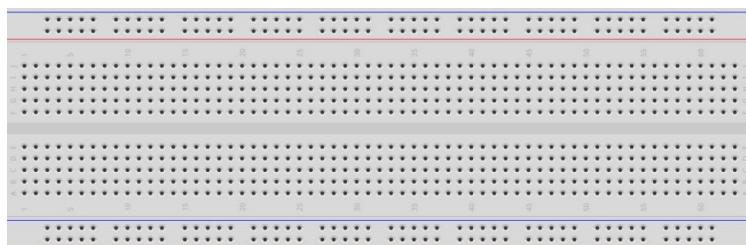


1 \* Keypad



Several Jumper Wires

1 \* Breadboard



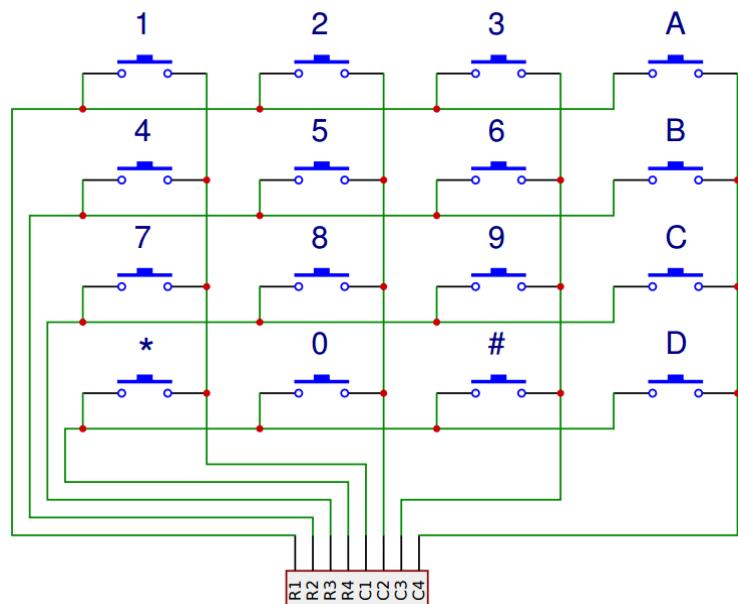
1 \* 40-pin Cable



### Principle

#### Keypad

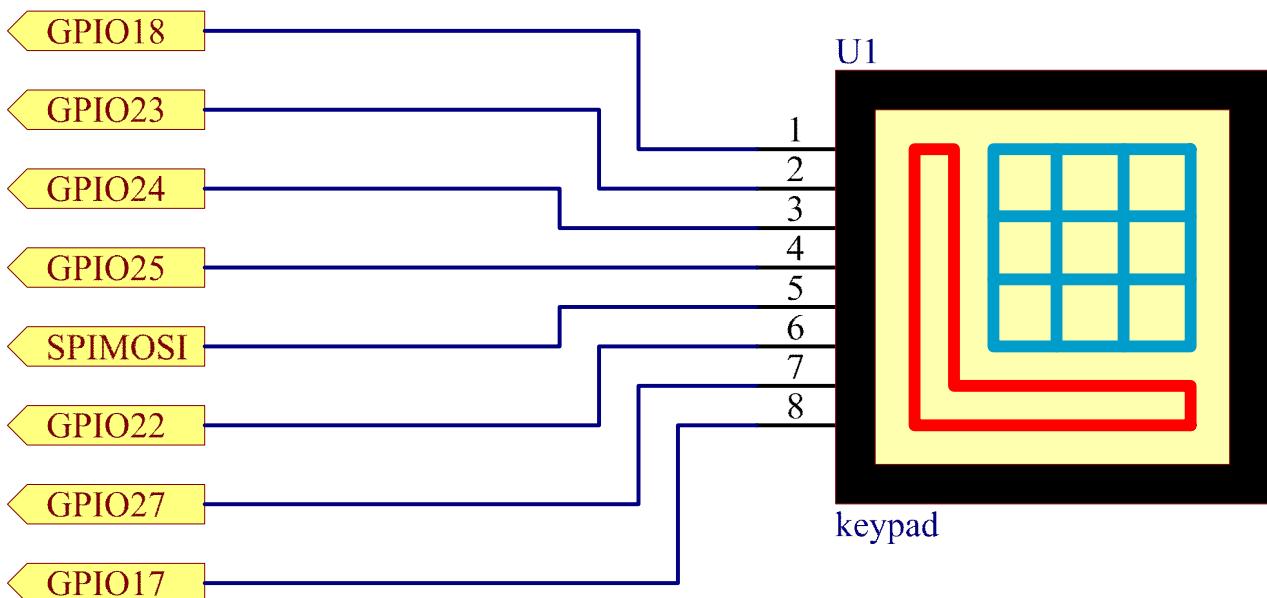
A keypad is a rectangular array of 12 or 16 OFF-(ON) buttons. Their contacts are accessed via a header suitable for connection with a ribbon cable or insertion into a printed circuit board. In some keypads, each button connects with a separate contact in the header, while all the buttons share a common ground.



More often, the buttons are matrix encoded, meaning that each of them bridges a unique pair of conductors in a matrix. This configuration is suitable for polling by a microcontroller, which can be programmed to send an output pulse to each of the four horizontal wires in turn. During each pulse, it checks the remaining four vertical wires in sequence, to determine which one, if any, is carrying a signal. Pullup or pulldown resistors should be added to the input wires to prevent the inputs of the microcontroller from behaving unpredictably when no signal is present.

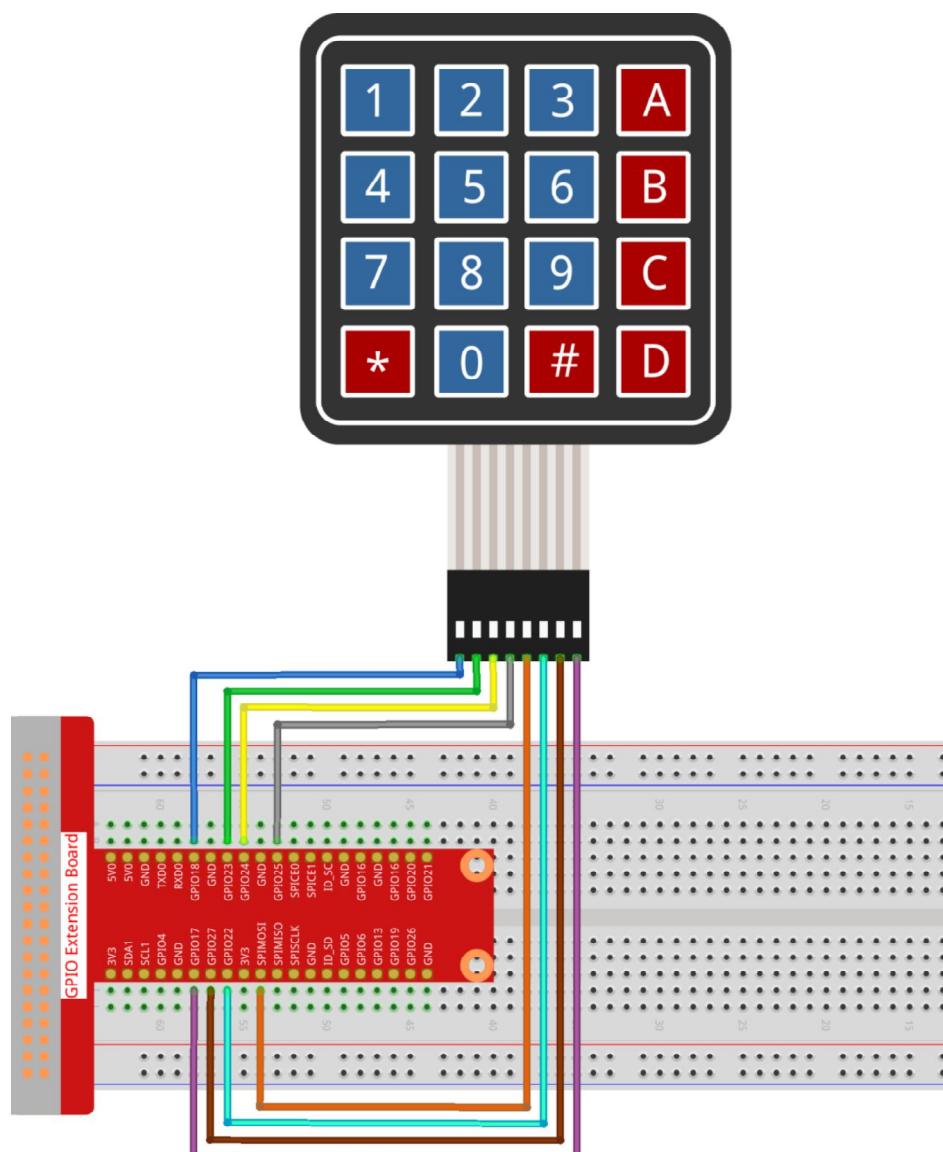
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17



## Experimental Procedures

## **Step 1:** Build the circuit.



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.5/
```

**Step 3:** Compile the code.

```
gcc 2.1.5_Keypad.cpp Key.cpp Keypad.cpp -lwiringPi
```

**Note:** The program contains custom headers that are compiled when CPP files are compiled.

You can use this method to simplify the instruction.

```
gcc *.cpp -lwiringPi
```

**Note:** Here we use the wildcard (\*), which causes all the files that conform to the format to be processed together.

**Step 4:** Run.

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

After the code runs, the values of pressed buttons on keypad (button Value) will be printed on the screen.

## Code

```
#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = { //key code
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
 {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
//create Keypad object
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

int main(){
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
```

```

printf("setup wiringPi failed !");
return 1;
}
char key = 0;
keypad.setDebounceTime(50);
while(1){
    key = keypad.getKey(); //get the state of keys
    if (key){ //if a key is pressed, print out its key code
        printf("You Pressed key : %c \n",key);
    }
}
return 1;
}

```

## Code Explanation

```
#include "Keypad.hpp"
```

These two files Keypad.hpp and key.hpp that is included from Keypad are in the same directory, and they come from Arduino which helps to use the Keypad easily.

```

char keys[ROWS][COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};
```

Declares a two-dimensional array for each key on the Keypad.

```
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

Read the values of buttons on keypad, pins and the number of rows and columns, then store them into the variable, keypad.

```
keypad.setDebounceTime(50);
```

This function is used to adjust the sensitivity of the keypad. 50 means that debounce time is 50ms.

```

key = keypad.getKey(); //get the state of keys
if (key){ //if a key is pressed, print out its key code
```

```
    printf("You Pressed key : %c \n",key);
}
```

Read and then store the values of pressed buttons into variable, key. If there is a button pressed, then the button value will be printed on the screen.

**Note:** The Keypad file is ported from the Arduino Keypad library and can be accessed for source code: <http://playground.arduino.cc/Code/Keypad>

## ➤ For Python Language Users

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

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

**Step 3:** Run.

```
sudo python 2.1.5_Keypad.py
```

After the code runs, the values of pressed buttons on keypad (button Value) will be printed on the screen.

## Code

```
import RPi.GPIO as GPIO
import time

class Key(object):
    .....
class Keypad(object):
    .....

ROWS = 4
COLS = 4
keys = [ '1','2','3','A',
         '4','5','6','B',
         '7','8','9','C',
         '*', '0', '#','D'   ]
rowsPins = [12,16,18,22]
colsPins = [19,15,13,11]
def loop():
    keypad = Keypad(keys,rowsPins,colsPins,ROWS,COLS)
    keypad.setDebounceTime(50)
    while(True):
```

```

key = keypad.getKey()
if(key != keypad.NULL):
    print ("You Pressed Key : %c %(key) )

if __name__ == '__main__':
    # Program start from here
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    pass
GPIO.cleanup()

```

## Code Explanation

```

class Key(object):
class Keypad(object):

```

These two classes come from the Arduino library. This makes it easier to use the Keypad.

```

keys = [ '1','2','3','A',
        '4','5','6','B',
        '7','8','9','C',
        '*', '0', '#', 'D' ]

```

Declares a two-dimensional array for each key on the Keypad.

```
keypad = Keypad(keys,rowsPins,colsPins,ROWS,COLS)
```

Read the values of buttons on keypad, pins and the number of rows and columns, then store them into the variable, keypad.

```
keypad.setDebounceTime(50)
```

This function is used to adjust the sensitivity of the keypad. 50 means that Debounce Time is 50ms.

```

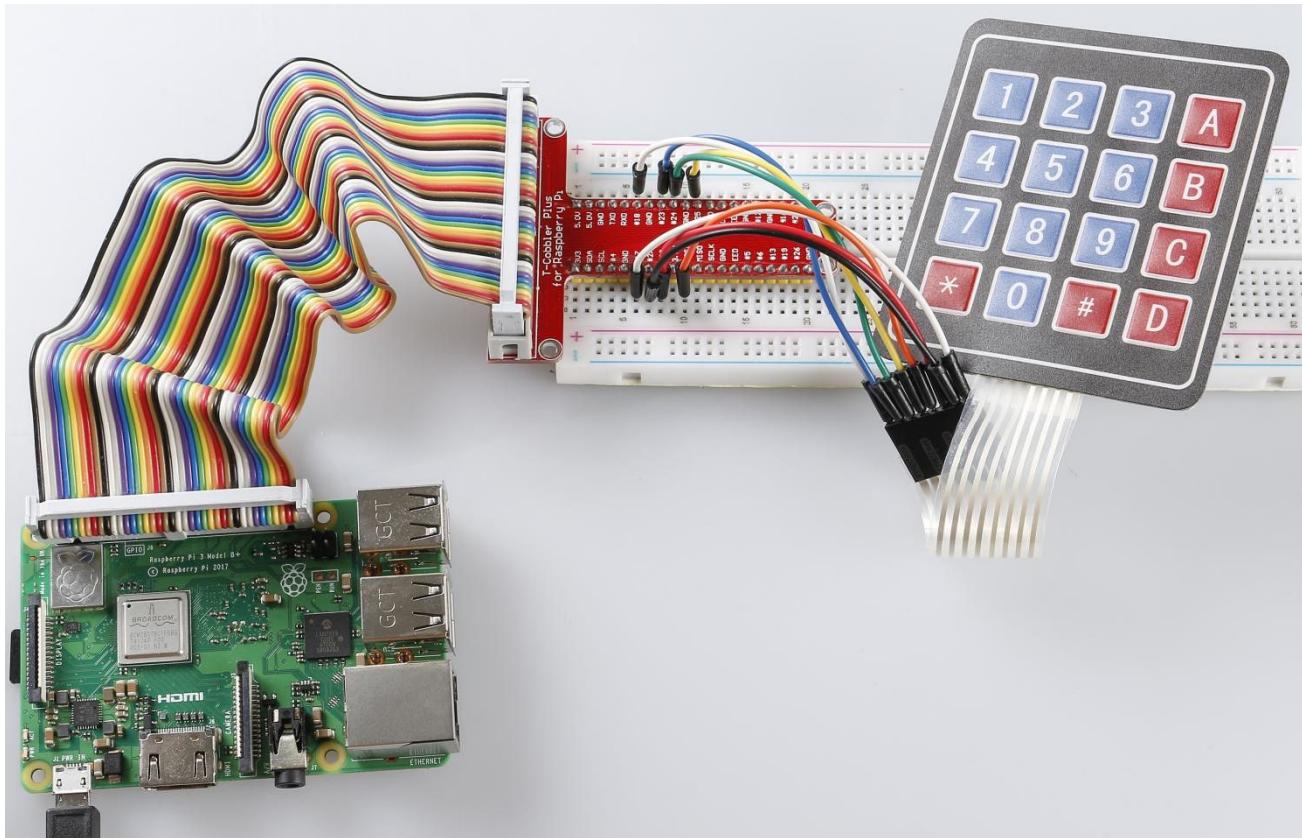
while(True):
    key = keypad.getKey()
    if(key != keypad.NULL):
        print ("You Pressed Key : %c %(key) )

```

Read and then store the values of pressed buttons into variable, key. If there is a button pressed, then the button value will be printed on the screen.

**Note:** Class Keypad and Class Key are ported from the Arduino Keypad library, which you can access the source code: <http://playground.arduino.cc/Code/Keypad>.

## Phenomenon Picture



## 2.1.6 Joystick

### Introduction

In this project, We're going to learn how joystick works. We manipulate the Joystick and display the results on the screen.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Joystick																																								
	 <table border="1"> <tr><td>3V3</td><td>5V0</td></tr> <tr><td>SDA1</td><td>5V0</td></tr> <tr><td>SCL1</td><td>GND</td></tr> <tr><td>GPIO4</td><td>TX00</td></tr> <tr><td>GND</td><td>RX00</td></tr> <tr><td>GPIO17</td><td>GPIO18</td></tr> <tr><td>GPIO27</td><td>GND</td></tr> <tr><td>GPIO22</td><td>GPIO23</td></tr> <tr><td>3V3</td><td>GPIO24</td></tr> <tr><td>SPIMOSI</td><td>GND</td></tr> <tr><td>SPIMISO</td><td>GPIO25</td></tr> <tr><td>SPISCLK</td><td>SPICE0</td></tr> <tr><td>GND</td><td>SPICE1</td></tr> <tr><td>ID_SD</td><td>ID_SC</td></tr> <tr><td>GPIO5</td><td>GND</td></tr> <tr><td>GPIO6</td><td>GPIO16</td></tr> <tr><td>GPIO13</td><td>GND</td></tr> <tr><td>GPIO19</td><td>GPIO16</td></tr> <tr><td>GPIO26</td><td>GPIO20</td></tr> <tr><td>GND</td><td>GPIO21</td></tr> </table>	3V3	5V0	SDA1	5V0	SCL1	GND	GPIO4	TX00	GND	RX00	GPIO17	GPIO18	GPIO27	GND	GPIO22	GPIO23	3V3	GPIO24	SPIMOSI	GND	SPIMISO	GPIO25	SPISCLK	SPICE0	GND	SPICE1	ID_SD	ID_SC	GPIO5	GND	GPIO6	GPIO16	GPIO13	GND	GPIO19	GPIO16	GPIO26	GPIO20	GND	GPIO21	
3V3	5V0																																									
SDA1	5V0																																									
SCL1	GND																																									
GPIO4	TX00																																									
GND	RX00																																									
GPIO17	GPIO18																																									
GPIO27	GND																																									
GPIO22	GPIO23																																									
3V3	GPIO24																																									
SPIMOSI	GND																																									
SPIMISO	GPIO25																																									
SPISCLK	SPICE0																																									
GND	SPICE1																																									
ID_SD	ID_SC																																									
GPIO5	GND																																									
GPIO6	GPIO16																																									
GPIO13	GND																																									
GPIO19	GPIO16																																									
GPIO26	GPIO20																																									
GND	GPIO21																																									
1 * 40-pin Cable		1 * PCF8591																																								
1 * Breadboard		Several Jumper Wires																																								
		2 * Resistor(10kΩ)																																								

### Principle

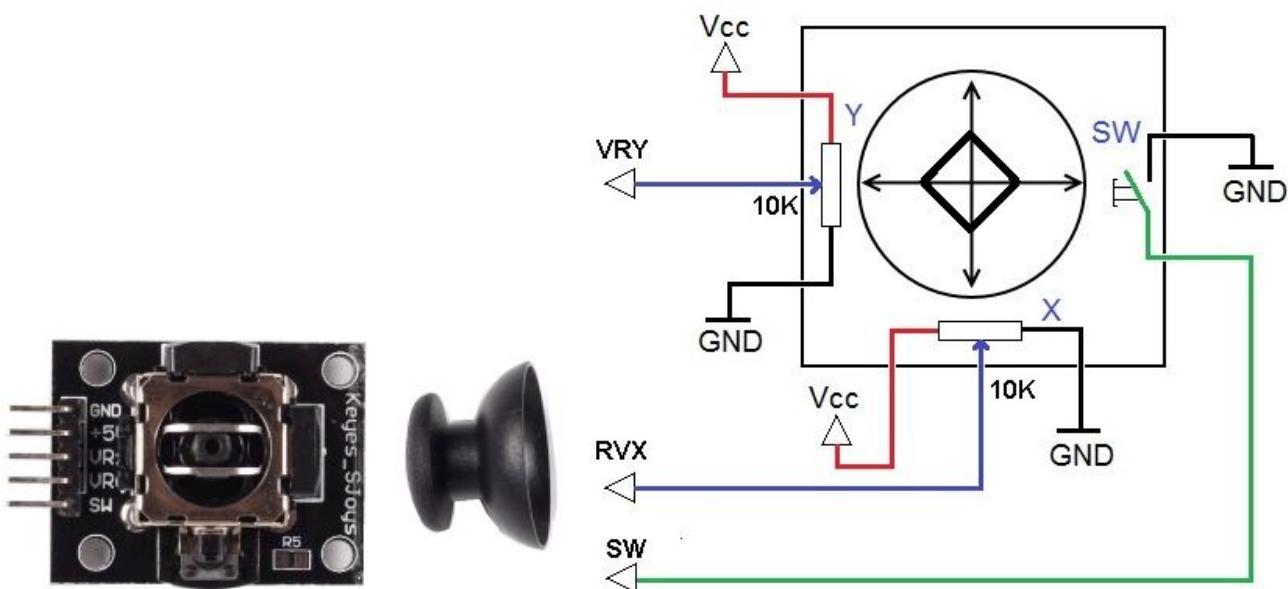
#### Joystick

The basic idea of a joystick is to translate the movement of a stick into electronic information that a computer can process.

In order to communicate a full range of motion to the computer, a joystick needs to measure the stick's position on two axes -- the X-axis (left to right) and the Y-axis (up and down). Just as in basic geometry, the X-Y coordinates pinpoint the stick's position exactly.

To determine the location of the stick, the joystick control system simply monitors the position of each shaft. The conventional analog joystick design does this with two potentiometers, or variable resistors.

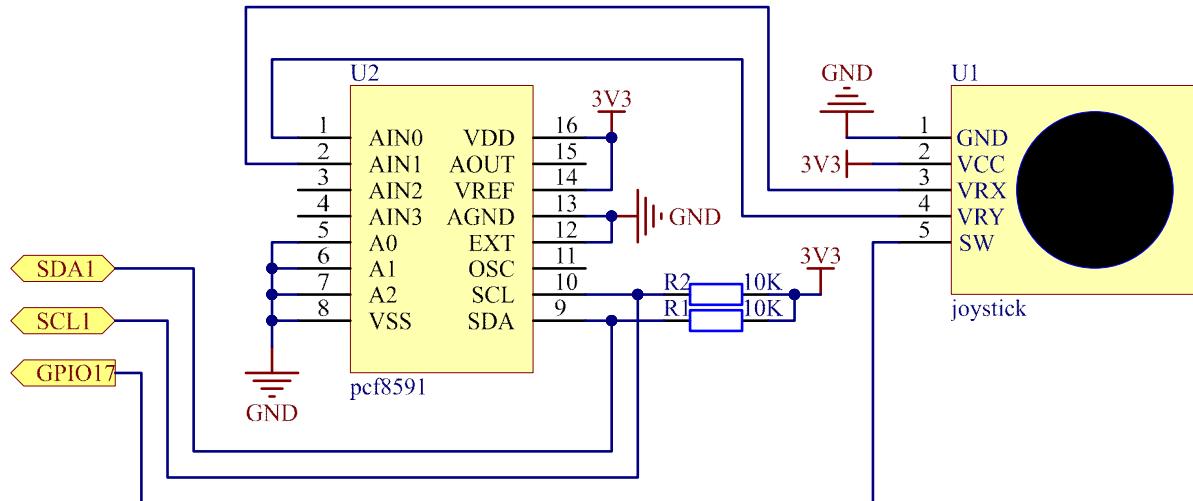
The joystick also has a digital input that is actuated when the joystick is pressed down.



## Schematic Diagram

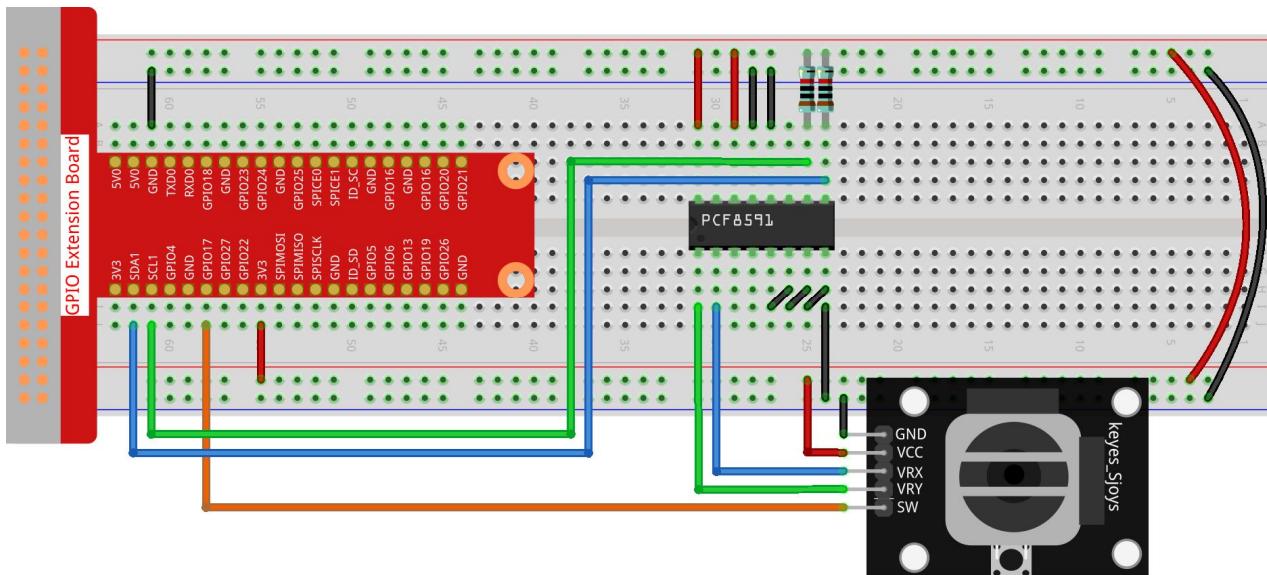
When the data of joystick is read, there are some differences between axis: data of X and Y axis is analog, which need to use PCF8591 to convert the analog value to digital value. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

T-Board Name	physical	wiringPi	BCM
SDA1	Pin 3		
SCL1	Pin 5		
GPIO17	Pin 11	0	17



## Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Setup I2C (see Appendix. If you have set I2C, skip this step.)

➤ **For C Language Users**

**Step 3:** Go to the folder of the code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.6/
```

**Step 4:** Compile the code.

```
gcc 2.1.6_Joystick.c -lwiringPi
```

**Step 5:** Run the executable file.

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

After the code runs, turn the Joystick, then the corresponding values of x, y, Btn are displayed on screen.

## Code

```
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>

#define PCF    120
#define uchar unsigned char

int AIN0 = PCF + 0;
int AIN1 = PCF + 1;

int direction(){
    int x, y, b;
    int tmp;
    x = analogRead(AIN1);
    y = analogRead(AIN0);
    b = digitalRead(0);
    //printf("%d , %d , %d \n",x,y,b);
    if (y <= 5)
        {tmp = 1;      // up
     }
    if (y >= 250)
        {tmp = 2;      // down
     }
    if (x <= 5)
        {tmp = 3;      // left
     }
    if (x >= 250)
        {tmp = 4;      // right
     }
    if (b == 0)
        {tmp = 5;      // button presesd
     }
    if (x-125<60 && x-125>-60 && y-125<60 && y-125>-60 && b == 1)
        {tmp = 0;      // home position
     }
    return tmp;
}

int main (void)
```

```
{  
    int tmp;  
    int status = 0;  
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen  
        printf("setup wiringPi failed !");  
        return 1;  
    }  
    pinMode(0,INPUT);  
    pullUpDnControl(0,PUD_UP);  
  
    // Setup pcf8591 on base pin 120, and address 0x48  
    pcf8591Setup (PCF, 0x48);  
    while(1) // loop forever  
    {  
        tmp = direction();  
        if (tmp != status)  
        {  
            switch (tmp)  
            {  
                case 1:printf("up \n");break;  
                case 2:printf("down \n");break;  
                case 3:printf("left \n");break;  
                case 4:printf("right \n");break;  
                case 5:printf("presssd \n");break;  
                case 0:printf("home \n");break;  
            }  
            status = tmp;  
        }  
    }  
    return 0;  
}
```

## Code Explanation

```
#define PCF    120
#define PCF    120
#define uchar  unsigned char

int AIN0 = PCF + 0;
int AIN1 = PCF + 1;
```

PCF is the basic pin of the device when you include it in the wiringPi device system. wiringPi is a pin based GPIO system and pin 0 ~ 63 are considered to be on the Pi, so you should pick a pin number when you add new devices into it. You can use any number you like from 64 to  $2^{31}-1$ . In this case, we picked 120. Then you have PCF + 0 which is the first channel, PCF + 1, the second channel, and so on.

```
int direction(){
    int x, y, b;
    int tmp;
    x = analogRead(AIN1);
    y = analogRead(AIN0);
    b = digitalRead(0);
    //printf("%d , %d , %d \n",x,y,b);
    if (y <= 5)
        {tmp = 1;      // up
     }
    if (y >= 250)
        {tmp = 2;      // down
    ...
}
```

This function is used to read the status of the joystick. When the joystick is moved forward and backward, the AIN0 reading of PCF8591 will change between 0 and 255. When the joystick is moved left and right, AIN1 will be changed. Pin 0 is used to read whether the joystick is pressed down. Then 5 if statements are used to judge the status of joystick at this moment which can be up, down or left.

```
switch (tmp)
{
    case 1:printf("up \n");break;
    case 2:printf("down \n");break;
    case 3:printf("left \n");break;
    case 4:printf("right \n");break;
    case 5:printf("presssd \n");break;
```

```
case 0:printf("home \n");break;
}
```

The switch statement prints the corresponding result based on the value returned by direction().

```
if (tmp != status)
{
    .....
    status = tmp;
}
```

The assignment and judgement of the variable, status ensures that printing is only performed when the state of the joystick changes.

## ➤ For Python Language Users

**Step 3:** Go to the folder of the code.

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

**Step 4:** Run.

```
sudo python 2.1.6_Joystick.py
```

After the code runs, turn the Joystick, then the corresponding values of x, y, Btn are displayed on screen.

## Code

```
import PCF8591 as ADC
import time
import RPi.GPIO as GPIO

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC.setup(0x48)          # Setup PCF8591
    global state

def direction(): #get joystick result
    state = ['home', 'up', 'down', 'left', 'right', 'pressed']
    i = 0

    if ADC.read(0) <= 5:
```

```

i = 1      #up
if ADC.read(0) >= 250:
    i = 2      #down
if ADC.read(1) >= 250:
    i = 3      #left
if ADC.read(1) <= 5:
    i = 4      #right
if GPIO.input(17) == 0:
    i = 5      # Button pressed
if ADC.read(0) - 125 < 15 and ADC.read(0) - 125 > -15 and ADC.read(1) - 125 < 15 and
ADC.read(1) - 125 > -15 and GPIO.input(17) == 1:
    i = 0
#print("%d %d %d \n")
return state[i]

def loop():
    status = ""
    while True:
        tmp = direction()
        if tmp != None and tmp != status:
            print (tmp)
            status = tmp
def destroy():
    pass

if __name__ == '__main__':    # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    destroy()

```

## Code Explanation

```

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC.setup(0x48)          # Setup PCF8591
    global state

```

Define the naming way of the pins of Raspberry Pi as BCM, the status of GPIO17 as input status, the mode of it as PUD\_UP. Then initialize PCF8591, and set the default address to 0x48.

```
def direction(): #get joystick result
    state = ['home', 'up', 'down', 'left', 'right', 'pressed']
    i = 0

    if ADC.read(0) <= 5:
        i = 1      #up
    if ADC.read(0) >= 250:
        i = 2      #down
    if ADC.read(1) >= 250:
```

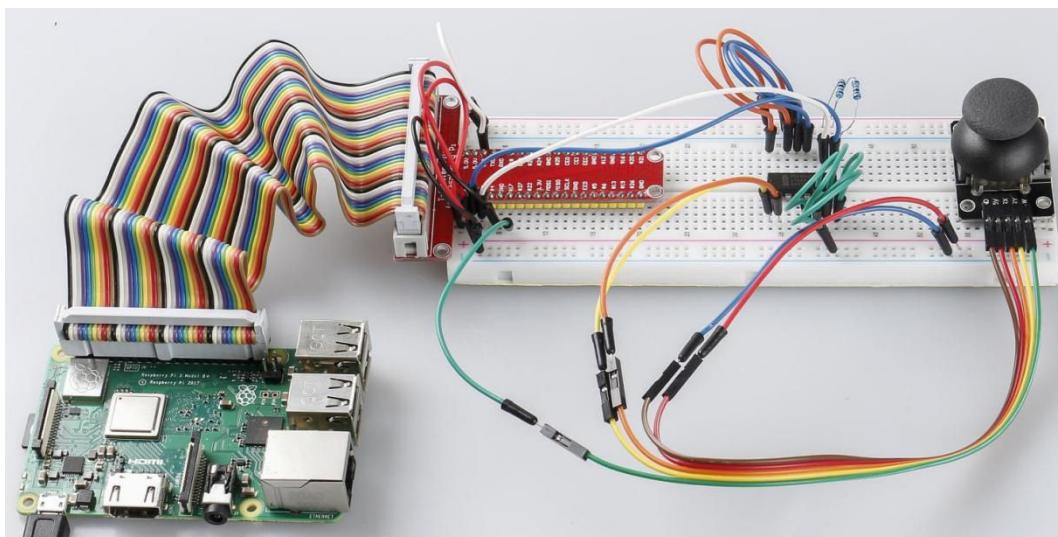
This function is used to read the state of PCF8591.

Set 6 statuses in the array, state[], when the read value of AIN0 (y axe) is less than or equal to 5, output the status of joystick that is 1 (up). The state of Joystick is 2 (down) when the read value of Y-axis is greater than or equal to 250, the rest can be done in the same manner.

