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**. You're also welcomed to share your projects on FORUM.

Contents

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

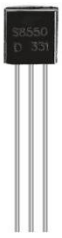
Yellow LED
4 pcs

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

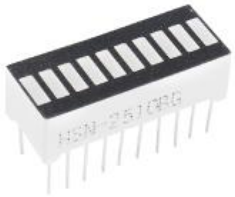
L293D
1 pcs

ADC0834
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

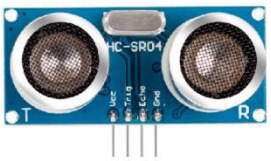
MFRC522 RFID Module
1 pcs

Relay
1 pcs

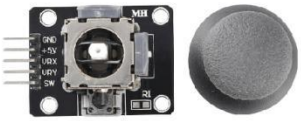
Stepping Motor Driver
1 pcs

Stepping Motor
1 pcs

Ultrasonic Ranging Module
1 pcs

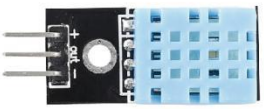


Joystick
1 pcs

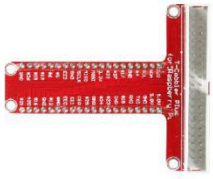


Infrare Motion Sensor
1 pcs

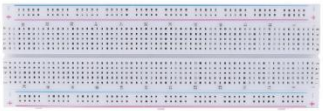
DHT-11
1 pcs



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.

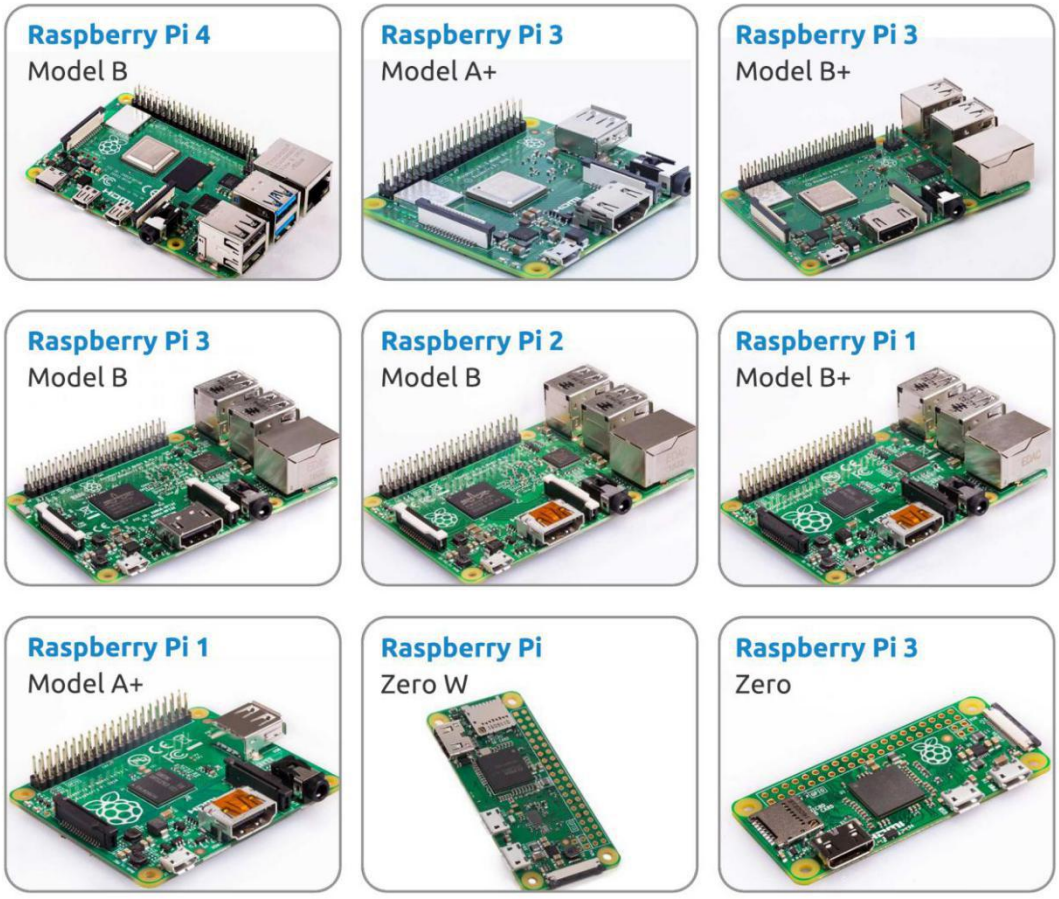
We are looking forward to your projects and hope that you can share your achievements or creation on our forum while reading this document.

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 Micro SD card to store all its files and the Raspberry Pi OS. 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 Monitor

If you have a monitor, you can use the NOOBS (New Out Of Box System) to install the Raspberry Pi OS.

Required Components

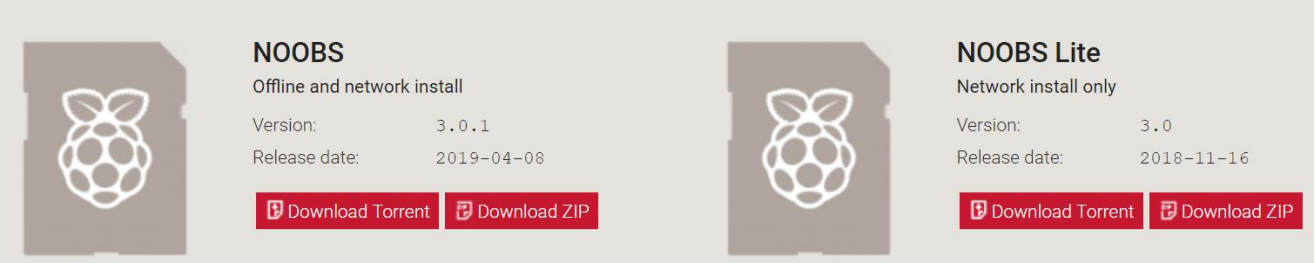
Any Raspberry Pi	1 * Power Adapter
1 * Monitor	1 * Monitor Power Adapter
1 * HDMI cable	1 * Micro SD card
1 * Mouse	1 * Keyboard
1 * Personal Computer	

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 Raspberry Pi OS 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

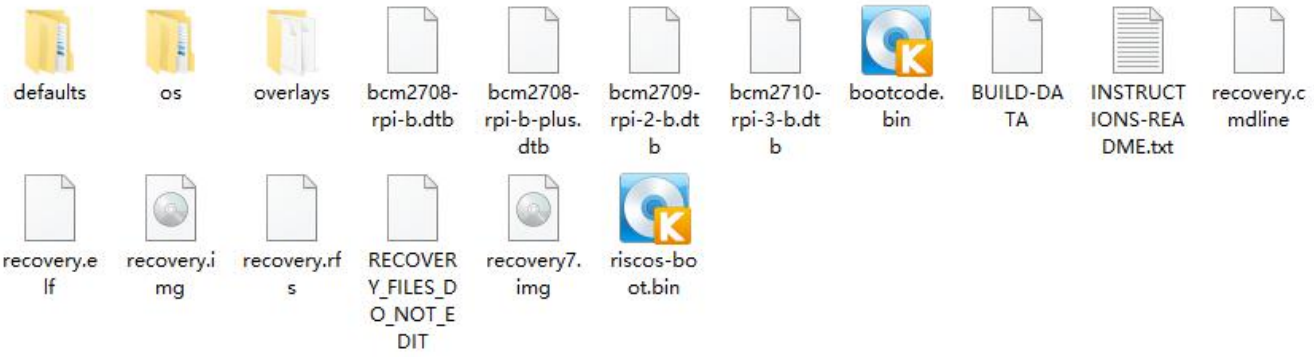
Plug in the Micro SD reader and format the Micro SD card with the **SD Formatter** (https://www.sdcard.org/downloads/formatter/index.html). If there are some important files in the Micro SD card, please backup them first.

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 Micro SD card.



Step 4

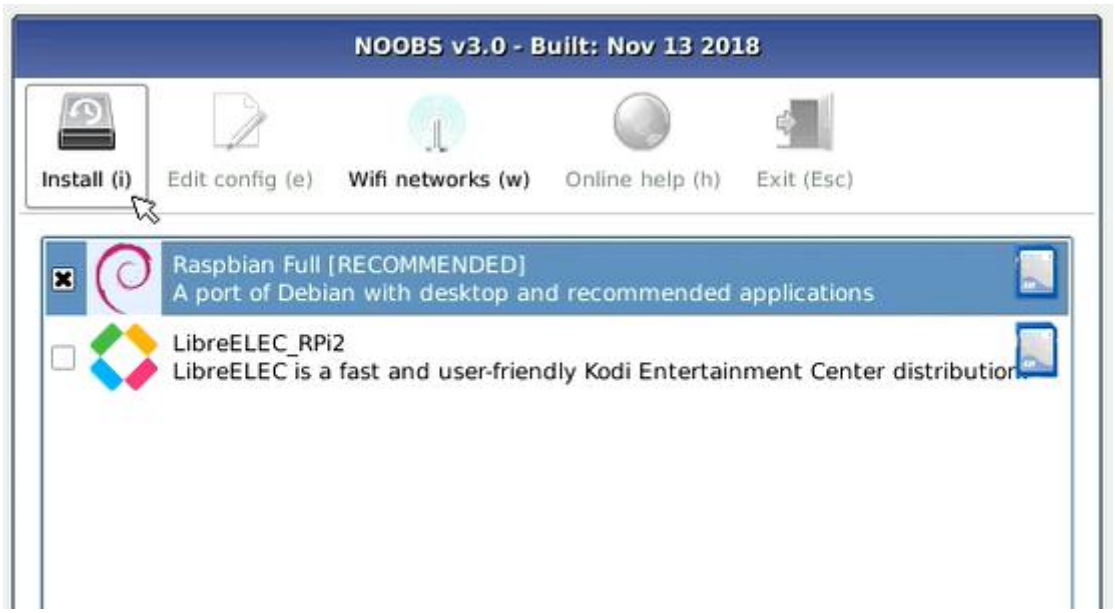
All the files transferred, the Micro SD card pops up.

Step 5

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

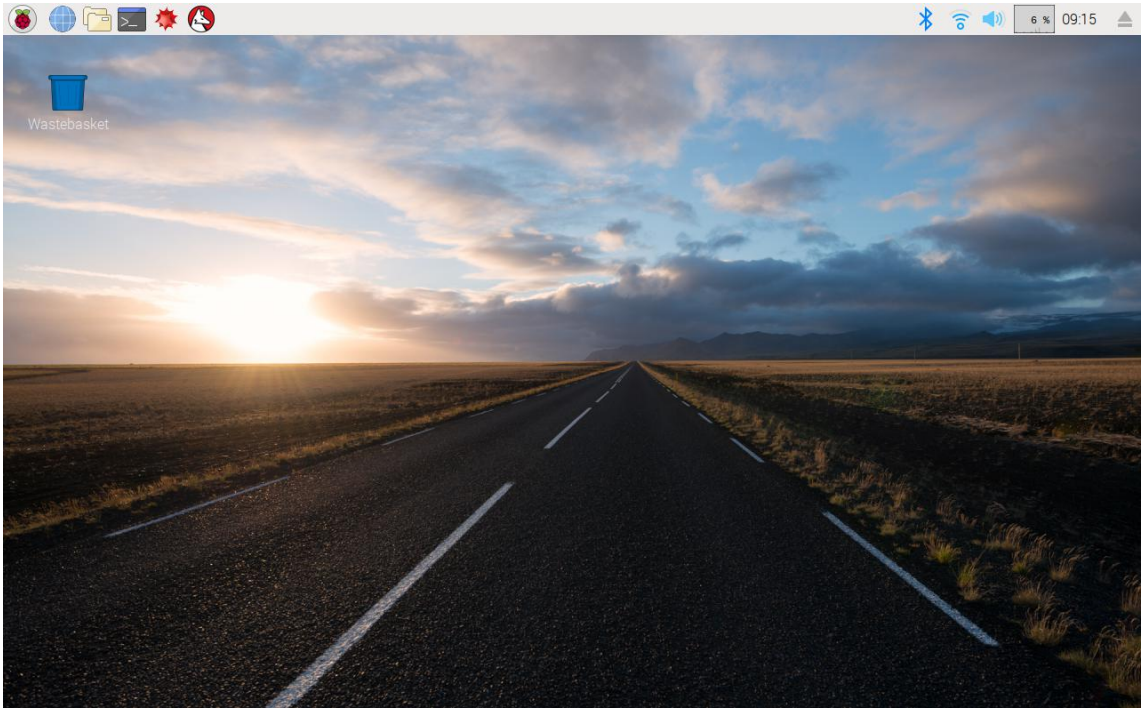
Step 6

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

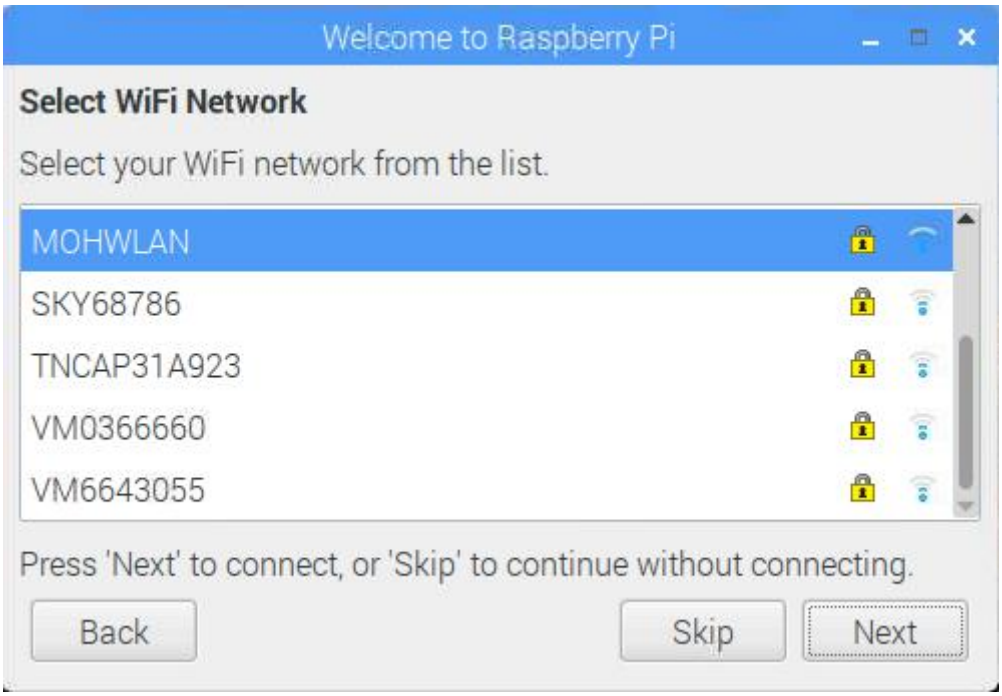
Set country/region, language and time zone, and then click "next" again.

Step 10

Input the new password of Raspberry Pi and click “Next”.

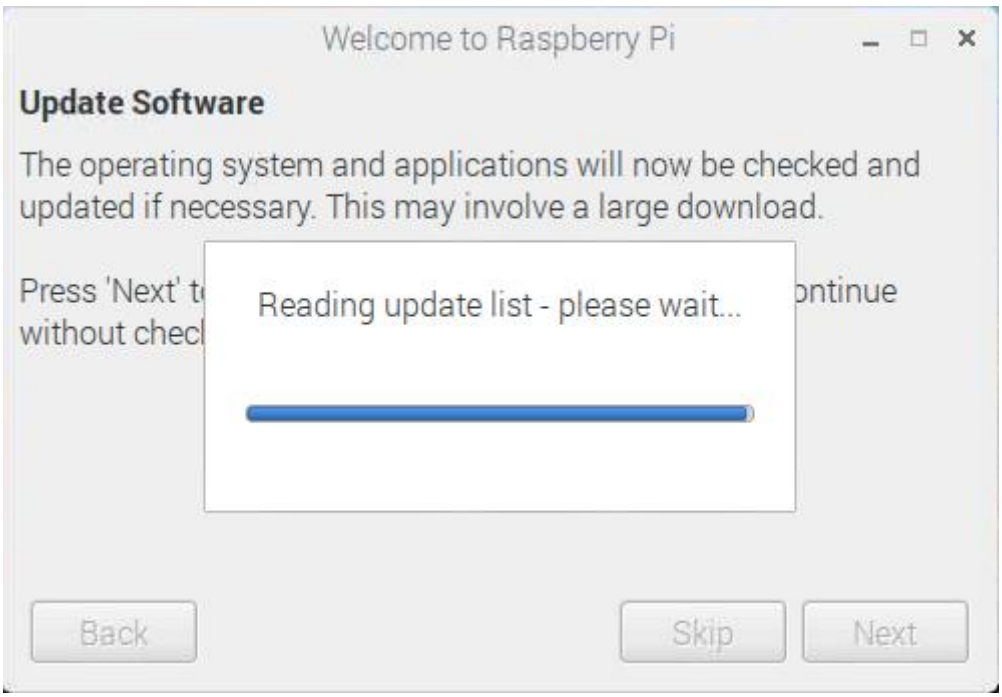
Step 11

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://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up*

If You Have No Monitor

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

Required Components

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

Installing System

There are 2 ways to install the system, **Using Raspberry Pi Imager** or **Using Raspberry Pi OS**. **Using Raspberry Pi Imager** is a kind of method recommended by Raspberry Pi official website for beginners with which you can directly write the Raspberry Pi OS into SD card after downloading Raspberry Pi Imager. However, each time the system is reinstalled, this method can take several hours.

In the later method, you need to download Raspberry Pi OS image at first, then use the tool to write it to your SD card, which can be confusing. But once you successfully finish the flashing at the first time, it only takes about 10 minutes to flash again.