```
while True:
    tmp = direction()
    if tmp != None and tmp != status:
        print (tmp)
    status = tmp
```

The assignment and determination of the variable, tmp ensure that printing is only performed when the status of the joystick changes.

## Phenomenon Picture



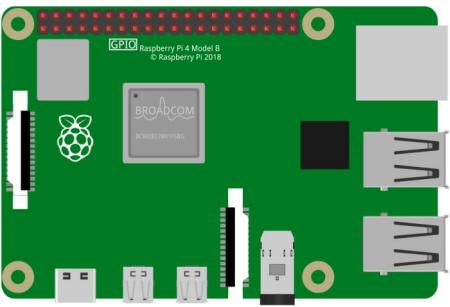
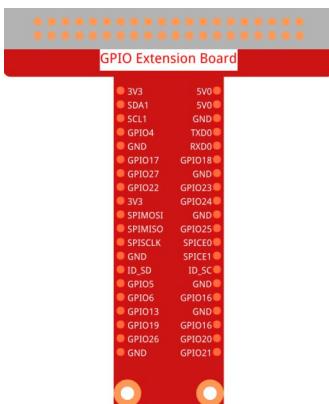
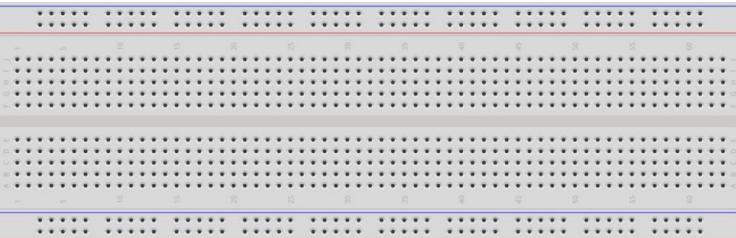
## 2.2 Sensors

### 2.2.1 Photoresistor

#### Introduction

Photoresistor is a commonly used component of ambient light intensity in life. It helps the controller to recognize day and night and realize light control functions such as night lamp. This project is very similar to potentiometer, and you might think it changing the voltage to sensing light.

#### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Photoresistor
	 GPIO Extension Board Pinout: 3V3 SDA1 SCL1 GND 5V0 GND TXD0 RXD0 GPIO17 GPIO18 GPIO27 GPIO22 GPIO23 3V3 GPIO24 GND SPI MOSI SPI MISO SPI SCLK GND ID_SD SPI CE1 GPIO5 GND GPIO6 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO26 GND GPIO20 GPIO21 GND	
1 * 40-pin Cable		1 * PCF8591
		
1 * Breadboard		1 * LED
		
		Several Jumper Wires
		
		1 * Resistor(220Ω)
		
		3 * Resistor 10KΩ
		

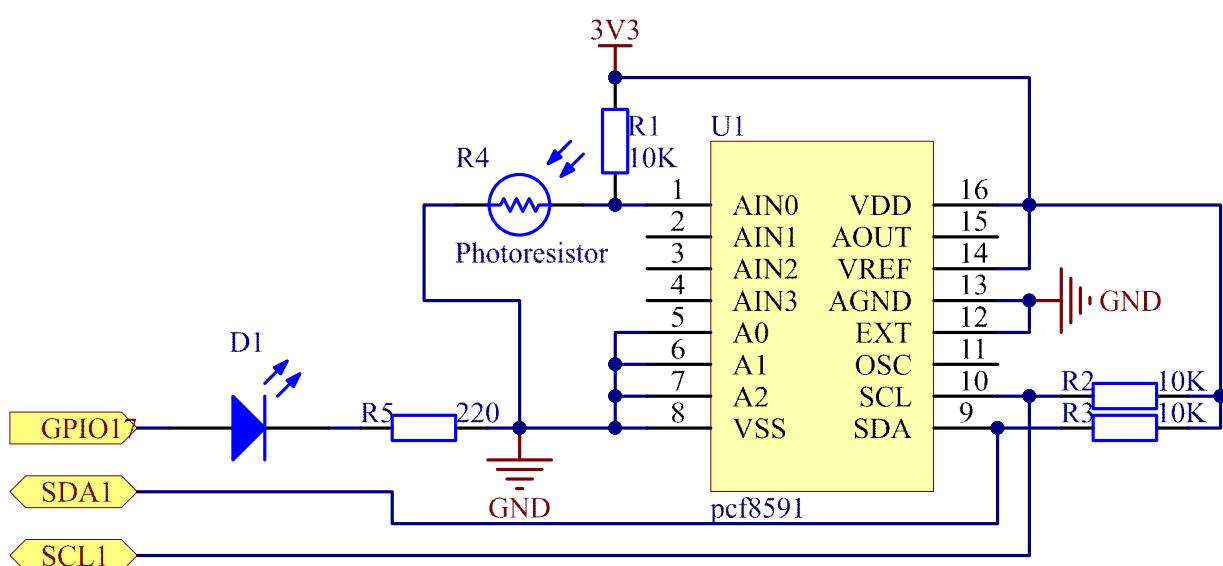
## Principle

A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity. A photoresistor can be applied in light-sensitive detector circuits, and light- and darkness-activated switching circuits.



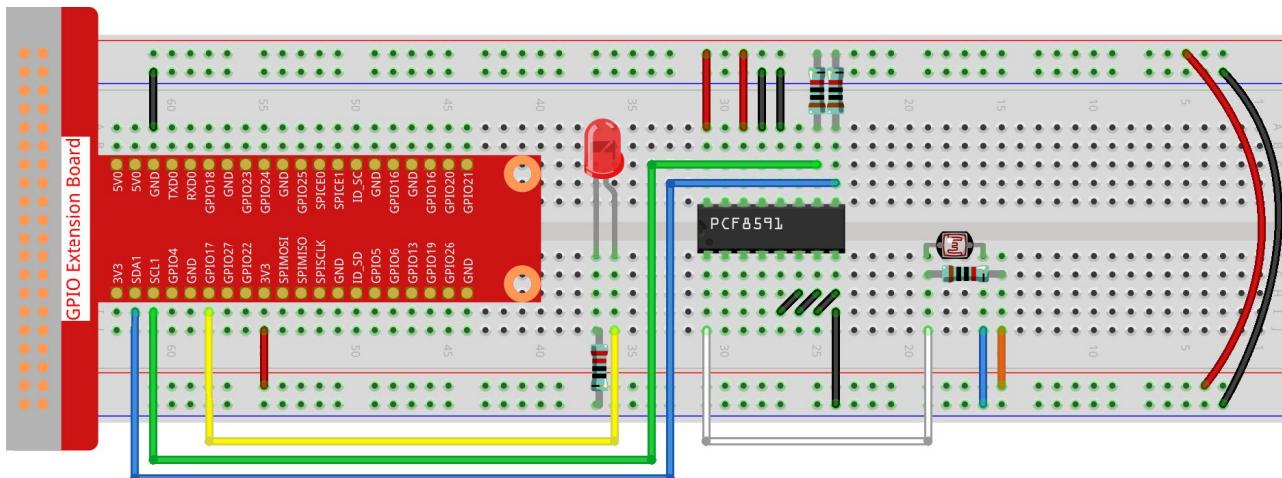
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
SDA1	Pin 3		SDA
SCL1	Pin 5		SCL
GPIO17	Pin 11	0	17



## Experimental Procedures

### Step 1: Build the circuit.



**Step 2:** Setup I2C (see Appendix. If you have set I2C, skip this step.)

### ➤ For C Language Users

**Step 3:** Go to the folder of the code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.1/
```

**Step 4:** Compile the code.

```
gcc 2.2.1_Photoresistor.c -lwiringPi
```

**Step 5:** Run the executable file.

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

The code run, the brightness of LED will vary depending on the intensity of light that the photoresistor senses.

### Code

```
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>
#include <math.h>
#include <softPwm.h>

#define PCF 120
#define ledPin 0
int main()
{
    int analogVal;
```

```

if(wiringPiSetup() == -1){
    printf("setup wiringPi failed !");
    return 1;
}
// Setup pcf8591 on base pin 120, and address 0x48
pcf8591Setup(PCF, 0x48);
softPwmCreate(ledPin,0,100);
while(1) // loop forever
{
    analogVal = analogRead(PCF + 0);
    softPwmWrite(ledPin,analogVal*100/255);
    printf("Value: %d\n", analogVal);

    delay (200);
}
return 0;
}

```

## Code Explanation

```

#define PCF 120
#define ledPin 0

```

Set the PCF to 120, you can also set it to  $64 \sim 2^{31}-1$  with LED connected to GPIO0.

```

pcf8591Setup(PCF, 0x48);
softPwmCreate(ledPin,0,100);

```

The address, 0x48 is the I2C device address of the PCF8591. Set the ledPin to a PWM pin whose initial pulse width is 0ms with a  $100 \times 100\mu s$  period.

```

while(1) // loop forever
{
    analogVal = analogRead(PCF + 0);
    softPwmWrite(ledPin,analogVal*100/255);
    printf("Value: %d\n", analogVal);

    delay (200);
}

```

Read the value, PCF+0 in the first input channel of PCF8591 and then assign it to the variable, analogVal. Since the variable analogVal ranges from 0 to 255 and the set

pwmRange of softPwmCreate is 100, the formula analogVal\*100/255 is used in order to map analogVal from 0-255 to 0-100; after that, get the value of analogVal and then write it to LED pins.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 2.2.1_Photoresistor.py
```

The code run, the brightness of LED will vary depending on the intensity of light that the photoresistor senses.

## Code

```
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time

ledPin = 17
GPIO.setmode(GPIO.BCM)

def setup():
    ADC.setup(0x48)
    GPIO.setup(ledPin, GPIO.OUT)
    GPIO.output(ledPin, GPIO.LOW)
    global p
    p = GPIO.PWM(ledPin, 1000)
    p.start(0)

def loop():
    while True:
        analogVal = ADC.read(0)
        p.ChangeDutyCycle(analogVal*100/255)
        print ('Value: ', analogVal)

        time.sleep(0.2)

if __name__ == '__main__':
    try:
```

```
setup()  
loop()  
except KeyboardInterrupt:  
    pass
```

## Code Explanation

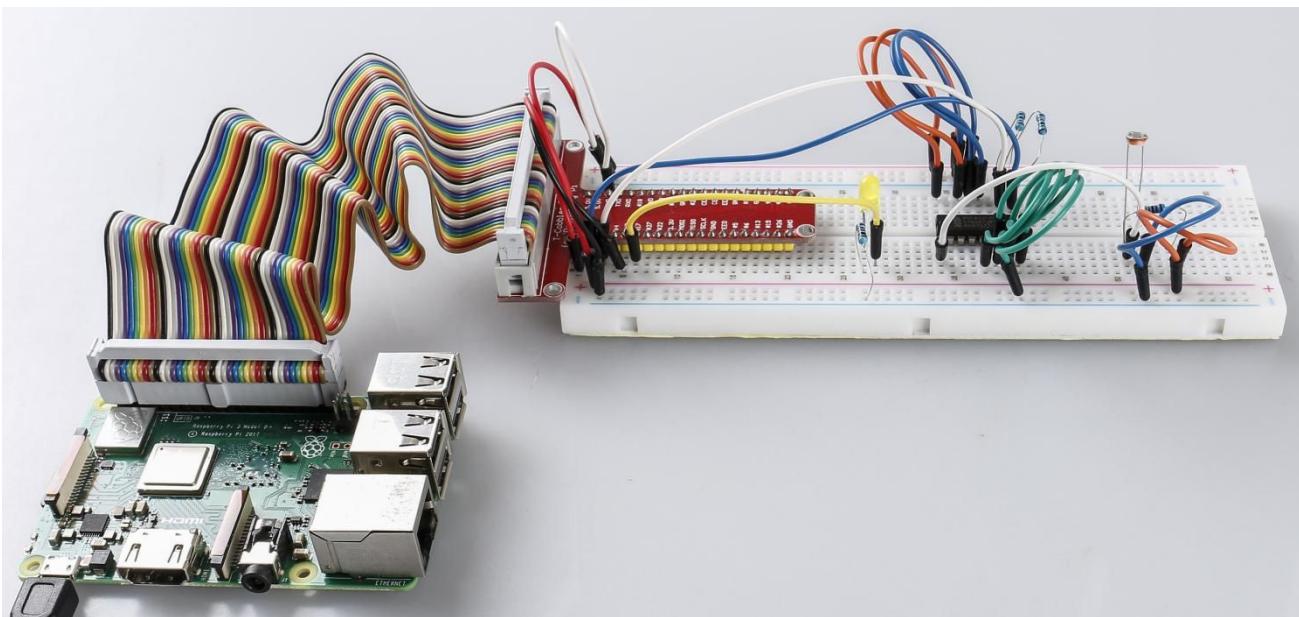
```
def setup():  
    ADC.setup(0x48)  
    GPIO.setup(ledPin, GPIO.OUT)  
    GPIO.output(ledPin,GPIO.LOW)  
    global p  
    p = GPIO.PWM(ledPin,1000)  
    p.start(0)
```

Initialize the I2C address of PCF8591 to 0x48. Set the ledPin to output state with a initial value LOW. Moreover, change the mode of ledPin into PWM, and set its frequency to 1000hz.

```
while True:  
    analogVal = ADC.read(0)  
    p.ChangeDutyCycle(analogVal*100/255)  
    print ('Value: ', analogVal)  
  
    time.sleep(0.2)
```

Read the value of photoresistor from AIN0 then assign it into variable, analogVal. Next, set the duty cycle of ledPin with the formula,  $\text{analogVal} * 100 / 255$ . Finally, call the function print() to print the read value.

## Phenomenon Picture

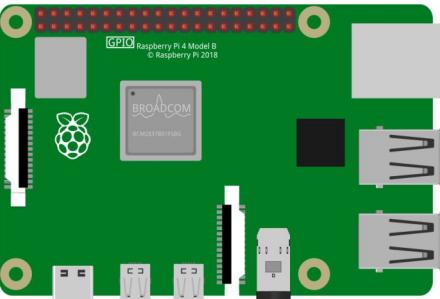
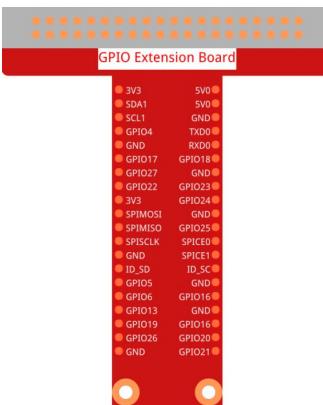
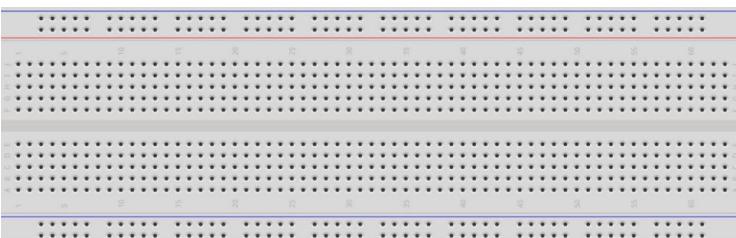


## 2.2.2 Thermistor

### Introduction

Just like photoresistor can sense light, thermistor is a temperature sensitive electronic device that can be used for realizing functions of temperature control, such as making a heat alarm.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Thermistor
	 GPIO Extension Board Pinout: <ul style="list-style-type: none"><li>● 3V3 5V0</li><li>● SDA1 GND0</li><li>● SCL1 GND0</li><li>● GPIO4 TXD0 0</li><li>● GND0 RXD0 0</li><li>● GPIO17 GPIO18</li><li>● GPIO27 GND0</li><li>● GPIO22 GPIO23</li><li>● 3V GND0</li><li>● SPI_MOSI GPIO25</li><li>● SPI_SCLK GPIO26</li><li>● GND0 SPI_CE1 0</li><li>● ID_SD ID_SD</li><li>● GPIO5 GND0</li><li>● GPIO6 GPIO16</li><li>● GPIO13 GND0</li><li>● GPIO19 GPIO16</li><li>● GPIO26 GPIO20</li><li>● GND0 GPIO21 0</li></ul>	
1 * 40-pin Cable		1 * PCF8591
		
1 * Breadboard	Several Jumper Wires	
	3 * Resistor 10KΩ	
		

## Principle

A thermistor is a thermally sensitive resistor that exhibits a precise and predictable change in resistance proportional to small changes in temperature. How much its resistance will change is dependent upon its unique composition. Thermistors are the parts of a larger group of passive components. And unlike their active component counterparts, passive devices are incapable of providing power gain, or amplification to a circuit.

Thermistor is a sensitive element, and it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also known as NTC and PTC. Its resistance varies significantly with temperature. The resistance of PTC thermistor increases with temperature ,while the condition of NTC is opposite to the former In this experiment we use NTC.



The principle is that the resistance of the NTC thermistor changes with the temperature of the outer environment. It detects the real-time temperature of the environment. When the temperature gets higher, the resistance of the thermistor decreases. Then the voltage data is converted to digital quantities by the A/D adapter. The temperature in Celsius or Fahrenheit is output via programming.

In this experiment, a thermistor and a 10k pull-up resistor are used. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature:

$$R_T = R_N \exp^{B(1/T_K - 1/T_N)}$$

$R_T$  is the resistance of the NTC thermistor when the temperature is  $T_K$ .

$R_N$  is the resistance of the NTC thermistor under the rated temperature  $T_N$ . Here, the numerical value of  $R_N$  is 10k.

$T_K$  is a Kelvin temperature and the unit is K. Here, the numerical value of  $T_K$  is 273.15 + degree Celsius.

$T_N$  is a rated Kelvin temperature; the unit is K too. Here, the numerical value of  $T_N$  is 273.15+25.

And **B**(beta), the material constant of NTC thermistor, is also called heat sensitivity index with a numerical value 3950.

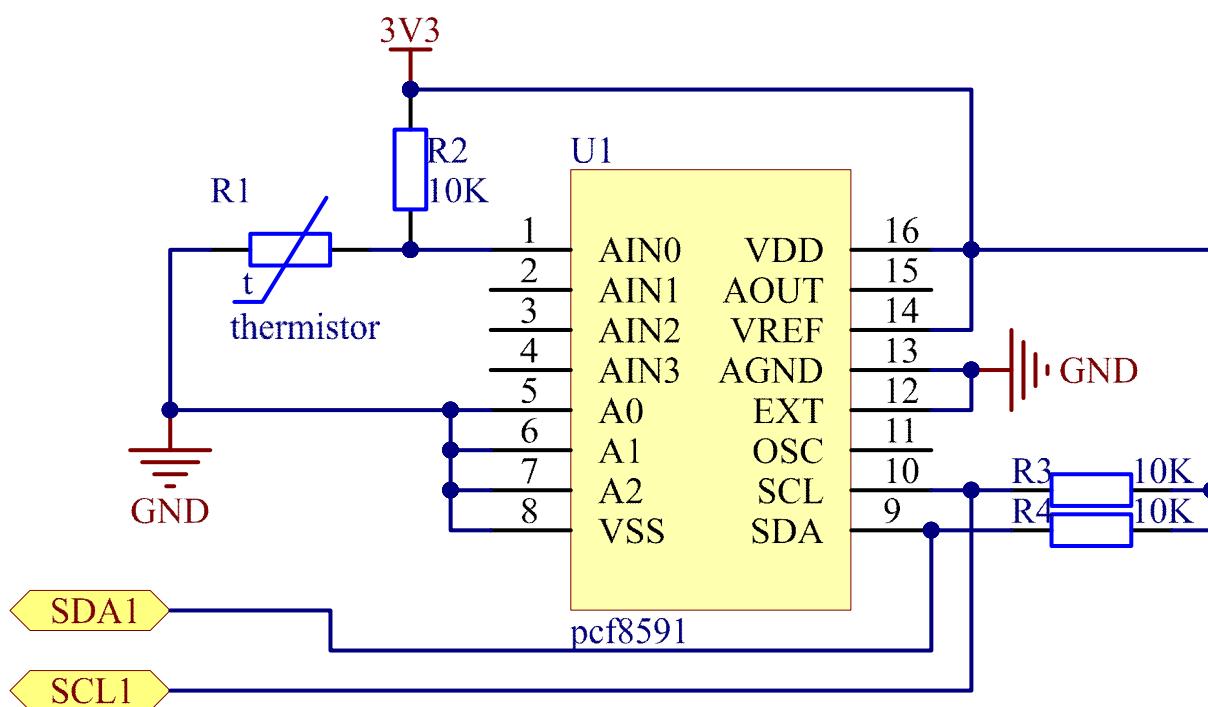
**exp** is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

Convert this formula  $T_K=1/(\ln(R_T/R_N)/B+1/T_N)$  to get Kelvin temperature that minus 273.15 equals degree Celsius.

This relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

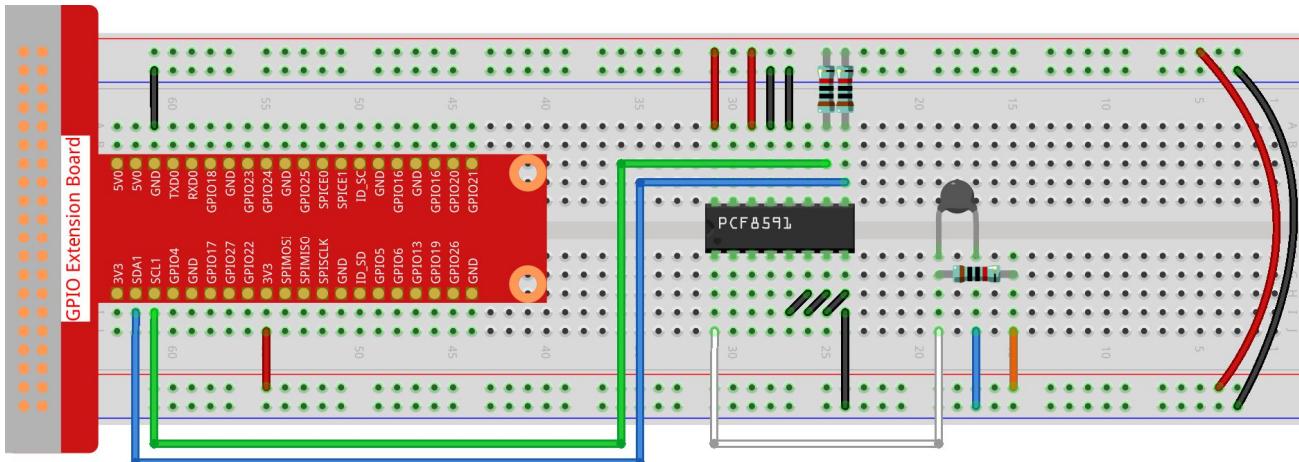
## Schematic Diagram

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



## Experimental Procedures

### Step 1: Build the circuit.



### Step 2: Setup I2C (see Appendix. If you have set I2C, skip this step.)

#### ➤ For C Language Users

### Step 3: Go to the folder of the code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.2/
```

### Step 4: Compile the code.

```
gcc 2.2.2_Thermistor.c -lwiringPi -lm
```

**Note:** `-lm` is to load the library math. Do not omit, or you will make an error.

### Step 5: Run the executable file.

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

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

## Code

```
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>
#include <math.h>

#define PCF 120

int main()
{
```

```

unsigned char analogVal;
double Vr, Rt, temp;
if(wiringPiSetup() == -1){
    printf("setup wiringPi failed !");
    return 1;
}
// Setup pcf8591 on base pin 120, and address 0x48
pcf8591Setup(PCF, 0x48);
while(1) // loop forever
{
    printf("loop");
    analogVal = analogRead(PCF + 0);
    Vr = 5 * (double)(analogVal) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    temp = temp - 273.15;
    printf("Current temperature : %lf\n", temp);

    delay (200);
}
return 0;
}

```

## Code Explanation

```
#include <math.h>
```

There is a C numerics library which declares a set of functions to compute common mathematical operations and transformations.

```
analogVal = analogRead(PCF + 0);
```

This function is used to read the value of the thermistor.

```
Vr = 5 * (double)(analogVal) / 255;
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
temp = temp - 273.15;
```

These calculations convert the thermistor values into Celsius values.

```
Vr = 5 * (double)(analogVal) / 255;
Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
```

These two lines of codes are calculating the voltage distribution with the read value analog so as to get Rt (resistance of thermistor).

```
temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
```

This code refers to plugging Rt into the formula  $T_k=1/(\ln(R_t/R_n)/B+1/T_n)$  to get Kelvin temperature.

```
temp = temp - 273.15;
```

Convert Kelvin temperature into degree Celsius.

## ➤ For Python Language Users

**Step 3:** Go to the folder of the code.

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

**Step 4:** Run the executable file

```
sudo python 2.2.2_Termistor.py
```

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

## Code

```
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time
import math

GPIO.setmode(GPIO.BCM)

def setup():
    ADC.setup(0x48)

def loop():
    while True:
        analogVal = ADC.read(0)
        Vr = 5 * float(analogVal) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
        temp = temp - 273.15
```

```

print ('temperature = ', temp, 'C')
time.sleep(0.2)

if __name__ == '__main__':
    try:
        setup()
        loop()
    except KeyboardInterrupt:
        pass

```

## Code Explanation

```
import math
```

There is a numerics library which declares a set of functions to compute common mathematical operations and transformations.

```
analogVal = ADC.read(0)
```

This function is used to read the value of the thermistor.

```

Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
temp = temp - 273.15

```

These calculations convert the thermistor values into Celsius values.

```

Vr = 5 * float(analogVal) / 255
Rt = 10000 * Vr / (5 - Vr)

```

These two lines of codes are calculating the voltage distribution with the read value analog so as to get Rt (resistance of thermistor).

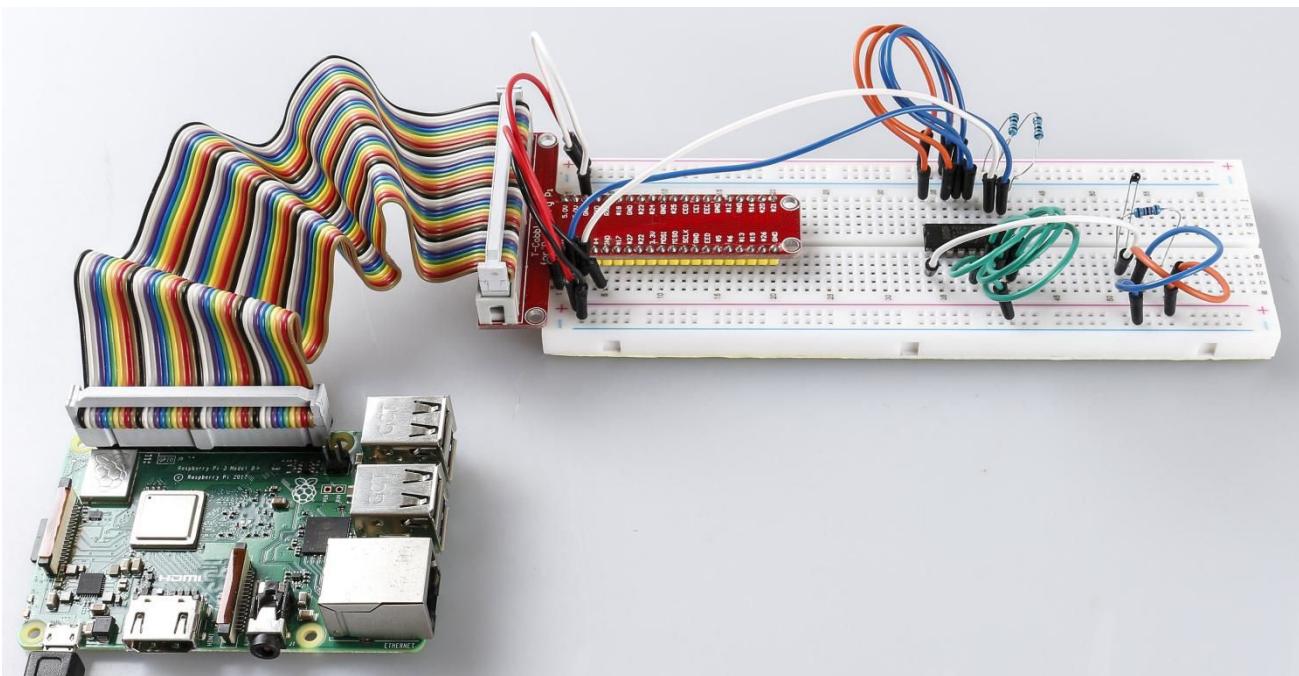
```
temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
```

This code refers to plugging Rt into the formula  $T_K=1/(\ln(R_T/R_N)/B+1/T_N)$  to get Kelvin temperature.

```
temp = temp - 273.15
```

Convert Kelvin temperature into degree Celsius.

## Phenomenon Picture



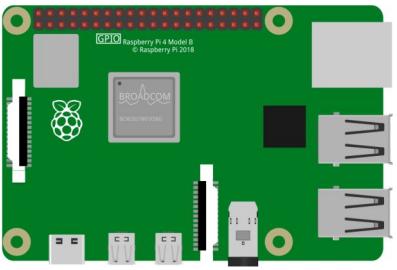
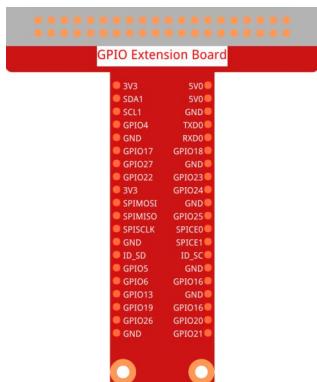
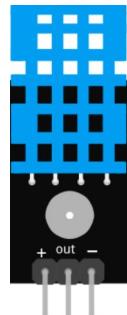
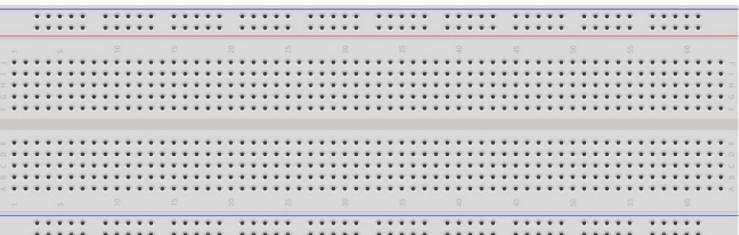
## 2.2.3 DHT-11

### Introduction

The digital temperature and humidity sensor DHT11 is a composite sensor that contains a calibrated digital signal output of temperature and humidity. The technology of a dedicated digital modules collection and the technology of the temperature and humidity sensing are applied to ensure that the product has high reliability and excellent stability.

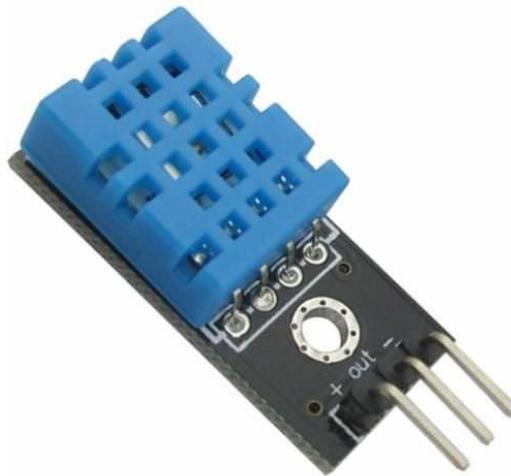
The sensors include a wet element resistive sensor and a NTC temperature sensor and they are connected to a high performance 8-bit microcontroller.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * DHT-11
	 GPIO Extension Board Pinout: <ul style="list-style-type: none"><li>■ 3V3 SVO</li><li>■ SDA1 SVO</li><li>■ SCL1 GND</li><li>■ GPIO4 TXD0</li><li>■ GND RXD0</li><li>■ GPIO17 GPIO18</li><li>■ GPIO27 GND</li><li>■ GPIO22 GPIO28</li><li>■ SDA GND</li><li>■ SPI0SI GND</li><li>■ SPI0SO GPIO15</li><li>■ SPI0CLK SPICE0</li><li>■ GND SPICE1</li><li>■ ID_SD ID_SC</li><li>■ GPIO5 GND</li><li>■ GPIO6 GPIO16</li><li>■ GPIO13 GND</li><li>■ GPIO19 GPIO16</li><li>■ GPIO26 GPIO20</li><li>■ GND GPIO21</li></ul>	
1 * 40-pin Cable	Several Jumper Wires	
		
1 * Breadboard	1 * Resistor 10KΩ	
		

## Principle

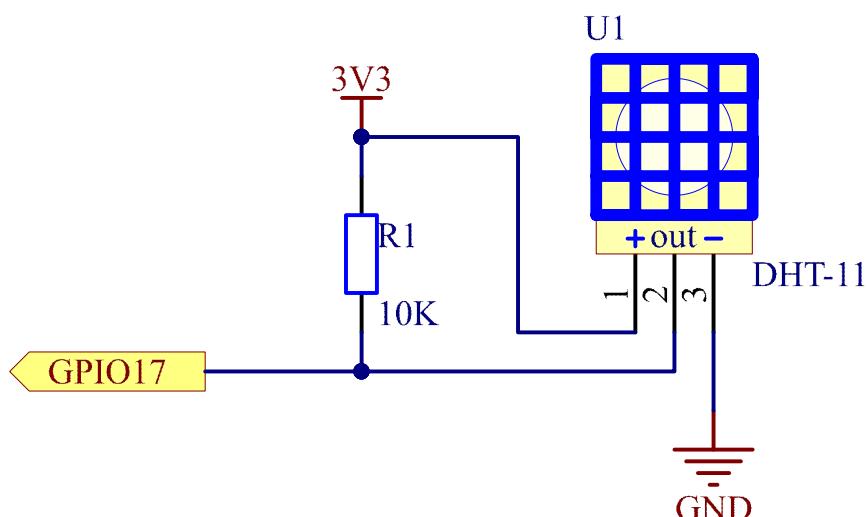
The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins are needed).



Only three pins are available: VCC, GND, and DATA. The communication process begins with the DATA line sending start signals to DHT11, and DHT11 receives the signals and returns an answer signal. Then the host receives the answer signal and begins to receive 40-bit humiture data (8-bit humidity integer + 8-bit humidity decimal + 8-bit temperature integer + 8-bit temperature decimal + 8-bit checksum). For more information, please refer to DHT11 datasheet.

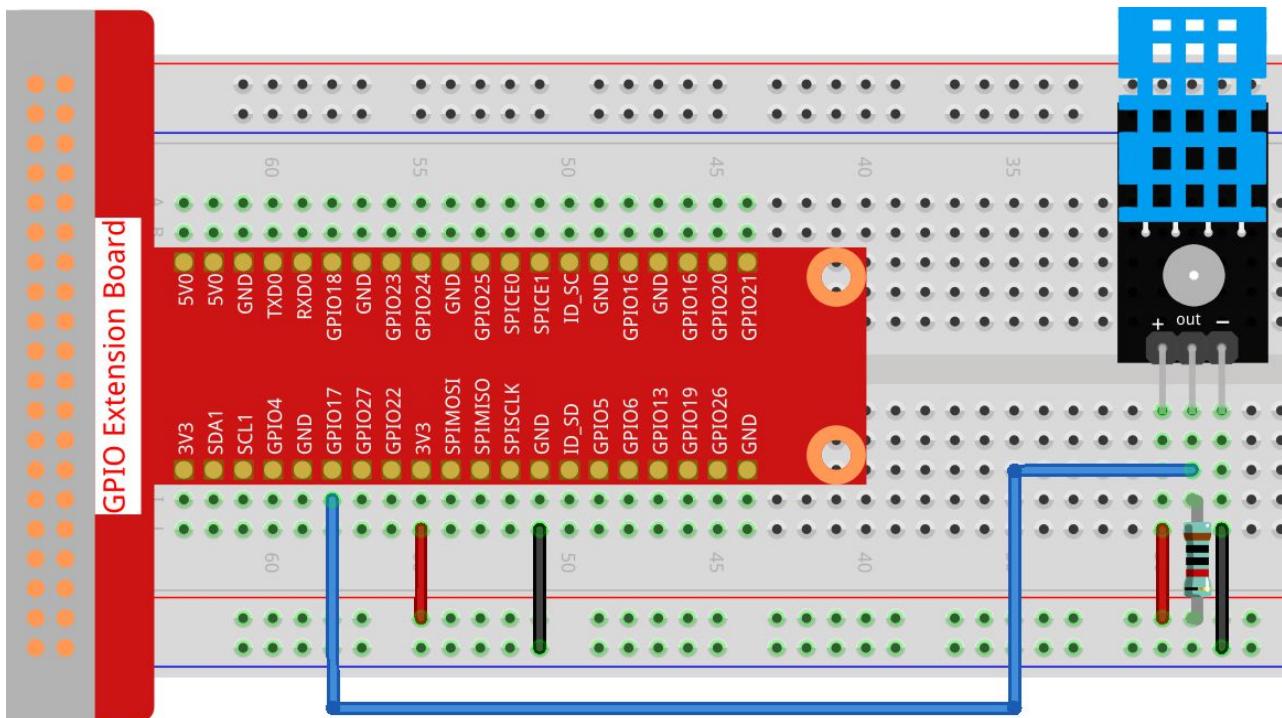
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



## Experimental Procedures

### Step 1: Build the circuit.



#### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.3/
```

### Step 3: Compile the code.

```
gcc 2.2.3_DHT.cpp -lwiringPi
```

**Note:** The program contains custom header files that are compiled when CPP files are compiled.

### Step 4: Run the executable file.

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

After the code runs, the program will print the temperature and humidity detected by DHT11 on the computer screen.

### Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 2.2.3\_DHT.cpp in **bash**.

```
#include <wiringPi.h>
#include <stdio.h>
```

```
#include <stdint.h>
#include "DHT.hpp"

#define DHT11_Pin 0

//Function: Read DHT sensor, store the original data in bits[]
int DHT::readSensor(int pin,int wakeupDelay){
    .....
}

//Function: Read DHT sensor, analyze the data of temperature and humidity
int DHT::readDHT11(int pin){
    .....
}

int main(){
    DHT dht;      //create a DHT class object
    int check;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    while(1){
        check = dht.readDHT11(DHT11_Pin);
        switch(check){
            case DHTLIB_OK: //if the return value is DHTLIB_OK, the data is normal.
                printf("Humidity : %.2f, \t Temperature : %.2f \n",dht.humidity,dht.temperature);
                break;
            case DHTLIB_ERROR_CHECKSUM: //data check has errors
                printf("Humidity : %.2f, \t Temperature : %.2f\t (this value may be
incorrect)\n",dht.humidity,dht.temperature);
                break;
            case DHTLIB_ERROR_TIMEOUT: //reading DHT times out
                printf("Timeout! \n");
                break;
            case DHTLIB_INVALID_VALUE: //other errors
                printf("Unknow problem! \n");
                break;
        }
        delay(2000);
    }
    return 1;
}
```

}

## Code Explanation

```
#include "DHT.hpp"
```

Dht.hpp is an open source file for reading sensor values of DHT which makes it easy for us to use the DHT sensor. Here, we write the invocation functionality directly below the open source file.

```
int DHT::readSensor(int pin,int wakeupDelay){}
int DHT::readDHT11(int pin){}
```

These two functions are the content of the open source file itself, and they will read the value of the sensor. After calculation we can understand the humidity and temperature in that they themselves have a return value that monitors the status of the sensor.

```
check = dht.readDHT11(DHT11_Pin);
switch(check){
    case DHTLIB_OK: //if the return value is DHTLIB_OK, the data is normal.
        printf("Humidity : %.2f, \t Temperature : %.2f \n",dht.humidity,dht.temperature);
        break;
    case DHTLIB_ERROR_CHECKSUM: //data check has errors
        printf("Humidity : %.2f, \t Temperature : %.2f \t (this value may be
incorrect)\n",dht.humidity,dht.temperature);
        break;
    case DHTLIB_ERROR_TIMEOUT: //reading DHT times out
        printf("Timeout! \n");
        break;
    case DHTLIB_INVALID_VALUE: //other errors
        printf("Unknown problem! \n");
        break;
}
```

Read the value of DHT11\_Pin, and store it in the variable, check. Then, if what you get is DHTLIB\_OK, it means that the DHT11 sensor works in good condition; accordingly, the printf function is called to print the temperature and humidity.

On the contrary, after finishing the operation of check, if the read value is DHTLIB\_ERROR\_CHECKSUM , DHTLIB\_ERROR\_TIMEOUT or DHTLIB\_INVALID\_VALUE,

it means that there is something wrong in the working process of modules. What's more, there appears an error message.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 2.2.3_DHT.py
```

After the code runs, the program will print the temperature and humidity detected by DHT11 on the computer screen.

## Code

```
import RPi.GPIO as GPIO
import time
DHTPin = 11

class DHT(object):
    .....
    #Read DHT sensor, store the original data in bits[]
    def readSensor(self,pin,wakeupDelay):
        .....
        #Read DHT sensor, analyze the data of temperature and humidity
    def readDHT11(self):
        .....

    def loop():
        dht = DHT(DHTPin)#create a DHT class object
        while(True):
            check = dht.readDHT11()
            if (check is dht.DHTLIB_OK):
                print("Humidity : %.2f, Temperature : %.2f %(dht.humidity,dht.temperature))
            elif (check is dht.DHTLIB_ERROR_CHECKSUM):
                print("Humidity : %.2f, Temperature : %.2f (this value may
incorrect)%(dht.humidity,dht.temperature))
            elif (check is dht.DHTLIB_ERROR_TIMEOUT):
                print("Timeout! ")
            else:
                print("unknow problem! ")
```

```
time.sleep(2)

if __name__ == '__main__':
    try:
        loop()
    except KeyboardInterrupt:
        pass
    exit()
```

## Explanation

```
class DHT(object):
    .....
    #Read DHT sensor, store the original data in bits[]
    def readSensor(self,pin,wakeupDelay):
        .....
        #Read DHT sensor, analyze the data of temperature and humidity
    def readDHT11(self):
        .....
```

This class is an open source code for reading sensor values of DHT. It makes it easy for us to use the DHT sensor. Here, we write the invocation functionality directly below the open source class.

These two functions are the content of the open source file itself, they will read the value of the sensor, and after calculation we can understand the humidity and temperature. They themselves have a return value that monitors the status of the sensor.

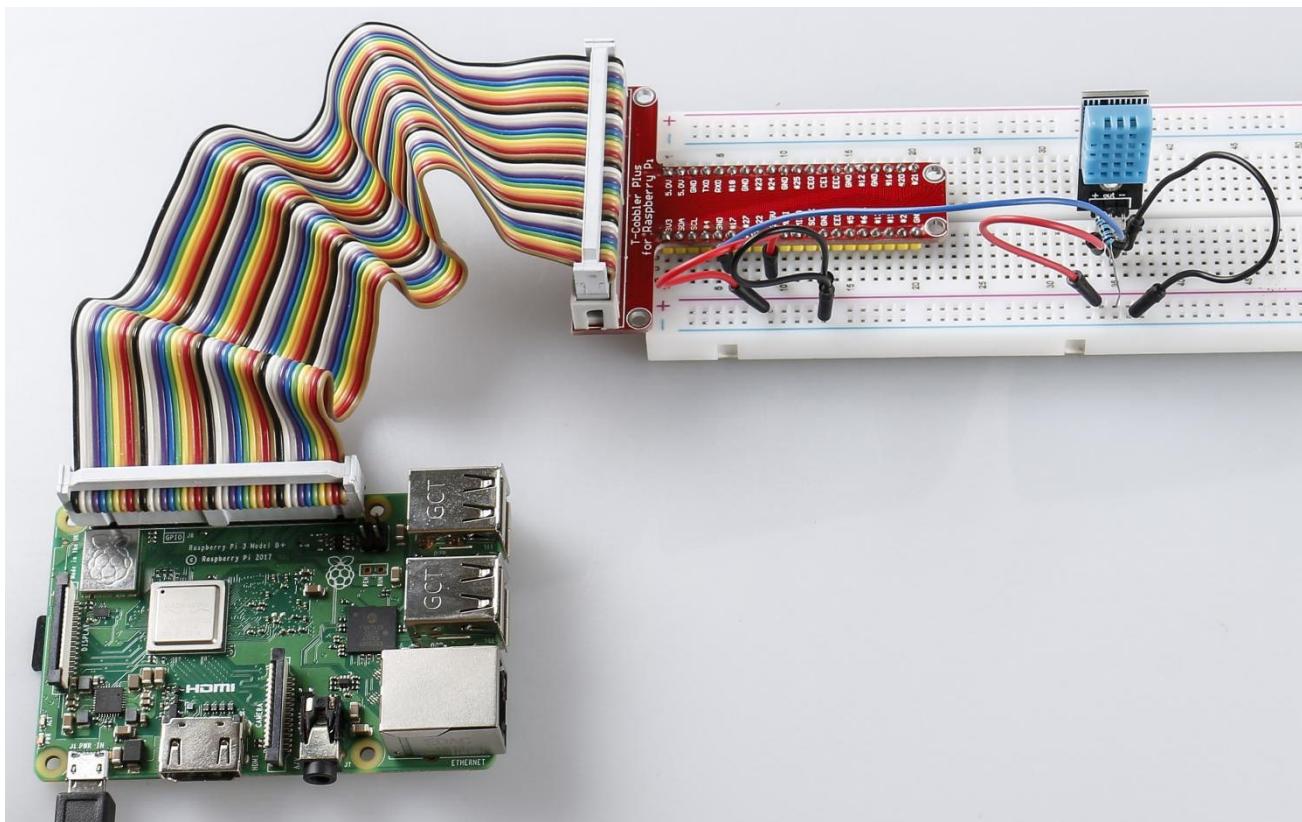
```
def loop():
    dht = DHT(DHTPin)#create a DHT class object
    while(True):
        check = dht.readDHT11()
        if (check is dht.DHTLIB_OK):
            print("Humidity : %.2f, Temperature : %.2f "%(dht.humidity,dht.temperature))
        elif (check is dht.DHTLIB_ERROR_CHECKSUM):
            print("Humidity : %.2f, Temperature : %.2f (this value may
incorrect)%(dht.humidity,dht.temperature))
        elif (check is dht.DHTLIB_ERROR_TIMEOUT):
            print("Timeout! ")
```

```
else:  
    print("unknow problem! ")  
    time.sleep(2)
```

Read the value of DHT11\_Pin, and store it in the variable, check. Then, if what you get is dht.DHTLIB\_OK, it means that the DHT11 sensor works in good condition; accordingly, the printf function is called to print the temperature and humidity.

On the contrary, after finishing the operation of check, if the read value is DHTLIB\_ERROR\_CHECKSUM or DHTLIB\_ERROR\_TIMEOUT, it means that there is something wrong in the working process of modules. What's more, there appears an error message.

## Phenomenon Picture

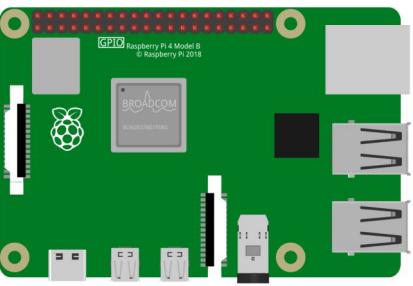
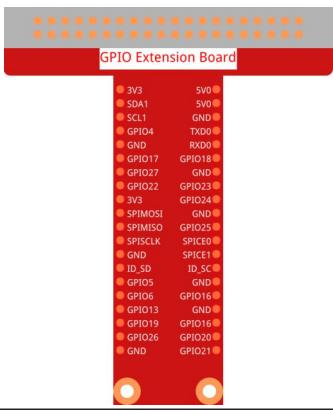
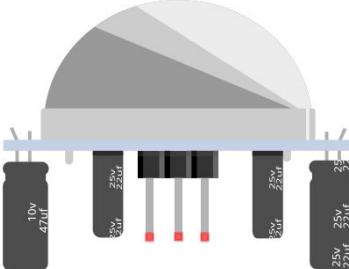
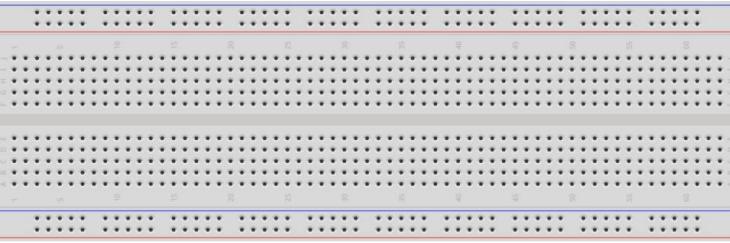


## 2.2.4 PIR

### Introduction

In this project, we will make a device by using the human body infrared pyroelectric sensors. When someone gets closer to the LED, the LED will turn on automatically. If not, the light will turn off. This infrared motion sensor is a kind of sensor that can detect the infrared emitted by human and animals.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * PIR
		
1 * 40-pin Cable	Several Jumper Wires	
		
1 * Breadboard	1 * Resistor 220Ω	1 * LED
		

### Principle

The PIR sensor detects infrared heat radiation that can be used to detect the presence of organisms that emit infrared heat radiation.

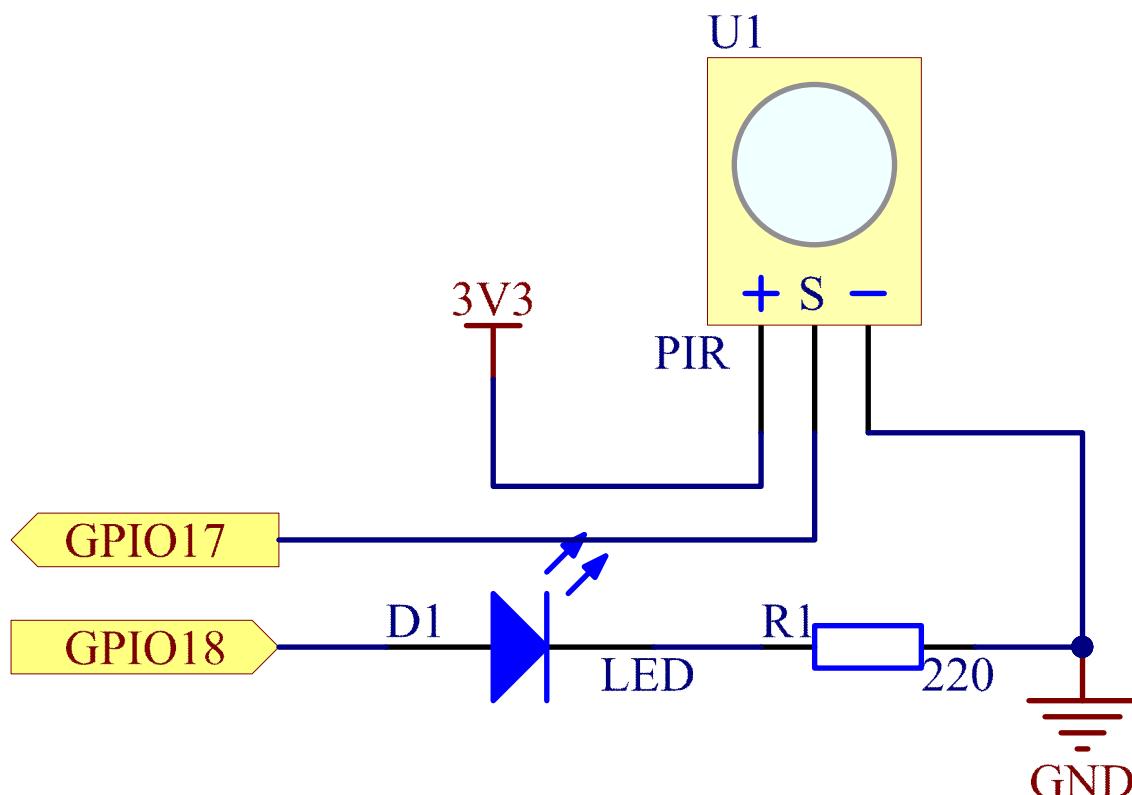
The PIR sensor is split into two slots that are connected to a differential amplifier. Whenever a stationary object is in front of the sensor, the two slots receive the same

amount of radiation and the output is zero. Whenever a moving object is in front of the sensor, one of the slots receives more radiation than the other , which makes the output fluctuate high or low. This change in output voltage is a result of detection of motion.



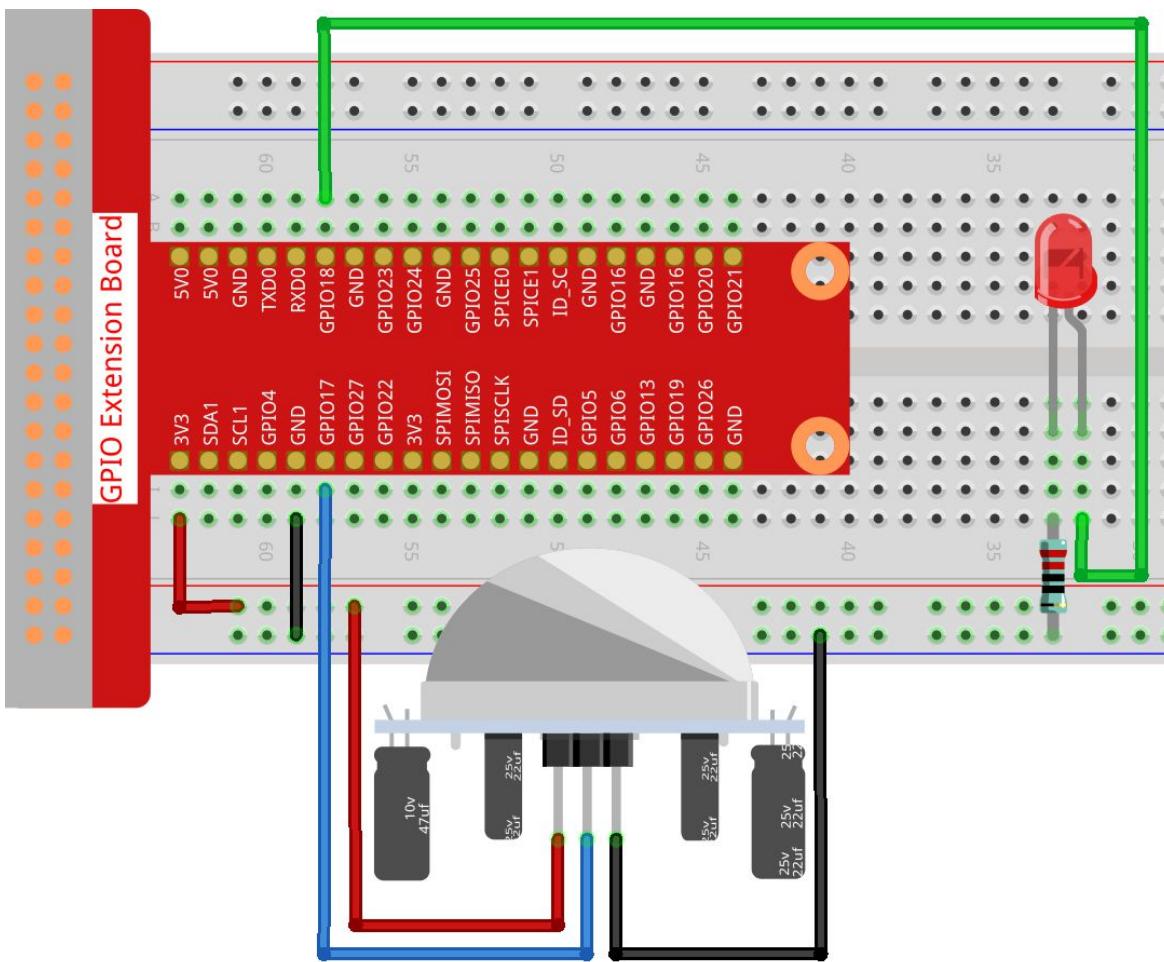
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18



## Experimental Procedures

**Step 1:** Build the circuit.



### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.4/
```

**Step 3:** Compile the code.

```
gcc 2.2.4_PIR.c -lwiringPi
```

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

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

After the code runs, PIR detects surroundings and lights an LED if it senses someone walking by. There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. In order to make the PIR module work better, you need to try to adjust these two potentiometers.

## Code

```
#include <wiringPi.h>
#include <stdio.h>

#define ledPin 1 //define the ledPin
#define sensorPin 0 //define the sensorPin

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ledPin, OUTPUT);
    pinMode(sensorPin, INPUT);
    while(1){
        if(digitalRead(sensorPin) == HIGH){ //if read sensor for high level
            digitalWrite(ledPin, HIGH); //led on
            printf("LED ON! \n");
        }
        else {
            digitalWrite(ledPin, LOW); //led off
            printf("LED OFF!\n");
        }
    }
    return 0;
}
```

## Code Explanation

```
if(digitalRead(sensorPin) == HIGH){ //if read sensor for high level
    digitalWrite(ledPin, HIGH); //led on
    printf("LED ON! \n");
}
```

The PIR works like a button, and we read its value to control the LED.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 2.2.4_PIR.py
```

After the code runs, PIR detects surroundings and lights an LED if it senses someone walking by. There are two potentiometers on the PIR module: one is to adjust sensitivity and the other is to adjust the detection distance. In order to make the PIR module work better, you need to try to adjust these two potentiometers.

## Code

```
import RPi.GPIO as GPIO

ledPin = 18 # define the ledPin
sensorPin = 17 # define the sensorPin

def setup():
    GPIO.setmode(GPIO.BCM)      # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT)  # Set ledPin's mode is output
    GPIO.setup(sensorPin, GPIO.IN) # Set sensorPin's mode is input

def loop():
    while True:
        if GPIO.input(sensorPin)==GPIO.HIGH:
            GPIO.output(ledPin,GPIO.HIGH)
            print ('led on ...')
        else :
            GPIO.output(ledPin,GPIO.LOW)
            print ('led off ...')

def destroy():
    GPIO.cleanup()          # Release resource

if __name__ == '__main__':  # Program start from here
    setup()
    try:
        loop()
```

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

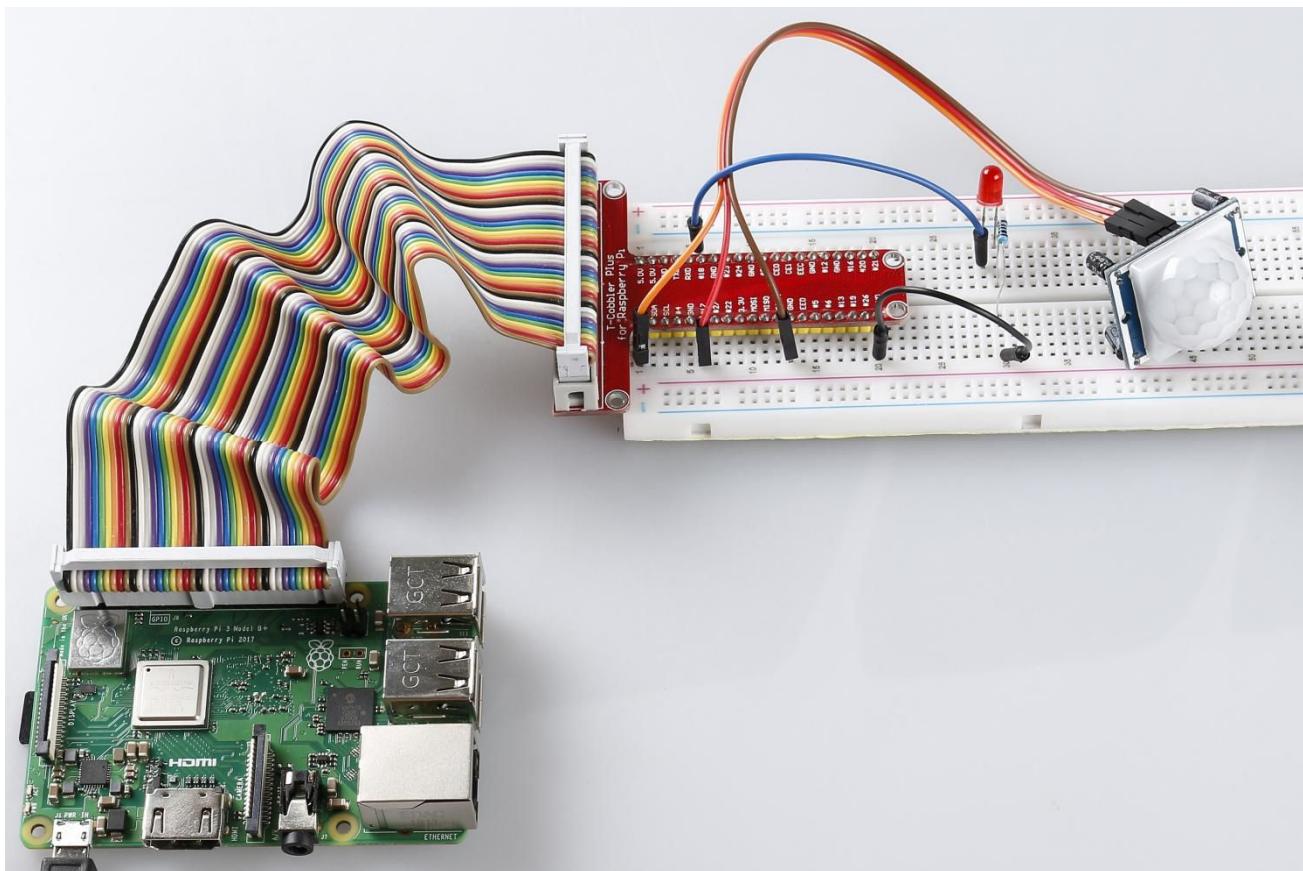
destroy()
```

## Code Explanation

```
if GPIO.input(sensorPin)==GPIO.HIGH:
    GPIO.output(ledPin,GPIO.HIGH)
    print ('led on ...')
```

The PIR works like a button, and we read its value to control the LED.

## Phenomenon Picture



## 2.2.5 Ultrasonic Sensor Module

## Introduction

The ultrasonic sensor uses ultrasonic to accurately detect objects and measure distances. It sends out ultrasonic waves and converts them into electronic signals.

# Components

The diagram illustrates the components needed for the project:

- 1 \* Raspberry Pi**: A green printed circuit board with a Broadcom BCM2837B1150 SoC, a USB port, and a 40-pin GPIO header.
- 1 \* T-Extension Board**: A red and grey board labeled "GPIO Extension Board". It has two circular pads at the top and a 2x20 pin header at the bottom. A table below lists the pin assignments:

Pin	Function
3V3	5V0
SDA1	5V0
SCL1	GND
GPIO4	TXD0
GND	RXD0
GPIO17	GPIO18
GPIO27	GND
GPIO22	GPIO23
3V	GPIO24
SPI MOSI	GND
SPI MISO	GPIO25
SPI SCLK	SPICE0
GND	SPICE1
ID_SD	ID_SC
GPIO5	GND
GPIO6	GPIO16
GPIO13	GND
GPIO19	GPIO16
GPIO26	GPIO20
GND	GPIO21
- 1 \* HC SR04**: A blue sensor module with two black cylindrical ultrasonic transducers. It has four pins labeled Ucc, Trig, Echo, and Gnd.
- Several Jumper Wires**: A long, thin black cable with a connector at one end.
- 1 \* 40-pin Cable**: A long, multi-colored ribbon cable with 40 pins on each end, used for connecting the Raspberry Pi to the T-Extension Board.
- 1 \* Breadboard**: A grey perforated breadboard with four horizontal rails and a central grid of connection points.

# Principle

# **Ultrasonic**

Ultrasonic ranging module provides 2cm - 400cm non-contact measurement function, and the ranging accuracy can reach to 3mm. It can ensure that the signal is stable within 5m, and the signal is gradually weakened after 5m, till the 7m position disappears.

The module includes ultrasonic transmitters, receiver and control circuit. The basic principles are as follows:

- (1) Use an IO flip-flop to process a high level signal of at least 10us;
  - (2) The module automatically sends eight 40khz and detects if there is a pulse signal return.

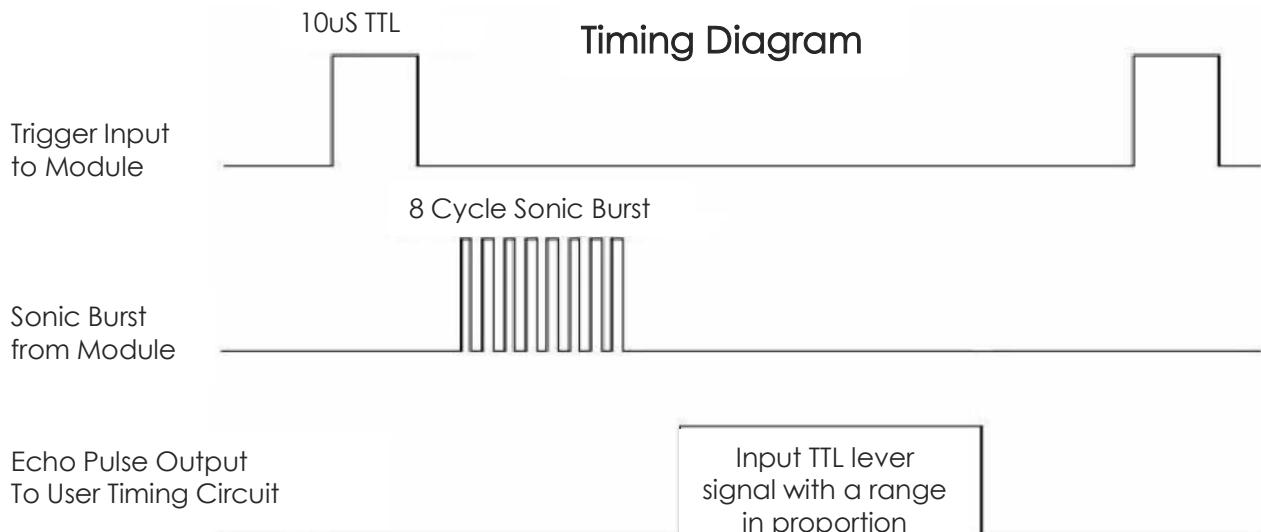
(3) If the signal returns, passing the high level, the high output IO duration is the time from the transmission of the ultrasonic wave to the return of it. Here, test distance = (high time x sound speed (340 m / s) / 2.



<b>TRIG</b>	Trigger Pulse Input
<b>ECHO</b>	Echo Pulse Output
<b>GND</b>	Ground
<b>VCC</b>	Supply

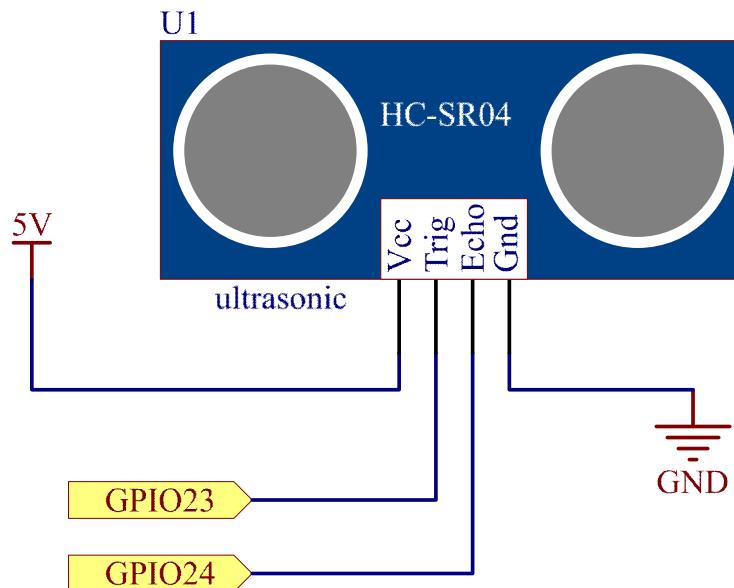
The timing diagram is shown below. You only need to supply a short 10us pulse for the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula:  $us / 58 = \text{centimeters}$  or  $us / 148 = \text{inch}$ ; or: the range = high level time \* velocity (340M/S) / 2; you are suggested to use measurement cycle over 60ms in order to prevent signal collisions of trigger signal and the echo signal.



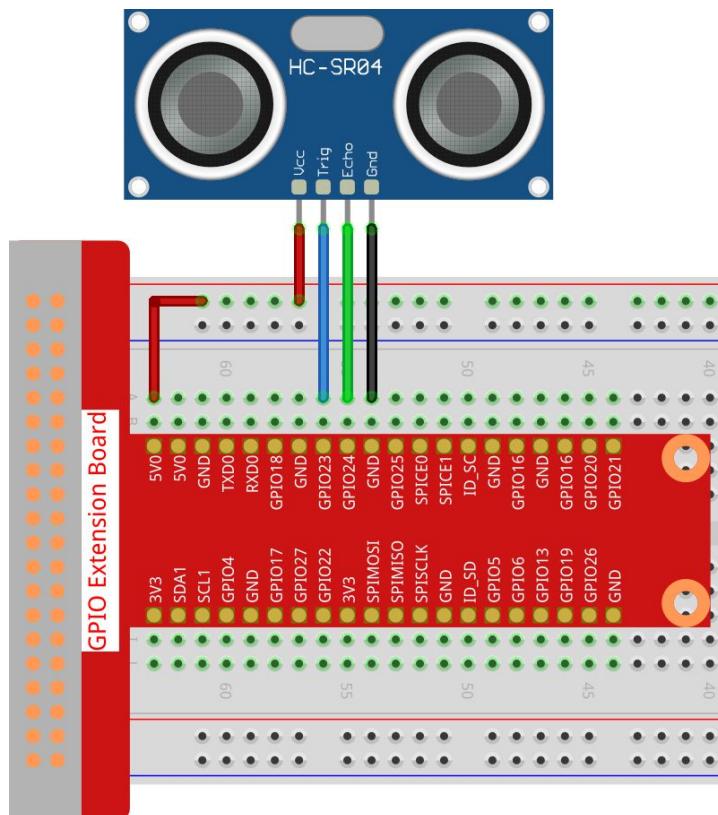
# Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24



## Experimental Procedures

## **Step 1: Build the circuit.**



## ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/2.2.5/
```

**Step 3:** Compile the code.

```
gcc 2.2.5_Ultrasonic.c -lwiringPi
```

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

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

With the code run, the ultrasonic sensor module detects the distance between the obstacle ahead and the module itself, then the distance value will be printed on the screen.

## Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define Trig  4
#define Echo  5

void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}

float disMeasure(void)
{
    struct timeval tv1;
    struct timeval tv2;
    long time1, time2;
    float dis;

    digitalWrite(Trig, LOW);
    delayMicroseconds(2);

    digitalWrite(Trig, HIGH);
    delayMicroseconds(10);
```

```
digitalWrite(Trig, LOW);

while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);

while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);

time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;
time2 = tv2.tv_sec * 1000000 + tv2.tv_usec;

dis = (float)(time2 - time1) / 1000000 * 34000 / 2;

return dis;
}

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

    ultraInit();

    while(1){
        dis = disMeasure();
        printf("%0.2f cm\n\n",dis);
        delay(300);
    }

    return 0;
}
```

## Code Explanation

```
void ultraInit(void)
{
    pinMode(Echo, INPUT);
    pinMode(Trig, OUTPUT);
}
```

Initialize the ultrasonic pin; meanwhile, set Echo to input, Trig to output.

```
float disMeasure(void){};
```

This function is used to realize the function of ultrasonic sensor by calculating the return detection distance.

```
struct timeval tv1;
struct timeval tv2;
```

Struct timeval is a structure used to store the current time. The complete structure is as follows:

```
struct timeval
{
    __time_t tv_sec;      /* Seconds. */
    __suseconds_t tv_usec; /* Microseconds. */
};
```

Here, tv\_sec represents the seconds that Epoch spent when creating struct timeval. Tv\_usec stands for microseconds or a fraction of seconds.

```
digitalWrite(Trig, HIGH);
delayMicroseconds(10);
digitalWrite(Trig, LOW);
```

A 10us ultrasonic pulse is being sent out.

```
while(!(digitalRead(Echo) == 1));
gettimeofday(&tv1, NULL);
```

This empty loop is used to ensure that when the trigger signal is sent, there is no interfering echo signal and then get the current time.

```
while(!(digitalRead(Echo) == 0));
gettimeofday(&tv2, NULL);
```

This empty loop is used to ensure that the next step is not performed until the echo signal is received and then get the current time.

```
time1 = tv1.tv_sec * 1000000 + tv1.tv_usec;  
time2 = tv2.tv_sec * 1000000 + tv2.tv_usec;
```

Convert the time stored by struct timeval into a full microsecond time.

```
dis = (float)(time2 - time1) / 1000000 * 34000 / 2;
```

The distance is calculated by the time interval and the speed of sound propagation. The speed of sound in the air: 34000cm/s.

## ➤ For Python Language Users

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

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

**Step 3:** Run the executable file.

```
sudo python 2.2.5_Ultrasonic.py
```

With the code run, the ultrasonic sensor module detects the distance between the obstacle ahead and the module itself, then the distance value will be printed on the screen.

## Code

```
import RPi.GPIO as GPIO  
import time  
  
TRIG = 16  
ECHO = 18  
  
def setup():  
    GPIO.setmode(GPIO.BOARD)  
    GPIO.setup(TRIG, GPIO.OUT)  
    GPIO.setup(ECHO, GPIO.IN)  
  
def distance():  
    GPIO.output(TRIG, 0)  
    time.sleep(0.000002)  
  
    GPIO.output(TRIG, 1)
```

```

time.sleep(0.00001)
GPIO.output(TRIG, 0)

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def loop():
    while True:
        dis = distance()
        print (dis, 'cm')
        print ("")
        time.sleep(0.3)

def destroy():
    GPIO.cleanup()

if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```
def distance():
```

This function is used to realize the function of ultrasonic sensor by calculating the return detection distance.

```

GPIO.output(TRIG, 1)
time.sleep(0.00001)
GPIO.output(TRIG, 0)

```

This is sending out a 10us ultrasonic pulse.

```
while GPIO.input(ECHO) == 0:  
    a = 0  
    time1 = time.time()
```

This empty loop is used to ensure that when the trigger signal is sent, there is no interfering echo signal and then get the current time.

```
while GPIO.input(ECHO) == 1:  
    a = 1  
    time2 = time.time()
```

This empty loop is used to ensure that the next step is not performed until the echo signal is received and then get the current time.

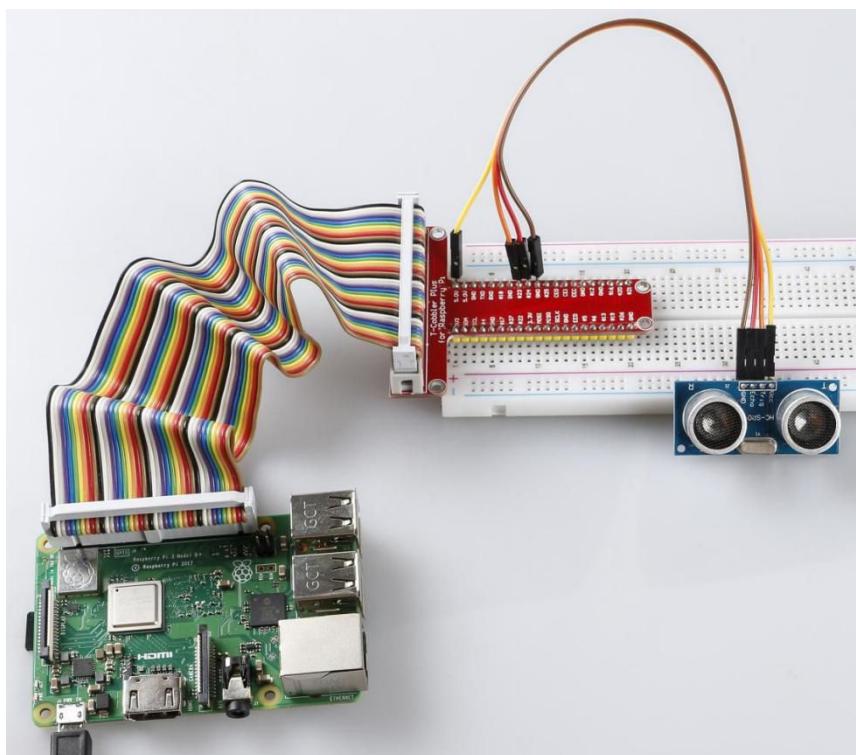
```
during = time2 - time1
```

Execute the interval calculation.

```
return during * 340 / 2 * 100
```

The distance is calculated in the light of time interval and the speed of sound propagation. The speed of sound in the air: 340m/s.

## Phenomenon Picture



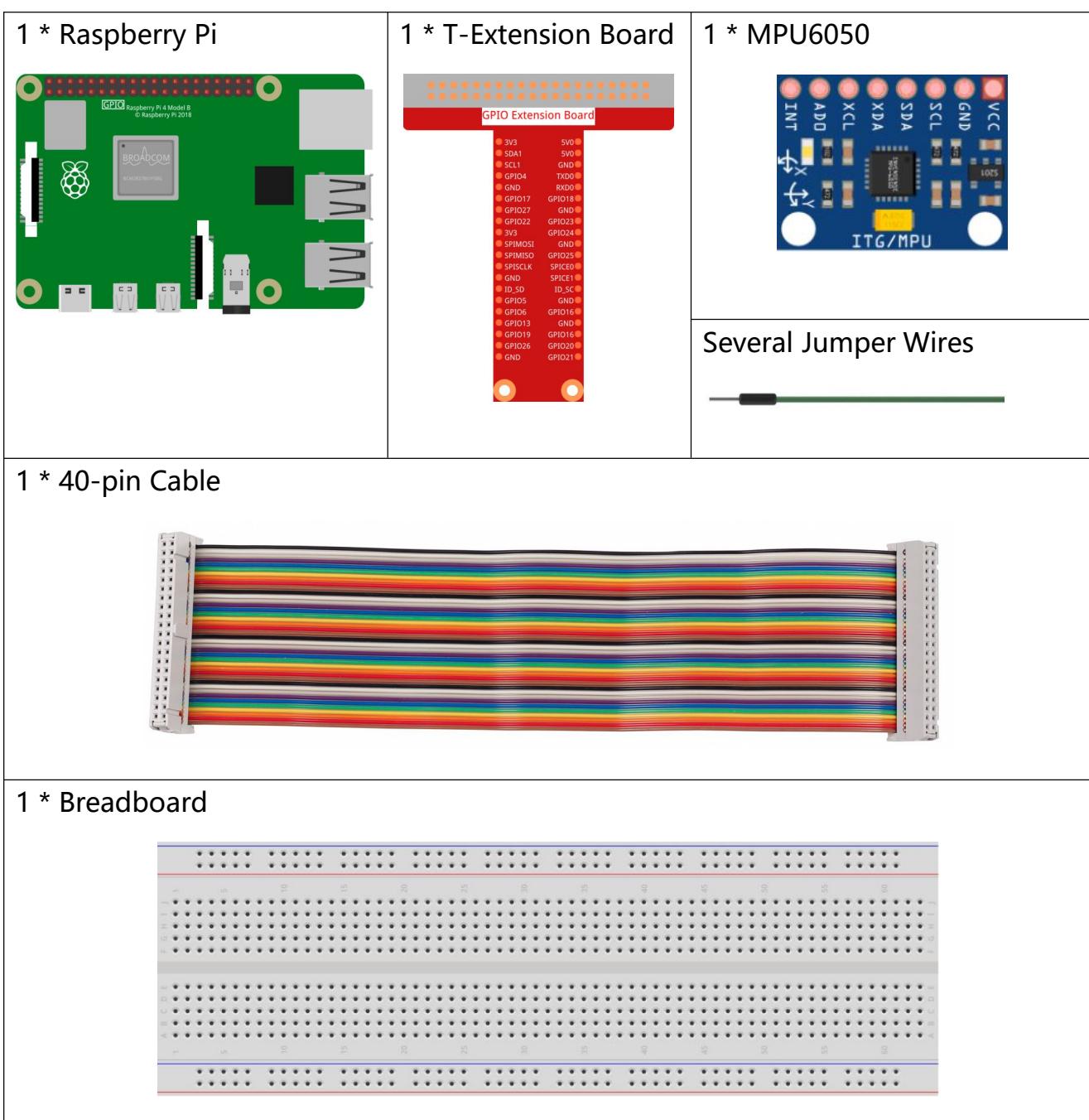
## 2.2.6 MPU6050 Module

# Introduction

The MPU-6050 is the world's first and only 6-axis motion tracking devices (combineing 3-axis Gyroscope, 3-axis Accelerometer) designed for smartphones, tablets and wearable sensors that have these features, including the low power, low cost, and high performance requirements.

In this experiment, use I<sup>2</sup>C to obtain the values of the three-axis acceleration sensor and three-axis gyroscope for MPU6050 and display them on the screen.

# Components



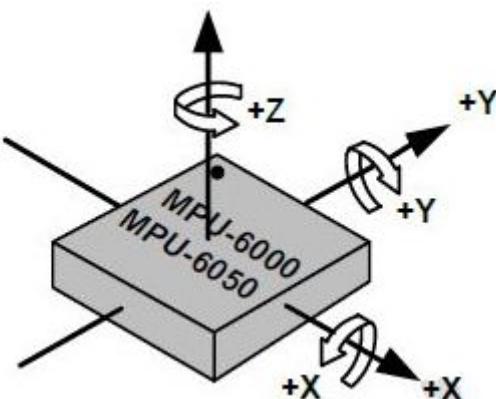
## Principle

### MPU6050

The MPU-6050 is a 6-axis(combines 3-axis Gyroscope, 3-axis Accelerometer) motion tracking devices.

Its three coordinate systems are defined as follows:

Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



### 3-axis Accelerometer

We can use the MPU6050 to detect its acceleration on each coordinate axis (in the stationary desktop state, the Z-axis acceleration is 1 gravity unit, and the X and Y axes are 0). If it is tilted or in a weightless/overweight condition, the corresponding reading will change.

There are four kinds of measuring ranges that can be selected programmatically: +/- 2g, +/- 4g, +/- 8g, and +/- 16g (2g by default) corresponding to each precision. Values range from -32768 to 32767.

The reading of accelerometer is converted to an acceleration value by mapping the reading from the reading range to the measuring range.

$$\text{Acceleration} = (\text{Accelerometer axis raw data} / 65536 * \text{full scale Acceleration range}) \text{ g}$$

Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as +/-2g:

$$\text{Acceleration along the X axis} = (16384 / 65536 * 4) \text{ g}$$

$$=1\text{g}$$

## 3-axis Gyroscope

The Gyroscope also has four kinds of measuring ranges: +/- 250, +/- 500, +/- 1000, +/- 2000. The calculation method and Acceleration are basically consistent.

The formula for converting the reading into angular velocity is as follows:

$$\text{Angular velocity} = (\text{Gyroscope axis raw data} / 65536 * \text{full scale Gyroscope range})^{\circ}/\text{s}$$

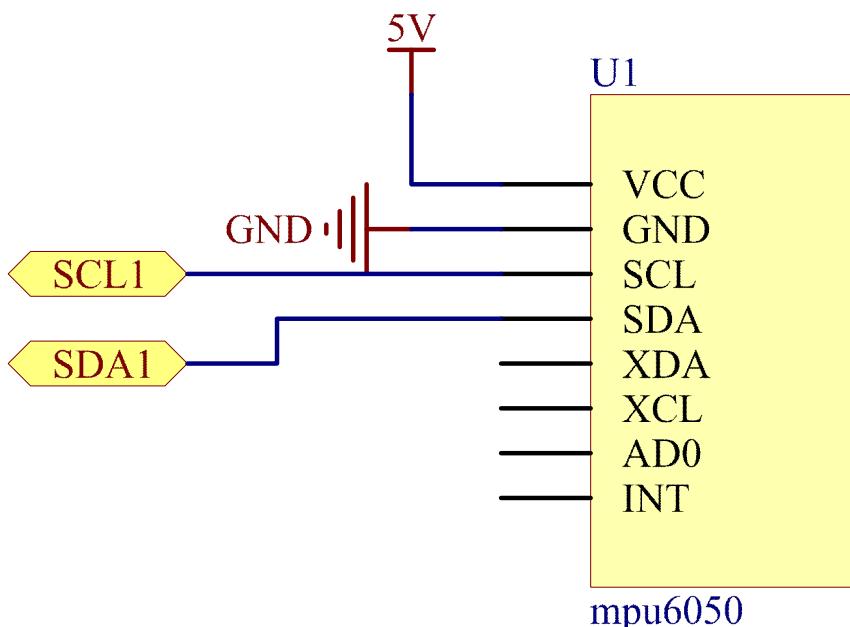
The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250 ° / s:

$$\begin{aligned}\text{Angular velocity along the X axis} &= (16384 / 65536 * 500)^{\circ}/\text{s} \\ &= 125^{\circ}/\text{s}\end{aligned}$$

## Schematic Diagram

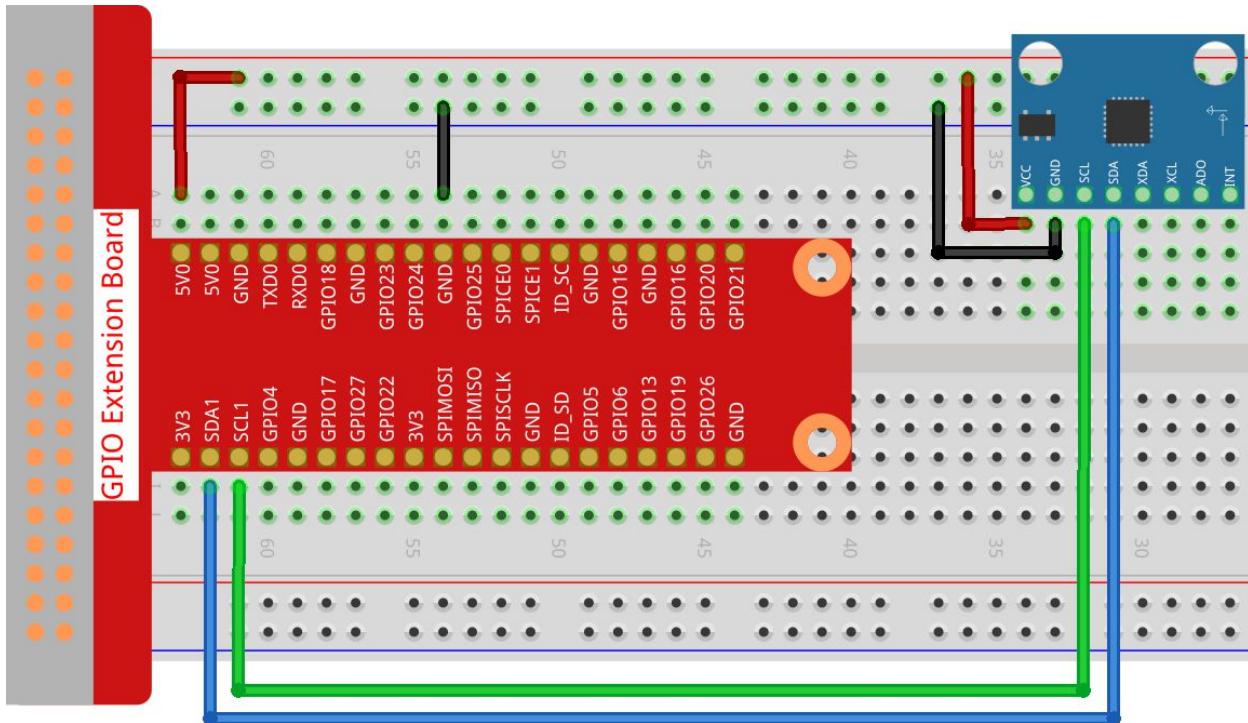
MPU6050 communicates with the microcontroller through the I2C bus interface. The SDA1 and SCL1 need to be connected to the corresponding pin.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.6/
```

#### Step 3: Compile the code.

```
gcc 2.2.6_MP6050.cpp I2Cdev.cpp MP6050.cpp -lwiringPi
```

**Note:** The program contains custom headers that are compiled when CPP files are compiled.

You can use this method to simplify the instruction.

```
gcc *.cpp -lwiringPi
```

**Note:** Here we use the wildcard (\*), which causes all the files that conform to the format to be processed together.

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

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

With the code run, the acceleration/angular velocity on each axis read by MP6050 will be printed on the screen after being calculating.

## Code

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 mpu;      //instantiate a MPU6050 class object
int16_t ax, ay, az; //store acceleration data
int16_t gx, gy, gz; //store gyroscope data

void loop() {
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    printf("Acceleration:\n");
    printf("X:%.2f g Y:%.2f g Z:%.2f g\n", (float)ax/16384,(float)ay/16384,(float)az/16384);
    printf("Angular velocity:\n");
    printf("X:%.2f d/s Y:%.2f d/s Z:%.2f d/s\n", (float)gx/131,(float)gy/131,(float)gz/131);
}

int main()
{
    mpu.initialize(); //initialize MPU6050
    // verify connection
    printf(mpu.testConnection() ? "" : "initialize failed\n");
    while(1){
        loop();
    }
    return 0;
}
```

## Code Explanation

```
#include "I2Cdev.h"
#include "MPU6050.h"
```

These two files are open source files used to drive MPU6050 which makes it easy to use MPU6050. They read the values of the 3-axis Accelerometer and the 3-axis Gyroscope so that we can calculate the acceleration and angular velocity.

```
MPU6050 mpu;
```

Instantiate a MPU6050 class object.

```
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

This function is used to obtain the accelerometer and gyro readings of the MPU6050.

```
printf("X:%.2f g Y:%.2f g Z:%.2f g\n", (float)ax/16384, (float)ay/16384, (float)az/16384);
```

ax /16384 is a simplified calculation that maps the reading from the value range to the acceleration range. The complete calculation is ax/65536\*4. Refer to the principle section of this lesson for the relationship between ranges and value range.

```
printf("X:%.2f d/s Y:%.2f d/s Z:%.2f d/s\n", (float)gx/131, (float)gy/131, (float)gz/131);
```

gx /131 is a simplified version of the angular velocity calculation. The complete calculation is gx/65536 \* 500.

```
mpu.initialize();
```

This function is used to initialize the mpu.

```
mpu.testConnection()
```

This function is used to test whether MPU6050 works

### ➤ For Python Language Users

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

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/2.2.6
```

**Step 3:** Run the executable file.

```
sudo python 2.2.6_MP6050.py
```

With the code run, the acceleration/angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

## Code

```

import MPU6050
import time

mpu = MPU6050.MPU6050()    #instantiate a MPU6050 class object
ac = [0]*3      #store accelerometer data
gy = [0]*3      #store gyroscope data

def setup():
    mpu.dmp_initialize()  #initialize MPU6050

def loop():
    while(True):
        ac = mpu.get_acceleration()    #get accelerometer data
        gy = mpu.get_rotation()        #get gyroscope data
        print("Acceleration:\n")
        print("X:%.2f g Y:%.2f g Z:%.2f g\n"%(ac[0]/16384.0,ac[1]/16384.0,ac[2]/16384.0))
        print("Angular velocity:\n")
        print("X:.2f d/s Y:.2f d/s Z:.2f d/s\n%"(gy[0]/131.0,gy[1]/131.0,gy[2]/131.0))
        time.sleep(0.1)

if __name__ == '__main__':  # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed,the program will exit.
        pass

```

## Code Explanation

```
import MPU6050
```

Import the MPU6050 library, the library is used to drive MPU6050 that makes it easy for us to use MPU6050. They read the values of the 3-axis Accelerometer and the 3-axis Gyroscope so that we can calculate the acceleration and angular velocity.

```
mpu = MPU6050.MPU6050()
```

Instantiate a MPU6050 class object.

```
mpu.dmp_initialize()
```

This function is used to initialize the mpu6050.

```
ac = mpu.get_acceleration()  
gy = mpu.get_rotation()
```

These two functions are used to obtain the accelerometer and gyro readings of the MPU6050.

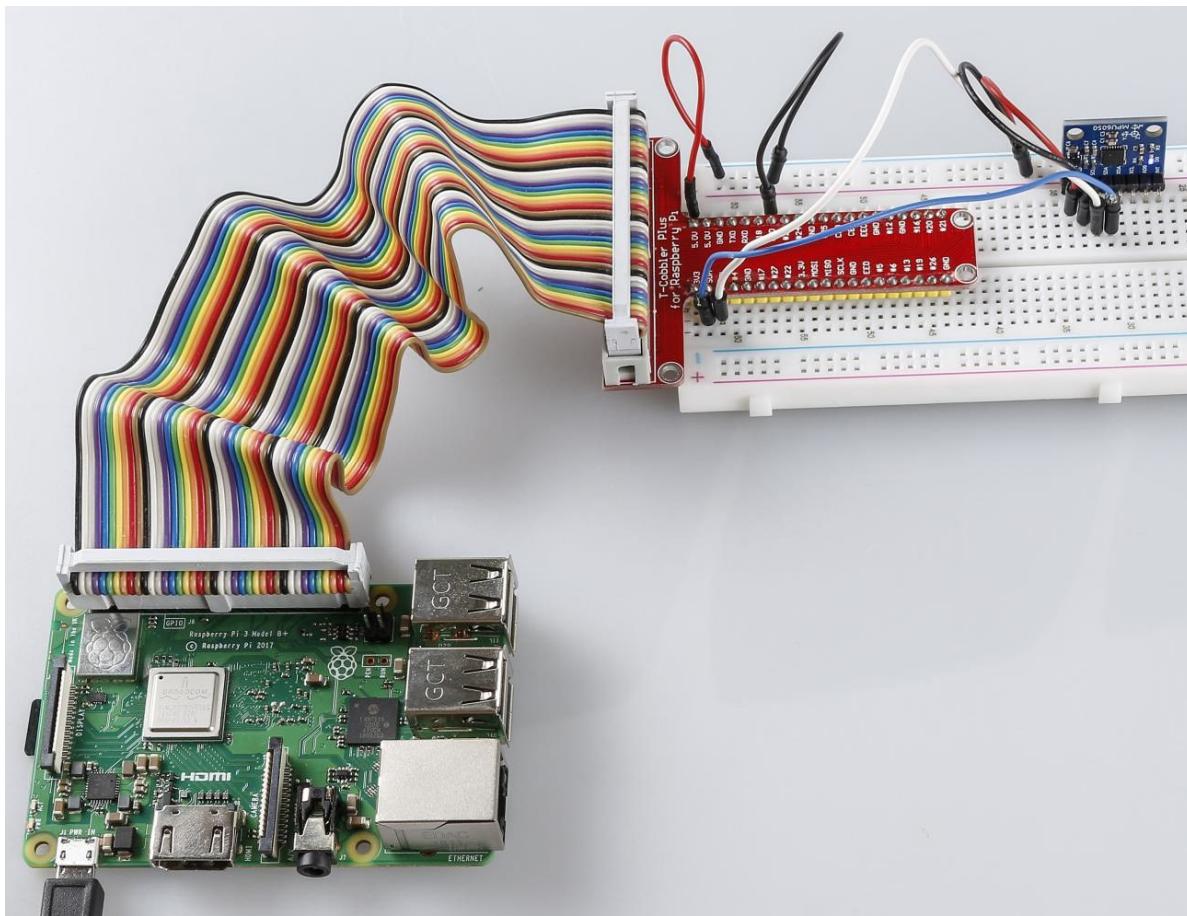
```
print("X:%.2f g Y:%.2f g Z:%.2f g\n"%(ac[0]/16384.0,ac[1]/16384.0,ac[2]/16384.0))
```

ac [0]/16384.0 is a simplified calculation that maps the reading from the value range to the acceleration range. The complete calculation is  $ax/65536*4$ . Refer to the principle section of this lesson for the relationship between ranges and values.

```
print("X:%.2f d/s Y:%.2f d/s Z:%.2f d/s\n"%(gy[0]/131.0,gy[1]/131.0,gy[2]/131.0))
```

gy [0]/131.0 is a simplified version of calculating angular velocity, and the complete calculation is  $gx/65536 * 500$ .

## Phenomenon Picture



## 2.2.7 MFRC522 RFID Module

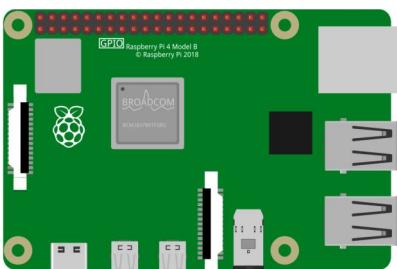
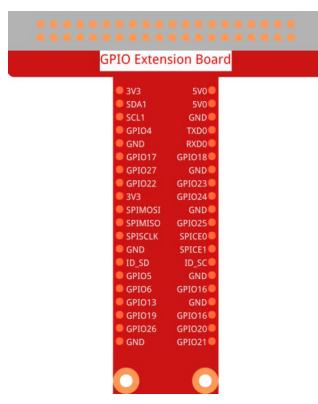
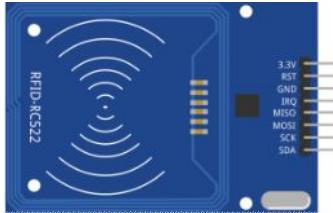
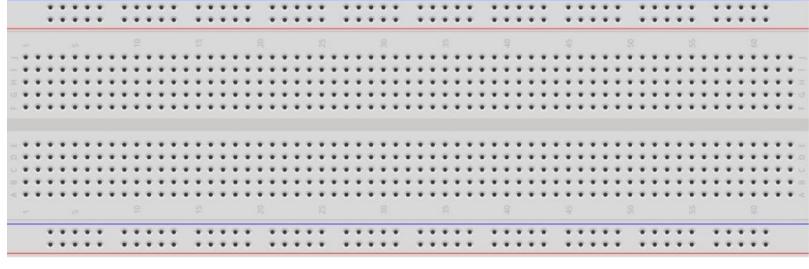
### Introduction

Radio Frequency Identification (RFID) refers to technologies that use wireless communication between an object (or tag) and interrogating device (or reader) to automatically track and identify such objects.

Some of the most common applications for this technology include retail supply chains, military supply chains, automated payment methods, baggage tracking and management, document tracking and pharmaceutical management, to name a few.

In this project, we will use RFID for reading and writing.

### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * RFID RC522 (with white card and key tag)
	 GPIO Extension Board Pinout: ● 3V3 SVO ● SDA1 SVO ● SCL1 GND ● GPIO4 TXD0 ● GND RXD0 ● GPIO17 GPIO18 ● GPIO27 GND ● GPIO22 GPIO23 ● 3V3 GND ● SPI-MOSI GND ● SPI-MISO GND ● SPI-CLK GND ● GND SPI-EI ● ID_SD ID_SC ● GPIO5 GND ● GPIO6 GPIO16 ● GPIO13 GND ● GPIO19 GPIO16 ● GPIO26 GPIO20 ● GND GPIO21	 RFID-RC522 3.3V RST GND IRQ MISO MOSI SCK SDA
Several Jumper Wires		
1 * 40-pin Cable		
1 * Breadboard		

## Principle

### RFID

Radio Frequency Identification (RFID) refers to technologies that involve using wireless communication between an object (or tag) and an interrogating device (or reader) to automatically track and identify such objects. The tag transmission range is limited to several meters from the reader. A clear line of sight between the reader and tag is not necessarily required.

Most tags contain at least one integrated circuit (IC) and an antenna. The microchip stores information and is responsible for managing the radio frequency (RF) communication with the reader. Passive tags do not have an independent energy source and depend on an external electromagnetic signal, provided by the reader, to power their operations. Active tags contain an independent energy source, such as a battery. Thus, they may have increased processing, transmission capabilities and range.



### MFRC522

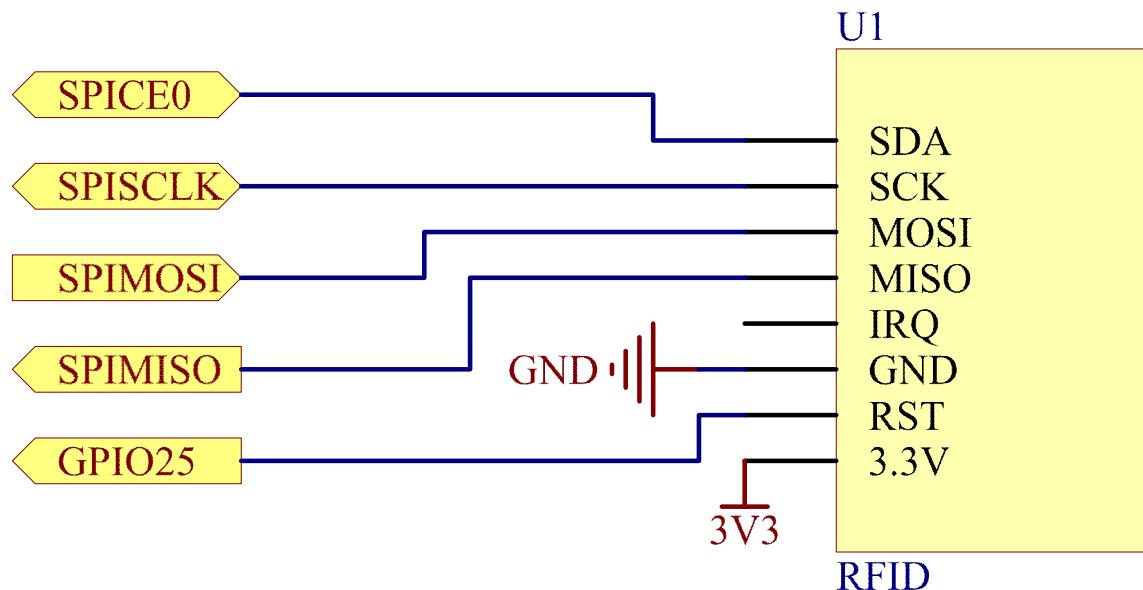
MF RC522 is a highly integrated read and write card chip applied to the 13.56MHz contactless communication. Launched by the NXP Company, it is a low-voltage, low-cost, and small-sized non-contact card chip, a best choice of intelligent instrument and portable handheld device.

The MF RC522 uses advanced modulation and demodulation concept which fully presented in all types of 13.56MHz passive contactless communication methods and protocols. In addition, it supports rapid CRYPTO1 encryption algorithm to verify MIFARE products. MFRC522 also supports MIFARE series of high-speed non-contact communication, with a two-way data transmission rate up to 424kbit/s. As a new member of the 13.56MHz highly integrated reader card series, MF RC522 is much similar to the existing MF RC500 and MF RC530 but there also exists great

differences. It communicates with the host machine via the serial manner which needs less wiring. You can choose between SPI, I2C and serial UART mode (similar to RS232), which helps reduce the connection, save PCB board space (smaller size), and reduce cost.

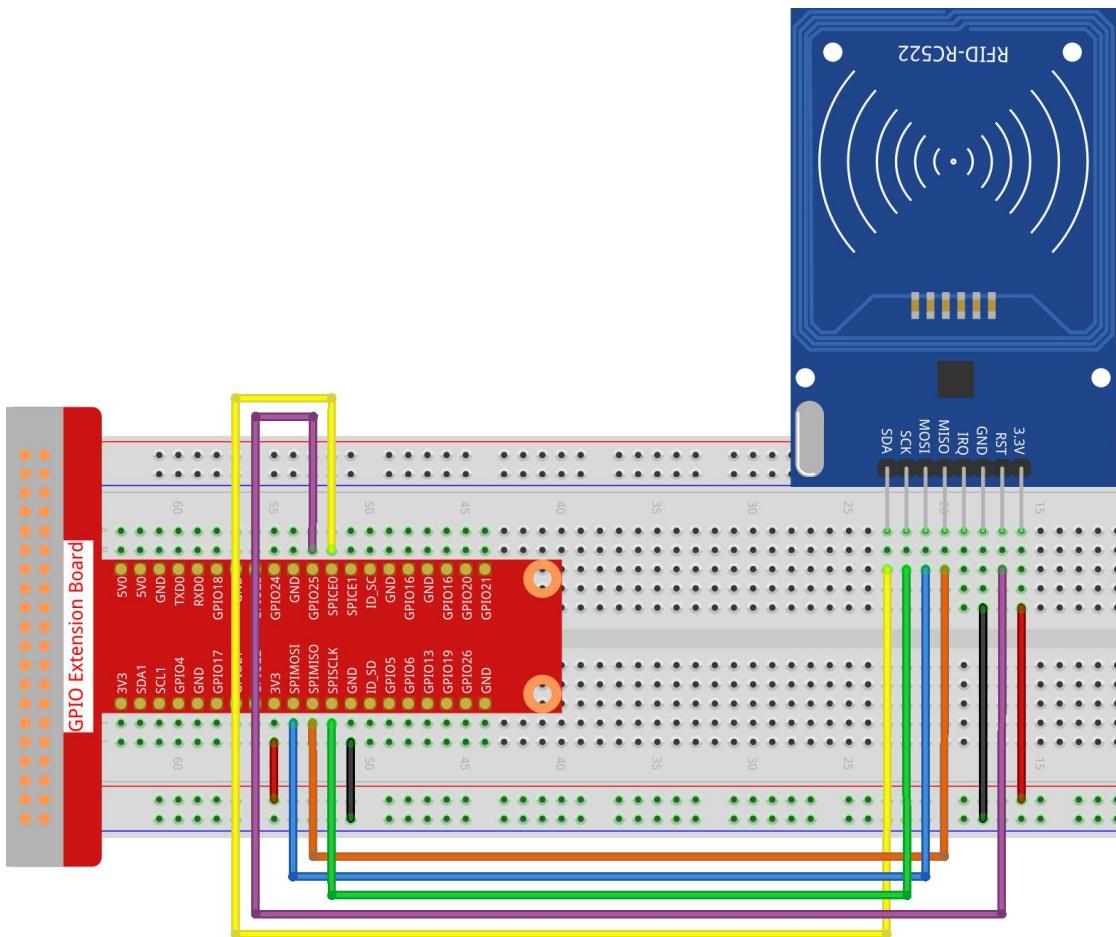
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
CE0	Pin 24	10	8
SCLK	Pin 23	14	11
SPIMOSI	Pin 19	12	10
SPIMISO	Pin 21	13	9
GPIO25	Pin 22	6	25



## Experimental Procedures

**Step 1:** Build the circuit.



**Step 2:** Set up SPI (refer to Appendix for more details. If you have set SPI, skip this step.)

### ➤ For C Language Users

**Step 3:** Go to the folder of the code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.7/
```

**Step 4:** Compile the code.

```
gcc *.c -lwiringPi
```

**Note:** The program includes mfrc522.c, mfrc522\_debug.c, mfrc522.c\_hal\_linux.c, dump.c, 2.2.7\_RFID.c and so on. We use the wildcard (\*), which causes all the files that conform to the format to be processed together.