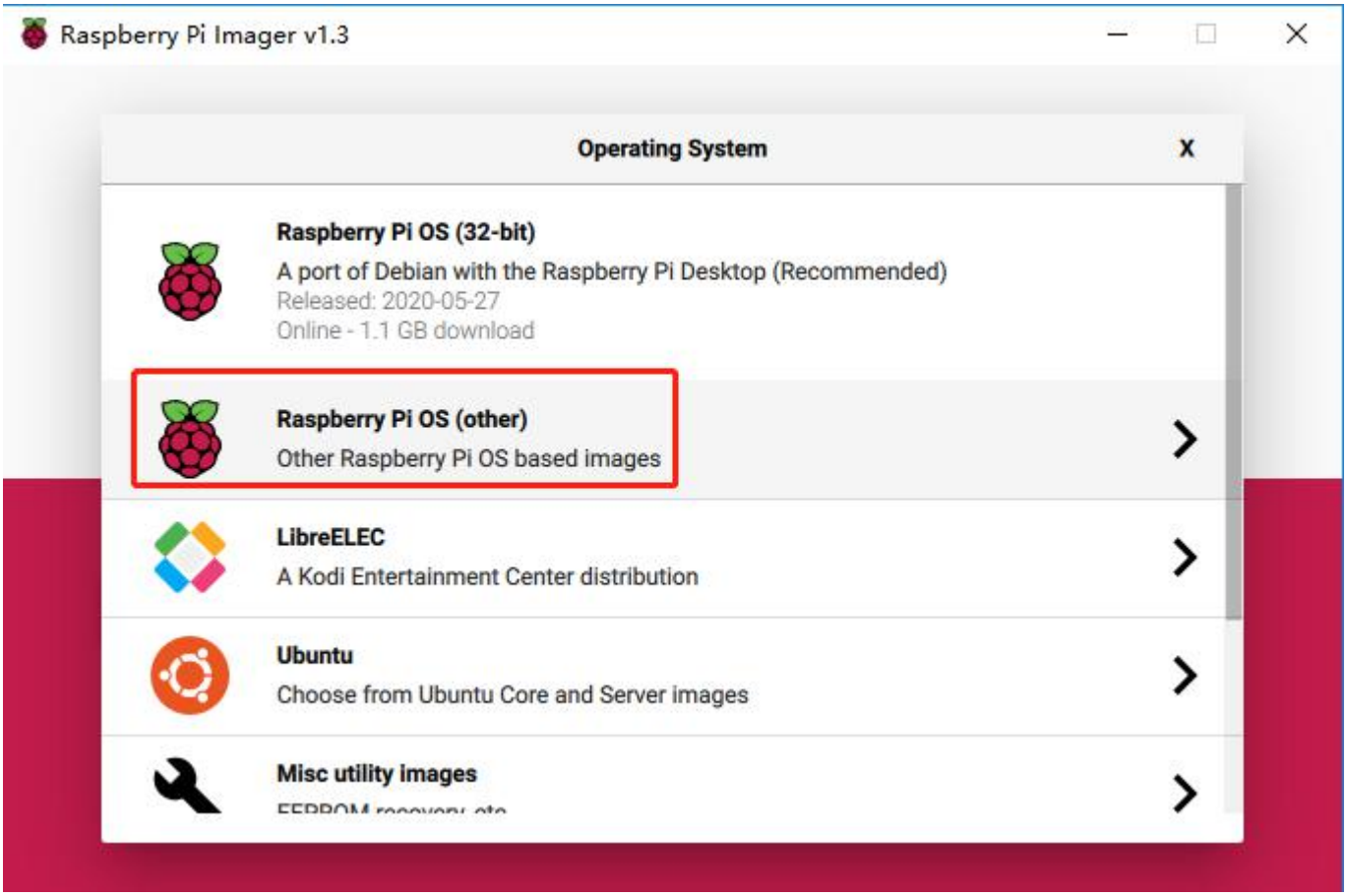
➢ Using Raspberry Pi Imager

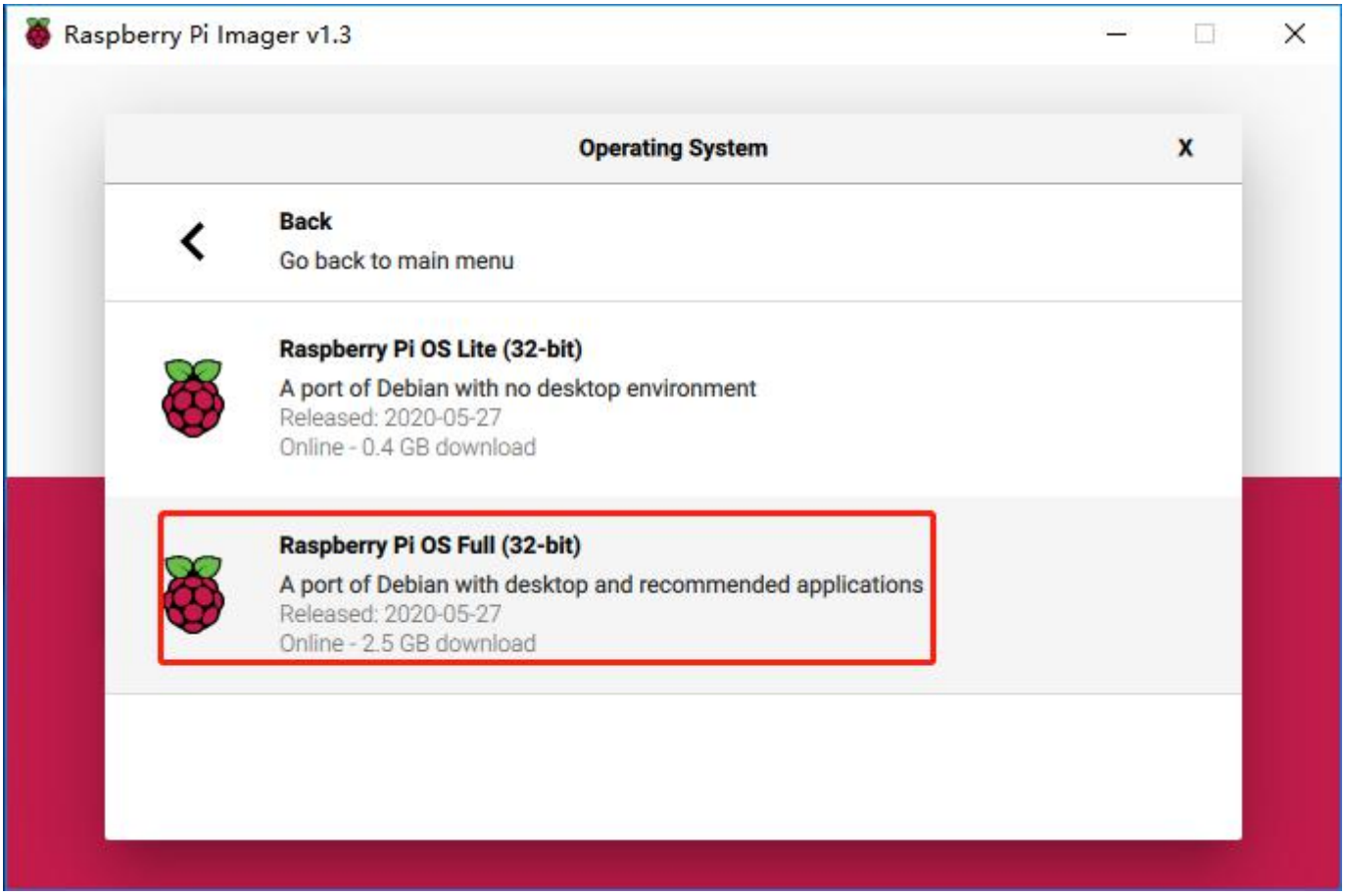
Raspberry Pi have developed a graphical SD card writing tool that works on Mac OS, Ubuntu 18.04 and Windows, and is the easiest option for most users as it will download the image and install it automatically to the SD card.

1) Download the latest version of **Raspberry Pi Imager**(*https://www.raspberrypi.org/downloads/*) and install it.

2) Connect an SD card reader with the SD card inside.

3) Open Raspberry Pi Imager and choose **Raspberry Pi OS (other) -> Raspberry Pi OS Full (32-bit)**.





4) Choose the SD card you wish to write your image to.

5) Review your selections and click 'WRITE' to begin writing data to the SD card.

Note: If using the Raspberry Pi Imager on Windows 10 with Controlled Folder Access enabled, you will need to explicitly allow the Raspberry Pi Imager permission to write the SD card. If this is not done, Raspberry Pi Imager will fail with a "failed to write" error.

➢ Using Raspberry Pi OS

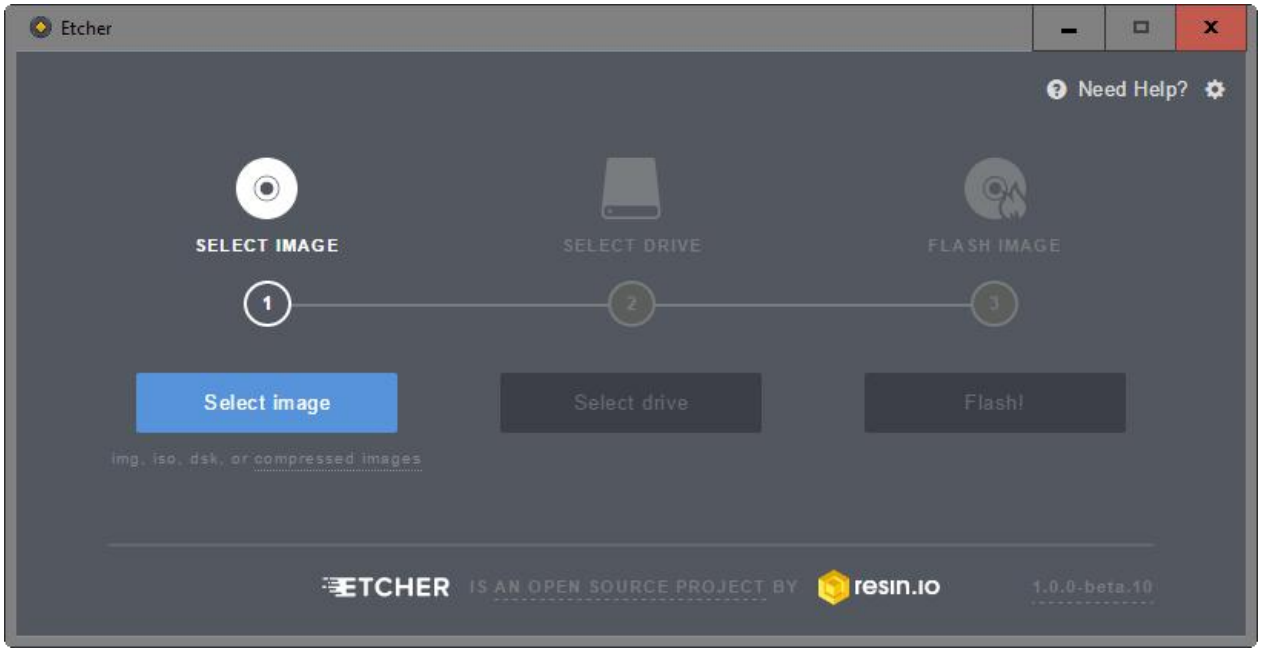
Step 1: Prepare the tool of image burning. Here we use the **balenaEtcher**. You can download the software from the link: *https://www.balena.io/etcher/*

Step 2: Download the complete image on the official website by clicking this link: *https://www.raspberrypi.org/downloads/raspberry-pi-os/*. There are three different kinds of Raspberry Pi OS system available, You are recommend to install the version: **Raspberry Pi OS with desktop and recommended software**.

Step 3: Unzip the package downloaded and you will see the *.img* file inside.

Note: The Raspberry Pi OS with desktop image contained in the ZIP archive is over 4GB in size and uses the ZIP64 format. To uncompress the archive, a unzip tool that supports ZIP64 is required. The following zip tools support ZIP64: 7-Zip (Windows), The Unarchiver (Mac) and Unzip (Linux).

Step 4: Plug the USB card reader into the computer, then you can burn the **.img** file with the Etcher.

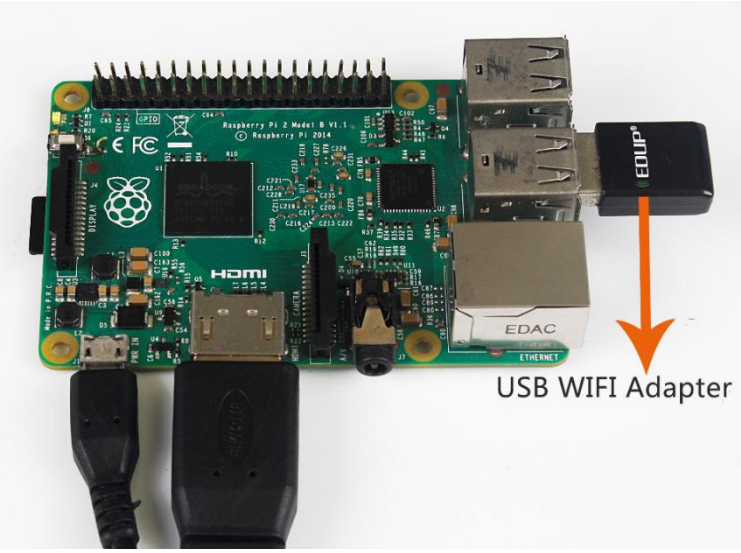


At this point, Raspberry Pi OS is installed. **Keep the USB card reader plug in your computer**. If you want to apply it, next you need to complete the settings accordingly.

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 for more.



If you want to use the WIFI function, you need to modify a WIFI configuration file wpa_supplicant.conf in the Micro 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.

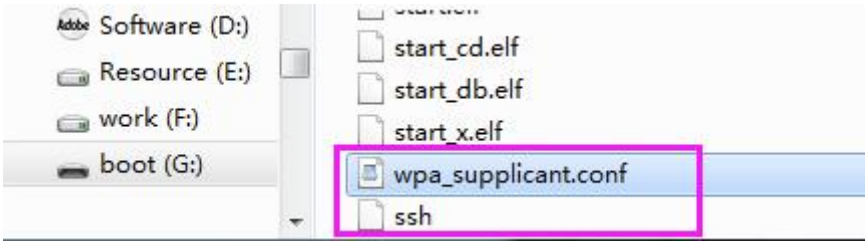
```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=COUNTRY
network={
ssid="SSID"
psk="PASSWORD"
key_mgmt=WPA-PSK
```

```
priority=1
}
```

You need to replace "**WiFi-A**" with your custom name of WiFi and "**Sunfounder**" with your password. By doing these, the Raspberry Pi OS 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 Raspberry Pi OS is configured. When the Micro 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 Raspberry Pi OS 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 Raspberry Pi OS is **raspberrypi**, now you need to find the hostname.

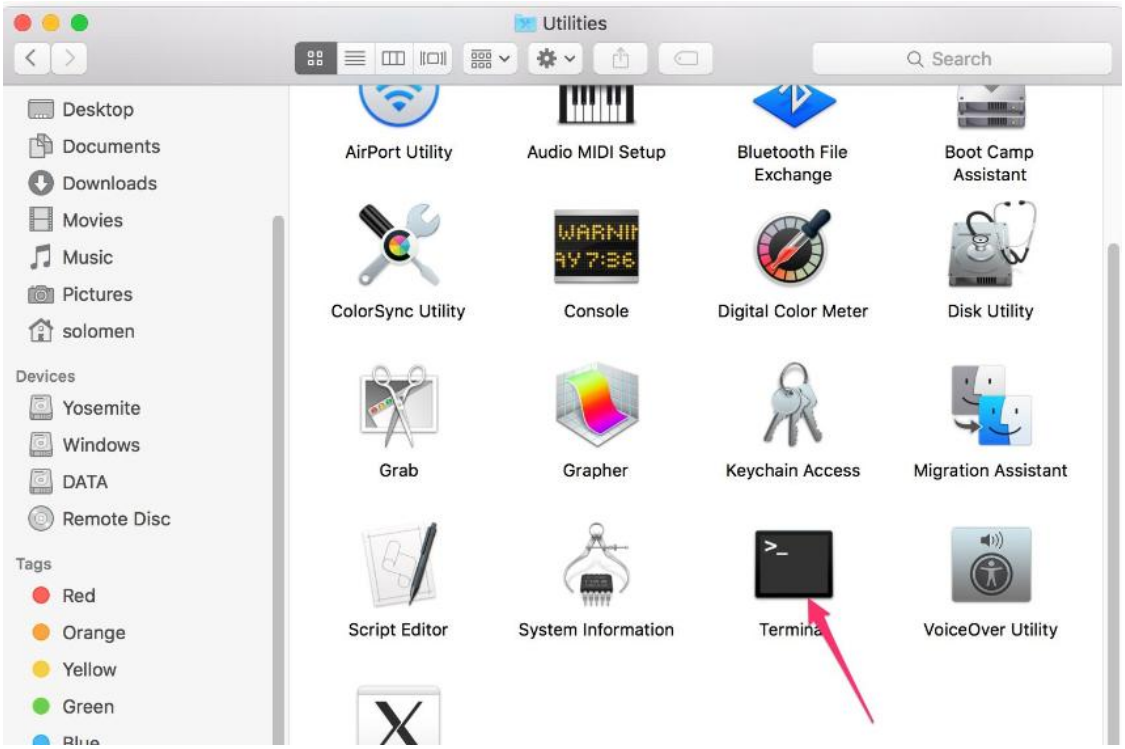
Use the SSH Remote Control

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.



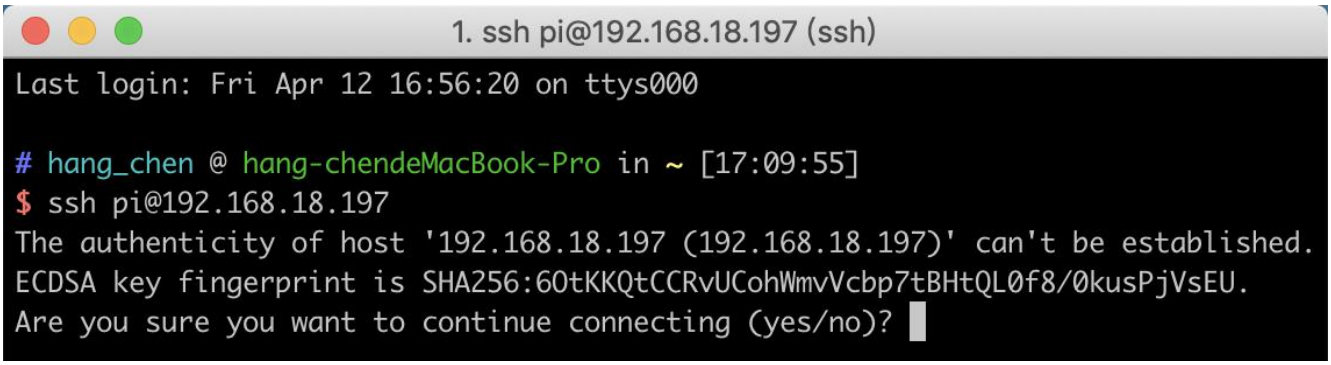
Step 2

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
```

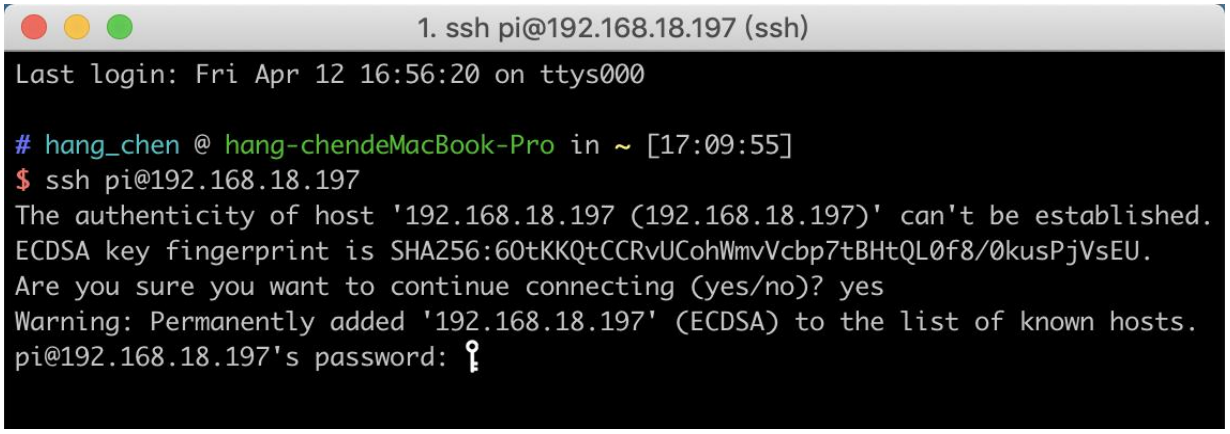
Step 3

Input"yes".



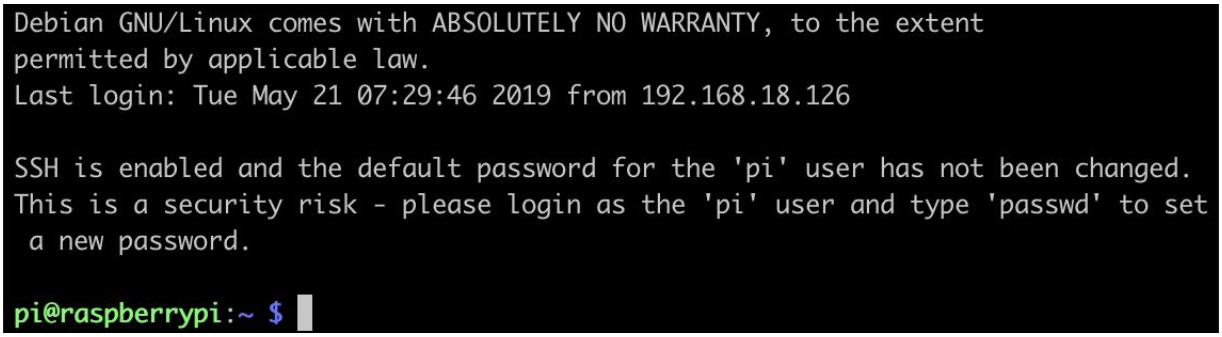
Step 4

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



Step 5

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



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

For Windows Users

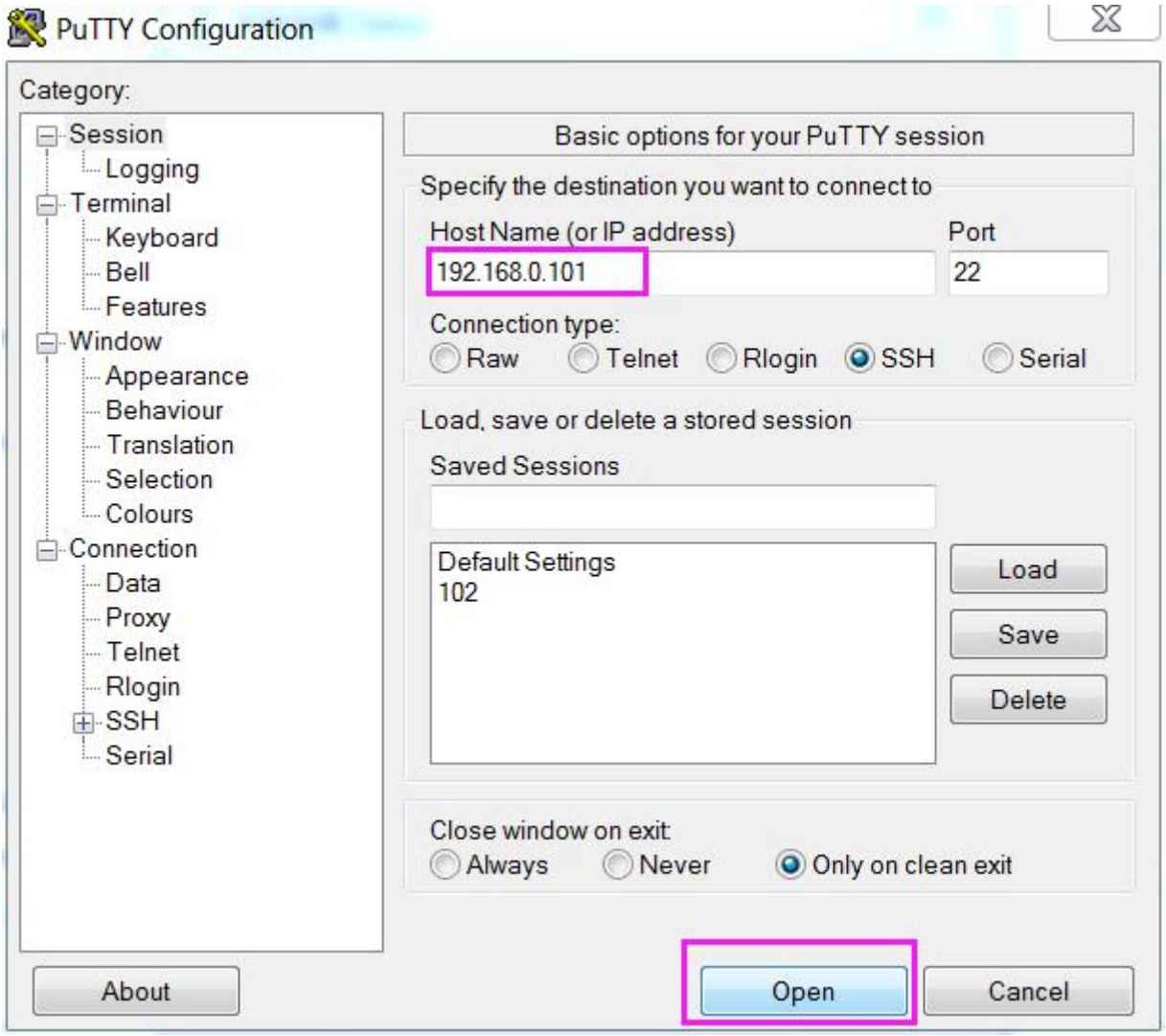
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

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).

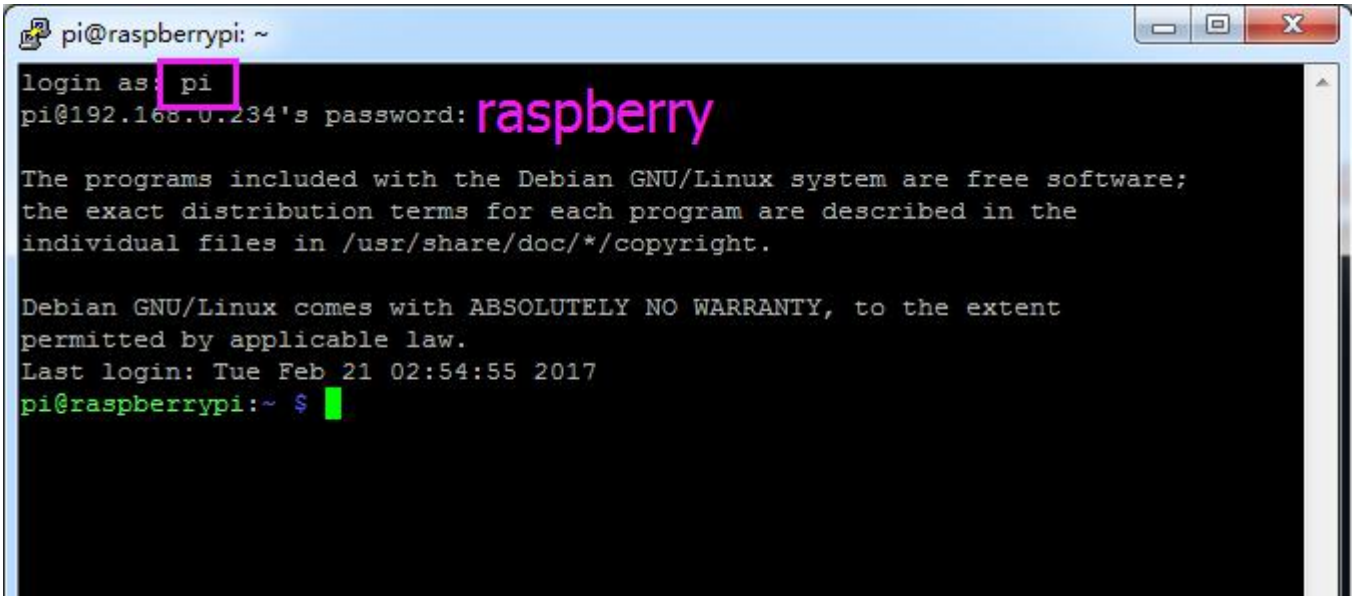


Step 3

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).



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 password.

Remote Desktop

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

VNC

You can use the function of remote desktop through VNC.

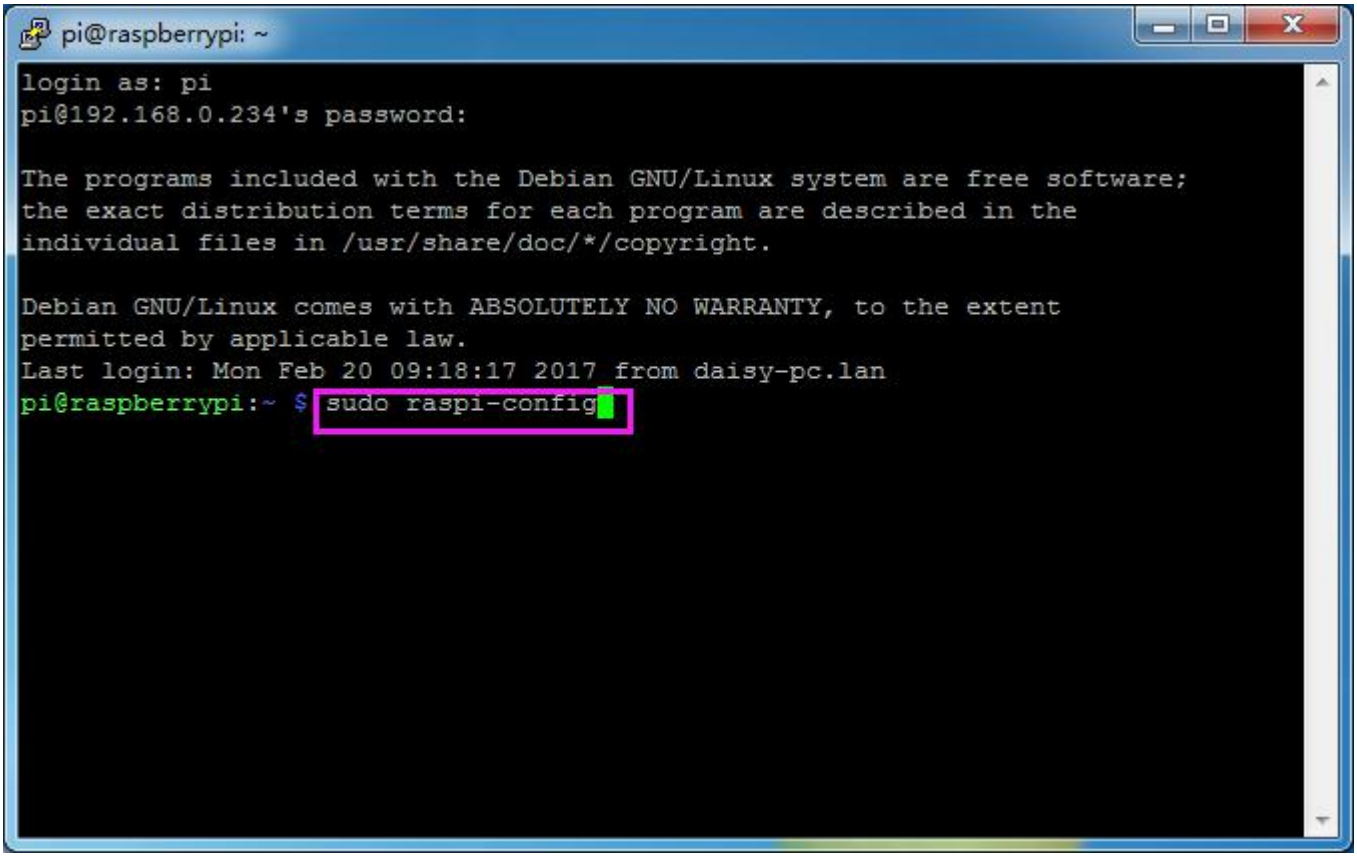
Enable VNC service

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

Step 1

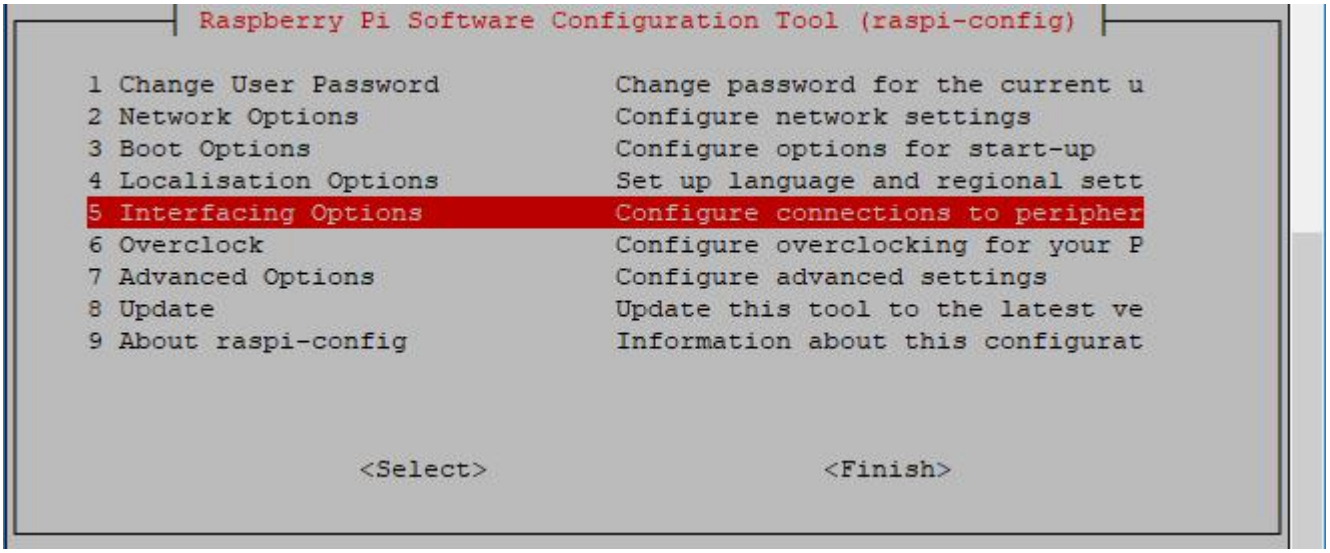
Input the following command:

```
sudo raspi-config
```



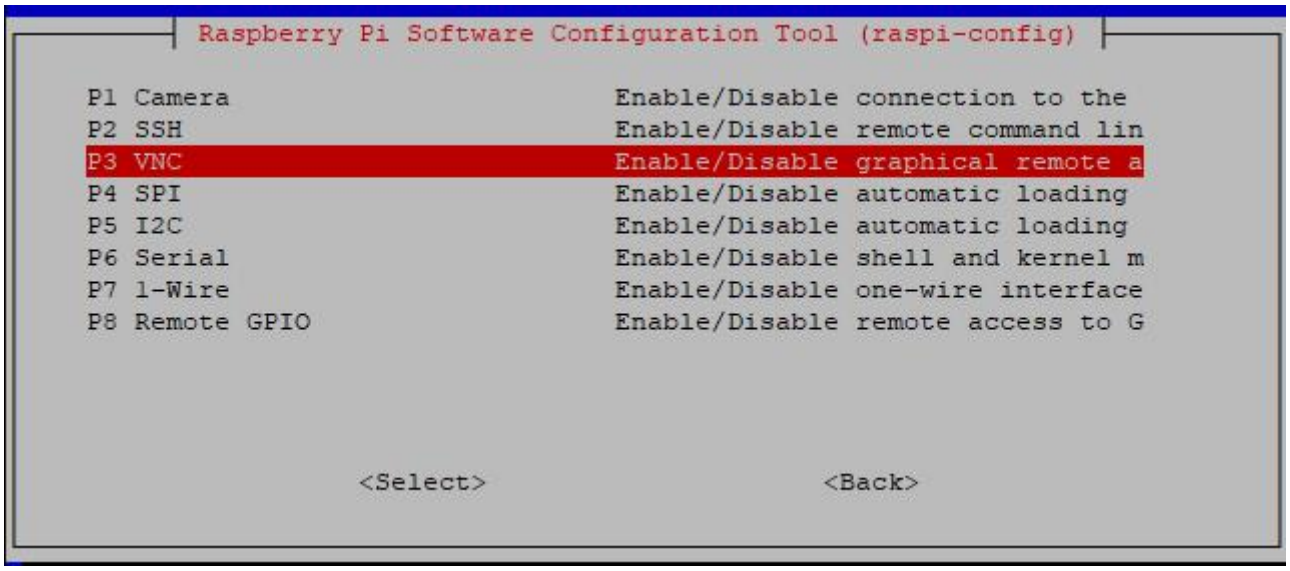
Step 2

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



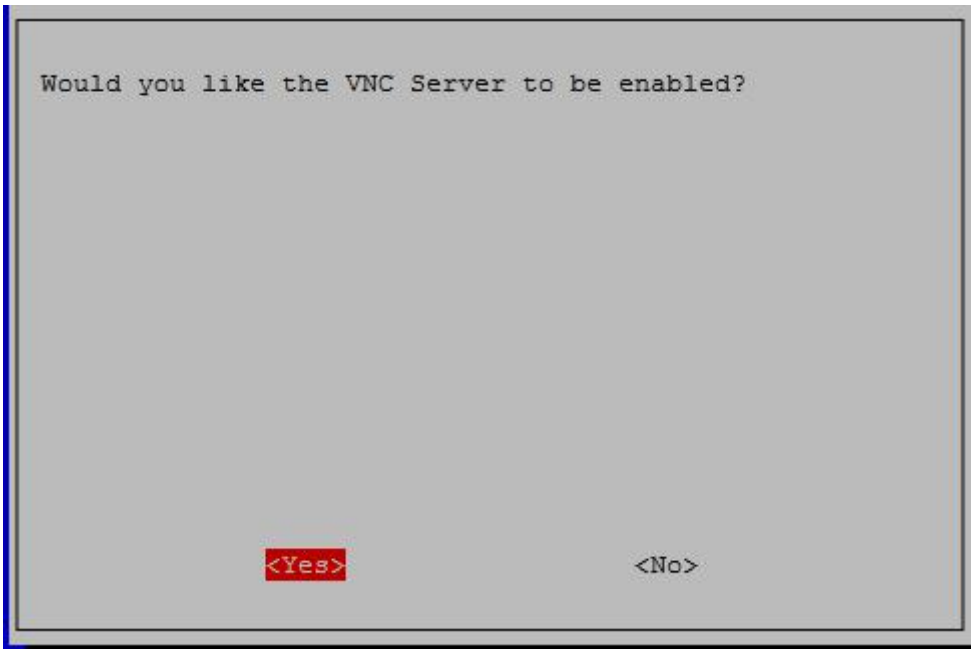
Step 3

Select **VNC**.



Step 4

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



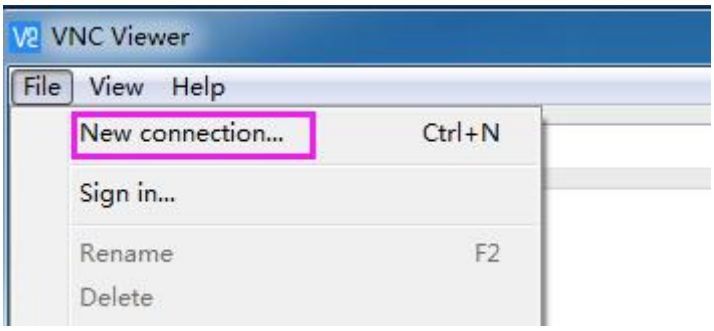
Login to VNC

Step 1

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

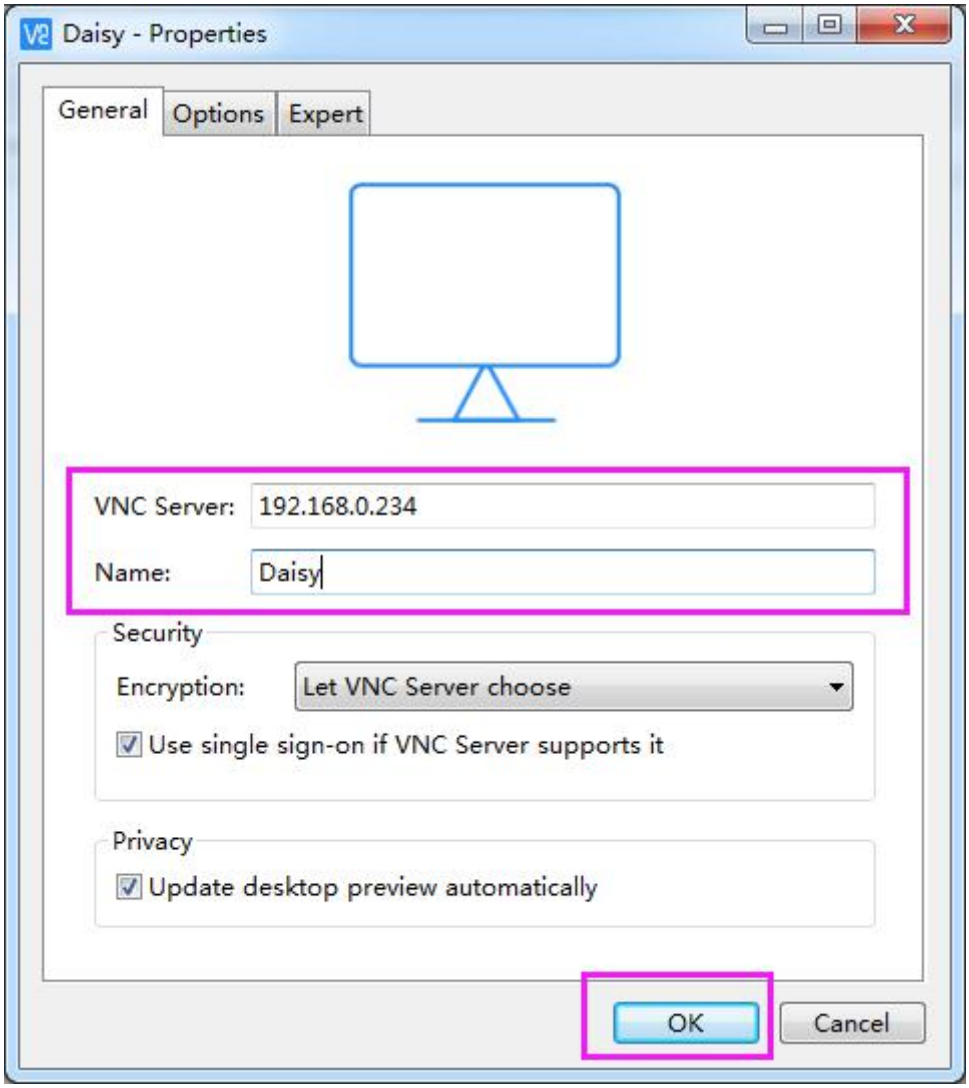
Step 2

Then select "**New connection**".



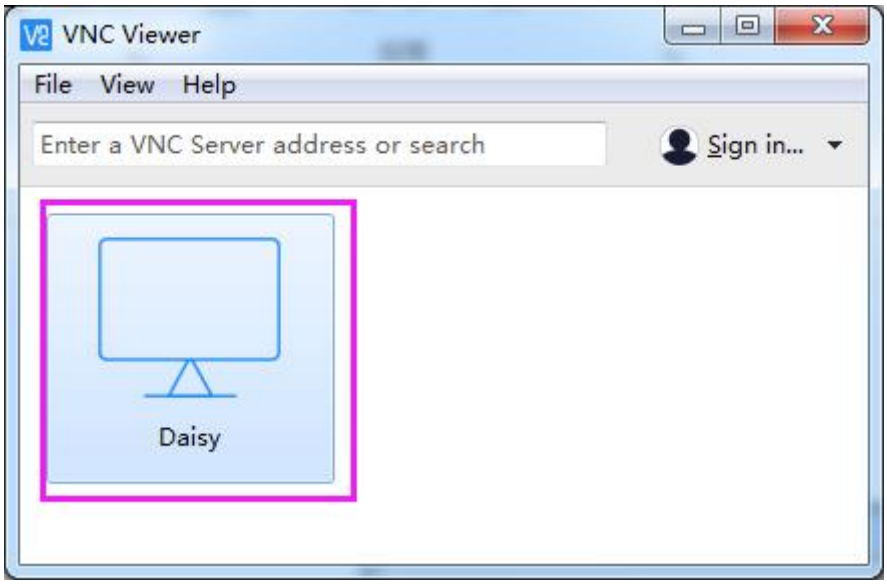
Step 3

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



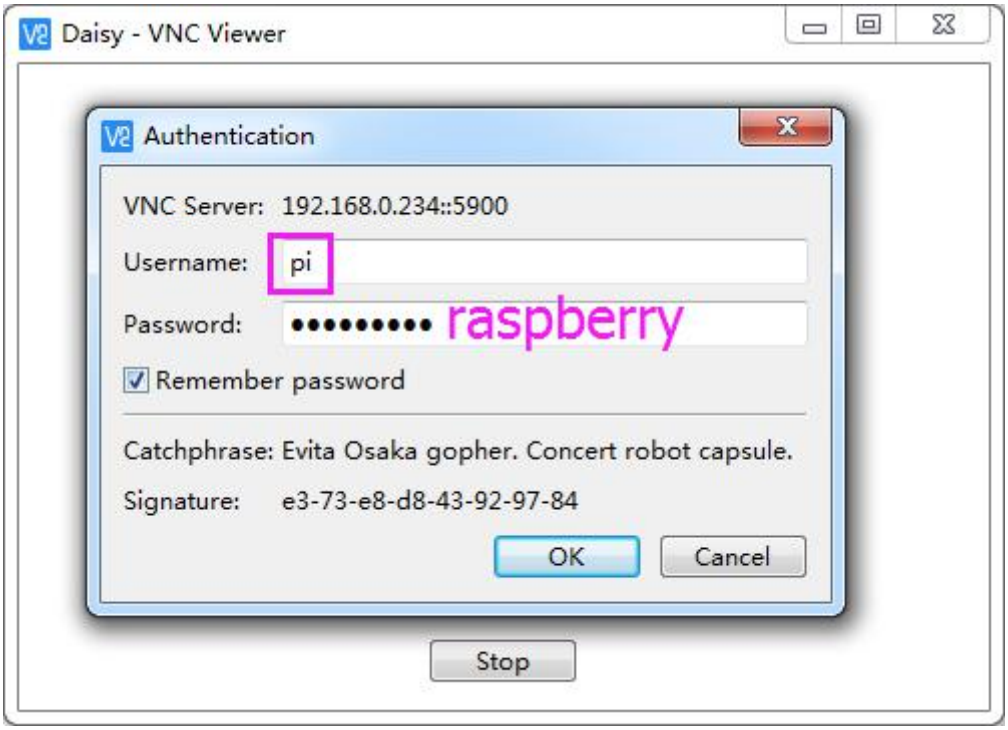
Step 4

Double click the **connection** just created:



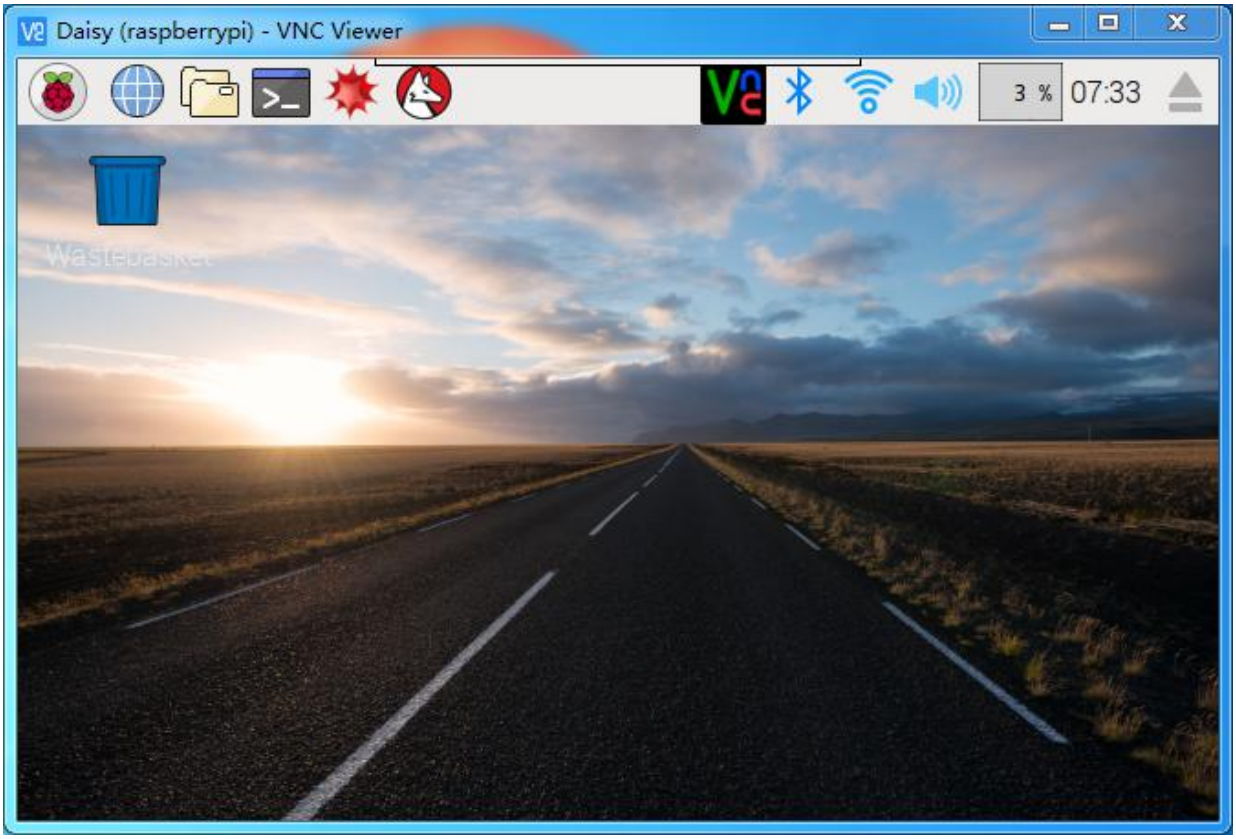
Step 5

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

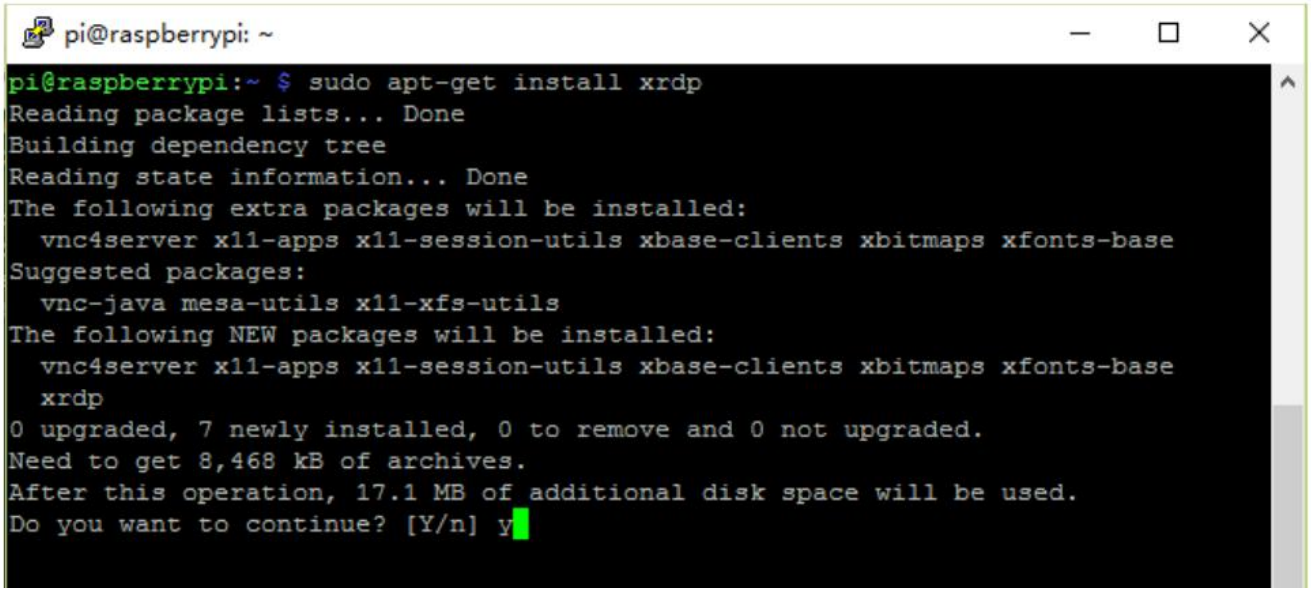
Input the following instructions to install XRDP.

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

Step 3

Later, the installation starts.

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



Step 4

Finished the installation, you should login to your Raspberry Pi by using Windows remote desktop applications.

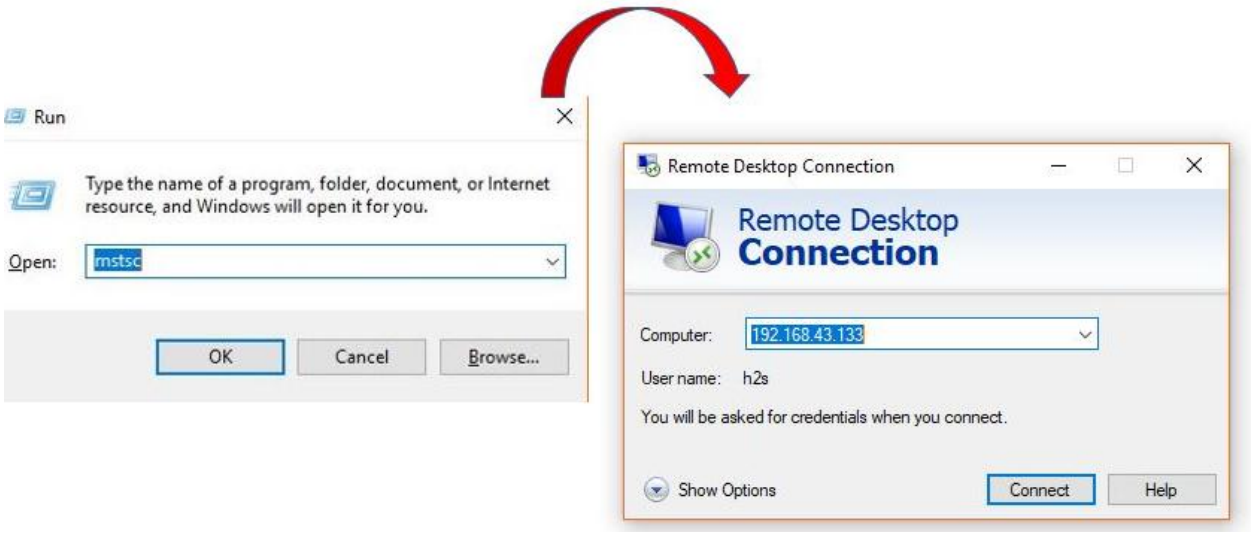
Login to XRDP

Step 1

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

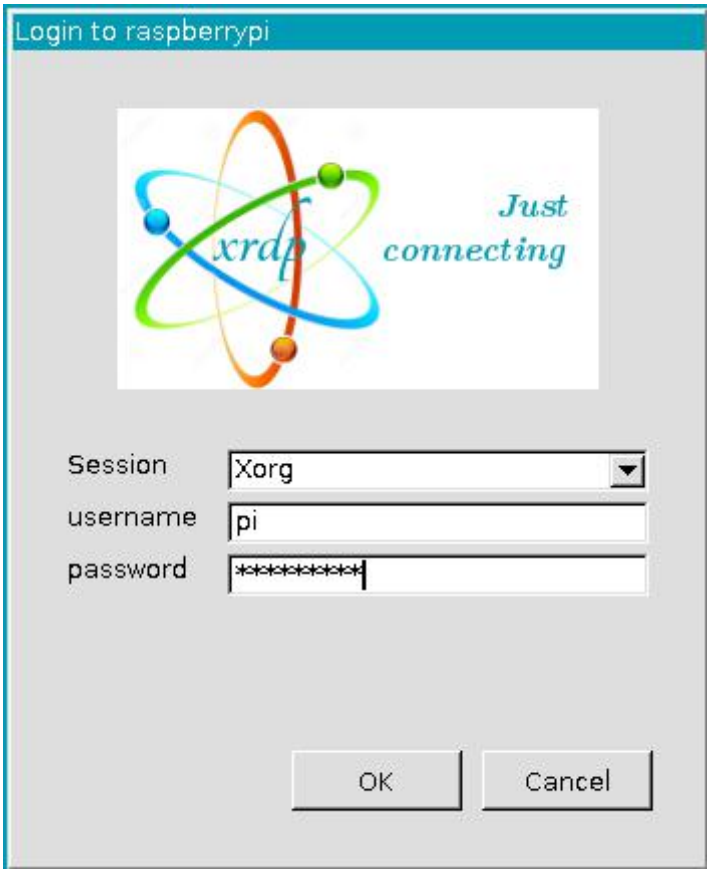
Step 2

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".



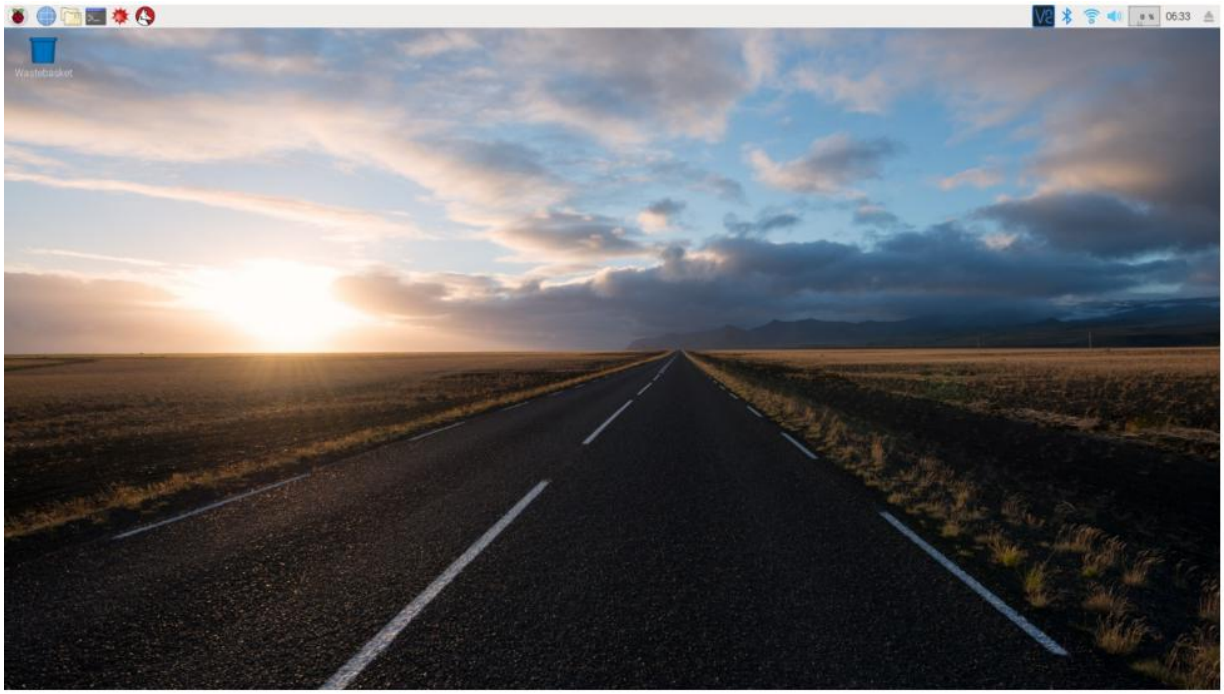
Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click "OK". At the first time you log in, your username 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 Raspberry Pi OS 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()
```

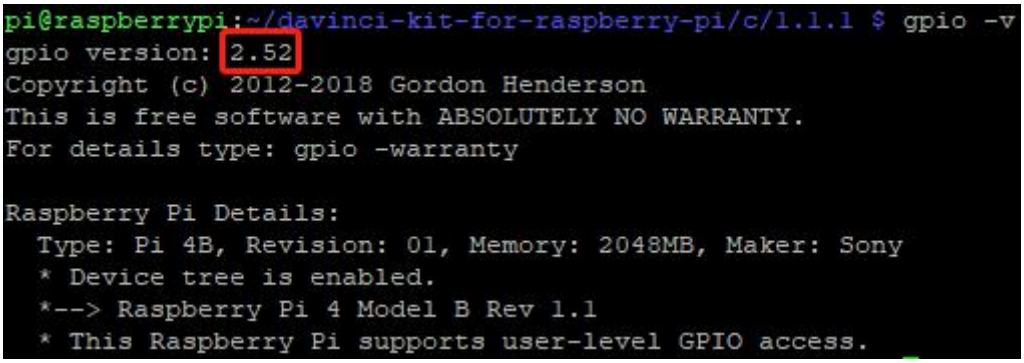
```
>>> exit()
pi@raspberrypi:~ $
```

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
```



Note:

If you are using Raspberry Pi 4B, but the GPIO version is **2.50**, it will cause no response after the C language code is running, that is, GPIO pins do not work. At this time, you need to manually update to version **2.52**, the update steps are as follows :

```
cd /tmp
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
```

Check with:

```
gpio -v
```

and make sure it's version 2.52.

gpio readall

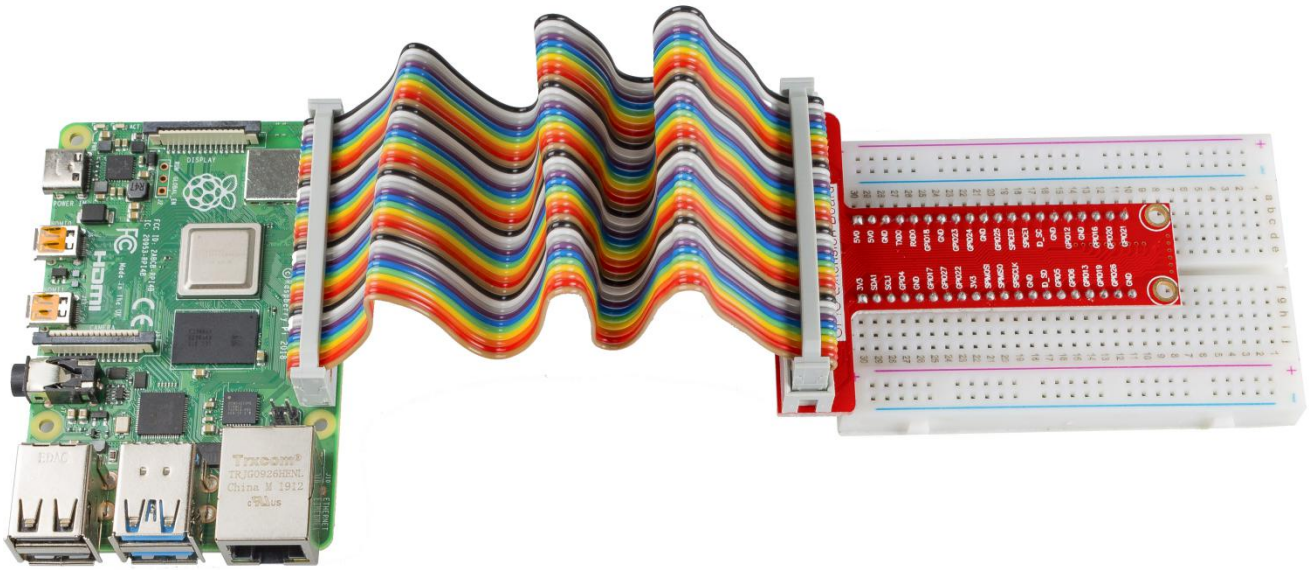
```
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |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

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

GPIO Extension Board

Before starting to learn the commands, you first need to know more about the pins of the Raspberry Pi, which is key to the subsequent study.