**Step 5:** Run the executable file.

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

When the program runs, terminal will pop up to notice you to print "Scanning Card...". At this time, we put the query object (matching tag or card) close to the reader (RFID-RC522) module, and then we can read the data inside.

```
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/2.2.7 $ sudo ./a.out
Device Number:3
Scanning Card...
```

When the query object is read, the following codes are printed.

```
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/C/2.2.7 $ sudo ./a.out
Device Number:3
Scanning Card...
Reading card 64002896
    0: 53 75 6e 66 6f 75 6e 64 65 72 00 00 00 00 00 00 : Sunfounder.....
    16: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
Input 'write' or 'clean' to modify the data
>
```

Reading card 64002896, here the number represents global unique identifier number of the query object(USN/UID).

The middle two lines of characters are the data stored in the query object. If the datum stored in the card is "Sunfounder", the hexadecimal number "53 75..." are the ASCII values of the data.

We can modify the data stored on the card by typing "write" or "clean".

```
Input 'write' or 'clean' to modify the data
>clean

Reading card 64002896
    0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
    16: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
Input 'write' or 'clean' to modify the data
>

Input 'write' or 'clean' to modify the data
>write

Input the Data (less than 16 character)
>sunfounder
Reading card 64002896
    0: 73 75 6e 66 6f 75 6e 64 65 72 00 00 00 00 00 00 : sunfounder.....
    16: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
Input 'write' or 'clean' to modify the data
>
```

## Code

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include <stdlib.h>
#include "mfrc522.h"

int scan_loop(uint8_t *CardID);
int tag_select(uint8_t *CardID);

int main(int argc, char **argv) {
    MFRC522_Status_t ret;
    uint8_t CardID[5] = { 0x00, };
    uint8_t tagType[16] = {0x00,};
    static char command_buffer[1024];

    ret = MFRC522_Init('B');
    if (ret < 0) {
        perror("Failed to initialize");
        exit(-1);
    }

    while (1) {
        puts("Scanning Card... ");
        while (1) {
            ret = MFRC522_Request(PICC_REQIDL, tagType);
            if (ret == MI_OK) {
                ret = MFRC522_Anticoll(CardID);
                if(ret == MI_OK){
                    ret = tag_select(CardID);
                    if (ret == MI_OK) {
                        ret = scan_loop(CardID);
                        if (ret < 0) {
                            printf("Error...\r\n");
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

```

        else{
            printf("Get Card ID failed!\r\n");
        }
    }
    MFRC522_Halt();
}
MFRC522_Halt();
MFRC522_Init('B');

}

int scan_loop(uint8_t *CardID) {
    while (1) {
        char input[32];
        int block_start=2;
        printf("Reading card %02X%02X%02X%02X \n", CardID[0], CardID[1], CardID[2],
CardID[3]);
        MFRC522_Debug_DumpSector(CardID, block_start);
        printf("Input 'write' or 'clean' to modify the data\n >");
        scanf("%s", input);
        printf("\n");
        if(strcmp(input, "clean") == 0){
            getchar();
            if (MFRC522_Debug_Clean(CardID, block_start)) {
                return -1;
            }
        } else if (strcmp(input, "write") == 0) {
            char write_buffer[256];
            size_t len = 0;
            getchar();
            printf("Input the Data (less than 16 character)\n > ");
            scanf("%s", write_buffer);
            if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
                strlen(write_buffer)) < 0) {
                return -1;
            }
        }
    }
    return 0;
}

```

```
int tag_select(uint8_t *CardID) {
    int ret_int;
    ret_int = MFRC522_SelectTag(CardID);
    if (ret_int == 0) {
        printf("Card Select Failed\r\n");
        return -1;
    }
    ret_int = 0;
    return ret_int;
}
```

## Code Explanation

```
#include "mfrc522.h"
```

This program is based on the open source file mfrc522, which helps us to use the RFID module easily.

```
while (1) {
    ret = MFRC522_Request(PICC_REQIDL, tagType);
    if (ret == MI_OK) {
        ret = MFRC522_Anticoll(CardID);
        if(ret == MI_OK){
            ret = tag_select(CardID);
            if (ret == MI_OK) {
                ret = scan_loop(CardID);
                if (ret < 0) {
                    printf("Error...\r\n");
                    break;
                }
            }
        }
    }
}
```

The nested judgment here is used to read the query object.

**ret = MFRC522\_Request(PICC\_REQIDL, tagType):** That detecting the presence of objects is a function in the MFRC522 file.

**ret = MFRC522\_Anticoll(CardID):** It is the function in MFRC522 that asking whether the query object is available and getting the CardID of the object.

**ret = tag\_select(CardID):** It is used to select the object, which is a process of calling the function, MFRC522\_SelectTag(CardID) in the MFRC522 file.

**ret = scan\_loop(CardID):** The actual reading and writing of the object is realized by this means.

```
int scan_loop(uint8_t *CardID) {
    while (1) {
        char input[32];
        int block_start=2;
        printf("Reading card %02X%02X%02X%02X \n", CardID[0], CardID[1], CardID[2],
CardID[3]);
        MFRC522_Debug_DumpSector(CardID, block_start);
        printf("Input 'write' or 'clean' to modify the data\n >");
        scanf("%s", input);
        printf("\n");
        if(strcmp(input, "clean") == 0){
            getchar();
            if (MFRC522_Debug_Clean(CardID, block_start)) {
                return -1;
            }
        } else if (strcmp(input, "write") == 0) {
            char write_buffer[256];
            size_t len = 0;
            getchar();
            printf("Input the Data (less than 16 character)\n > ");
            scanf("%s", write_buffer);
            if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
                strlen(write_buffer)) < 0) {
                return -1;
            }
        }
        return 0;
    }
}
```

This function displays the stored data and modifies them based on the input instructions.

```
char input[32];
```

The variable input is used to get the input instructions from the keyboard.

```
int block_start=2;
```

block\_start is a storage address of the query object that corresponds to the initial position of every block. Tag and card generally have 64 blocks (0~63). Here, we conduct the experiment with block<2>.

Each block can store 16 bytes of data. If you have larger data storage requirements, you can rewrite the code to assign a value to the variable block\_start.

Note that four blocks together form one sector and the fourth block of each sector is a control module (eg. sector<0>block<3>). In general, we do not recommend that it be modified in that improper operation may render permanent unavailability of sectors.

In addition, the UID stored in sector<0>block<0> is fixed and cannot be modified.

```
MFRC522_Debug_DumpSector(CardID, block_start);
```

This function is used to display the data stored by the query object. It displays the data stored from the block\_start to the end position of sector.

```
if(strcmp(input, "clean") == 0){
    getchar();
    if (MFRC522_Debug_Clean(CardID, block_start)) {
        return -1;
    }
}
```

When the directive "clean" is received, the data of the corresponding block are cleaned up.

```
if (strcmp(input, "write") == 0) {
    char write_buffer[256];
    size_t len = 0;
    getchar();
    printf("Input the Data (less than 16 character)\n");
    scanf("%s", write_buffer);
    if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
        strlen(write_buffer)) < 0) {
        return -1;
    }
}
```

When the "write" instruction is received, write data to the corresponding blocks.

## ➤ For Python Language Users

**Step 3:** Go to the folder of the code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/python/2.2.7
```

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

```
sudo python 2.27_RFID.py
```

When the program runs, terminal will pop up to notice you to print "Scanning Card...". At this time, we put the query object (matching tag or card) close to the reader (RFID-RC522) module, and then we can read the data inside.

```
pi@raspberrypi:~/SunFounder_Da_Vinci_Kit_for_Raspberry_Pi/python/2.2.7 $ python3 2.2.7_RFID.py
Program is starting ...
Press Ctrl-C to exit.

Scanning Card...
```

When the query object is read, the following codes are printed.

```
Scanning Card...
Reading card 64 02896DA >

Block 2 > sunfounder

Input'write' or 'clean' to modify the data
>
```

The code "Reading card 64002896DA" represents global unique identifier number (USN/UID) of inquiry objects, The middle characters are the data stored in the query object. If the datum stored in this card is "sunfounder", the "Block 2" in front represents the current position of data reading.

We can modify the data stored on the card by typing "write" or "clean".

```
Input'write' or 'clean' to modify the data
>write

Input the Data (less than 16 character)
>Sunfounder

Input'write' or 'clean' to modify
>clean
Data written
Reading card 64 02896DA >

Block 2 > Sunfounder

Input'write' or 'clean' to modify the data
>
```

## Code

```

import RPi.GPIO as GPIO
import MFRC522
import sys
import os

mfrc = MFRC522.MFRC522()

def setup():
    print ("Program is starting ... ")
    print ("Press Ctrl-C to exit.\n")
    pass

def loop():
    global mfrc
    while(True):
        print ("Scanning Card... ")
        mfrc = MFRC522.MFRC522()
        isScan = True
        while isScan:
            (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
            (status,uid) = mfrc.MFRC522_Anticoll()
            if status == mfrc.MI_OK:
                if mfrc.MFRC522_SelectTag(uid) == 0:
                    print ("MFRC522_SelectTag Failed!")
                if cmdloop(uid) < 1 :
                    isScan = False

def cmdloop(cardID):
    blockAddr = 2
    while(True):
        print ("Reading card %2X%2X%2X%2X%2X >
\n"%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]))
        key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
        status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
        if status == mfrc.MI_OK:
            mfrc.MFRC522_Read(blockAddr)
            print(" ")
        else:
            print ("Authentication error")

```

```

    return 0

    print ("\nInput'write' or 'clean' to modify the data\n> ",end="")
    inCmd = input()
    cmd = inCmd.split(" ")

    if cmd[0] == "write":
        print ("\nInput the Data (less than 16 character)\n> ",end="")
        inCmd = input()
        cmd = inCmd.split(" ")
        data = [0]*16
        data = cmd[0][0:17]
        data = map(ord,data)
        data = list(data)
        lenData = len(list(data))
        if lenData<16:
            data+=[0]*(16-lenData)
        key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
        status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
        if status == mfrc.MI_OK:
            mfrc.MFRC522_Write(blockAddr, data)
        else:
            print ("Authentication error")
            return 0

    elif cmd[0] == "clean":
        data = [0]*16
        key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
        status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
        if status == mfrc.MI_OK:
            mfrc.MFRC522_Write(blockAddr, data)
        else:
            print ("Authentication error")
            return 0
    else :
        return 0

def destroy():
    GPIO.cleanup()

if __name__ == "__main__":

```

```
setup()
try:
    loop()
except KeyboardInterrupt: # Ctrl+C captured, exit
    destroy()
```

## Code Explanation

```
import MFRC522
```

This program is based on the open source file mfrc522, which helps us to use the RFID module easily.

```
mfrc = MFRC522.MFRC522()
isScan = True
while isScan:
    (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
    (status,uid) = mfrc.MFRC522_Anticoll()
    if status == mfrc.MI_OK:
        if mfrc.MFRC522_SelectTag(uid) == 0:
            print ("MFRC522_SelectTag Failed!")
        if cmdloop(uid) < 1 :
            isScan = False
```

The nested judgment here is used to read the query object.

```
(status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
```

That detecting the presence of objects is a function in the MFRC522 file.

```
(status,uid) = mfrc.MFRC522_Anticoll()
```

The function in the file, MFRC522 is to judge whether the inquiry object is available and to get the UID of objects.

```
mfrc.MFRC522_SelectTag(uid)
```

It is used to select the object, which is a process of calling the function, MFRC522\_SelectTag(CardID) in the MFRC522 file.

```
cmdloop(uid)
```

The actual reading and writing of the object is realized by this means.

```

def cmdloop(cardID):
    blockAddr = 2
    while(True):
        print ("Reading card %2X%2X%2X%2X%2X >
\n"%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]))
        key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
        status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
        if status == mfrc.MI_OK:
            mfrc.MFRC522_Read(blockAddr)
            print(" ")
        else:
            print ("Authentication error")
            return 0
        print ("\nInput'write' or 'clean' to modify the data\n> ",end="")
        inCmd = input()
        cmd = inCmd.split(" ")

        if cmd[0] == "write":
            print ("\nInput the Data (less than 16 character)\n> ",end="")
            inCmd = input()
            cmd = inCmd.split(" ")
            data = [0]*16
            data = cmd[0][0:17]
            data = map(ord,data)
            data = list(data)
            lenData = len(list(data))
            if lenData<16:
                data+=[0]*(16-lenData)
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            if status == mfrc.MI_OK:
                mfrc.MFRC522_Write(blockAddr, data)
            else:
                print ("Authentication error")
                return 0

        elif cmd[0] == "clean":
            data = [0]*16

```

```

key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
if status == mfrc.MI_OK:
    mfrc.MFRC522_Write(blockAddr, data)
else:
    print ("Authentication error")
    return 0
else :
    return 0

```

This function displays the stored data and modifies them based on the input instructions.

blockAddr = 2

The “blockAddr” is a storage address of the query object that corresponds to the initial position of every block. Tag and card generally have 64 blocks (0~63). Here, we conduct the experiment with block<2>.

Each block can store 16 bytes of data. If you have larger data storage requirements, you can rewrite the code to assign a value to the variable block\_start.

Note that four blocks together form one sector and the fourth block of each sector is a control module (eg. sector<0>block<3>) In general, we do not recommend that it be modified in that improper operation may render permanent unavailability of sectors.

In addition, the UID stored in sector<0>block<0> is fixed and cannot be modified.

```

status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
if status == mfrc.MI_OK:
    mfrc.MFRC522_Read(blockAddr)
    print(" ")
else:
    print ("Authentication error")
    return 0

```

This function is used to display the data stored by the query object. It displays the data stored from the block\_start to the end position of sector.

```

inCmd = input()
cmd = inCmd.split(" ")

```

It is the instruction used to obtain keyboard input.

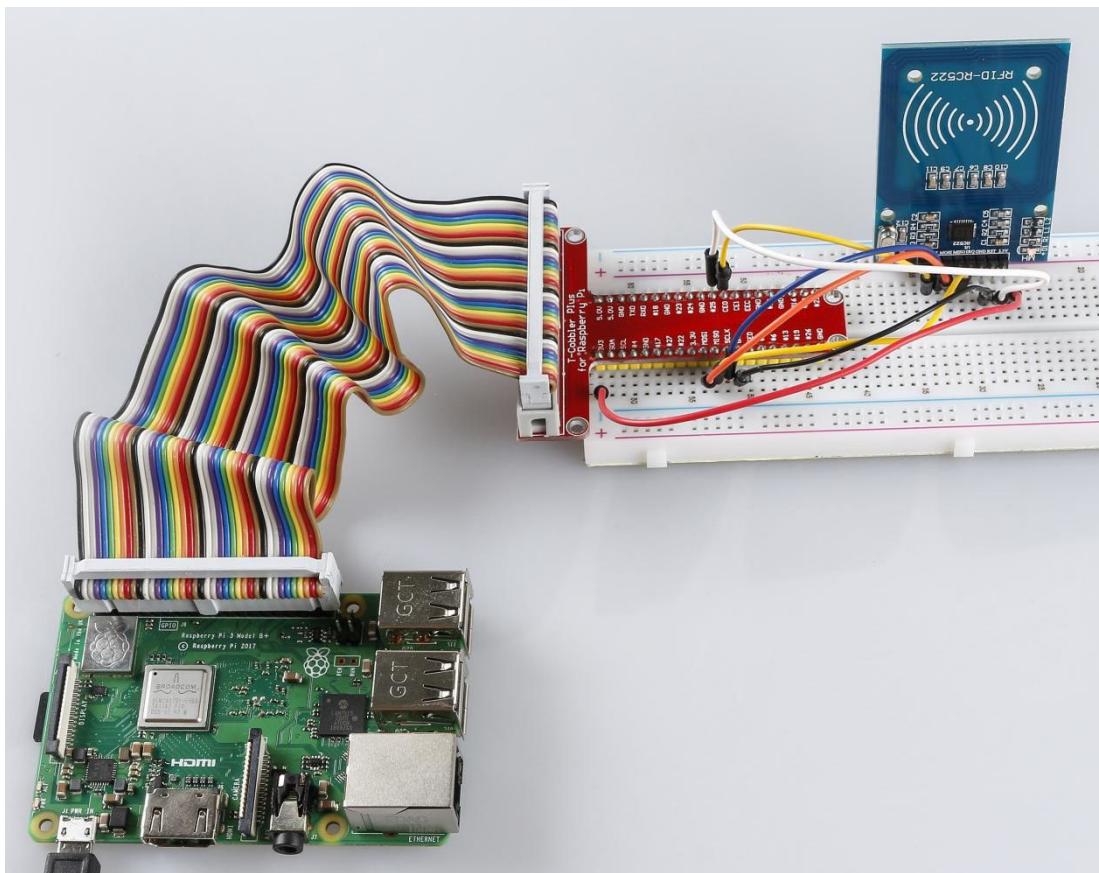
```
if cmd[0] == "clean":
    data = [0]*16
    key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
    status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
    if status == mfrc.MI_OK:
        mfrc.MFRC522_Write(blockAddr, data)
    else:
        print ("Authentication error")
        return 0
```

When the directive "clean" is received, the data of the corresponding blocks are cleaned up.

```
if cmd[0] == "write":
    print ("\nInput the Data (less than 16 character)\n>",end="")
    inCmd = input()
    cmd = inCmd.split(" ")
    data = [0]*16
    data = cmd[0][0:17]
    data = map(ord,data)
    data = list(data)
    lenData = len(list(data))
    if lenData<16:
        data+=[0]*(16-lenData)
    key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
    status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
    if status == mfrc.MI_OK:
        mfrc.MFRC522_Write(blockAddr, data)
    else:
        print ("Authentication error")
        return 0
```

When the instruction "write" is received, write data to the corresponding block.

## Phenomenon Picture

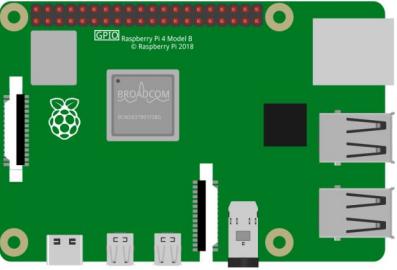
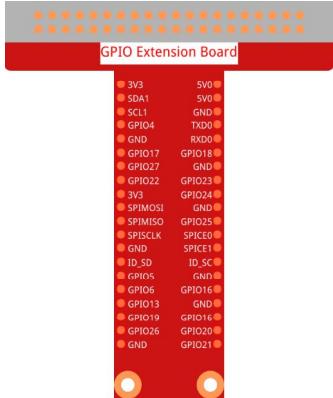


## 2.2.8 Tilt Switch

### Introduction

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

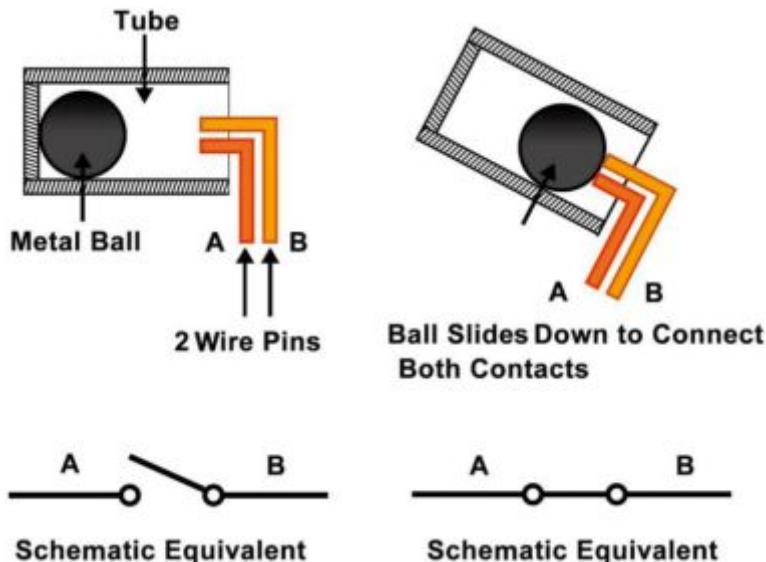
### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Tilt	2 * LED
	 GPIO Extension Board Pinout: 3V3 SDA1 SDA1 SCL1 GND GND GPI04 TXD0 TXD0 GND RXD0 RXD0 GPI017 GPIO18 GPIO18 GPI027 GND GND GPI022 GPIO23 GPIO23 3V3 GPIO24 GPIO24 SPIMOSI GND GND SPIMISO GPIO25 GPIO25 SPISCLK SPICE0 SPICE0 GND SPICE1 SPICE1 ID_SD ID_SD ID_SD GPI05 GND GND GPI06 GPIO16 GPIO16 GPI013 GND GND GPI019 GPIO16 GPIO16 GPI026 GPIO20 GPIO20 GND GPIO21 GPIO21		
1 * 40-pin Cable	Several Jumper Wires		
1 * Breadboard	2 * Resistor(220Ω)		
	1 * Resistor 10KΩ		

## Principle

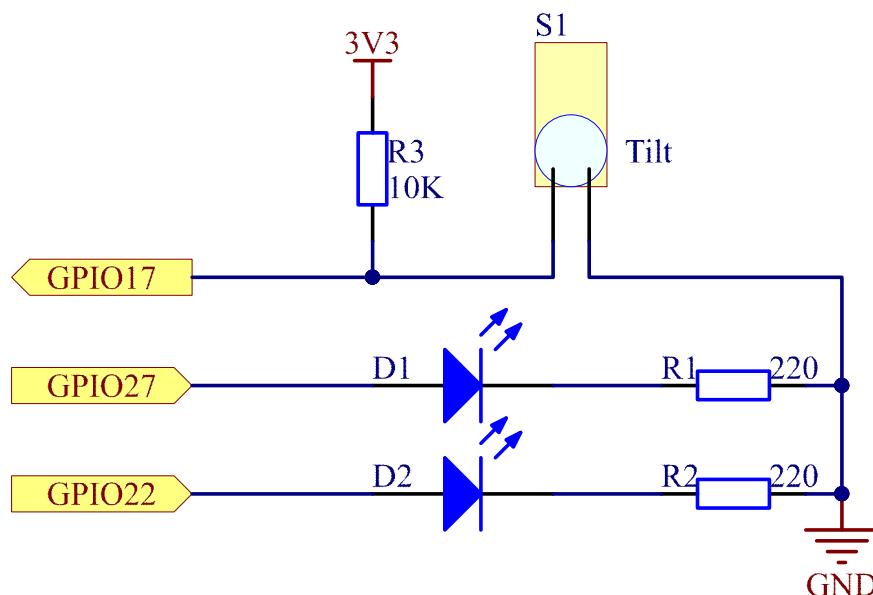
### Tilt

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



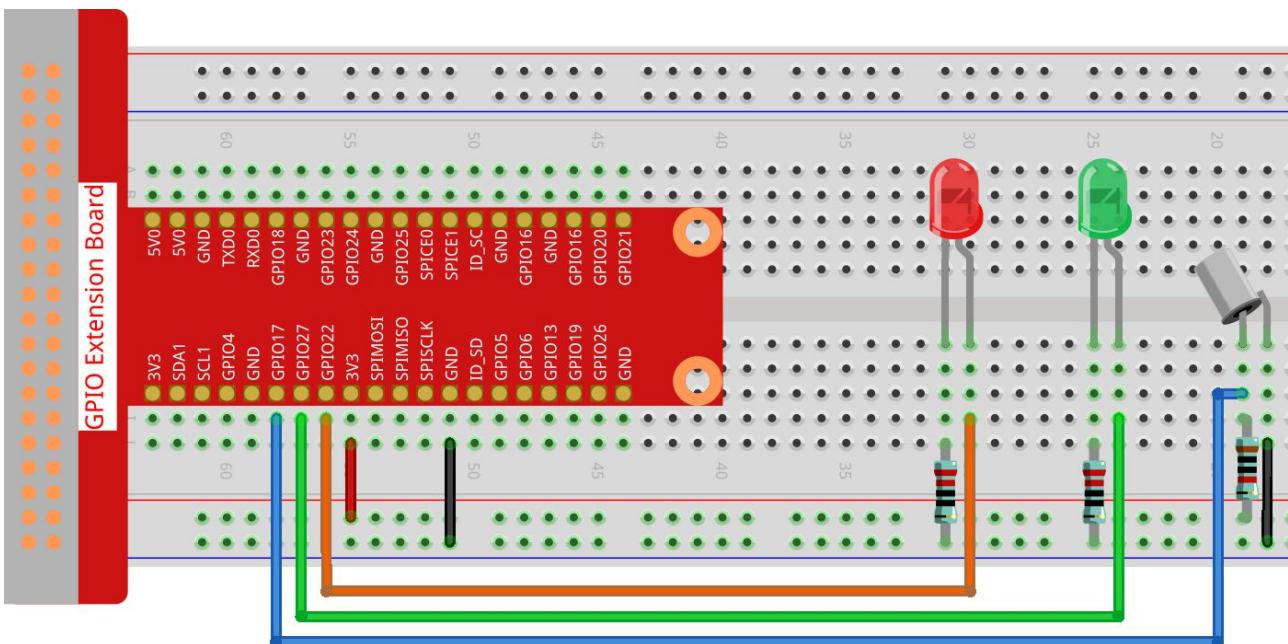
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

#### Step 2: Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.2.8/
```

#### Step 3: Compile.

```
gcc 2.2.8_Tilt.c -lwiringPi
```

#### Step 4: Run.

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

Place the tilt horizontally, and the green LED will turn on. If you tilt it, "Tilt!" will be printed on the screen and the red LED will light up. Place it horizontally again, and the green LED will turn on again.

### Code

```
#include <wiringPi.h>
#include <stdio.h>

#define TiltPin    0
#define Gpin       2
#define Rpin       3

void LED(char* color)
```

```

{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == "RED")
    {
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}

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

    pinMode(TiltPin, INPUT);
    LED("GREEN");

    while(1){
        if(0 == digitalRead(TiltPin)){
            delay(10);
            if(0 == digitalRead(TiltPin)){
                LED("RED");
                printf("Tilt!\n");
            }
        }
        else if(1 == digitalRead(TiltPin)){
            delay(10);
            if(1 == digitalRead(TiltPin)){
                LED("GREEN");
            }
        }
    }
}

```

```

    }
}

return 0;
}
}
```

## Code Explanation

```

void LED(char* color)
{
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);
    if (color == "RED")
    {
        digitalWrite(Rpin, HIGH);
        digitalWrite(Gpin, LOW);
    }
    else if (color == "GREEN")
    {
        digitalWrite(Rpin, LOW);
        digitalWrite(Gpin, HIGH);
    }
    else
        printf("LED Error");
}
```

Define a function LED () to turn the two LEDs on or off. If the parameter color is RED, the red LED lights up; similarly, if the parameter color is GREEN, the green LED will turns on.

```

while(1){
    if(0 == digitalRead(TiltPin)){
        delay(10);
        if(0 == digitalRead(TiltPin)){
            LED("RED");
            printf("Tilt!\n");
        }
    }
    else if(1 == digitalRead(TiltPin)){
        delay(10);
        if(1 == digitalRead(TiltPin)){
```

```

        LED("GREEN");
    }
}
}
```

If the read value of tilt switch is 0, it means that the tilt switch is tilted then you write the parameter "RED" into function LED to get the red LED lighten up; otherwise, the green LED will lit.

## ➤ For Python Language Users

**Step 2:** Change directory.

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

**Step 3:** Run.

```
sudo python 2.2.8_Tilt.py
```

Place the tilt horizontally, and the green LED will turns on. If you tilt it, "Tilt!" will be printed on the screen and the red LED will turns on. Place it horizontally again, and the green LED will lights on.

## Code

```

import RPi.GPIO as GPIO

TiltPin = 11
Gpin  = 13
Rpin  = 15

def setup():
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(Gpin, GPIO.OUT)     # Set Green Led Pin mode to output
    GPIO.setup(Rpin, GPIO.OUT)     # Set Red Led Pin mode to output
    GPIO.setup(TiltPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set BtnPin's mode is input,
and pull up to high level(3.3V)
    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
```

```

GPIO.output(Rpin, 0)
GPIO.output(Gpin, 1)

def Print(x):
    if x == 0:
        print (' ****')
        print (' * Tilt! *')
        print (' ****')

def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.HIGH)      # Green led off
    GPIO.output(Rpin, GPIO.HIGH)      # Red led off
    GPIO.cleanup()                  # Release resource

if __name__ == '__main__':  # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
        destroy()

```

## Code Explanation

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

Set up a detect on TiltPin, and callback function to detect.

```

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)

```

```
if x == 1:  
    GPIO.output(Rpin, 0)  
    GPIO.output(Gpin, 1)
```

Define a function Led() to turn the two LEDs on or off. If x=0, the red LED lights up; otherwise, the green LED will be lit.

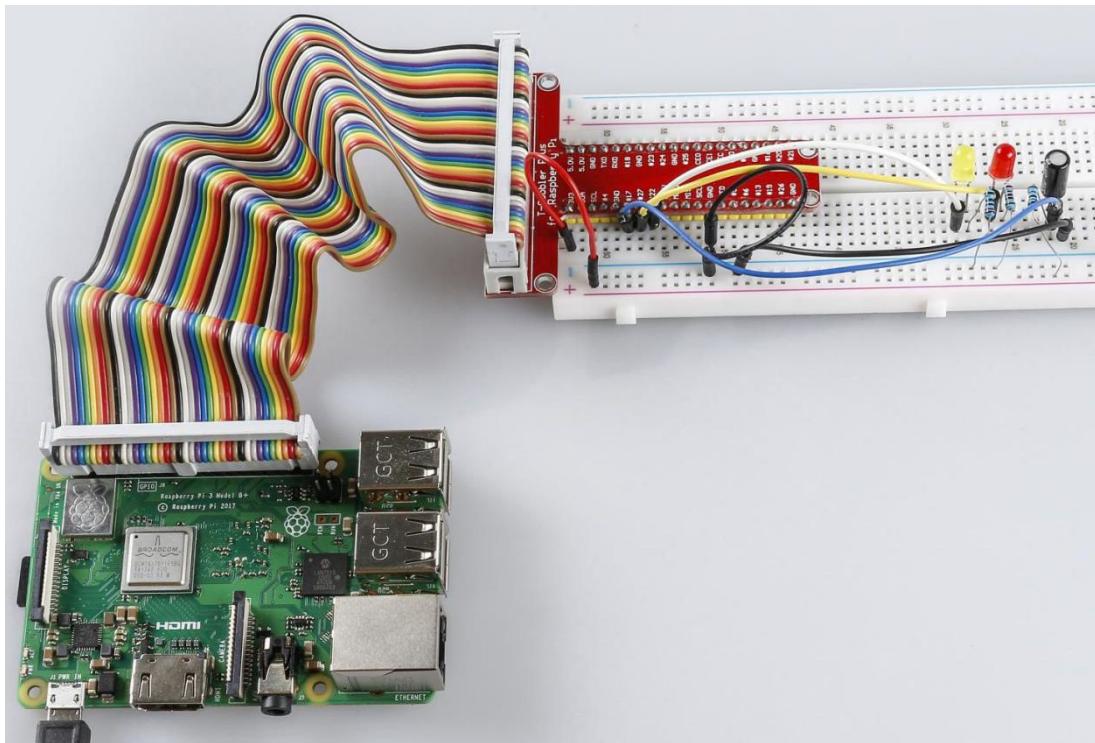
```
def Print(x):  
    if x == 0:  
        print (' ****' * 10)  
        print (' * Tilt! *')  
        print (' ****' * 10)
```

Create a function, Print() to print the characters above on the screen.