We can easily lead out pins of the Raspberry Pi to breadboard by GPIO Extension Board to avoid GPIO damage caused by frequent plugging in or out. This is our 40-pin GPIO Extension Board and GPIO cable for Raspberry Pi model B+, 2 model B and 3, 4 model B.



Pin Number

The pins of Raspberry Pi have three kinds of ways to name and they are wiringPi, BCM and Board. Among these naming methods, 40-pin GPIO Extension board uses the naming method, BCM. But for some special pins, such as I2C port and SPI port, they use the Name that comes with themselves. The following table shows us the naming methods of WiringPi, Board and the intrinsic Name of each pin on GPIO Extension board. For example, for the GPIO17, the Board naming method of it is 11, the wiringPi naming method is 0, and the intrinsic naming method of it is GPIO0.

Note:

1）In C Language, what is used is the naming method WiringPi.

2）In Python Language, the applied naming methods are Board and BCM, and the function GPIO.setmode() is used to set them.

Name	WiringPi	Board	BCM		Board	WiringPi	Name
		GPIO Extention Board					
3.3V	3V3	1	3V3	5.0V	2	5.0V	5V
SDA	8	3	SDA	5.0V	4	5.0V	5V
SCL	9	5	SCL	GND	6	GND	0V
GPIO7	7	7	GPIO4	TXD	8	15	TXD
0V	GND	9	GND	RXD	10	16	RXD
GPIO0	0	11	GPIO17	GPIO18	12	1	GPIO1
GPIO2	2	13	GPIO27	GND	14	GND	0V
GPIO3	3	15	GPIO22	GPIO23	16	4	GPIO4
3.3V	3.3V	17	3.3V	GPIO24	18	5	GPIO5
MOSI	12	19	MOSI	GND	20	GND	0V
MISO	13	21	MISO	GPIO25	22	6	GPIO6
SCLK	14	23	SCLK	CE0	24	10	CEO
0V	GND	25	GND	CE1	26	11	CE1
IN_SDA	30	27	EED	EEC	28	31	ID_SCL
GPIO21	21	29	GPIO5	GND	30	GND	0V
GPIO22	22	31	GPIO6	GPIO12	32	26	GPIO26
GPIO23	23	33	GPIO13	GND	34	GND	0V
GPIO24	24	35	GPIO19	GPIO16	36	27	GPIO27
GPIO25	25	37	GPIO26	GPIO20	38	28	GPIO28
0V	GND	39	GND	GPIO21	40	29	GPIO29

Download the Code

Before you download the code, please note that the example code is **ONLY** test on Raspberry Pi OS. 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/
```

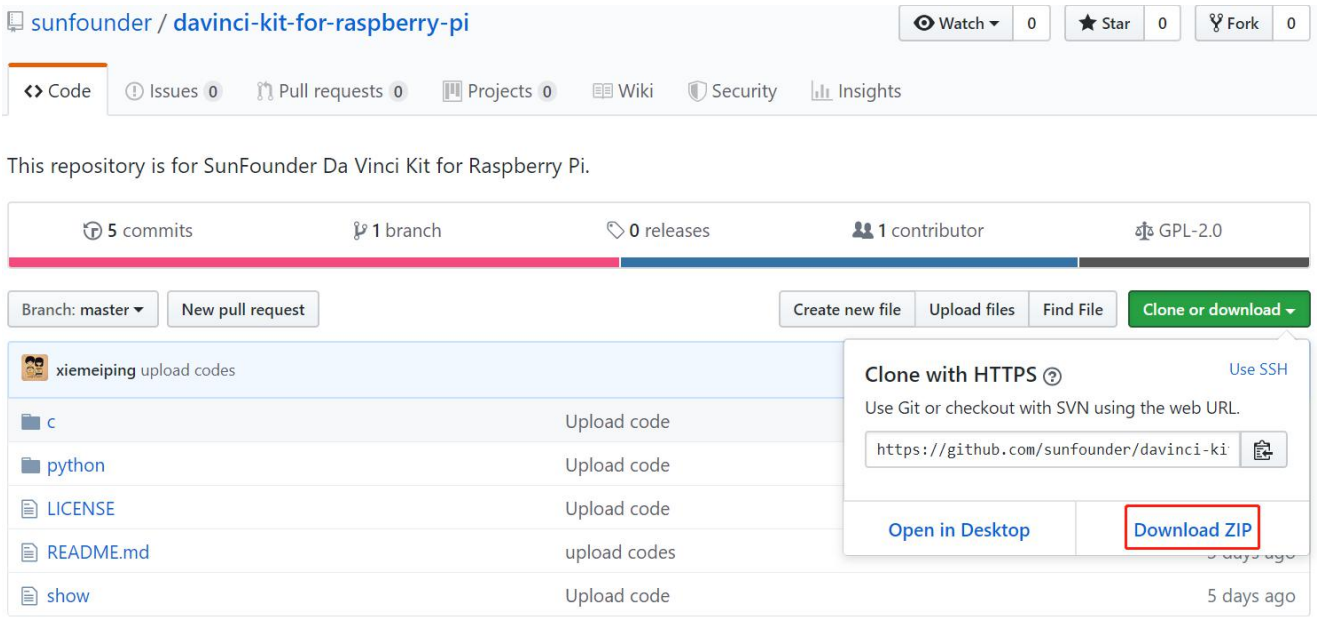
Note: cd to change to the intended directory from the current path. Informally, here is to go to the path /home/pi/.

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*



1 Output

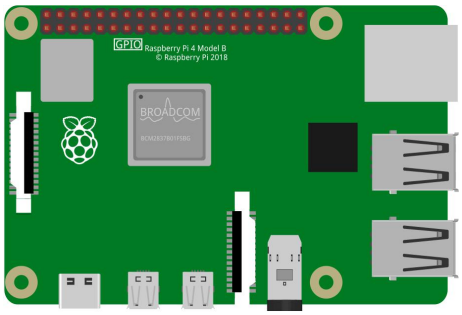
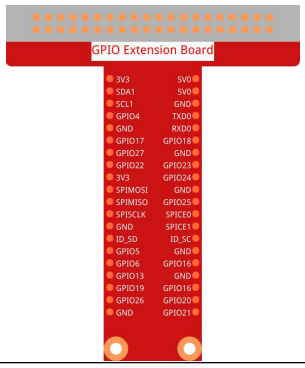
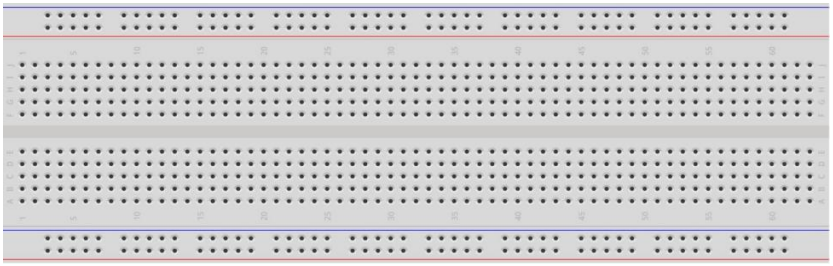
1.1 Displays

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 
1 * 40-pin Cable 	Several Jumper Wires	
1 * Breadboard 	1 * Resistor(220Ω) 	

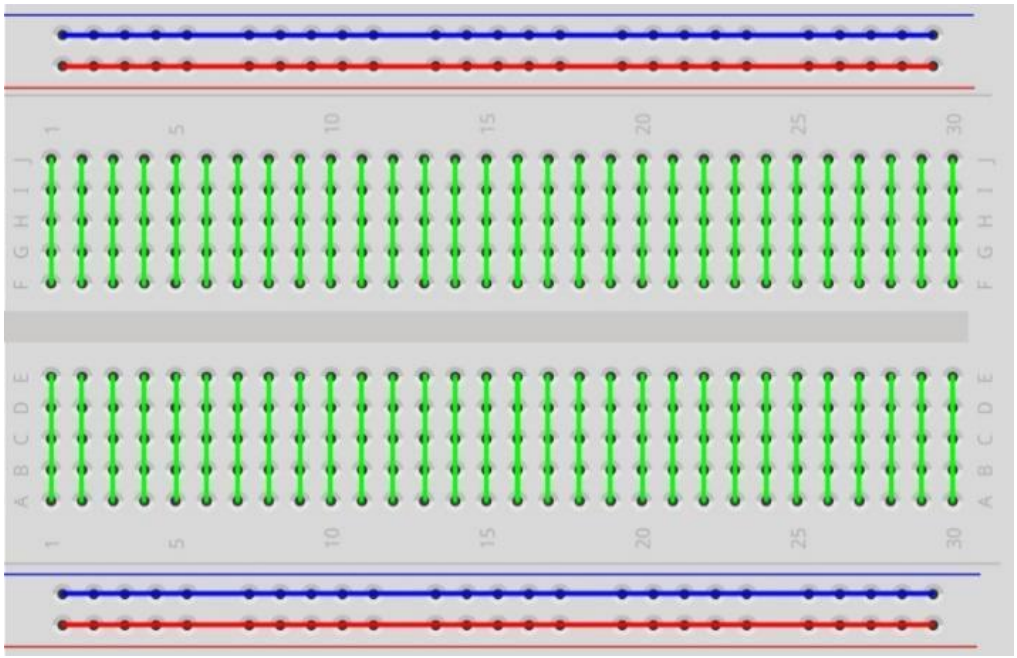
Note: In order to proceed smoothly, you need to bring your own Raspberry Pi, TF card and Raspberry Pi power.

Principle

Breadboard

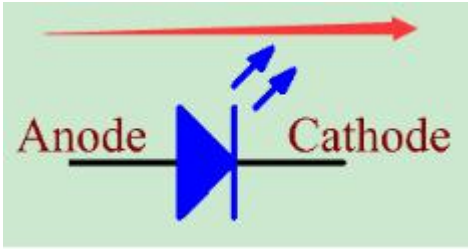
A breadboard is a construction base for prototyping of electronics. It is used to build and test circuits quickly before finishing any circuit design. And it has many holes into which components mentioned above can be inserted like ICs and resistors as well as jumper wires. The breadboard allows you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

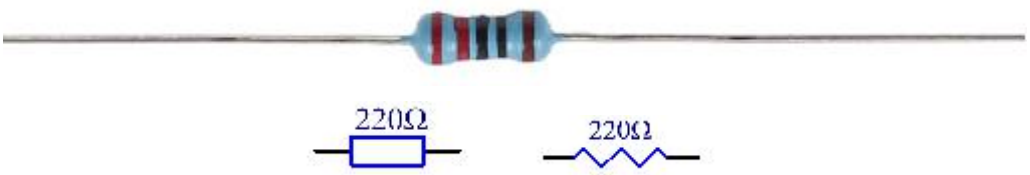


The LED can not be directly connected to power supply, which can damage component. A resistor with 160Ω or larger (work in 5V) must be connected in series in the circuit of LED.

Resistor

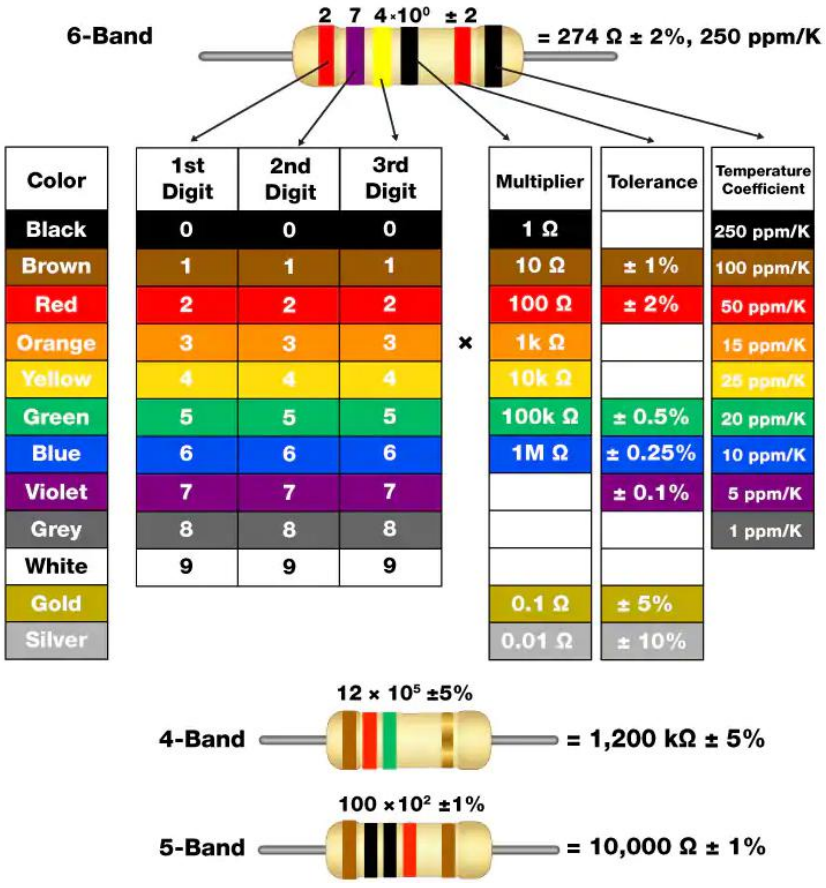
Resistor is an electronic element that can limit the branch current. A fixed resistor is a kind of resistor whose resistance cannot be changed, while that of a potentiometer or a variable resistor can be adjusted.

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, 220Ω resistor and two generally used circuit symbols of resistor. Ω is the unit of resistance and the larger units include KΩ, MΩ, etc. Their relationship can be shown as follows: 1 MΩ =1000 KΩ, 1 KΩ = 1000 Ω. 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.



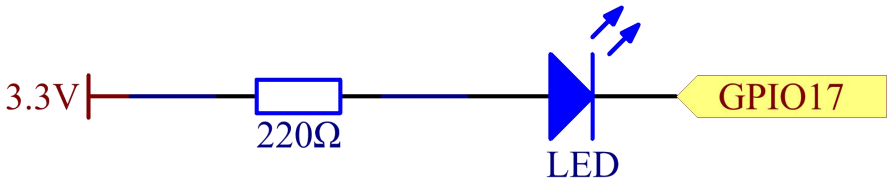
Schematic Diagram

In this experiment, connect a 220Ω resistor to the anode (the long pin of the LED), then the resistor to 3.3 V, and connect the cathode (the short pin) of the LED to 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.

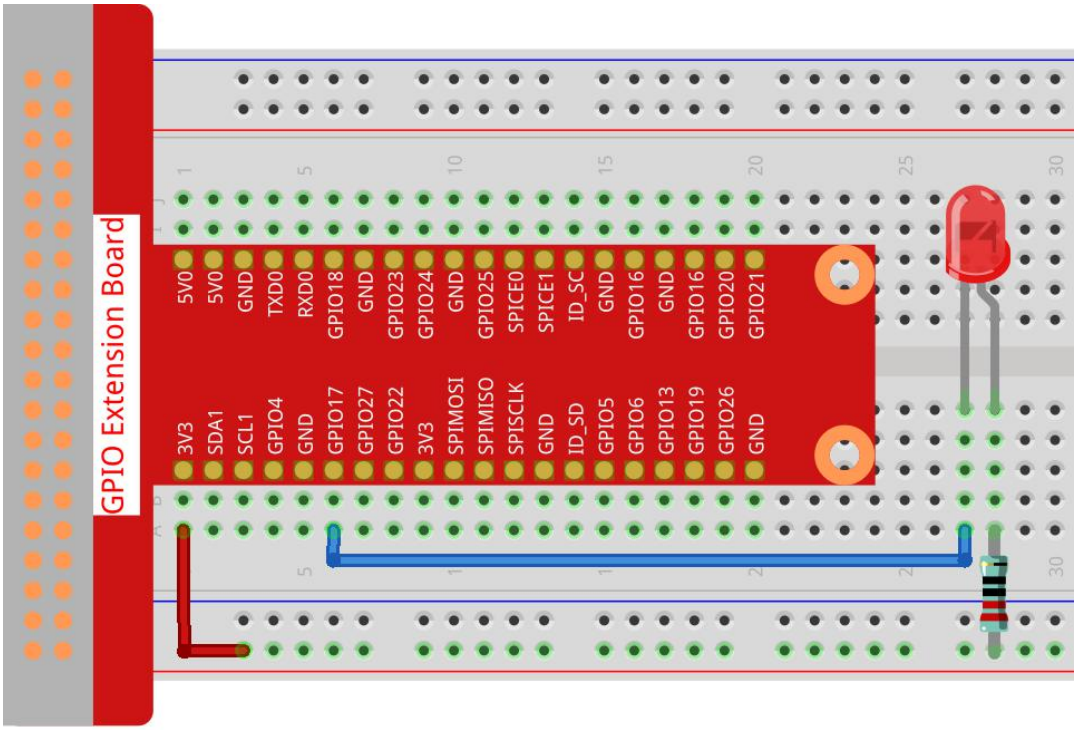
In the C language related content, we make GPIO0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi, Pin 11.

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



Experimental Procedures

Step 1: Build the circuit.



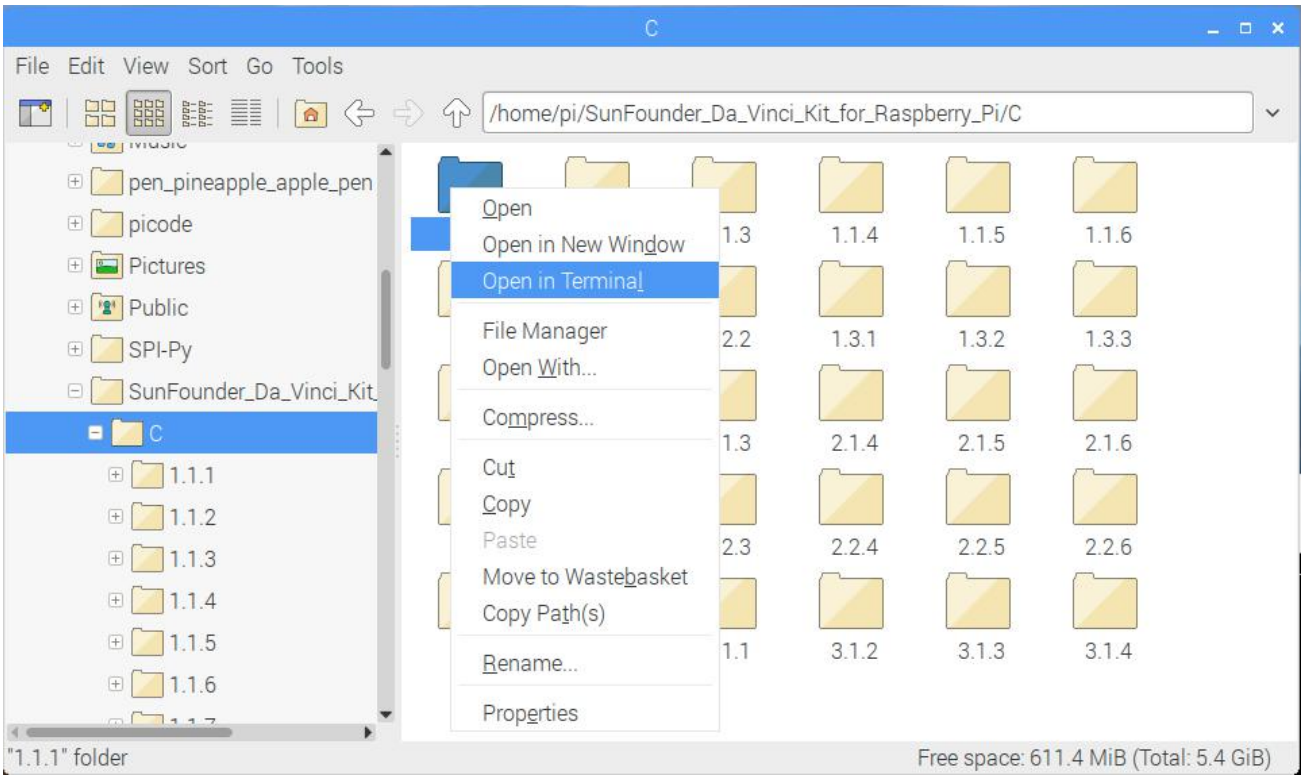
➢ For C Language Users

Step 2: Go to the folder of the code.

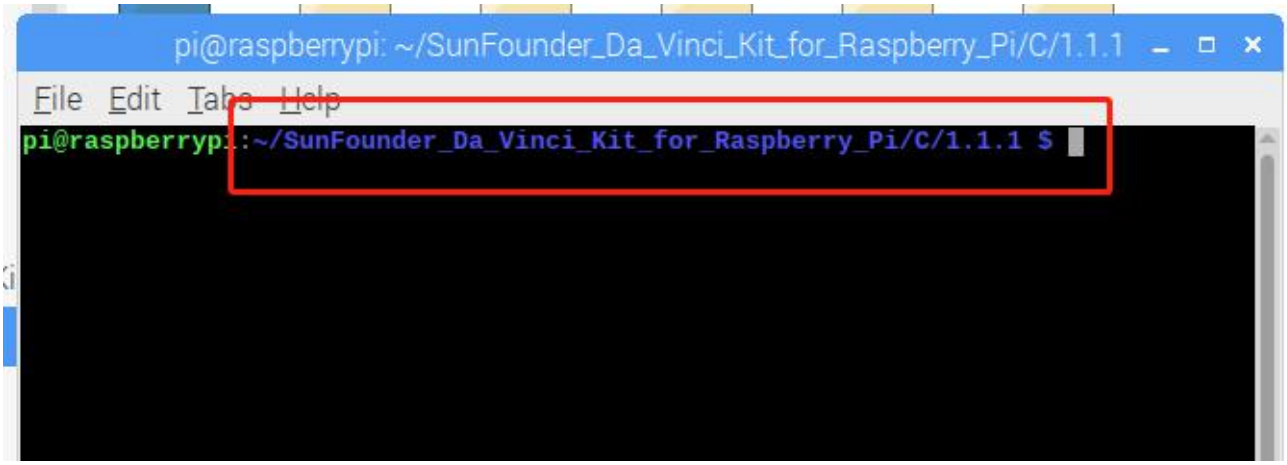
If you use a monitor, you're recommended to take the following steps.

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.

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
```

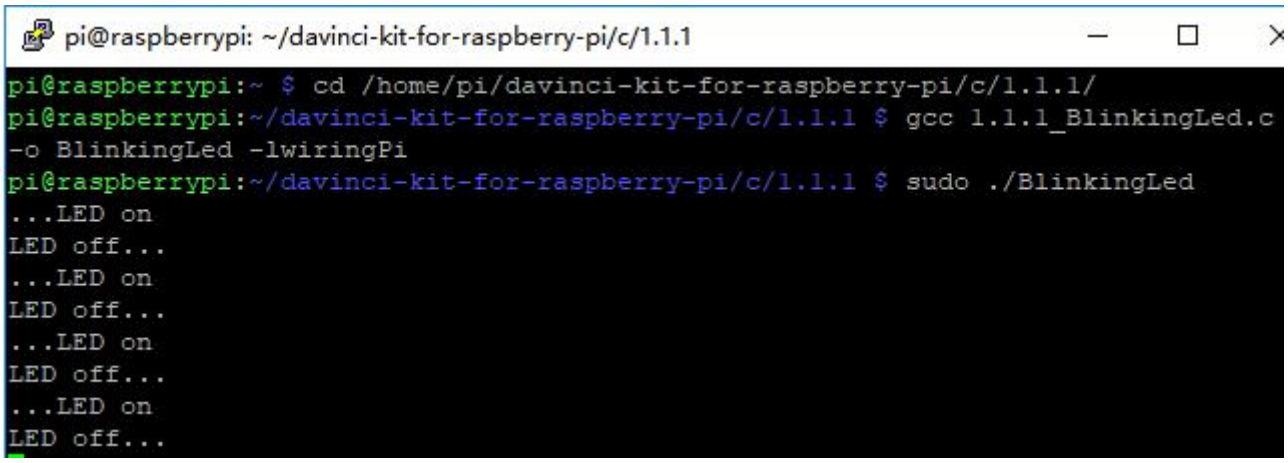
Note: gcc is GNU Compiler Collection. Here, it functions like compiling the C language file *1_BlinkingLed.c* and outputting an executable file.

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 (**l** 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.



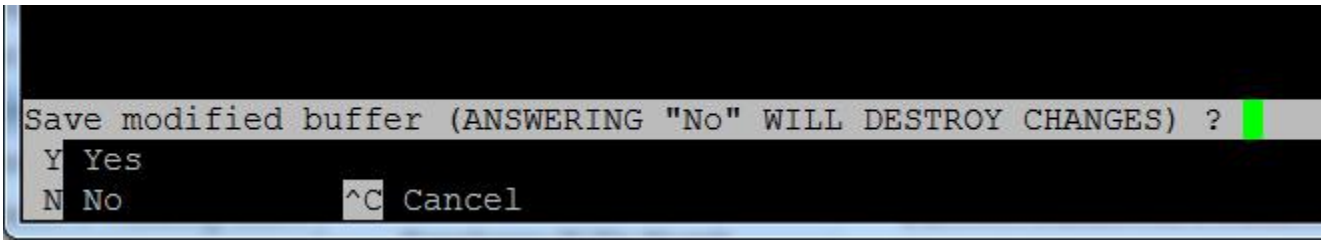
After the code runs, you will see the LED flashing.

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.

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 pintf 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.

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

To stop it from running, just click the X button on the top right to close it and then you'll back to the code. 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 python3 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.

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)
        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.

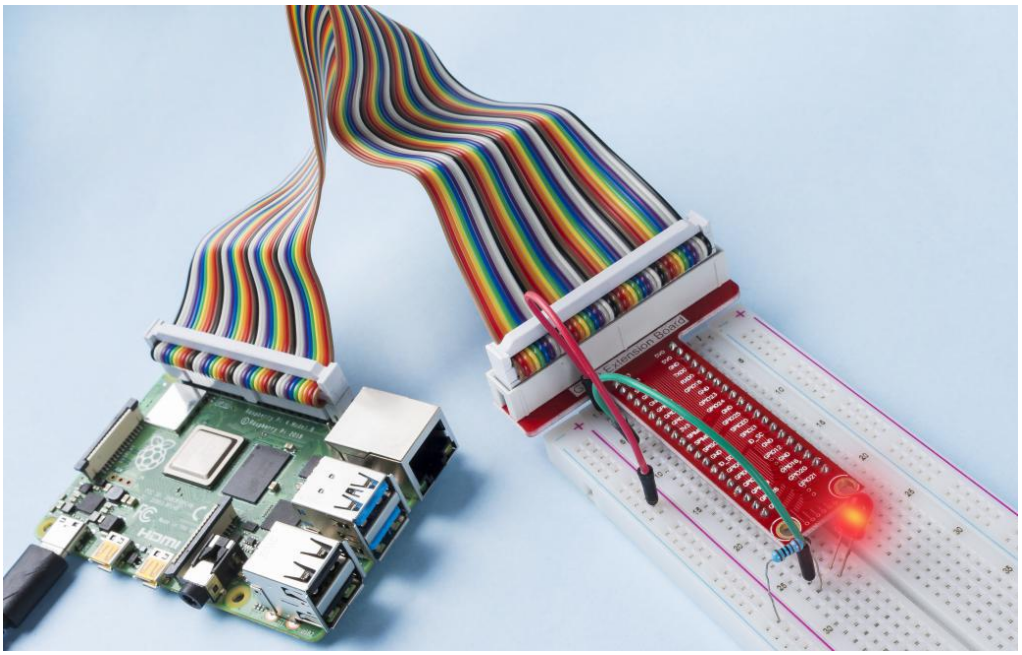
```
def destroy():
        GPIO.cleanup()
```

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

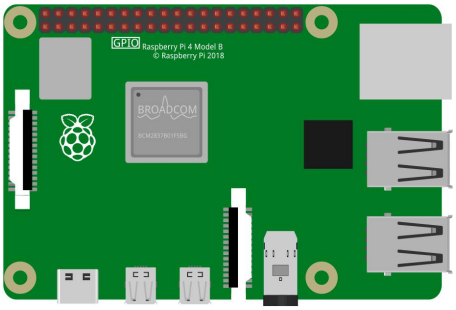
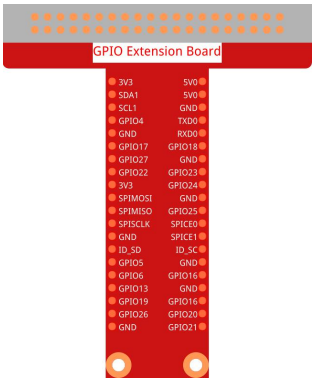
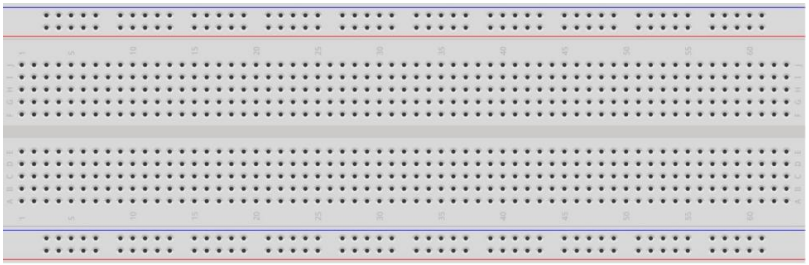


1.1.2 RGB LED

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 
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

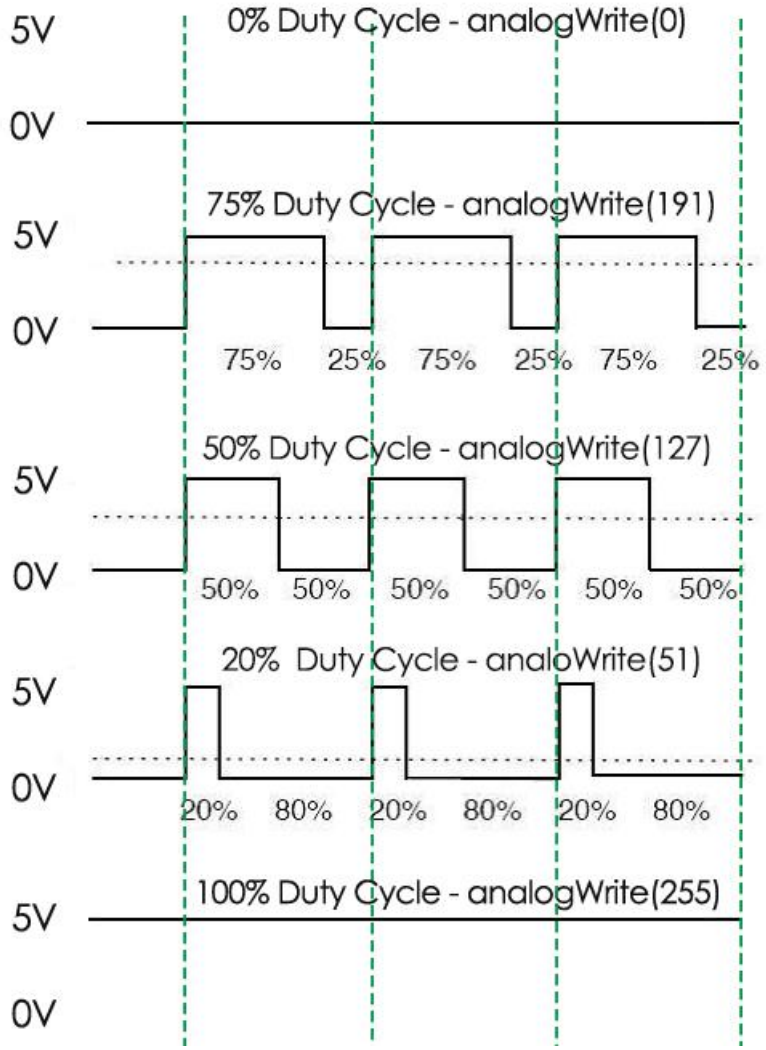
repeat this on-off pattern fast enough with some device, an LED for example, the result would be like this: the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

Duty Cycle

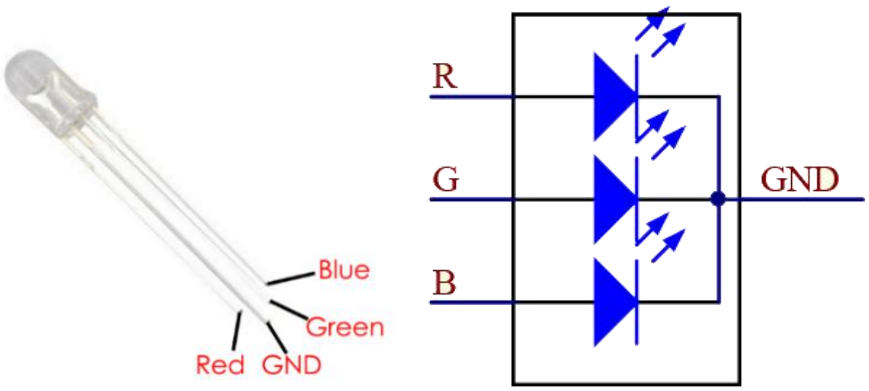
A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

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

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

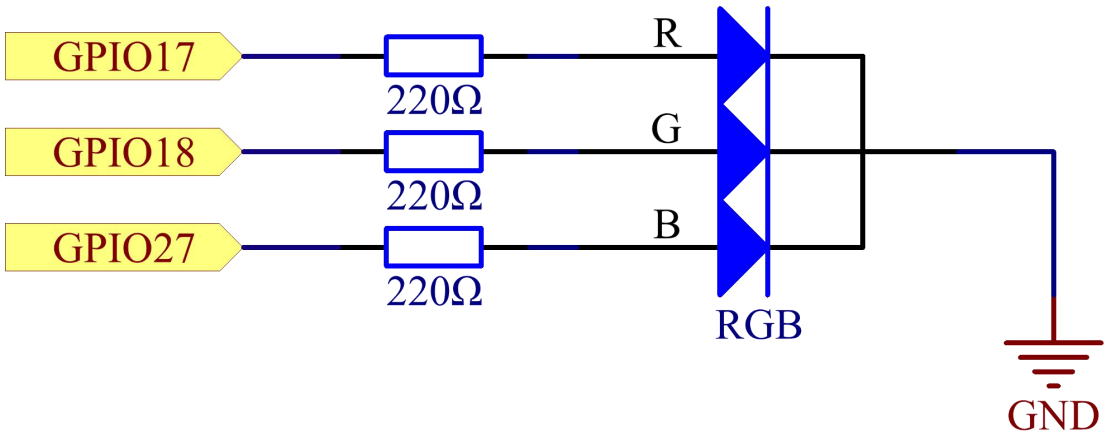


The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with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

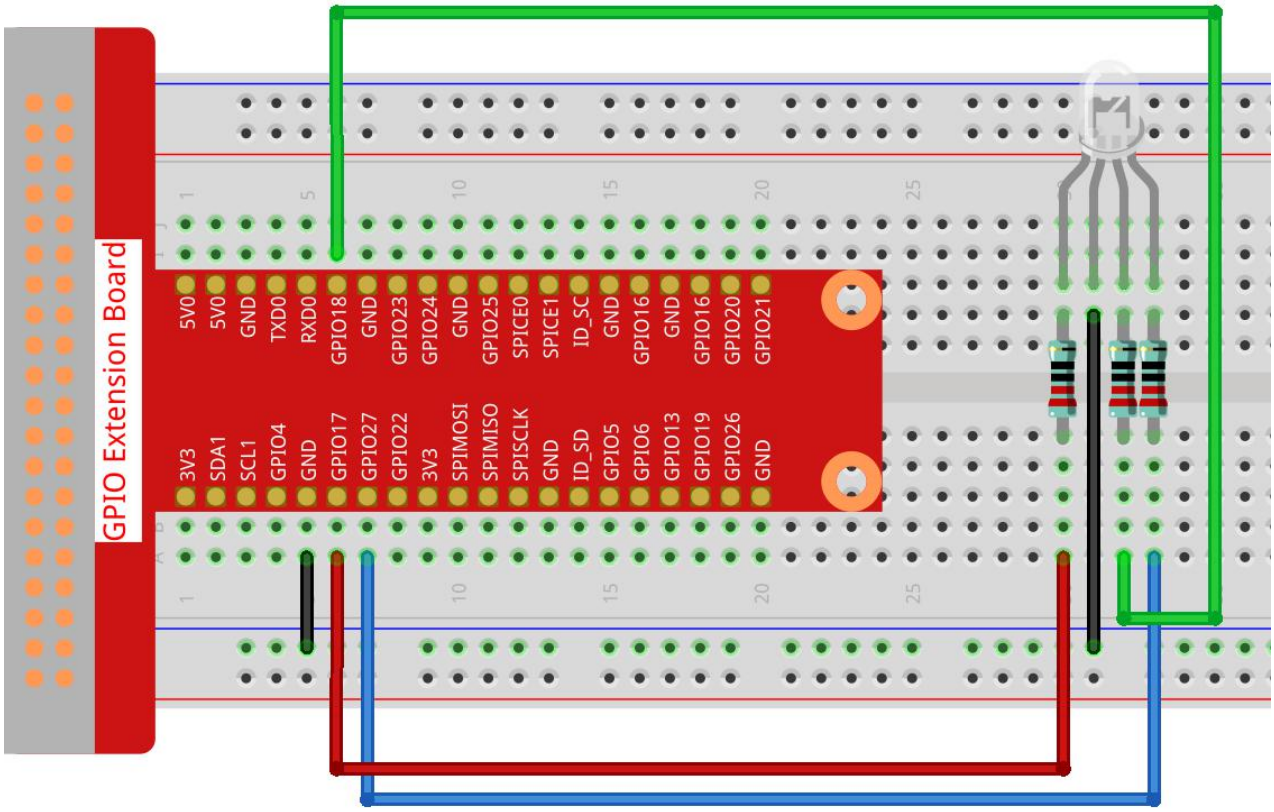
After connecting the pins of R, G, and B to a current limiting resistor, connect them to the GPIO17, GPIO18, and GPIO27 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.

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 messageto 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 python3 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 & 0xFF0000) >> 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.