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

Define a callback function for tilt callback. Get the read value of the tilt switch then the function Led () controls the turning on or off of the two LEDs that is depended on the read value of the tilt switch.

## Phenomenon Picture



# 3 Extension

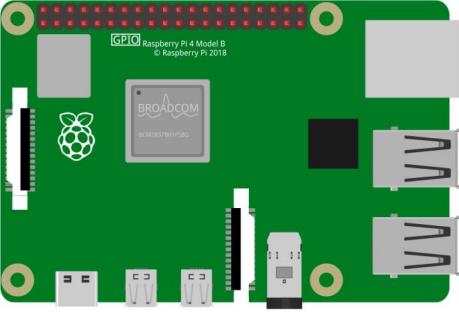
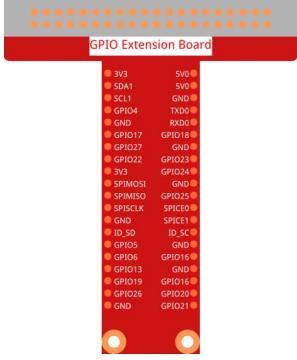
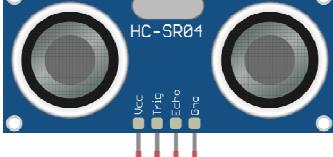
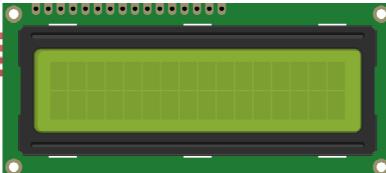
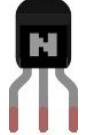
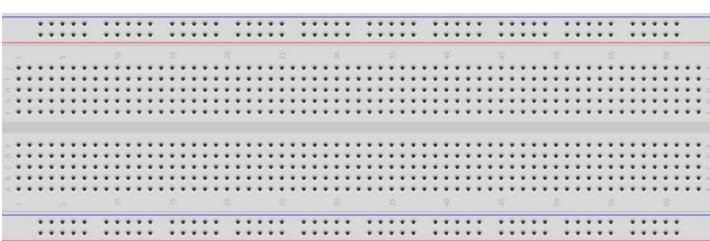
## 3.1 Application

### 3.1.1 Reversing Alarm

#### Introduction

In this project, we will use LCD, buzzer and ultrasonic sensors to make a reverse assist system. We can put it on the remote control vehicle to simulate the actual process of reversing the car into the garage.

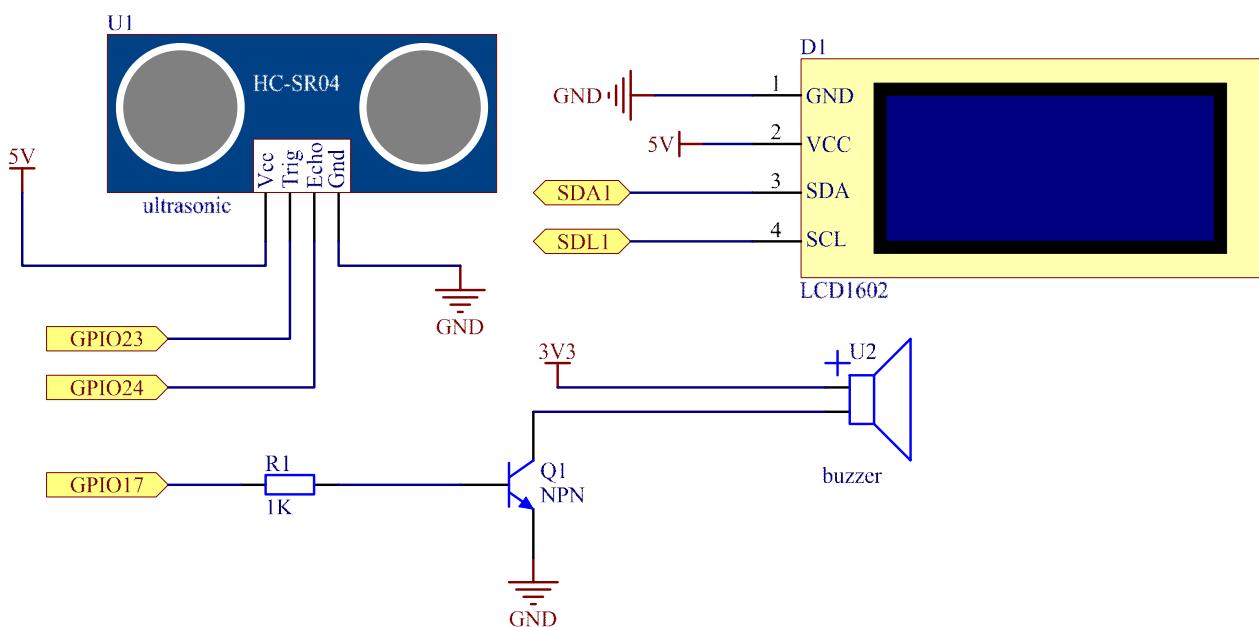
#### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * Active Buzzer
	 GPIO Extension Board Pinout: 3V3 SDA1 SVO0 SCL1 GND1 GPIO4 TXD00 GND RXD00 GPIO17 GPIO18 GPIO27 GPIO23 GPIO22 GPIO24 3V3 GND SPIMOSI GND SPIMOSO GPIO25 SPISCLK SPICE00 GND SPICE11 GPIO25 GND0 GPIO26 GND1 GPIO6 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO1026 GPIO20 GND GPIO21	
1 * HC SR04	1 * I2C LCD1602	1 * NPN transistor
		
1 * 40-pin Cable		Several Jumper Wires
		
1 * Breadboard		1 * Resistor(1kΩ)
		

## Schematic Diagram

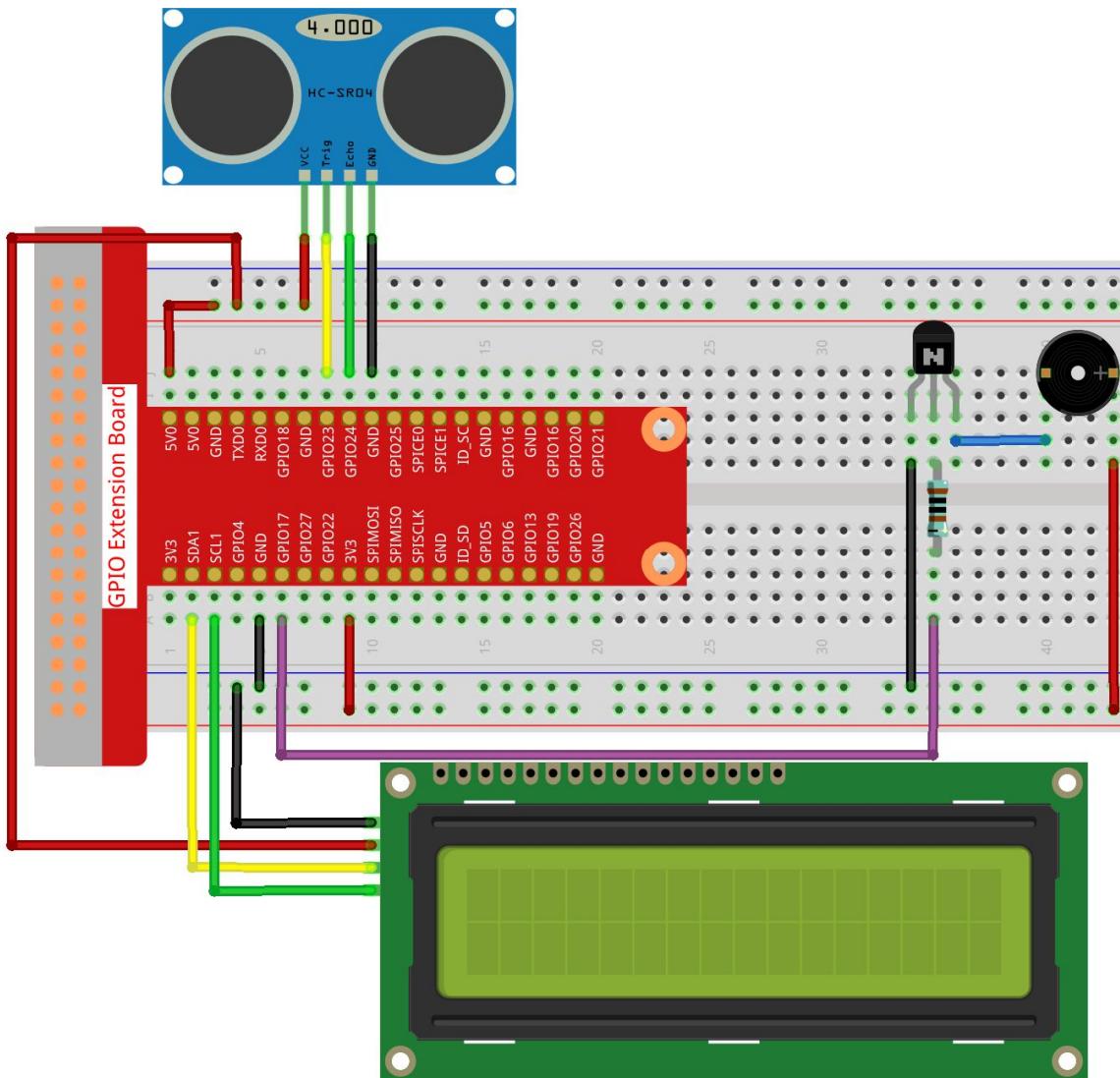
Ultrasonic sensor detects the distance between itself and the obstacle that will be displayed on the LCD in the form of code. At the same time, the ultrasonic sensor let the buzzer issue prompt sound of different frequency according to different distance value.

T-Board Name	physical	wiringPi	BCM
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO17	Pin 11	0	17
SDA1	Pin 3		
SCL1	Pin 5		



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

#### Step 2: Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.1/
```

#### Step 3: Compile.

```
gcc 3.1.1_ReversingAlarm.c -lwiringPi
```

#### Step 4: Run.

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

As the code runs, ultrasonic sensor module detects the distance to the obstacle and then displays the information about the distance on LCD1602; besides, buzzer emits warning tone whose frequency changes with the distance.

## Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.1\_ReversingAlarm.c.

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <string.h>

#define Trig 4
#define Echo 5
#define Buzzer 0

int LCDAddr = 0x27;
int BLEN = 1;
int fd;

//here is the function of LCD
void write_word(int data){.....}

void send_command(int comm){.....}

void send_data(int data){.....}

void lcdInit(){.....}

void clear(){.....}

void write(int x, int y, char data[]){.....}

//here is the function of Ultrasonic
void ultralnit(void){.....}

float disMeasure(void){.....}

//here is the main function
int main(void)
{
```

```

float dis;
char result[10];
if(wiringPiSetup() == -1){
    printf("setup wiringPi failed !");
    return 1;
}

pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();

clear();
write(0, 0, "Ultrasonic Starting");
write(1, 1, "By Sunfounder");

while(1){
    dis = disMeasure();
    printf("%.2f cm \n",dis);
    digitalWrite(Buzzer,LOW);
    if (dis > 400){
        clear();
        write(0, 0, "Error");
        write(3, 1, "Out of range");
        delay(500);
    }
    else
    {
        clear();
        write(0, 0, "Distance is");
        sprintf(result, "%.2f cm",dis);
        write(5, 1, result);

        if(dis>=50)
        {delay(500);}
        else if(dis<50 & dis>20) {
            for(int i=0;i<2;i++){
                digitalWrite(Buzzer,HIGH);
                delay(50);
                digitalWrite(Buzzer,LOW);
            }
        }
    }
}

```

```

        delay(200);
    }
}

else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}

return 0;
}

```

## Code Explanation

```

pinMode(Buzzer,OUTPUT);
fd = wiringPiI2CSetup(LCDAddr);
lcdInit();
ultraInit();

```

In this program, we apply previous components synthetically. Here we use buzzers, LCD and ultrasonic. We can initialize them the same way as we did before.

```

dis = disMeasure();
printf("%.2f cm \n",dis);
digitalWrite(Buzzer,LOW);
if (dis > 400){
    write(0, 0, "Error");
    write(3, 1, "Out of range");
}
else
{
    write(0, 0, "Distance is");
    sprintf(result, "%.2f cm",dis);
    write(5, 1, result);
}

```

Here we get the value of the ultrasonic sensor and get the distance through calculation.

If the value of distance is greater than the range value to be detected, an error message is printed on the LCD. And if the distance value is within the range, the corresponding results will be output.

```
sprintf(result,"%.2f cm",dis);
```

Since the output mode of LCD only supports character type, and the variable dis stores the value of float type, we need to use sprintf(). The function converts the float type value to a character and stores it on the string variable result[]. %.2f means to keep two decimal places.

```
if(dis>=50)
{delay(500);}
else if(dis<50 & dis>20) {
    for(int i=0;i<2;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(200);
    }
}
else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}
```

This judgment condition is used to control the sound of the buzzer. According to the difference in distance, it can be divided into three cases, in which there will be different sound frequencies. Since the total value of delay is 500, all of the cases can provide a 500ms interval for the ultrasonic sensor.

## ➤ For Python Language Users

**Step 2:** Change directory.

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

**Step 3:** Run.

```
sudo python 3.1.1_ReversingAlarm.py
```

As the code runs, ultrasonic sensor module detects the distance to the obstacle and then displays the information about the distance on LCD1602; besides, buzzer emits warning tone whose frequency changes with the distance.

### Code

```
import LCD1602
import time
import RPi.GPIO as GPIO

TRIG = 16
ECHO = 18
BUZZER = 11

def lcdsetup():
    LCD1602.init(0x27, 1) # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'Ultrasonic Starting')
    LCD1602.write(1, 1, 'By SunFounder')
    time.sleep(2)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
    lcdsetup()

def distance():
    GPIO.output(TRIG, 0)
    time.sleep(0.000002)

    GPIO.output(TRIG, 1)
    time.sleep(0.00001)
```

```

GPIO.output(TRIG, 0)

while GPIO.input(ECHO) == 0:
    a = 0
time1 = time.time()
while GPIO.input(ECHO) == 1:
    a = 1
time2 = time.time()

during = time2 - time1
return during * 340 / 2 * 100

def destroy():
    GPIO.output(BUZZER, GPIO.LOW)
    GPIO.cleanup()
    LCD1602.clear()

def loop():
    while True:
        dis = distance()
        print (dis, 'cm')
        print ('')
        GPIO.output(BUZZER, GPIO.LOW)
        if (dis > 400):
            LCD1602.clear()
            LCD1602.write(0, 0, 'Error')
            LCD1602.write(3, 1, 'Out of range')
            time.sleep(0.5)
        else:
            LCD1602.clear()
            LCD1602.write(0, 0, 'Distance is')
            LCD1602.write(5, 1, str(round(dis,2)) + ' cm')
            if(dis>=50):
                time.sleep(0.5)
            elif(dis<50 and dis>20):
                for i in range(0,2,1):
                    GPIO.output(BUZZER, GPIO.HIGH)
                    time.sleep(0.05)
                    GPIO.output(BUZZER, GPIO.LOW)
                    time.sleep(0.2)

```

```

elif(dis<=20):
    for i in range(0,5,1):
        GPIO.output(BUZZER, GPIO.HIGH)
        time.sleep(0.05)
        GPIO.output(BUZZER, GPIO.LOW)
        time.sleep(0.05)

if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```

def lcdsetup():
    LCD1602.init(0x27, 1) # init(slave address, background light)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(TRIG, GPIO.OUT)
    GPIO.setup(ECHO, GPIO.IN)
    GPIO.setup(BUZZER, GPIO.OUT, initial=GPIO.LOW)
    lcdsetup()

```

In this program, we apply the previously used components synthetically. Here we use buzzers, LCD and ultrasonic. We can initialize them in the same way as we did before.

```

dis = distance()
print (dis, 'cm')
print ('')
GPIO.output(BUZZER, GPIO.LOW)
if (dis > 400):
    LCD1602.clear()
    LCD1602.write(0, 0, 'Error')
    LCD1602.write(3, 1, 'Out of range')
    time.sleep(0.5)
else:

```

```
LCD1602.clear()
LCD1602.write(0, 0, 'Distance is')
LCD1602.write(5, 1, str(round(dis,2)) +' cm')
```

Here we get the values of the ultrasonic sensor and get the distance through calculation. If the value of distance is greater than the range of value to be detected, an error message is printed on the LCD. And if the distance is within the working range, the corresponding results will be output.

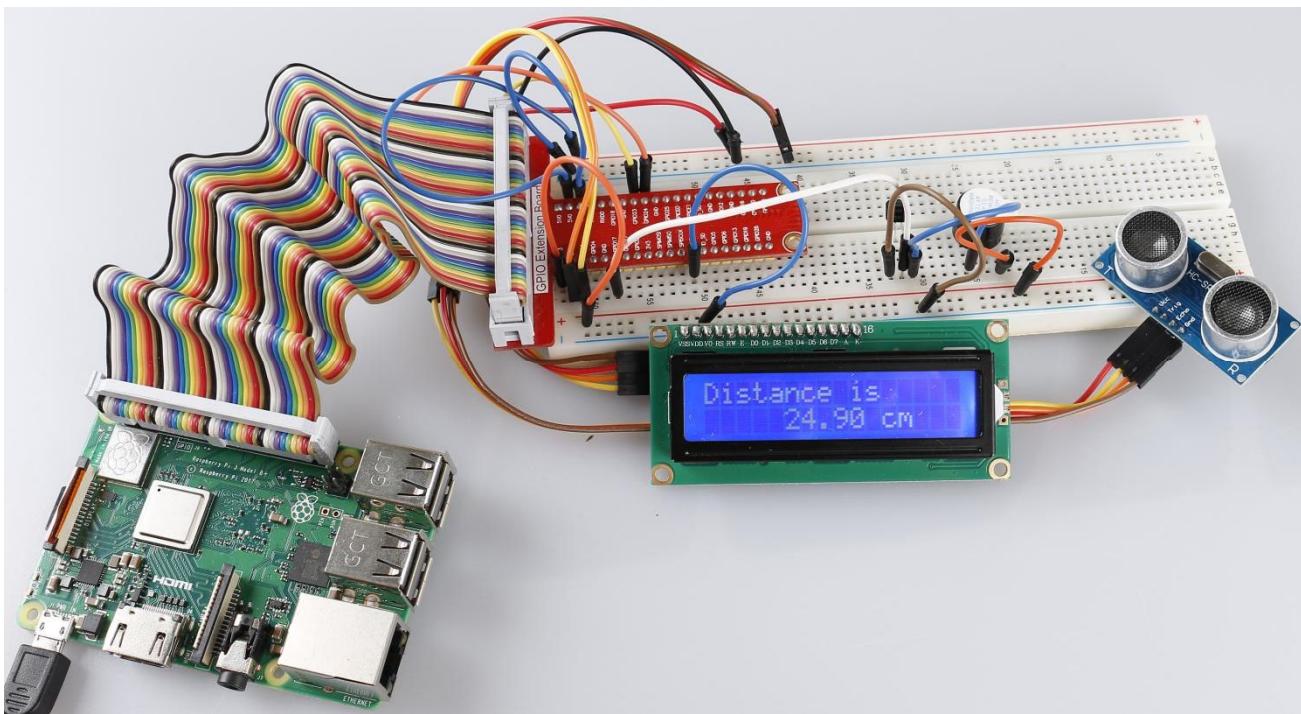
LCD1602.write(5, 1, str(round(dis,2)) +' cm')

Since the LCD output only supports character types, we need to use **str ()** to convert numeric values to characters. We are going to round it to two decimal places.

```
if(dis>=50)
{delay(500);}
else if(dis<50 & dis>20) {
    for(int i=0;i<2;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(200);
    }
}
else if(dis<=20){
    for(int i=0;i<5;i++){
        digitalWrite(Buzzer,HIGH);
        delay(50);
        digitalWrite(Buzzer,LOW);
        delay(50);
    }
}
```

This judgment condition is used to control the sound of the buzzer. According to the difference in distance, it can be divided into three cases, in which there will be different sound frequencies. Since the total value of delay is 500, all of them can provide a 500ms interval for the ultrasonic sensor to work.

## Phenomenon Picture

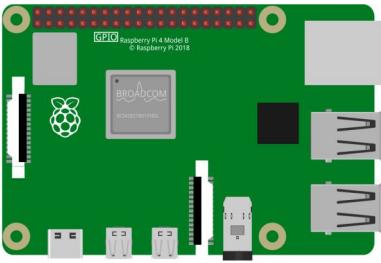
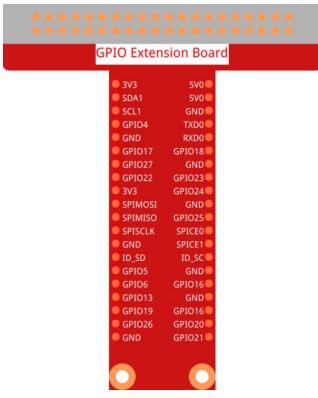
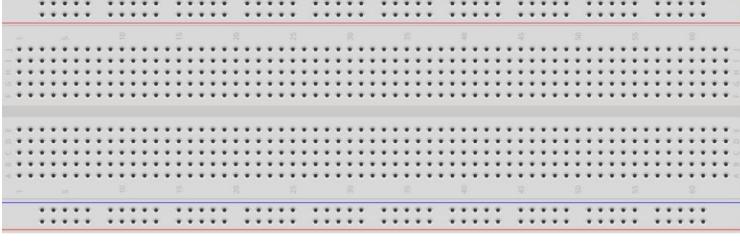


### 3.1.2 Traffic Light

#### Introduction

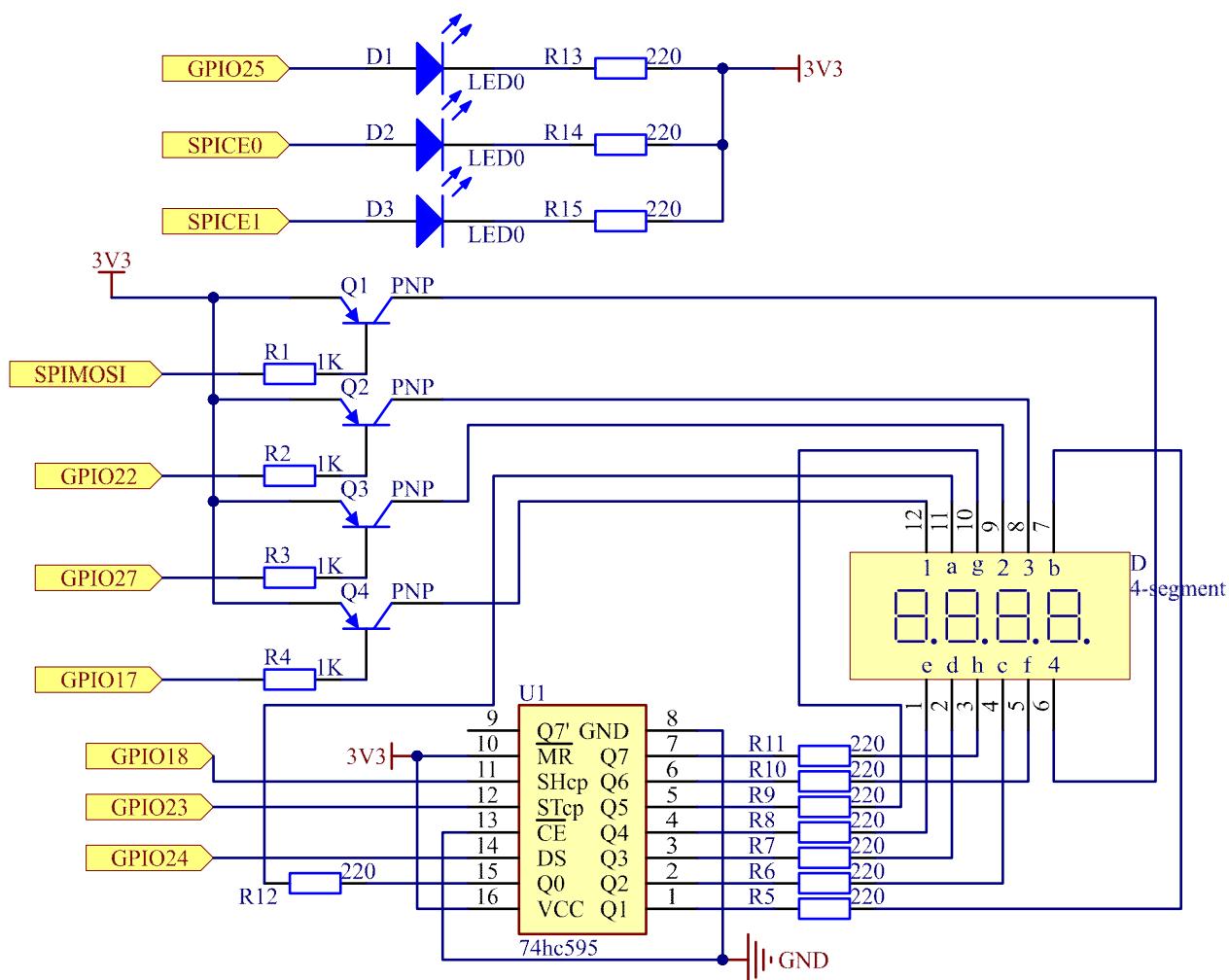
In this project, we will use LED lights of three colors to realize the change of traffic lights and a four-digit 7-segment display will be used to display the timing of each traffic state.

#### Components

1 * Raspberry Pi	1 * T-Extension Board	1 * 4-Digit 7-segment display
	 GPIO Extension Board Pinout: 3V3 SDA1 SVO GND SCL1 GND 3V3 GPIO4 TXD0 GND RXD0 GPIO17 GPIO18 GPIO27 GND GPIO22 GPIO23 3V3 GPIO24 GND SPIMOSI GPIO25 SPIMISO GPIO26 SPISCLK GPIO27 GND SPICE0 ID_SD ID_SC GND GPIO16 GPIO05 GND GPIO06 GPIO16 GPIO13 GND GPIO19 GPIO16 GPIO26 GPIO20 GND GPIO21	
1 * 40-pin Cable	4 * PNP Transistor	3 * LED
		
1 * Breadboard	1 * 74HC595	Several Jumper Wires
	11 * Resistor(220Ω)	
	4 * Resistor 1KΩ	

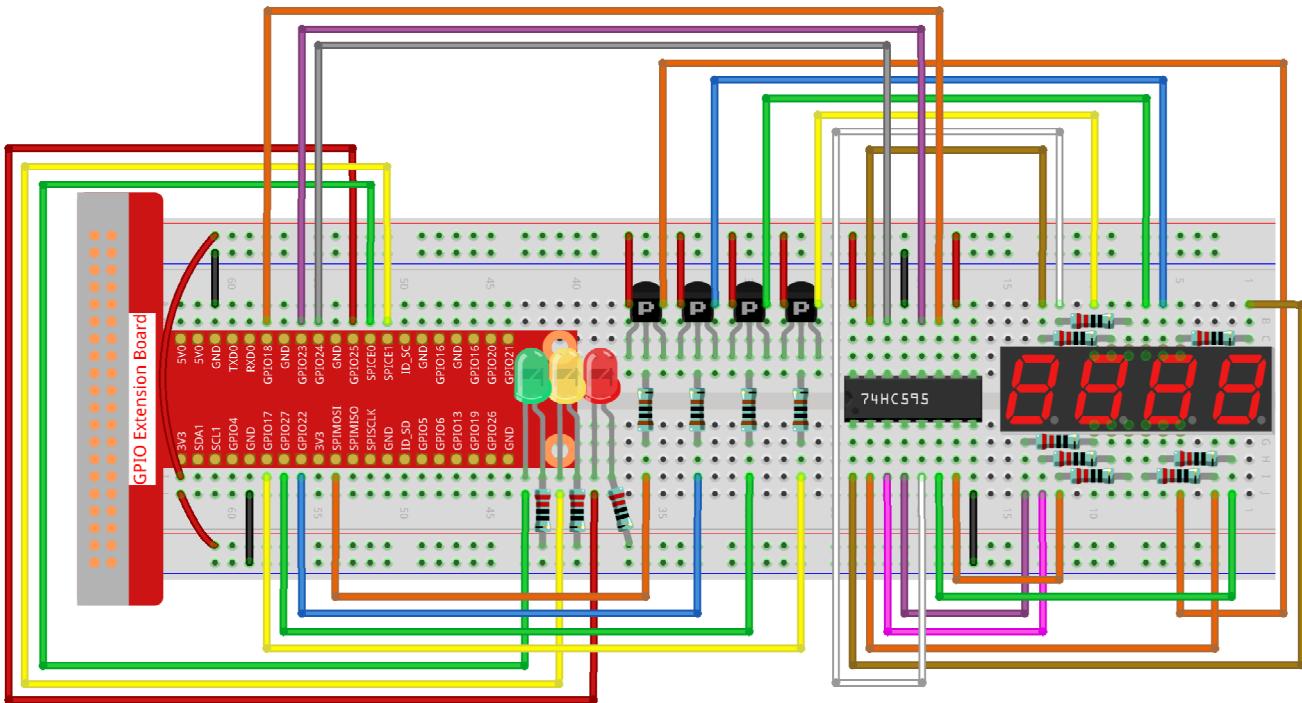
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPI MOSI	Pin 19	12	10
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
CE0	Pin 24	10	8
CE1	Pin 26	11	7



## Experimental Procedures

**Step 1:** Build the circuit.



### ➤ For C Language Users

**Step 2:** Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.2/
```

**Step 3:** Compile.

```
gcc 3.1.2_TrafficLight.c -lwiringPi
```

**Step 4:** Run.

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

As the code runs, LEDs will simulate the color changing of traffic lights. Firstly, the red LED lights up for 60s, then the green LED lights up for 30s; next, the yellow LED lights up for 5s. After that, the red LED lights up for 60s once again. In this way, this series of actions will be executed repeatedly.

## Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.2\_TrafficLight.c.

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define SDI 5 //serial data input(DS)
#define RCLK 4 //memory clock input(STCP)
#define SRCLK 1 //shift register clock input(SHCP)
const int placePin[]={0,2,3,12}; // Define 4 digit's common pin
const int ledPin[]={6,10,11}; //Define 3 LED pin
unsigned char number[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]{"Red", "Green", "Yellow"};
int counter = 60;

//here is the 4 digital function
void selectPlace(int digit){.....}

void hc595_shift(int8_t data){.....}

void display(int num){.....}

//here is the timer for countdown
void timer(int sig){ //Timer function
    if(sig == SIGALRM){
        counter--;
        alarm(1);
        if(counter == 0){
            if(colorState == 0) counter = greenLight;
            if(colorState == 1) counter = yellowLight;
            if(colorState == 2) counter = redLight;
            colorState = (colorState+1)%3;
        }
    }
}
```

```

    }
}

void lightup(int state)
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],HIGH);
    }
    digitalWrite(ledPin[state],LOW);
}

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){.....}
    pinMode(SDI,OUTPUT);      //set the pin connected to74HC595 for output mode
    pinMode(RCLK,OUTPUT);
    pinMode(SRCLK,OUTPUT);
    //set the pin connected to 7-segment display common end to output mode
    for(i=0;i<4;i++){
        pinMode(placePin[i],OUTPUT);
        digitalWrite(placePin[i],HIGH);
    }
    //set the led pin
    for(i=0;i<3;i++){
        pinMode(ledPin[i],OUTPUT);
        digitalWrite(ledPin[i],HIGH);
    }
    signal(SIGALRM,timer); //configure the timer
    alarm(1);              //set the time of timer to 1s
    while(1){
        display(counter); //display the number counter
        lightup(colorState); //turn on the traffic light
    }
    return 0;
}

```

## Code Explanation

```
int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 60;
```

These variables, greenLight, yellowLight and redLight store the duration time of LEDs in different colors respectively, the Unit is second.

The variable **colorState** corresponds to the state of the traffic lights, and we only need to do a simple calculation of colorState to indicate the order change of the state of the traffic lights.

**Counter** is used to count down the time to each traffic light status and will be output on a four 7-segment display.

```
void timer(int sig){ //Timer function
    if(sig == SIGALRM){
        counter--;
        alarm(1);
        if(counter == 0){
            if(colorState == 0) counter = greenLight;
            if(colorState == 1) counter = yellowLight;
            if(colorState == 2) counter = redLight;
            colorState = (colorState+1)%3;
        }
    }
}
```

On this timer, **counter** decreases gradually with every second passing, and when it goes to 0, the state of the traffic light changes accordingly.

```
void lightup(int state)
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],HIGH);
    }
    digitalWrite(ledPin[state],LOW);
}
```

The function is to turn off all the lights first, and then light up the corresponding LED according to the value of the traffic light **state**.

## ➤ For Python Language Users

**Step 2:** Change directory.

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

**Step 3:** Run.

```
sudo python 3.1.2_TrafficLight.py
```

As the code runs, LEDs will simulate the color changing of traffic lights. Firstly, the red LED lights up for 60s, then the green LED lights up for 30s; next, the yellow LED lights up for 5s. After that, the red LED lights up for 60s once again. In this way, this series of actions will be executed repeatedly. Meanwhile, the 4-digit 7-segment display displays the countdown time continuously.

## Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.2\_TrafficLight.py.

```
import RPi.GPIO as GPIO
import time
import threading

#define the pins connect to 74HC595
SDI = 18    #serial data input(DS)
RCLK = 16    #memory clock input(STCP)
SRCLK = 12    #shift register clock input(SHCP)
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)

placePin = (11,13,15,19)
ledPin =(22,24,26)

greenLight = 30
yellowLight = 5
redLight = 60
lightColor=("Red","Green","Yellow")

colorState=0
counter = 60
t = 0
```

```
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    for pin in placePin:
        GPIO.setup(pin,GPIO.OUT)
    for pin in ledPin:
        GPIO.setup(pin,GPIO.OUT)

def hc595_shift(dat):
    .....

def selectPlace(digit):
    .....

def display(num):
    .....

def timer():      #timer function
    global counter
    global colorState
    global t
    t = threading.Timer(1.0,timer)
    t.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
        colorState=(colorState+1)%3

def lightup(state):
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.HIGH)
```

```

GPIO.output(ledPin[state], GPIO.LOW)

def loop():
    global t
    global counter
    global colorState
    t = threading.Timer(1.0,timer)
    t.start()
    while True:
        display(counter)
        lightup(colorState)

def destroy(): # When "Ctrl+C" is pressed, the function is executed.
    global t
    GPIO.cleanup()
    t.cancel()    #cancel the timer

if __name__ == '__main__': # Program starting from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

## Code Explanation

```

greenLight = 30
yellowLight = 5
redLight = 60
colorState=0
counter = 60

```

These variables, greenLight, yellowLight and redLight store the duration time of LEDs in different colors respectively, the Unit is second.

The variable colorState corresponds to the state of the traffic lights, and we only need to do a simple calculation of colorState to indicate the order change of the state of the traffic lights.

Counter is used to count down the time to each traffic light status and will be output on a 4-digit 7-segment display.

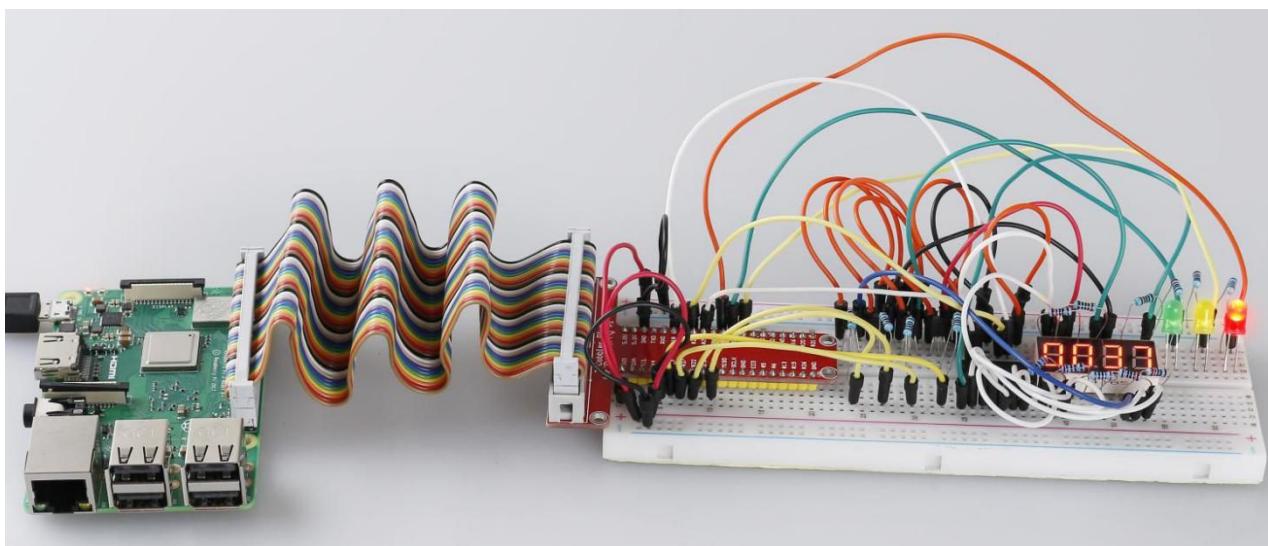
```
def timer():      #timer function
    global counter
    global colorState
    global t
    t = threading.Timer(1.0,timer)
    t.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
    colorState=(colorState+1)%3
```

On this timer, counter decreases gradually with every second passing, and when it goes to 0, the state of the traffic light changes accordingly.

```
def lightup(state):
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.HIGH)
    GPIO.output(ledPin[state], GPIO.LOW)
```

The function is to turn off all the lights first, and then light up the corresponding LED according to the value of the traffic light state.

## Phenomenon Picture



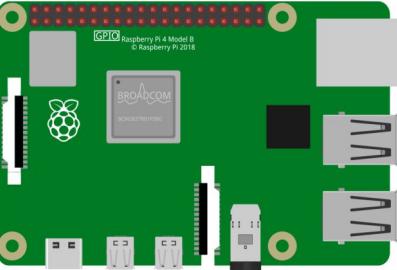
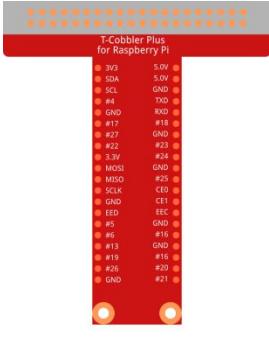
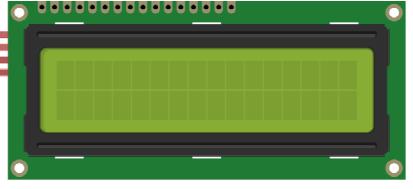
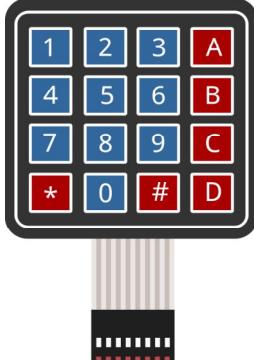
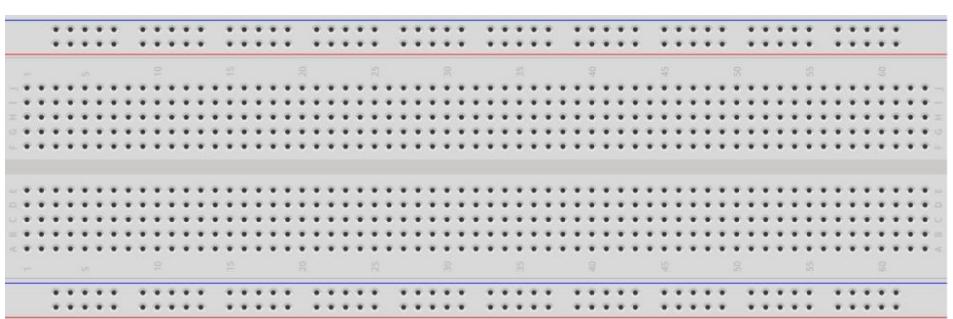
### 3.1.3 Password Lock

#### Introduction

In this project, we will use a keypad and a LCD to make a combination lock. The LCD will display a corresponding prompt for you to type your password on the Keypad. If the password is input correctly, "Correct" will be displayed.

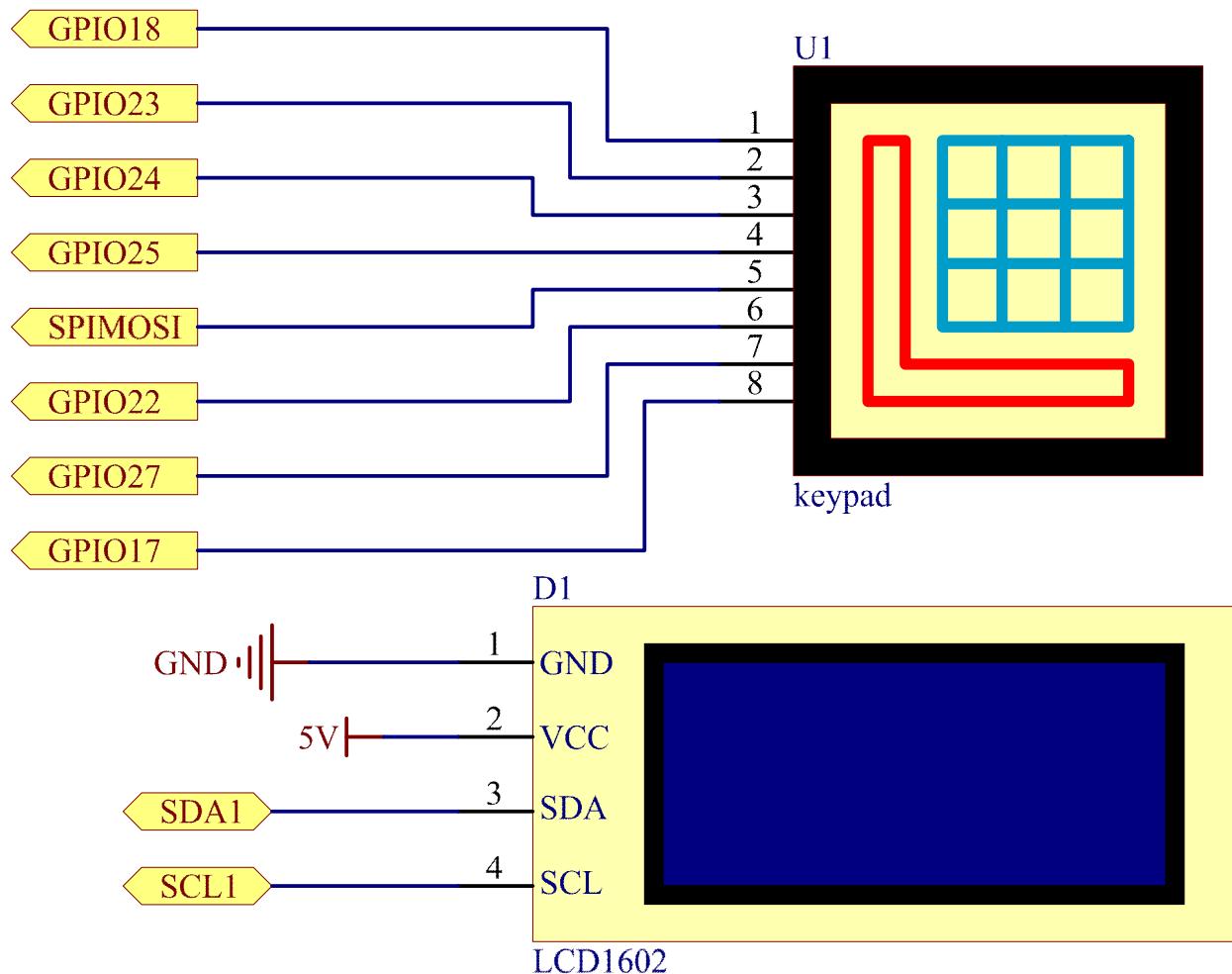
On the basis of this project, we can add additional electronic components, such as buzzer, LED and so on, to add different experimental phenomena for password input.

#### Components

1 * Raspberry Pi	1* T-Extension Board	1 * I2C LCD1602
	 T-Cobbler Plus for Raspberry Pi Pinout: 3V 5.0V SDA GND SCL GND #4 TxD #5 RxD #17 #18 #27 GND #22 #23 #24 #25 MOSI GND MISO GND SCLK CE0 GND CE1 LED D1 #3 #16 #6 #16 #13 GND #19 #16 #26 #20 #27 GND	
Several Jumper Wires		
1 * 40-pin Cable	1 * Keypad	
	 1 2 3 A 4 5 6 B 7 8 9 C * 0 # D	
1 * Breadboard		

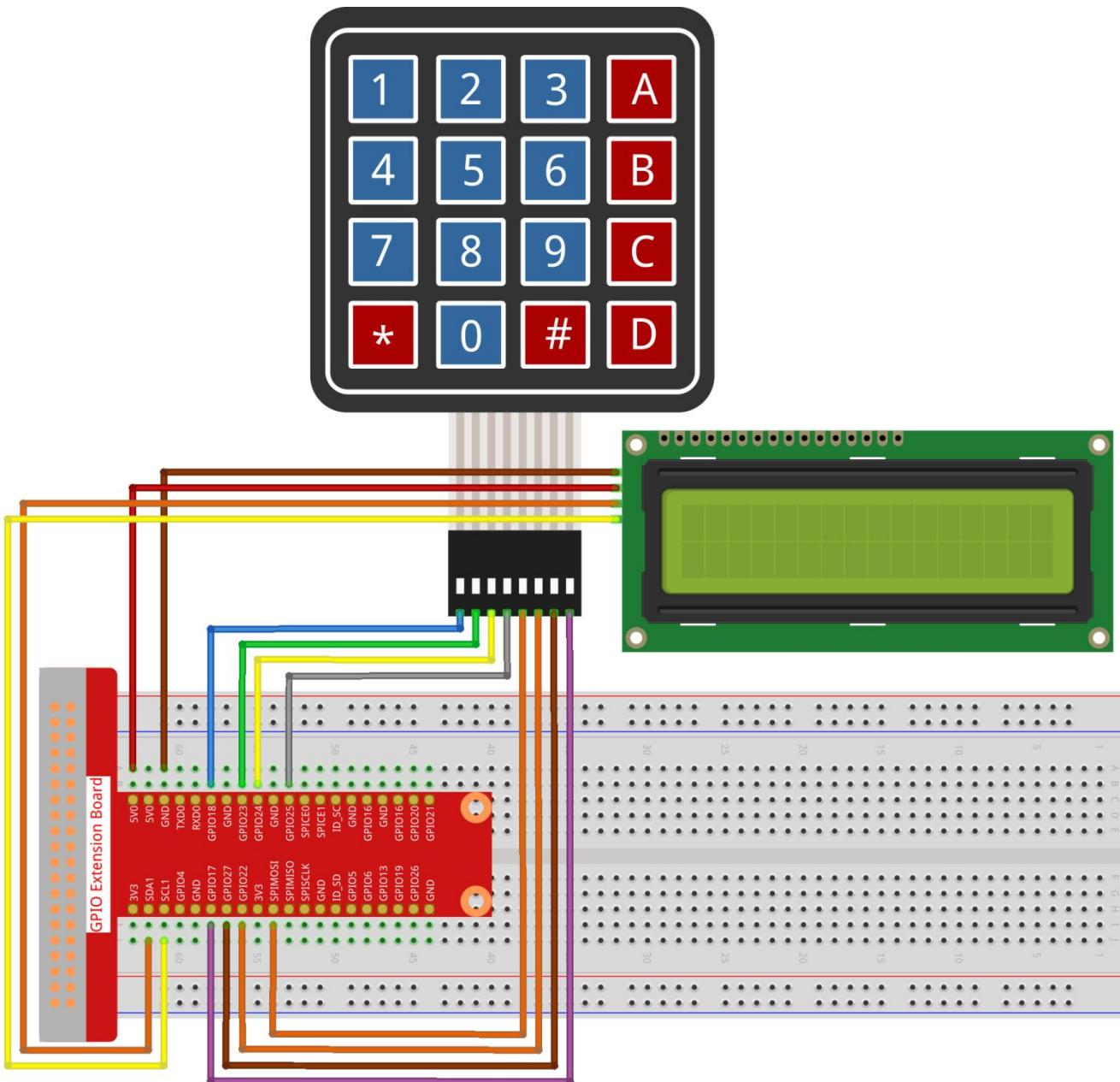
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPI MOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		



## Experimental Procedures

**Step 1:** Build the circuit.



## ➤ For C Language Users

**Step 2:** Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.3/
```

**Step 3:** Compile.

```
gcc 3.1.3_PasswordLock.cpp Key.cpp Keypad.cpp -lwiringPi
```

**Note:** The program contains custom headers that are compiled when CPP files are compiled.

You can use this method to simplify the instruction.

```
gcc *.cpp -lwiringPi
```

**Note:** Here we use the wildcard (\*), which causes all the files that conform to the format to be processed together.

**Step 4:** Run.

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

After the code runs, keypad is used to input password. If the “CORRECT” appears on LCD1602, there is no wrong with the password; otherwise, “WRONG KEY” will appear.

## Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.3\_PasswordLock.cpp.

```
#include "Keypad.hpp"
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPi2C.h>
#include <string.h>

int LCDAddr = 0x27;
int BLEN = 1;
int fd;

const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
const byte LENS = 4; //password length
char keys[ROWS][COLS] = { //key code
```

```

{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};

char password[LENS]={'1','9','8','4'}; //password
char testword[LENS]={};
int keyIndex=0;

byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
//create Keypad object
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void write_word(int data){.....}
void send_command(int comm){.....}
void send_data(int data){.....}
void lcdInit(){.....}
void clear(){.....}
void write(int x, int y, char const data[]){.....}

int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
            {return 0;}
    }
    return 1;
}

int main(){
    if(wiringPiSetup() == -1){.....}
    fd = wiringPiI2CSetup(LCDAddr);
    lcdInit();
    clear();
    write(0, 0, "WELCOME!");
    write(2, 1, "Enter password");
    char key = 0;
    keypad.setDebounceTime(50);
    while(1){

```

```

key = keypad.getKey(); //get the state of keys
if (key){      //if a key is pressed, print out its key code
    clear();
    write(0, 0, "Enter password:");
    write(15-keyIndex, 1, "****");
    testword[keyIndex]=key;
    keyIndex++;
    if(keyIndex==LENS){
        if(check()==0){
            clear();
            write(3, 0, "WRONG KEY!");
            write(0, 1, "please try again");
        }
        else{
            clear();
            write(4, 0, "CORRECT!");
            write(2, 1, "welcome back");
        }
    }
    keyIndex=keyIndex%LENS;
}
}

return 1;
}

```

## Code Explanation

```

const byte LENS = 4; //password length
char password[LENS]={'1','9','8','4'}; //password
char testword[LENS]={};
int keyIndex=0;

```

The variable, password[LENS] is the correct password that we've set in the program; testword[LENS] is used to store characters input during program operation. KeyIndex is used to indicate the number of digits input.

```

int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
        {return 0;}
    }
}

```

```

    }
    return 1;
}

```

We use a loop to compare the input result with the default password bit by bit. The true value is returned only if they correspond exactly, otherwise the return value is false.

```

while(1){
    key = keypad.getKey(); //get the state of keys
    if (key){      //if a key is pressed, print out its key code
        clear();
        write(0, 0, "Enter password:");
        write(15-keyIndex, 1, "*****");
        testword[keyIndex]=key;
        keyIndex++;
        if(keyIndex==LENS){
            if(check()==0){
                clear();
                write(3, 0, "WRONG KEY!");
                write(0, 1, "please try again");
            }
            else{
                clear();
                write(4, 0, "CORRECT!");
                write(2, 1, "welcome back");
            }
        }
        keyIndex=keyIndex%LENS;
    }
}

```

Get the value of keypad when it is pressed. Then the value input will be displayed on LCD. The variable keyIndex is used to change the display position of "\*\*\*\*\*" according to the password digits input, and since positions greater than 15 are beyond the screen. the effect is to add a "\*" on the LCD display for each digit input.

Then assign the input value to the testword[] array and add 1 to keyIndex. When keyIndex reaches the length of the password, the program judges whether the return value of the check() function is 0. If so, it indicates that the password is wrong and an error message will be displayed on LCD1602; otherwise, LCD1602 displays "CORRECT" and "welcome back".

## ➤ For Python Language Users

**Step 2:** Change directory.

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

**Step 3:** Run.

```
sudo python 3.1.3_PasswordLock.py
```

After the code runs, keypad is used to input password. If the "CORRECT" appears on LCD1602, there is no wrong with the password; otherwise, "WRONG KEY" will appear.

### Code

**Note:** The following codes are incomplete. If you want to check the complete codes, you are suggested to use command nano 3.1.3\_PasswordLock.py.

```
import RPi.GPIO as GPIO
import time
import LCD1602

##### HERE IS THE KEYPAD#####
class Key(object):.....  
  
class Keypad(object):.....  
  
##### EXAMPLE CODE START HERE #####
ROWS = 4
COLS = 4
LENS = 4
keys = [ '1','2','3','A',
         '4','5','6','B',
         '7','8','9','C',
         '*', '0','#','D'   ]
password=['1','9','8','4']
testword=['0','0','0','0']
keyIndex=0  
  
rowsPins = [12,16,18,22]
colsPins = [19,15,13,11]  
  
def check():
    for i in range(0,LENS):
```

```

if(password[i]!=testword[i]):
    return 0
return 1

def setup():
    LCD1602.init(0x27, 1)  # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'WELCOME!')
    LCD1602.write(2, 1, 'Enter password')
    time.sleep(2)

def destroy():
    LCD1602.clear()

def loop():
    keypad = Keypad(keys,rowsPins,colsPins,ROWS,COLS)
    keypad.setDebounceTime(50)
    global keyIndex
    global LENS
    while(True):
        key = keypad.getKey()
        if(key != keypad.NULL):
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, "*****")
            testword[keyIndex]=key
            keyIndex+=1
            if (keyIndex is LENS):
                if (check() is 0):
                    LCD1602.clear()
                    LCD1602.write(3, 0, "WRONG KEY!")
                    LCD1602.write(0, 1, "please try again")
                else:
                    LCD1602.clear()
                    LCD1602.write(4, 0, "CORRECT!")
                    LCD1602.write(2, 1, "welcome back")
            keyIndex=keyIndex%LENS

```

```

if __name__ == '__main__': # Program start from here
    try:
        setup()
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
        destroy()

```

## Code Explanation

```

LENS = 4
password=['1','9','8','4']
testword=['0','0','0','0']
keyIndex=0

```

The variable, password is the correct password that we've set in the program; testword is used to store characters entered during program operation. KeyIndex is used to indicate the number of digits input.

```

def check():
    for i in range(0,LENS):
        if(password[i]!=testword[i]):
            return 0
    return 1

```

We use a loop to compare the input result with the default password bit by bit. The true value is returned only if they correspond exactly, otherwise the return value is false.

```

def setup():
    LCD1602.init(0x27, 1) # init(slave address, background light)
    LCD1602.clear()
    LCD1602.write(0, 0, 'WELCOME!')
    LCD1602.write(2, 1, 'Enter password')
    time.sleep(2)

```

The function setup() is used to desplay I2C LCD1602. LCD1602.init(0x27, 1) is used to initialize the address of LCD1602 and turn on the backlight. Lcd1602.clear() is applied

to clear the screen if it is necessary, or the characters will overlap and affect the display result. LCD1602.write(0, 0, 'WELCOME!') is used to display the position of characters. In row 0, column 0, character 'WELCOME' will appear.

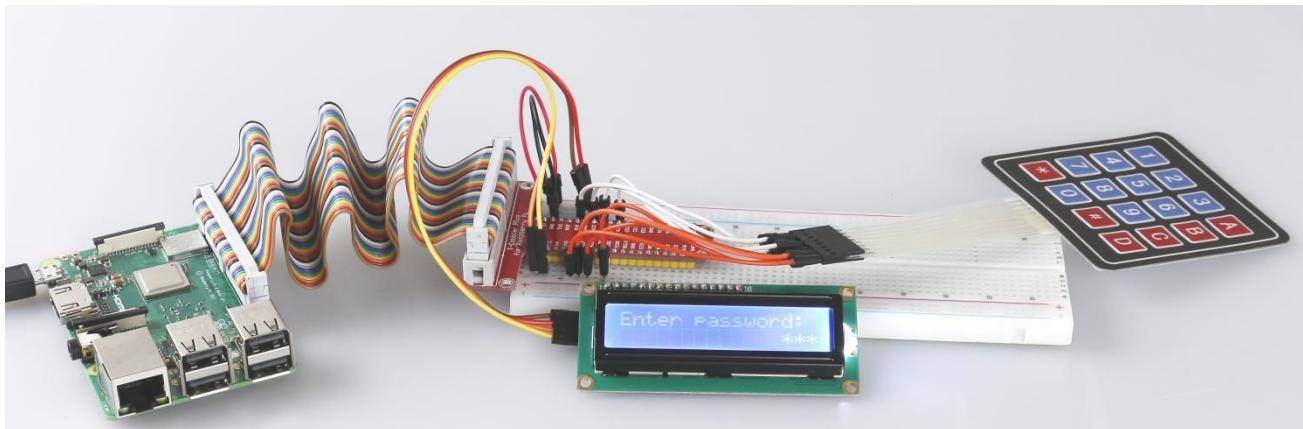
```
key = keypad.getKey()
if(key != keypad.NULL):
    LCD1602.clear()
    LCD1602.write(0, 0, "Enter password:")
    LCD1602.write(15-keyIndex,1, "*****")
    testword[keyIndex]=key
    keyIndex+=1
    if (keyIndex is LENS):
        if (check() is 0):
            LCD1602.clear()
            LCD1602.write(3, 0, "WRONG KEY!")
            LCD1602.write(0, 1, "please try again")
        else:
            LCD1602.clear()
            LCD1602.write(4, 0, "CORRECT!")
            LCD1602.write(2, 1, "welcome back")
    keyIndex=keyIndex%LENS
```

Read the key of keypad when it is pressed, then the key input will appear on LCD.

The variable keyIndex is used to change the display position of "\*\*\*\*" according to the password digits input since positions greater than 15 are beyond the screen. The effect is to add a "\*" on the LCD display for each digit input.

Then assign the input value to the testword[] array and add 1 to keyIndex. When keyIndex reaches the length of the password, judge whether the return value of check() function is 0; if the condition is met, it means that the password input is wrong; accordingly, the notice of error displays on LCD1602. Otherwise, there appears "CORRECT" and "welcome back".

## Phenomenon Picture

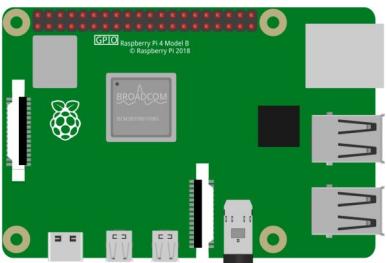
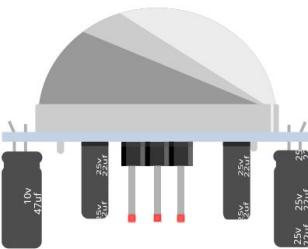
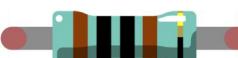
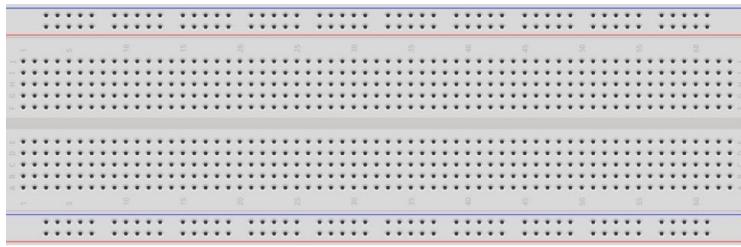


### 3.1.4 Welcome

#### Introduction

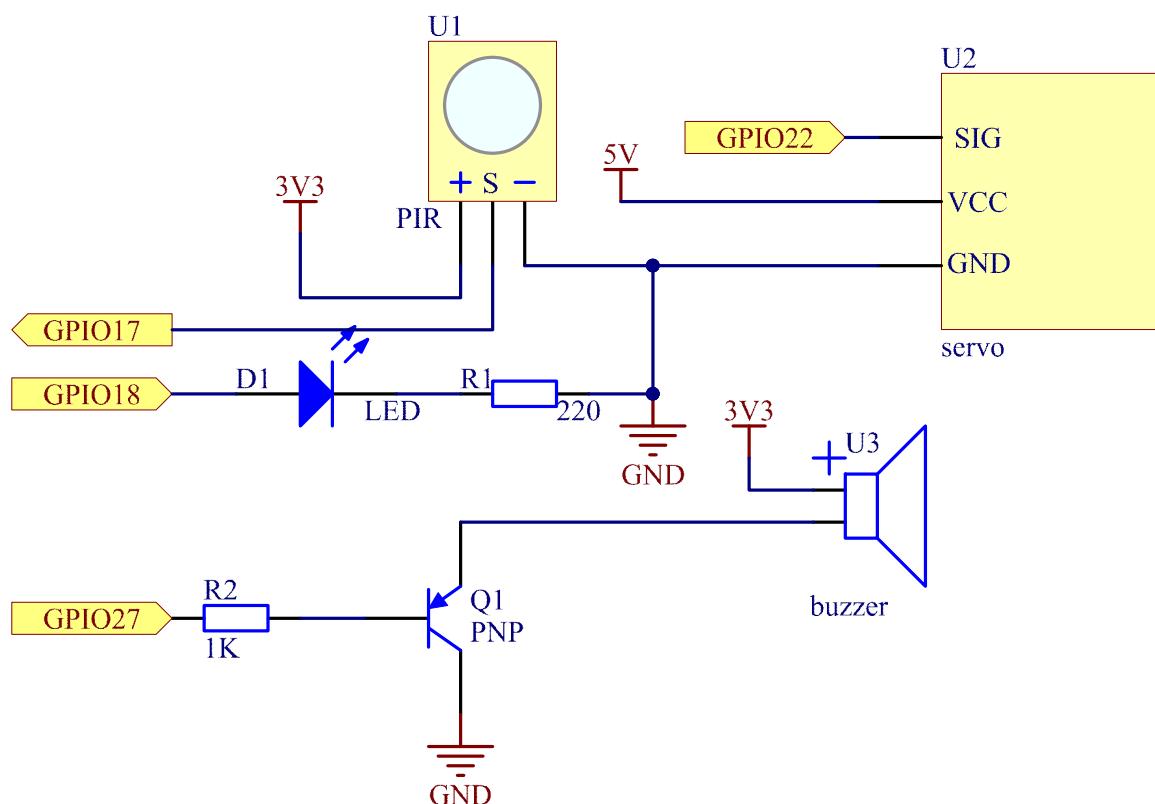
In this project, we will use PIR to sense the movement of pedestrians, and use servos, LED, buzzer to simulate the work of the sensor door of the convenience store. When the pedestrian appears within the sensing range of the PIR, the indicator light will be on, the door will be opened, and the buzzer will play the opening bell.

#### Components

1* Raspberry Pi	1* T-Extension Board	1 * PIR			
					
1 * Servo	1 * Active Buzzer	1 * PNP Transistor	1 * LED		
					
1 * Resistor 1kΩ	1 * Resistor 220Ω	Several Jumper Wires			
					
1 * 40-pin Cable					
1 * Breadboard					

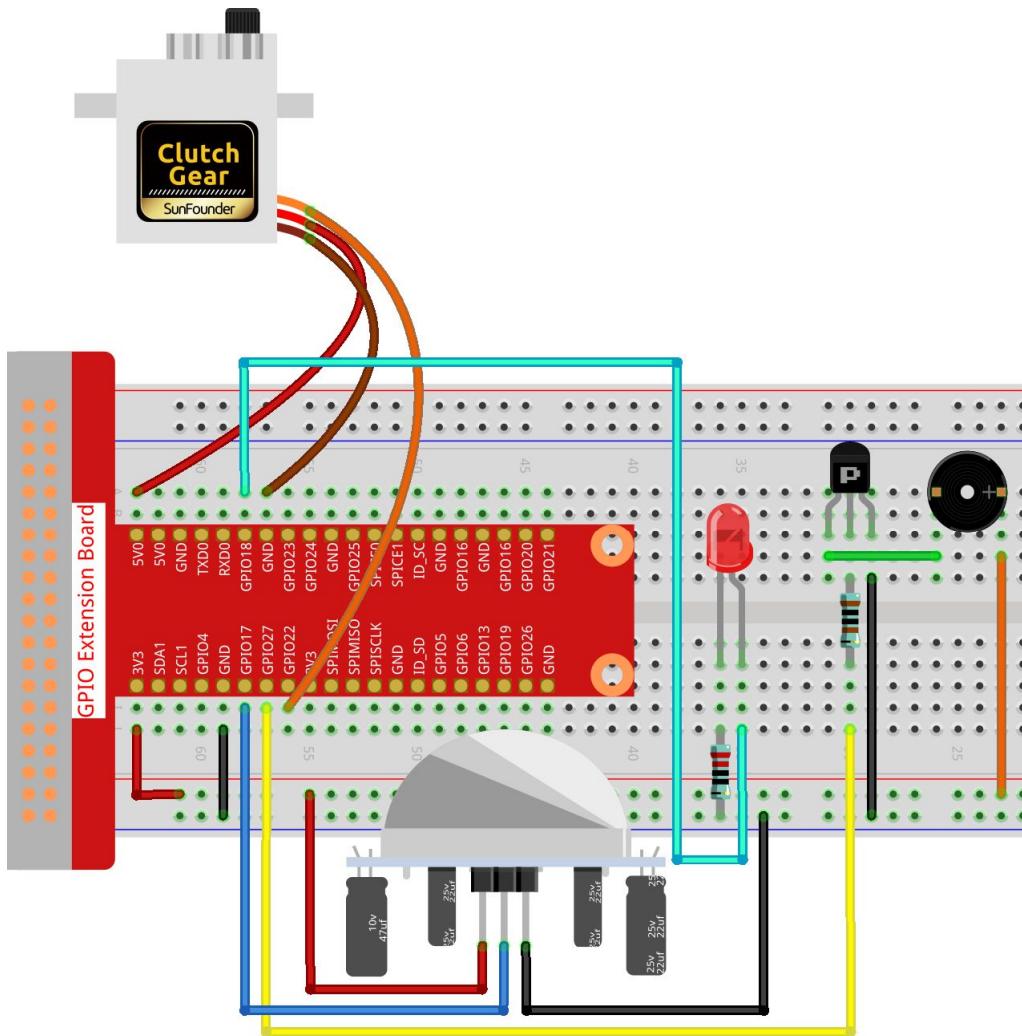
## Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22



## Experimental Procedures

### Step 1: Build the circuit.



### ➤ For C Language Users

#### Step 2: Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.4/
```

#### Step 3: Compile.

```
gcc 3.1.4_Welcome.c -lwiringPi
```

#### Step 4: Run.

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

After the code runs, if the PIR sensor detects someone passing by, the door will automatically open (simulated by the servo), turn on the indicator and play the doorbell music. After the doorbell music plays, the system will automatically close the door and turn off the indicator light, waiting for the next time someone passes by.

## Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <softTone.h>
#include <softPwm.h>

#define CL1 131
.....
#define CH7 990

#define ledPin 1 //define the ledPin
#define pirPin 0 //define the PIR_SensorPin
#define BuzPin 2 //define the buzzerPin
#define servoPin 3 //define the servoPin

int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};

int beat[] = {1,1,1,1,1,2,1,1,1,1,3};

long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}

void servoWrite(int pin, int angle){
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,map(angle,0,180,5,25));
}

void doorbell(){
    for(int i=0;i<sizeof(song)/4;i++){
        softToneWrite(BuzPin, song[i]);
        delay(beat[i] * 250);
    }
}

void closedoor(){
    digitalWrite(ledPin, LOW); //led off
```

```
for(int i=180;i>-1;i--){ //make servo rotate from maximum angle to minimum angle
    servoWrite(servoPin,i);
    delay(1);
}
}

void opendoor(){
    digitalWrite(ledPin, HIGH); //led on
    for(int i=0;i<181;i++){ //make servo rotate from minimum angle to maximum angle
        servoWrite(servoPin,i);
        delay(1);
    }
    doorbell();
    closedoor();
}

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    if(softToneCreate(BuzPin) == -1){
        printf("setup softTone failed !");
        return 1;
    }
    pinMode(ledPin, OUTPUT);
    pinMode(pirPin, INPUT);
    softPwmCreate(servoPin, 0, 200);

    while(1){
        if(digitalRead(pirPin) == HIGH){ //if read sensor for high level
            opendoor();
        }
    }

    return 0;
}
```

## Code Explanation

```
void servoWrite(int pin, int angle){
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,map(angle,0,180,5,25));
}
```

Create a function, servowrite to write the angle in the servo that is 0-180.

```
void doorbell(){
    for(int i=0;i<sizeof(song)/4;i++){
        softToneWrite(BuzPin, song[i]);
        delay(beat[i] * 250);
    }
}
```

Create a function, doorbell to enable the buzzer to play music.

```
void closedoor(){
    digitalWrite(ledPin, LOW); //led off
    for(int i=180;i>-1;i--){ //make servo rotate from maximum angle to minimum angle
        servoWrite(servoPin,i);
        delay(1);
    }
}
```

Create a closedoor function to simulate closing the door, turn off the LED and let the servo turn from 180 degrees to 0 degree.

```
void opendoor(){
    digitalWrite(ledPin, HIGH); //led on
    for(int i=0;i<181;i++){ //make servo rotate from minimum angle to maximum angle
        servoWrite(servoPin,i);
        delay(1);
    }
    doorbell();
    closedoor();
}
```

The function opendoor() includes several parts: turn on the indicator light, turn the servo (simulate the action of opening the door), play the doorbell music of the convenience store, and call the function closedoor() after playing music.