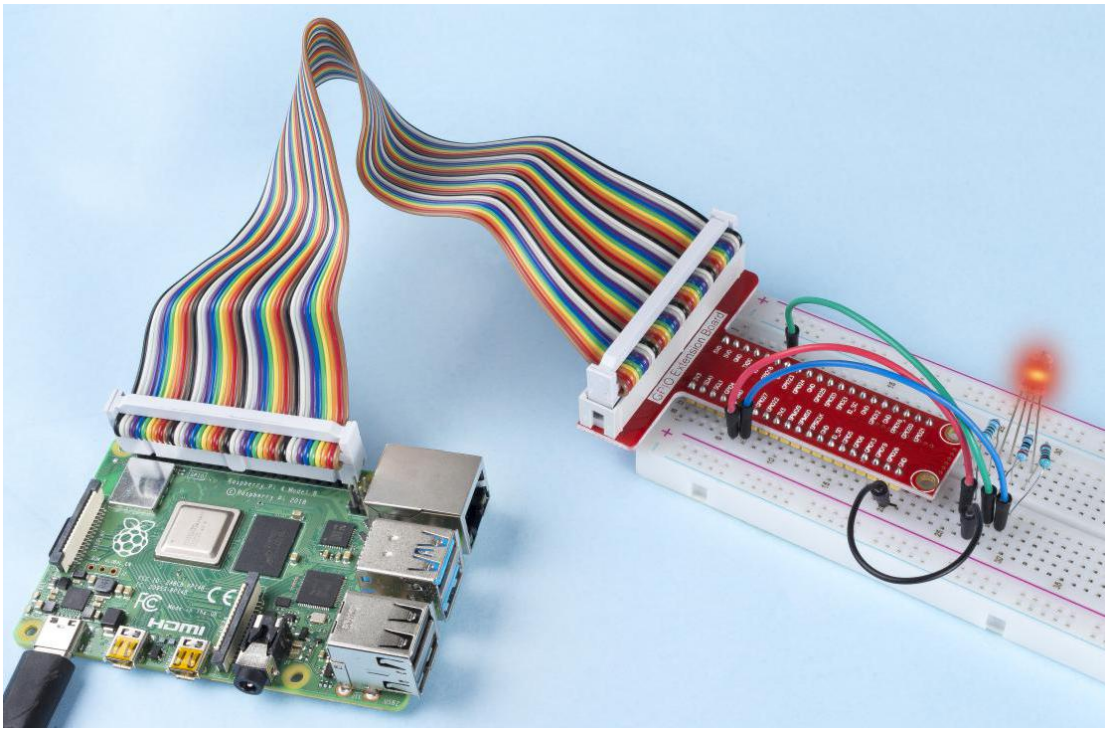
```
p_R.ChangeDutyCycle(R_val)
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

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

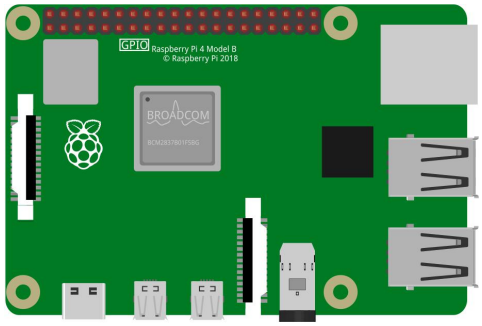
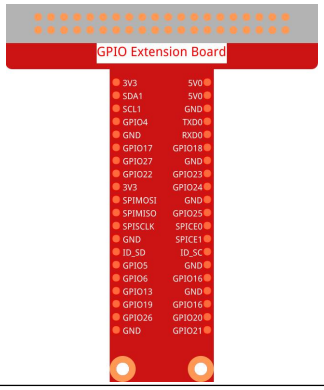
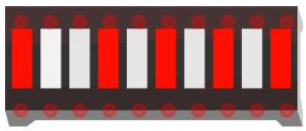
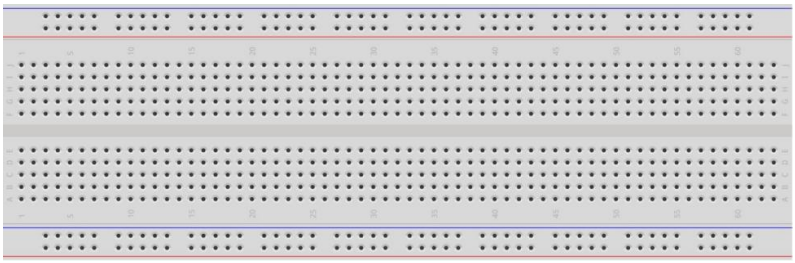


1.1.3 LED Bar Graph

Introduction

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

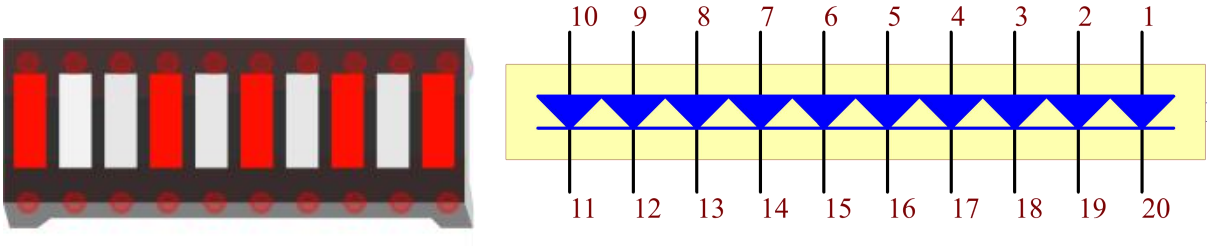
Components

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Bargraph 
1 * 40-pin Cable 		Several Jumper Wires 
1 * Breadboard 		10 * Resistor(220Ω) 

Principle

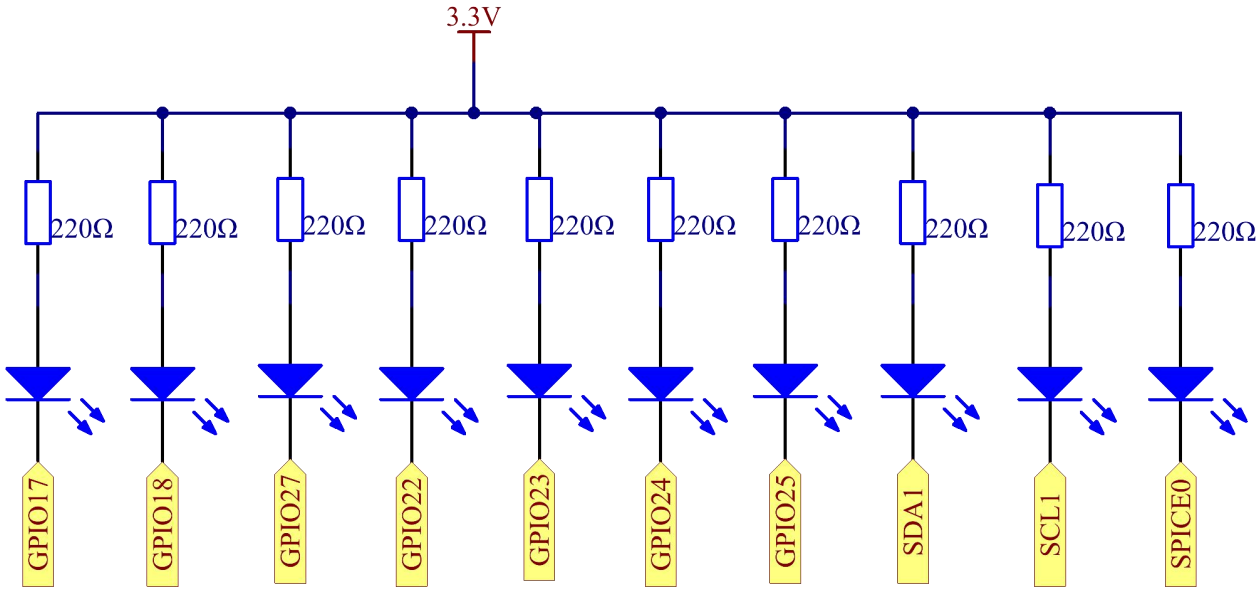
LED Bar Graph

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.



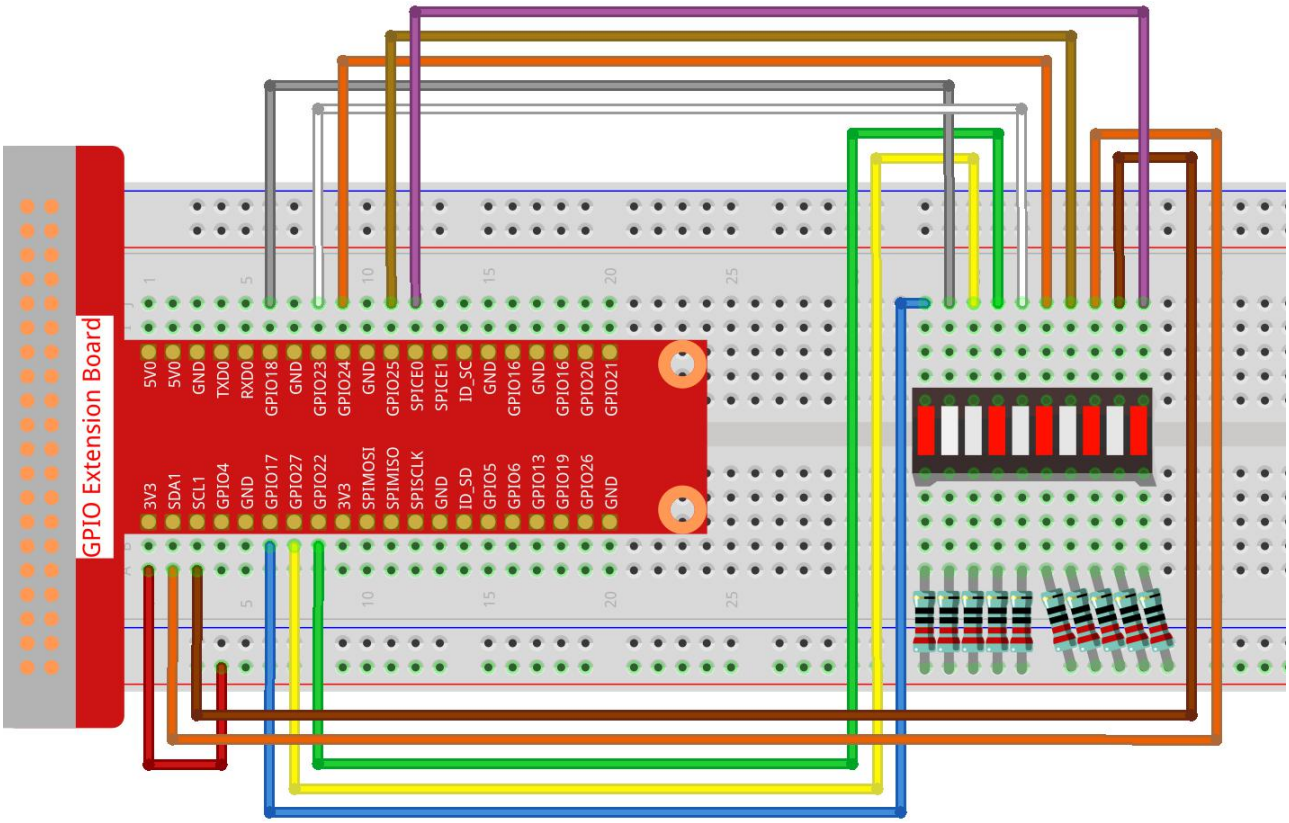
Schematic Diagram

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
SPICE0	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/
```

Step 3: Compile the code.

```
gcc 1.1.3_LedBarGraph.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, you will see the LEDs on the LED bar turn on and off regularly.

Code

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

int pins[10] = {0,1,2,3,4,5,6,8,9,10};
void oddLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2;
```

```
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
void evenLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2+1;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
void allLedBarGraph(void){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
        delay(300);
        digitalWrite(pins[i],LOW);
    }
}
int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to
screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(int i=0;i<10;i++){       //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],LOW);
    }
    while(1){
        oddLedBarGraph();
        delay(300);
        evenLedBarGraph();
        delay(300);
        allLedBarGraph();
        delay(300);
    }
    return 0;
}
```

Code Explanation

```
int pins[10] = {0,1,2,3,4,5,6,8,9,10};
```

Create an array and assign it to the pin number corresponding to the LED Bar Graph (0,1,2,3,4,5,6,8,9,10) and the array will be used to control the LED.

```
void oddLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
```

Let the LED on the odd digit of the LED Bar Graph light on in turn.

```
void evenLedBarGraph(void){
    for(int i=0;i<5;i++){
        int j=i*2+1;
        digitalWrite(pins[j],HIGH);
        delay(300);
        digitalWrite(pins[j],LOW);
    }
}
```

Make the LED on the even digit of the LED Bar Graph light on in turn.

```
void allLedBarGraph(void){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
        delay(300);
        digitalWrite(pins[i],LOW);
    }
}
```

Let the LED on the LED Bar Graph light on one by one.

➢ 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 python3 1.1.3_LedBarGraph.py
```

After the code runs, you will see the LEDs on the LED bar turn on and off regularly.

Code

```
import RPi.GPIO as GPIO
import time

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

def oddLedBarGraph():
    for i in range(5):
        j = i*2
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)

def allLedBarGraph():
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(i,GPIO.LOW)

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

def loop():
    while True:
        oddLedBarGraph()
        time.sleep(0.3)
        evenLedBarGraph()
```

```
        time.sleep(0.3)
        allLedBarGraph()
        time.sleep(0.3)

def destroy():
    for pin in ledPins:
        GPIO.output(pin, GPIO.LOW)    # 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

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

Create an array and assign it to the pin number corresponding to the LED Bar Graph (11, 12, 13, 15, 16, 18, 22, 3, 5, 24) and the array will be used to control the LED.

```
def oddLedBarGraph():
    for i in range(5):
        j = i*2
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)
```

Let the LED on the odd digit of the LED Bar Graph light on in turn.

```
def evenLedBarGraph():
    for i in range(5):
        j = i*2+1
        GPIO.output(ledPins[j],GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(ledPins[j],GPIO.LOW)
```

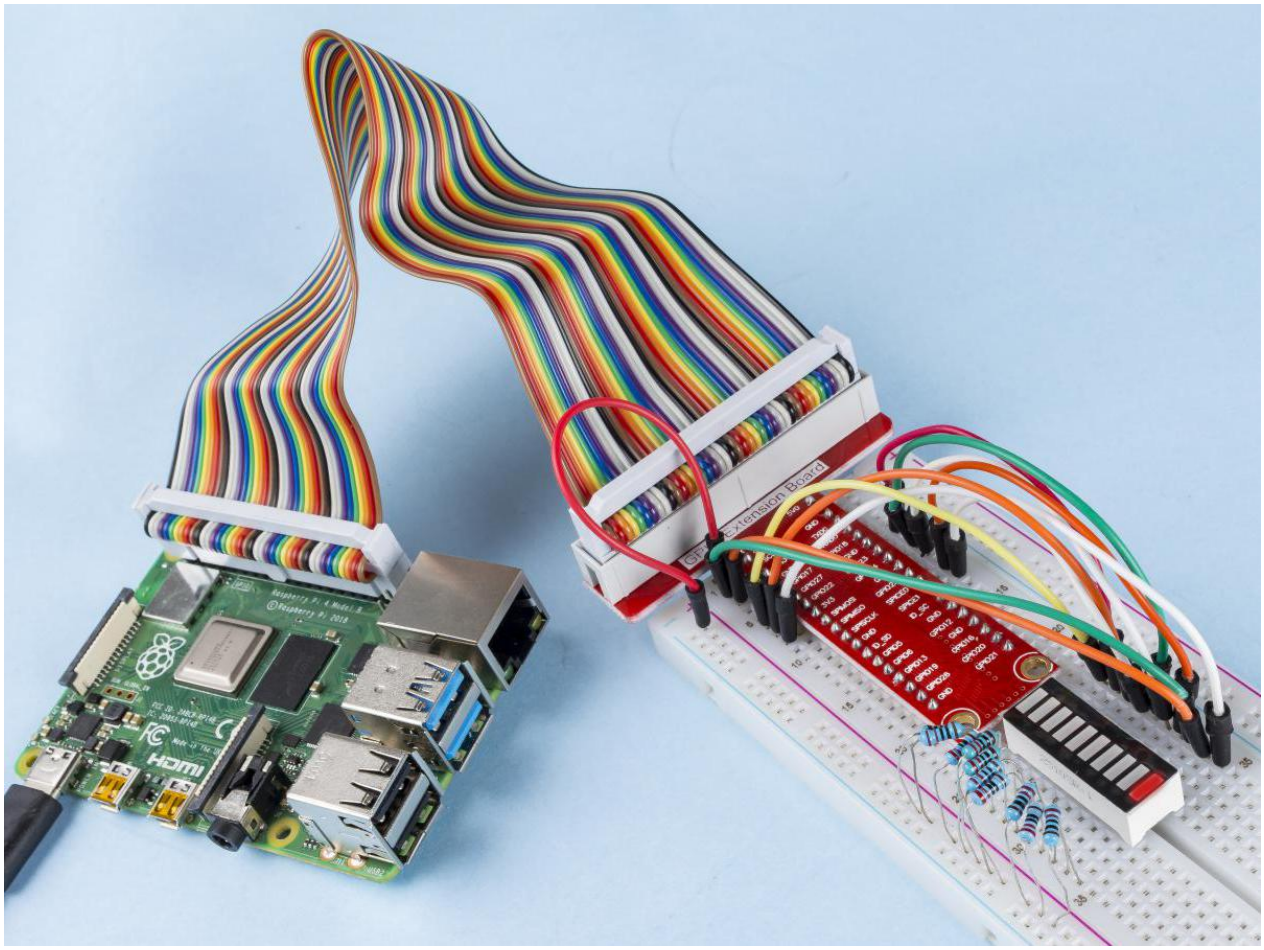
Make the LED on the even digit of the LED Bar Graph light on in turn.

```
def allLedBarGraph():
    for i in ledPins:
```

```
        GPIO.output(i,GPIO.HIGH)
        time.sleep(0.3)
        GPIO.output(i,GPIO.LOW)
```

Let the LED on the LED Bar Graph light on one by one.

Phenomenon Picture

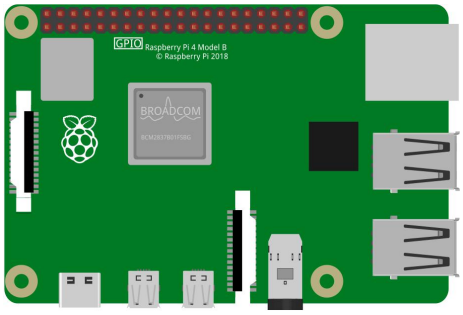
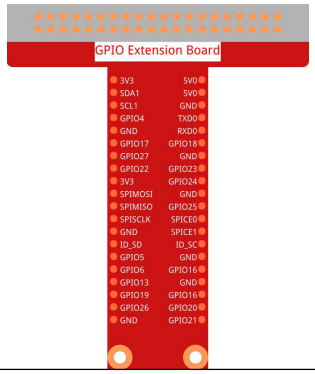
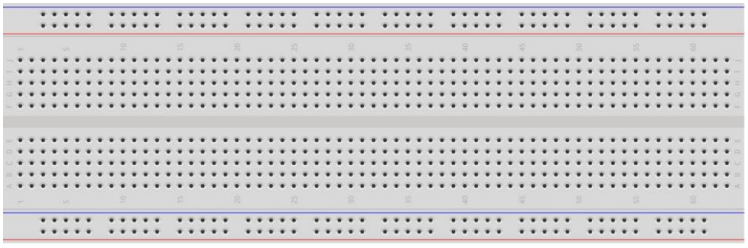


1.1.4 7-segment Display

Introduction

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

Components

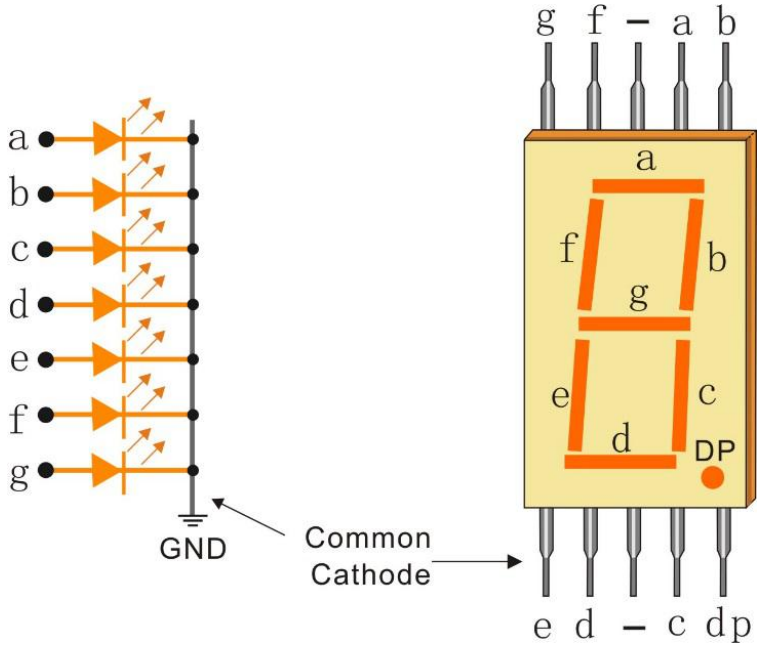
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 7-segment display
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>1 * Resistor(220Ω)
 