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

In the function main(), initialize library wiringPi and setup softTone, then set ledPin to output state and pirPin to input state. If the PIR sensor detects someone passing by, the function opendoor will be called to simulate opening the door.

## ➤ For Python Language Users

**Step 2:** Change directory.

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

**Step 3:** Run.

```
sudo python 3.1.4_Welcome.py
```

After the code runs, if the PIR sensor detects someone passing by, the door will automatically open (simulated by the servo), turn on the indicator and play the doorbell music. After the doorbell music plays, the system will automatically close the door and turn off the indicator light, waiting for the next time someone passes by.

## Code

```
import RPi.GPIO as GPIO
import time

ledPin = 18 # define the ledPin
pirPin = 17 # define the sensorPin
servoPin = 22 # define the servoPin
buzPin = 27 # define the buzzerpin

CL = [0, 131, 147, 165, 175, 196, 211, 248] # Frequency of Low C notes
```

```

CM = [0, 262, 294, 330, 350, 393, 441, 495]      # Frequency of Middle C notes

CH = [0, 525, 589, 661, 700, 786, 882, 990]      # Frequency of High C notes

song = [ CH[5],CH[2],CM[6],CH[2],CH[3],CH[6],CH[3],CH[5],CH[3],CM[6],CH[2] ] 

beat = [ 1,1,1,1,1,1,1,1,1,1,]

def setup():
    global p
    global Buzz           # Assign a global variable to replace GPIO.PWM
    GPIO.setmode(GPIO.BCM)   # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT) # Set ledPin's mode is output
    GPIO.setup(sensorPin, GPIO.IN) # Set sensorPin's mode is input
    GPIO.setup(buzzerPin, GPIO.OUT) # Set pins' mode is output
    Buzz = GPIO.PWM(buzzerPin, 440) # 440 is initial frequency.
    Buzz.start(50)            # Start Buzzer pin with 50% duty ration
    GPIO.setup(servoPin, GPIO.OUT) # Set servoPin's mode is output
    GPIO.output(servoPin, GPIO.LOW) # Set servoPin to low
    p = GPIO.PWM(servoPin, 50)   # set Freqeuce to 50Hz
    p.start(0)                # Duty Cycle = 0

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

def servoWrite(angle):    # make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))#map the angle to duty cycle and output it

def doorbell():
    for i in range(1, len(song)):      # Play song 1
        Buzz.ChangeFrequency(song[i])  # Change the frequency along the song note
        time.sleep(beat[i] * 0.25)     # delay a note for beat * 0.25s
    time.sleep(1)                    # Wait a second for next song.

```

```
def closedoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
        servoWrite(i)
        time.sleep(0.001)

def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1): #make servo rotate from 0 to 180 deg
        servoWrite(i)    # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()

def loop():
    while True:
        if GPIO.input(pirPin)==GPIO.HIGH:
            opendoor()

def destroy():
    GPIO.cleanup()          # Release resource
    p.stop()
    Buzz.stop()

if __name__ == '__main__':  # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be
executed.
    destroy()
```

## Code Explanation

```
def setup():
    global p
    global Buzz           # Assign a global variable to replace GPIO.PWM
    GPIO.setmode(GPIO.BCM)   # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT) # Set ledPin's mode is output
    GPIO.setup(sensorPin, GPIO.IN) # Set sensorPin's mode is input
    GPIO.setup(buzzerPin, GPIO.OUT) # Set pins' mode is output
    Buzz = GPIO.PWM(buzzerPin, 440) # 440 is initial frequency.
    Buzz.start(50)            # Start Buzzer pin with 50% duty ration
    GPIO.setup(servoPin, GPIO.OUT) # Set servoPin's mode is output
    GPIO.output(servoPin, GPIO.LOW) # Set servoPin to low
    p = GPIO.PWM(servoPin, 50) # set Frequency to 50Hz
    p.start(0)                # Duty Cycle = 0
```

These statements are used to initialize the pins of each component.

```
def servoWrite(angle):    # make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,2.5,12.5))#map the angle to duty cycle and output it
```

Create a function, servowrite to write the angle in the servo that is 0-180.

```
def doorbell():
    for i in range(1, len(song)):    # Play song 1
        Buzz.ChangeFrequency(song[i]) # Change the frequency along the song note
        time.sleep(beat[i] * 0.25)    # delay a note for beat * 0.25s
```

Create a function, doorbell to enable the buzzer to play music.

```
def closedoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
        servoWrite(i)
        time.sleep(0.001)
```

Close the door and turn off the indicator light.

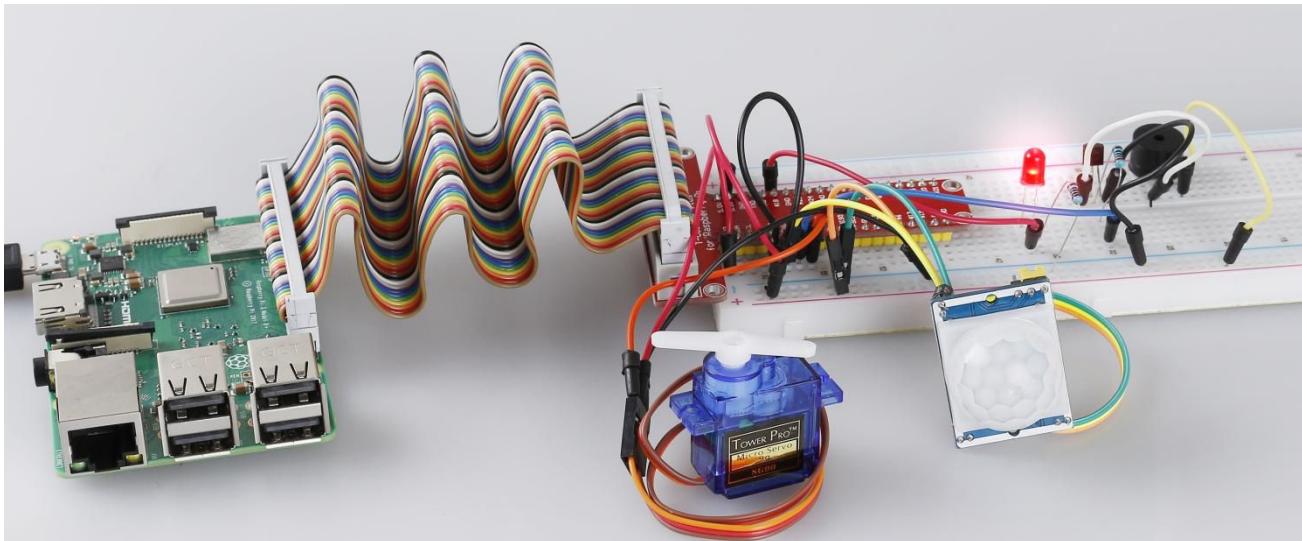
```
def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1): #make servo rotate from 0 to 180 deg
        servoWrite(i) # Write to servo
        time.sleep(0.001)
    doorbell()
    closedoor()
```

The function, opendoor() consists of several parts: turn on the indicator light, turn the servo (to simulate the action of opening the door), play the doorbell music of the convenience store, and call the function , closedoor() after playing music.

```
def loop():
while True:
    if GPIO.input(pirPin)==GPIO.HIGH:
        opendoor()
```

When RIP senses that someone is passing by, it calls the function, opendoor().

## Phenomenon Picture



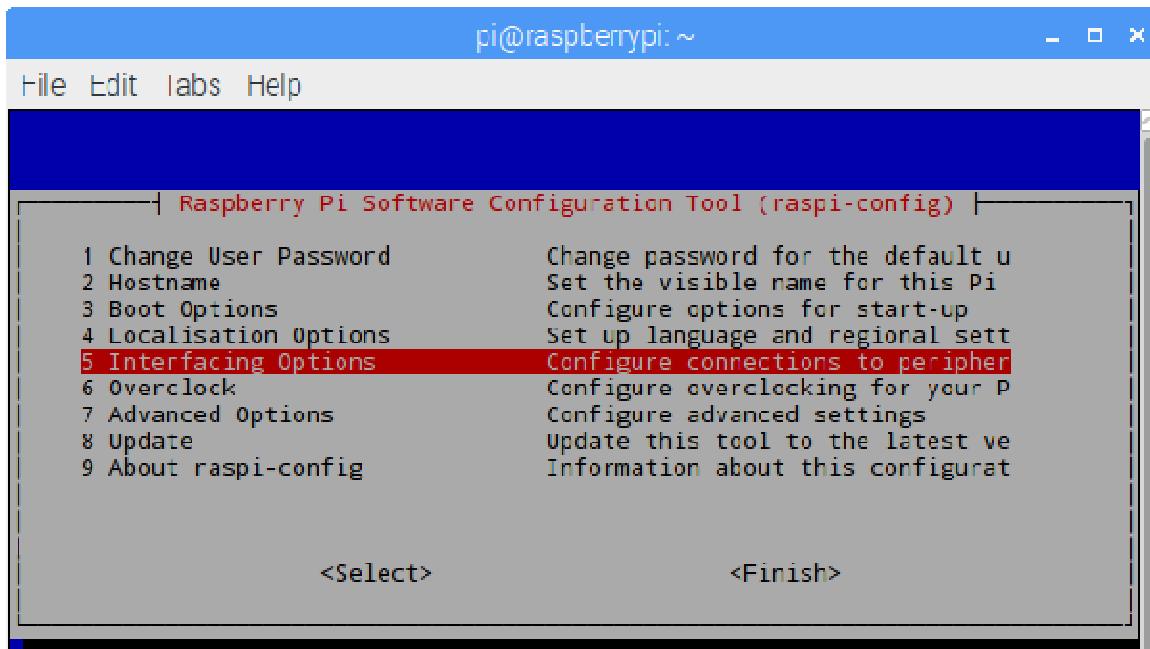
# Appendix

## I2C Configuration

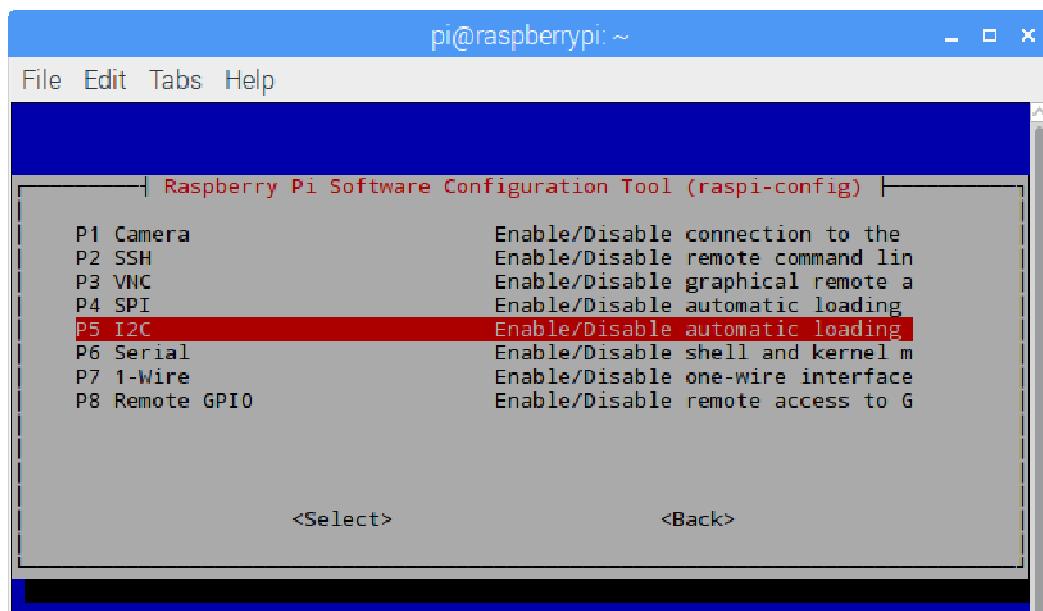
**Step 1:** Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

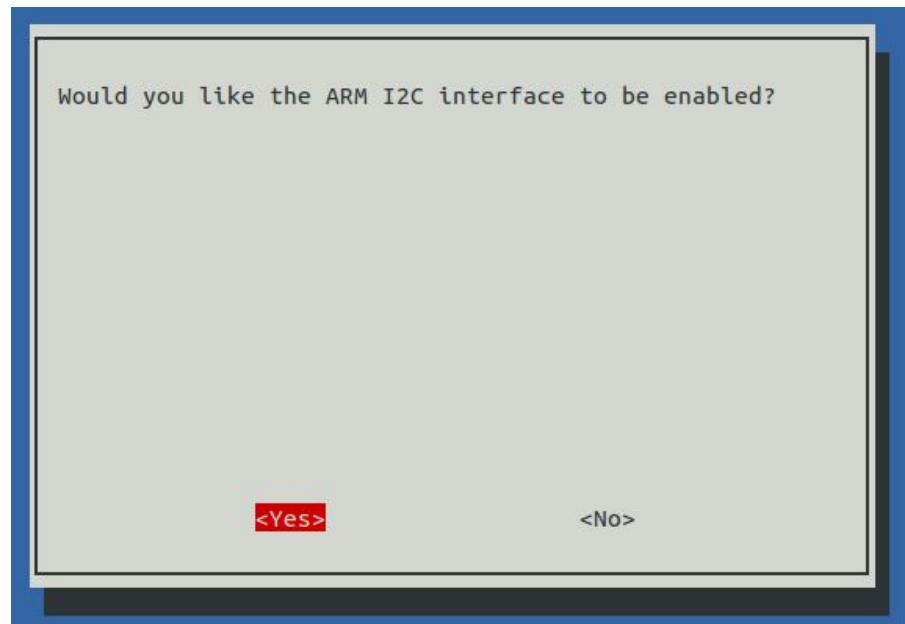
### 5 Interfacing options



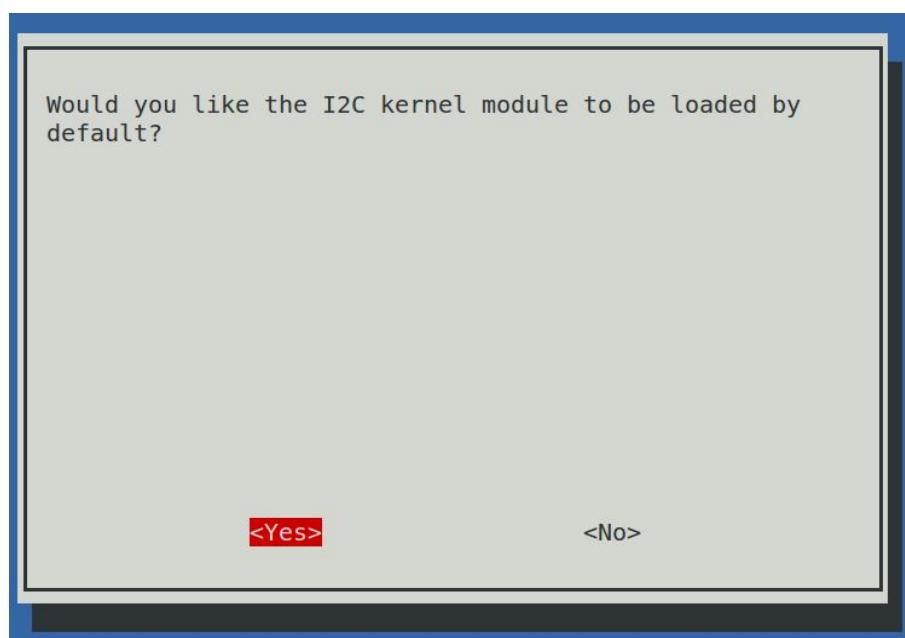
### P5 I2C



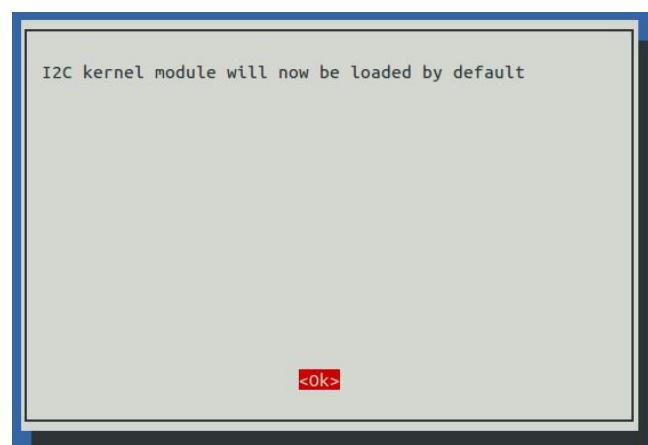
<Yes>



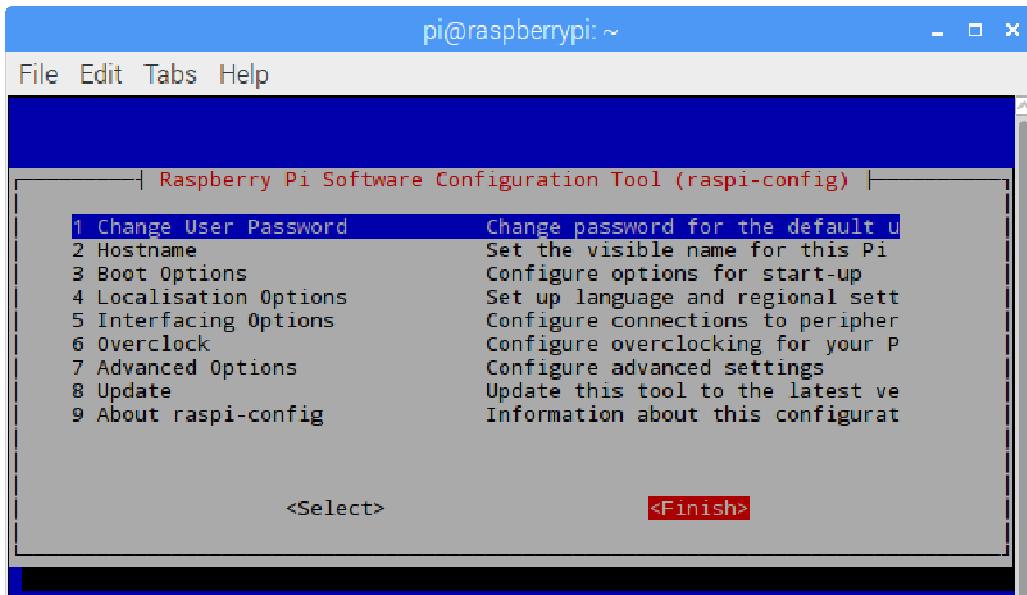
<Yes>



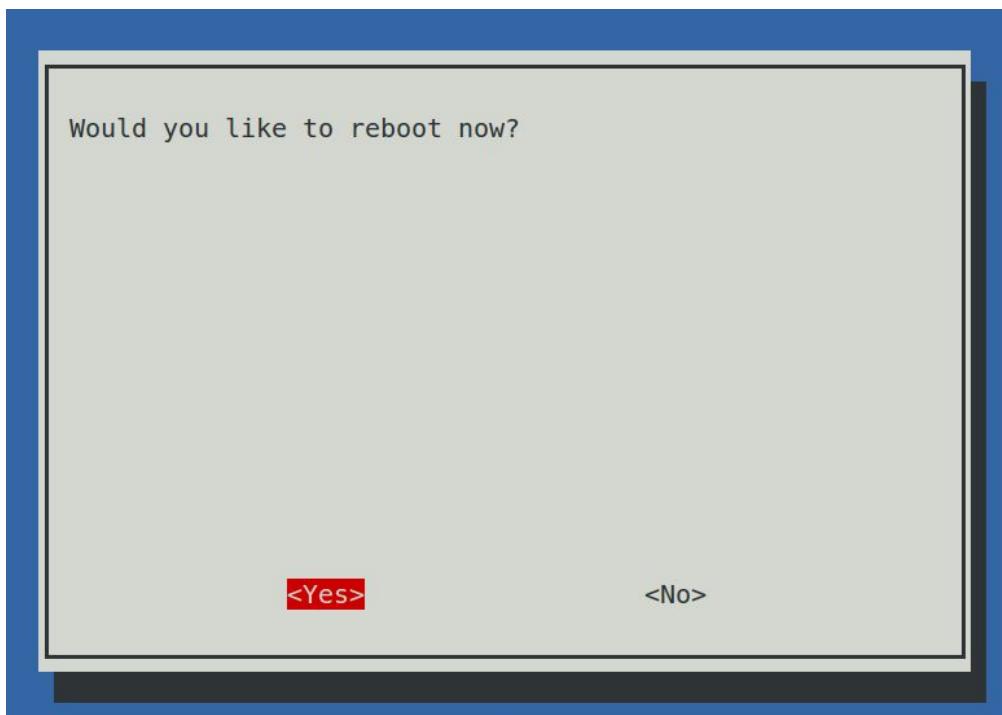
<Ok>



## <Finish>



<Yes> (If you do not see this page, continue to the next step)



**Step 2:** Check whether the i2c modules are loaded and active.

```
lsmod | grep i2c
```

Then the following codes will appear (the number may be different)

```
i2c_dev          6276  0
i2c_bcm2708      4121  0
```

**Step 3:** Install i2c-tools.

```
sudo apt-get install i2c-tools
```

#### Step 4: Check the address of the I2C device.

```
i2cdetect -y 1    # For Raspberry Pi 2  
i2cdetect -y 0    # For Raspberry Pi 1  
pi@raspberrypi ~ $ i2cdetect -y 1  
 0 1 2 3 4 5 6 7 8 9  a b c d e f  
00: -----  
10: -----  
20: -----  
30: -----  
40:      48 -----  
50: -----  
60: -----  
70: -----
```

If there's an I2C device connected, the results will be similar as shown above – since the address of the device is 0x48, 48 is printed.

#### Step 5:

**For C language users:** Install libi2c-dev.

```
sudo apt-get install libi2c-dev
```

**For Python users:** Install smbus for I2C.

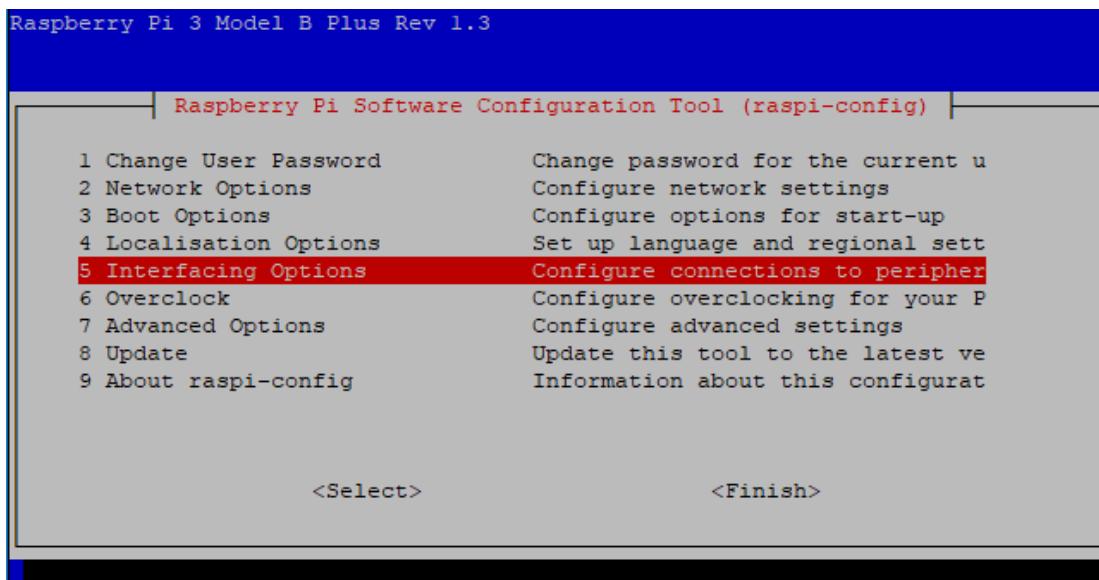
```
sudo apt-get install python-smbus
```

# SPI Configuration

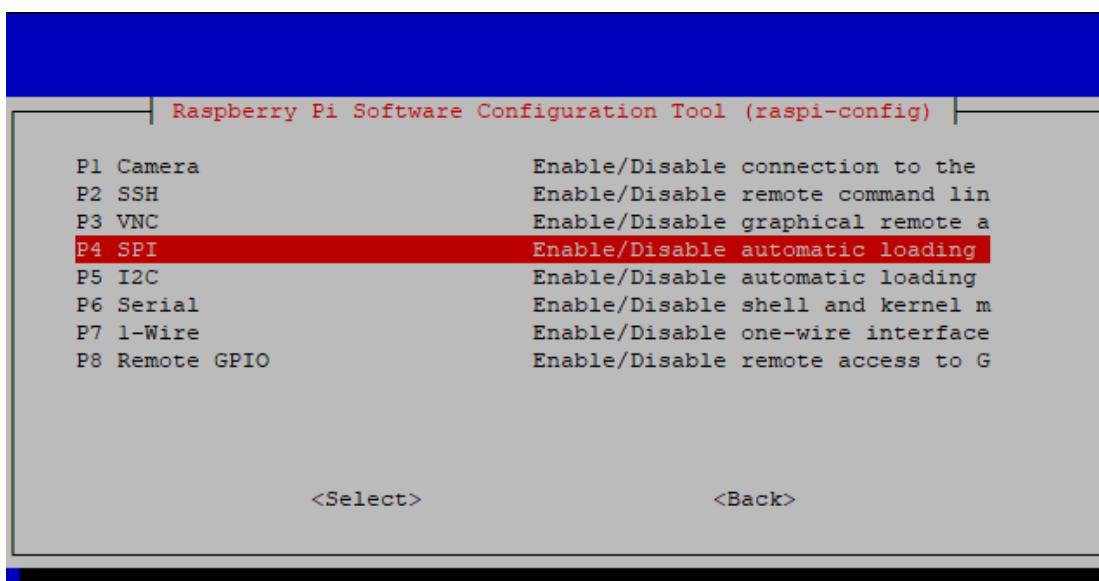
**Step 1:** Enable the SPI port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

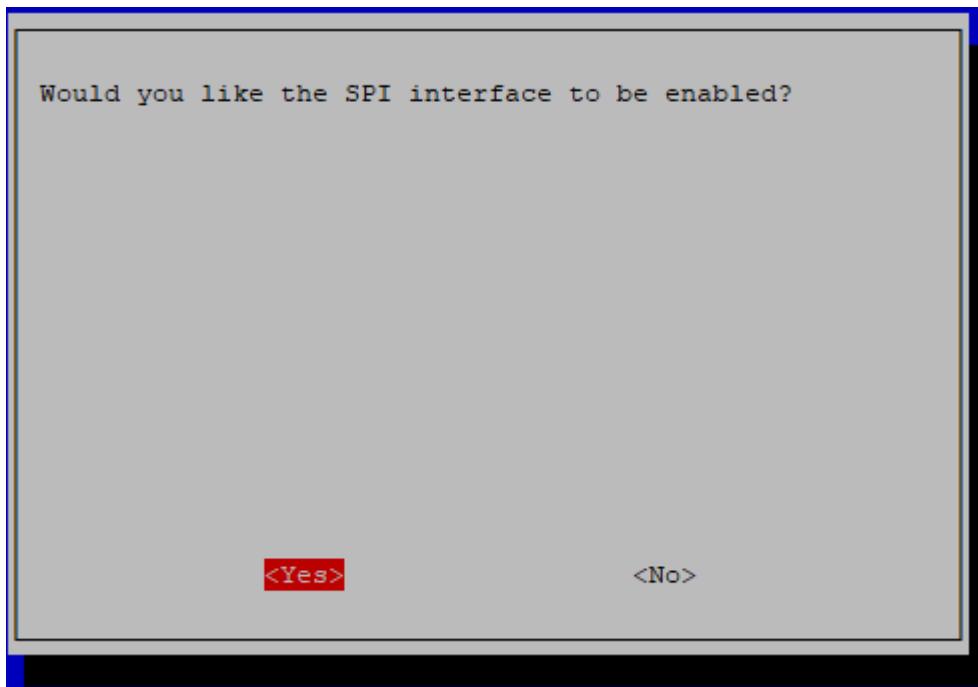
## 5 Interfacing options



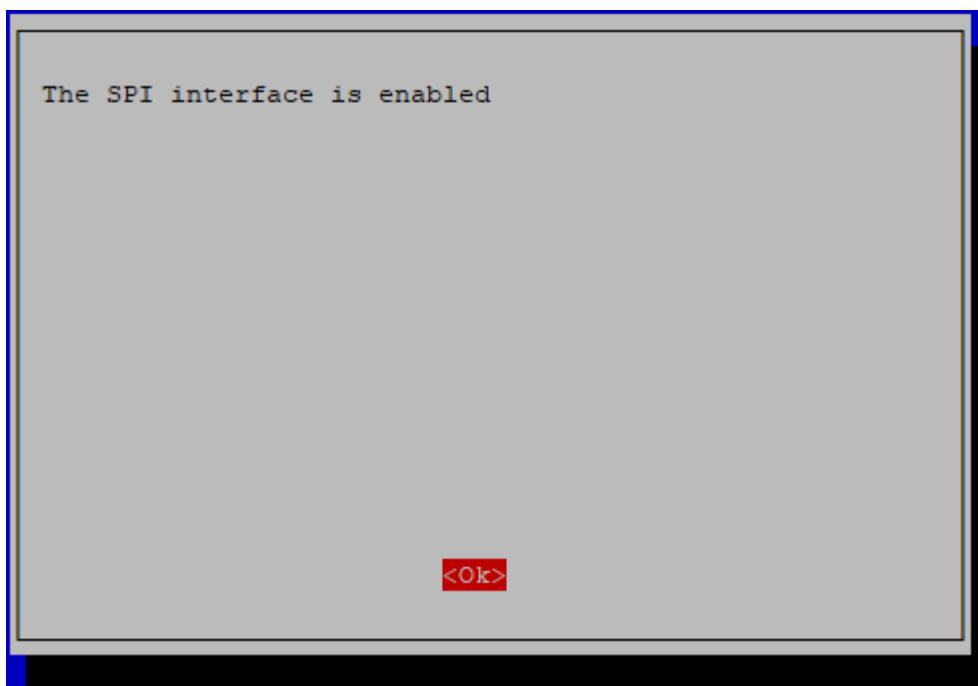
## P4 SPI



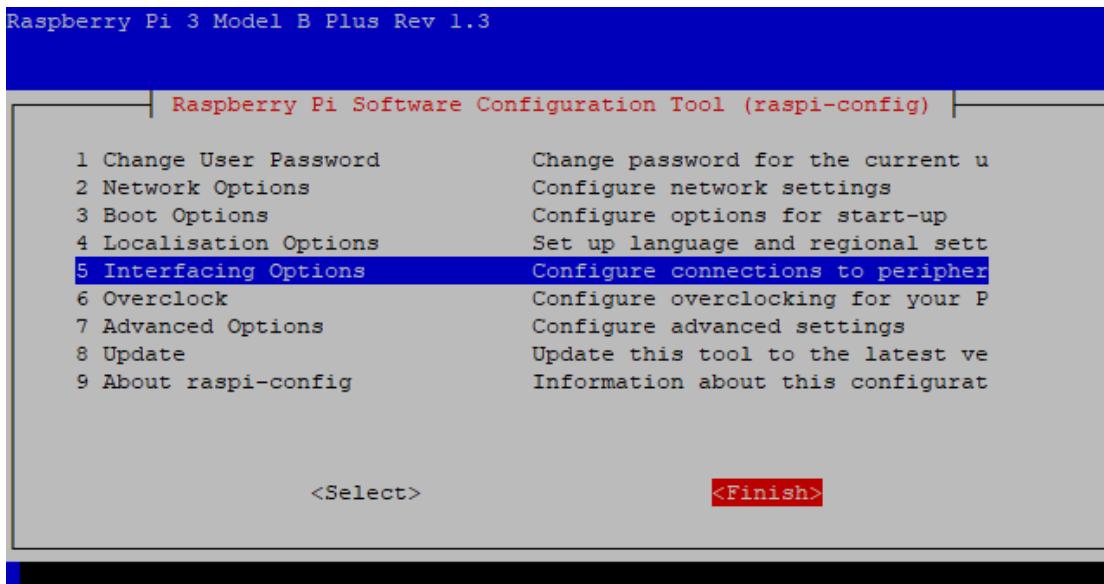
<YES>



<OK>



## <Finish>



**Step 2:** Check that the i2c modules are loaded and active.

```
ls /dev/sp*
```

Then the following codes will appear (the number may be different).

```
/dev/spidev0.0 /dev/spidev0.1
```

**Step 3:** Install Python module SPI-Py.

```
git clone https://github.com/lthiery/SPI-Py.git  
cd SPI-Py  
sudo python setup.py install
```

**Note:** This step is for python users, if you use C language, please skip.

## 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.