Several Jumper Wires
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * 74HC595
  </td></tr>
</table>

Principle

7-Segment Display

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

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



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

Display Codes

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

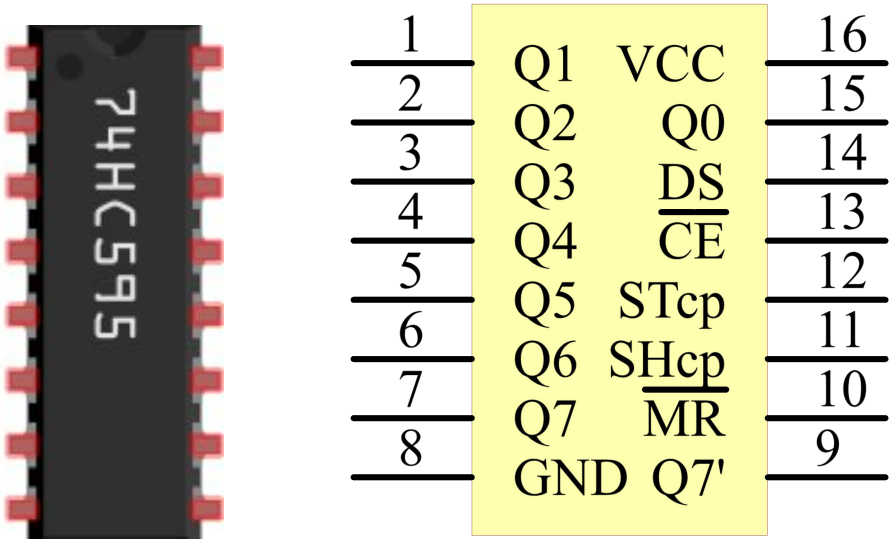
Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39

3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d			
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

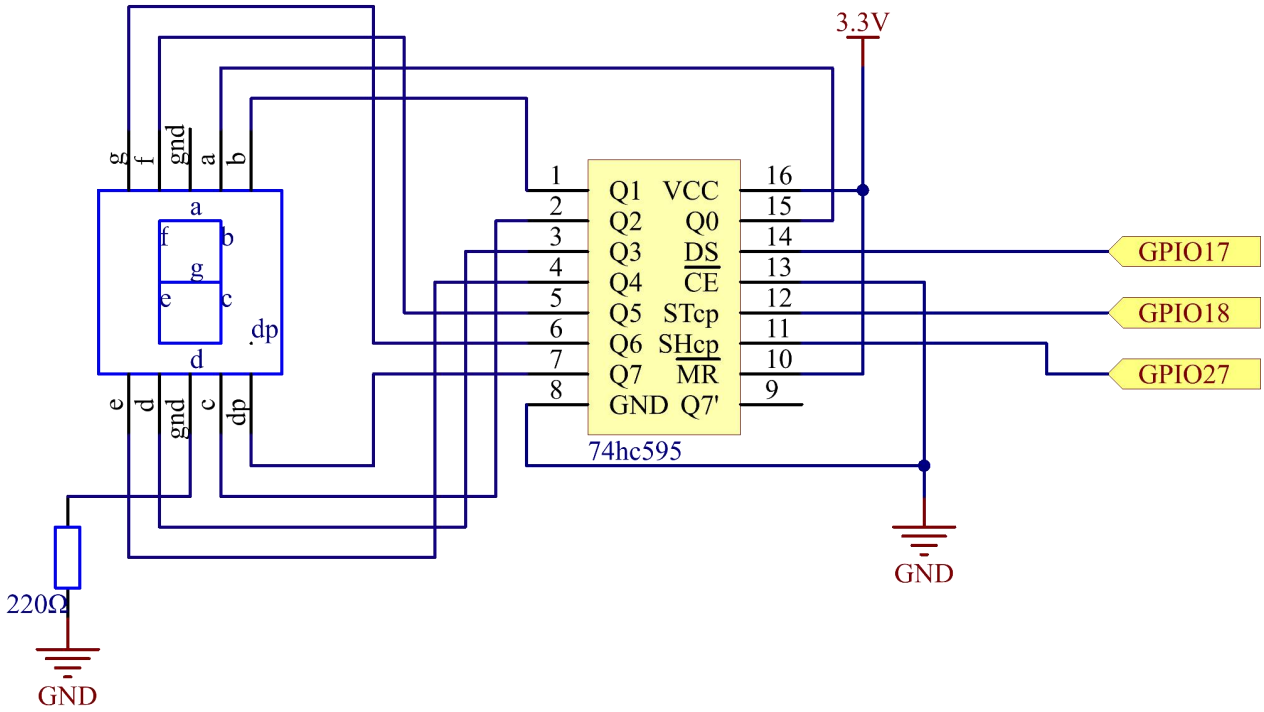
VCC: Positive supply voltage

GND: Ground

Schematic Diagram

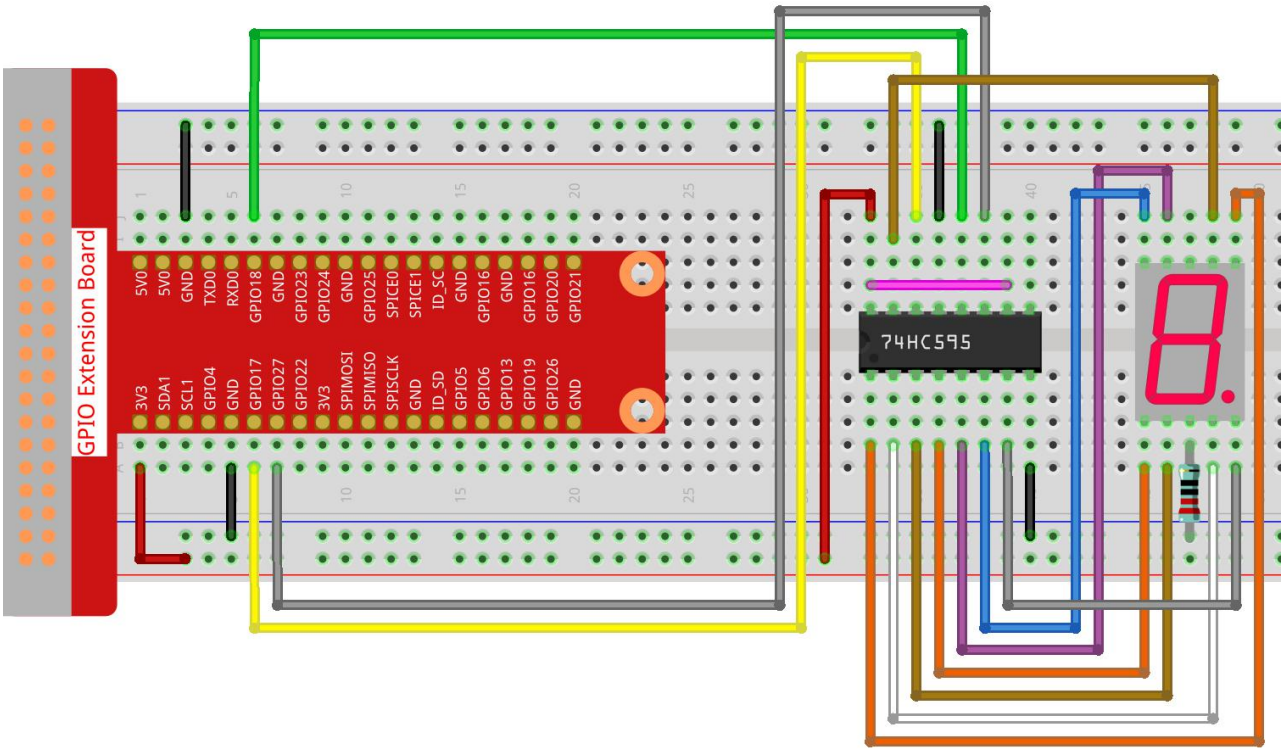
Connect pin ST_CP of 74HC595 to Raspberry Pi GPIO18, SH_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST_CP (the 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[16] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};

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 messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<16;i++){
            printf("Print %1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
    return 0;
}
```

Code Explanation

```
unsigned char SegCode[16] =
{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).

```
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<16;i++){
            printf("Print %1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
```

In this for loop, we use "%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 python3 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)
```

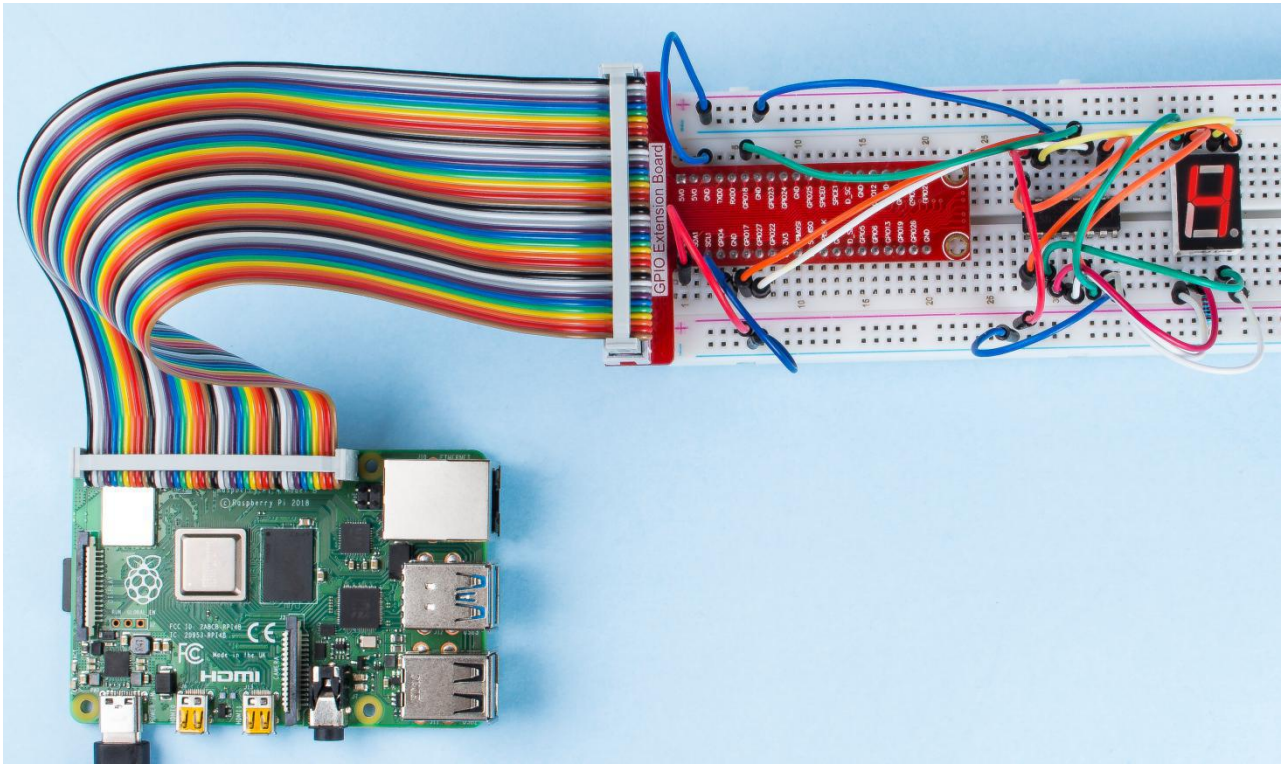
SRCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge pulse, then shift the DS date to shift register.

```
    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

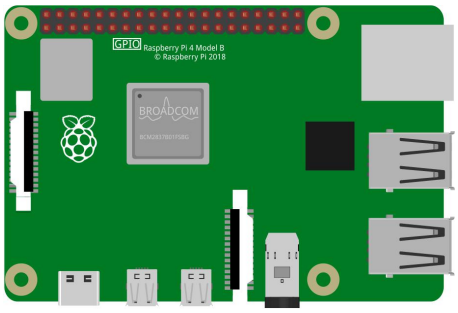
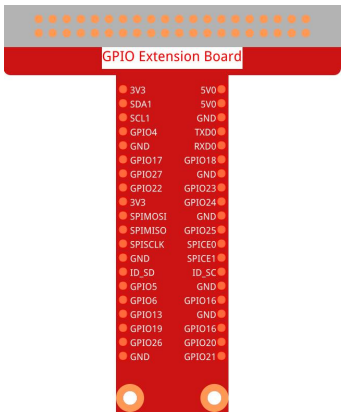
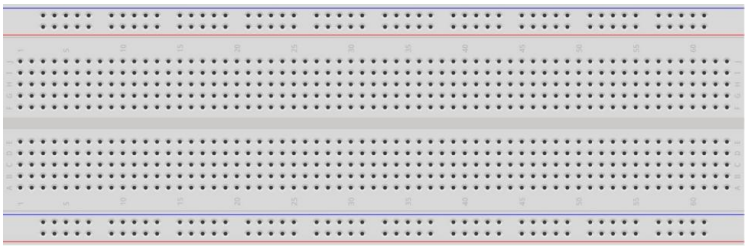


1.1.5 4-Digit 7-Segment Display

Introduction

Next, follow me to try to control the 4-digit 7-segment display.

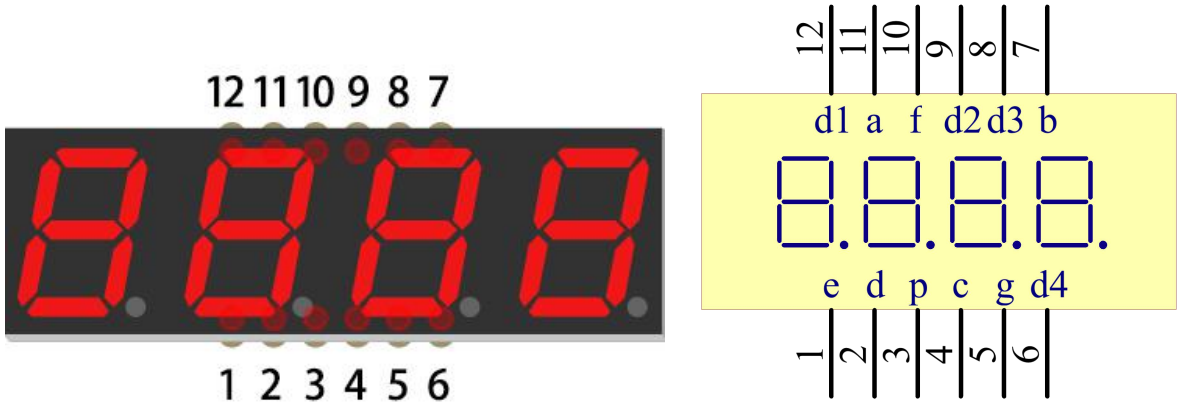
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 4-Digit 7-Segment Display
  </td></tr>
<tr><td colspan="2"></td><td>1 * 74HC595
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>4 * Resistor(220Ω)
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>Several Jumper Wires
</td></tr>
</table>

Principle

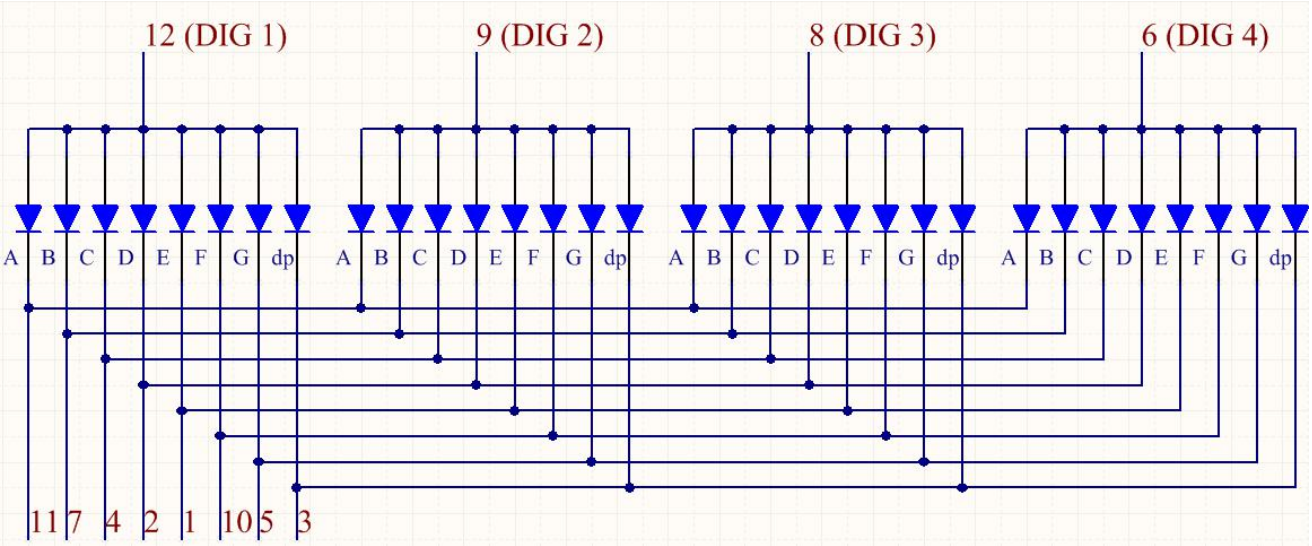
4-Digit 7-Segment Display

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



The 4-digtal 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.



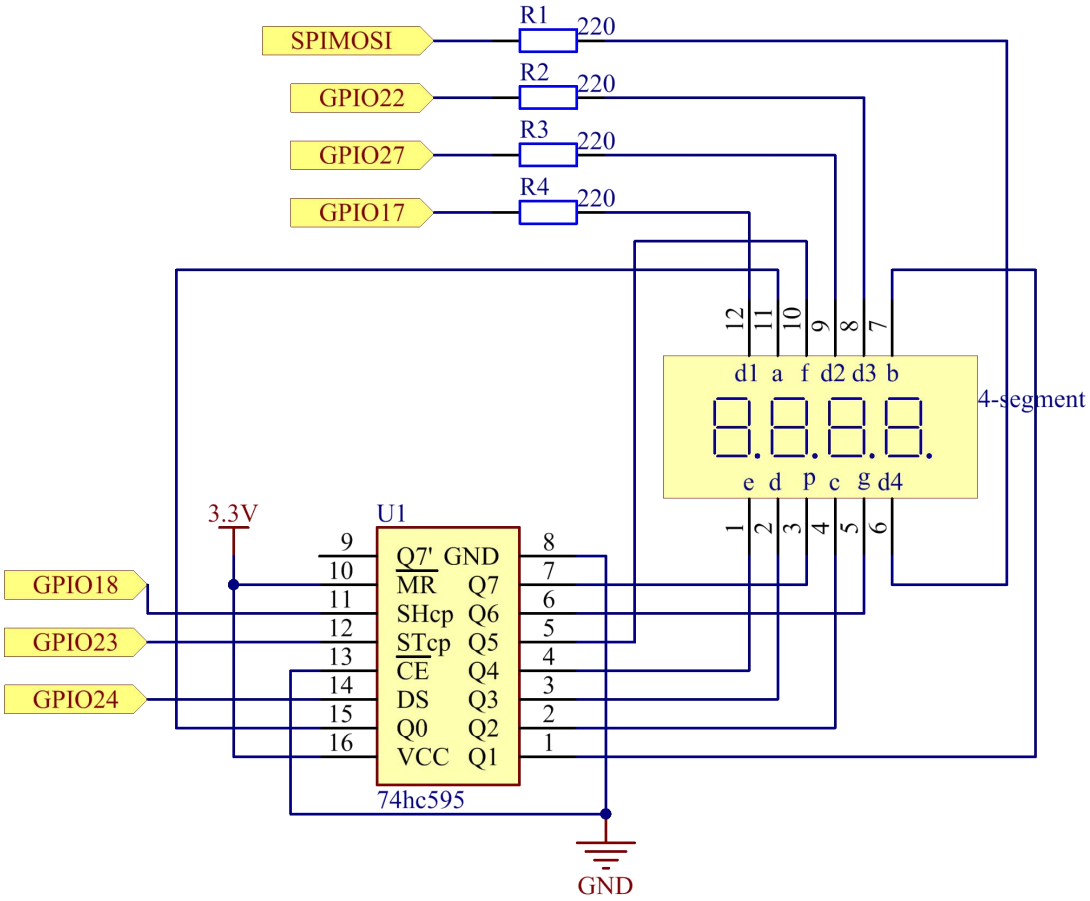
Display Codes

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)GFEDCBA	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			

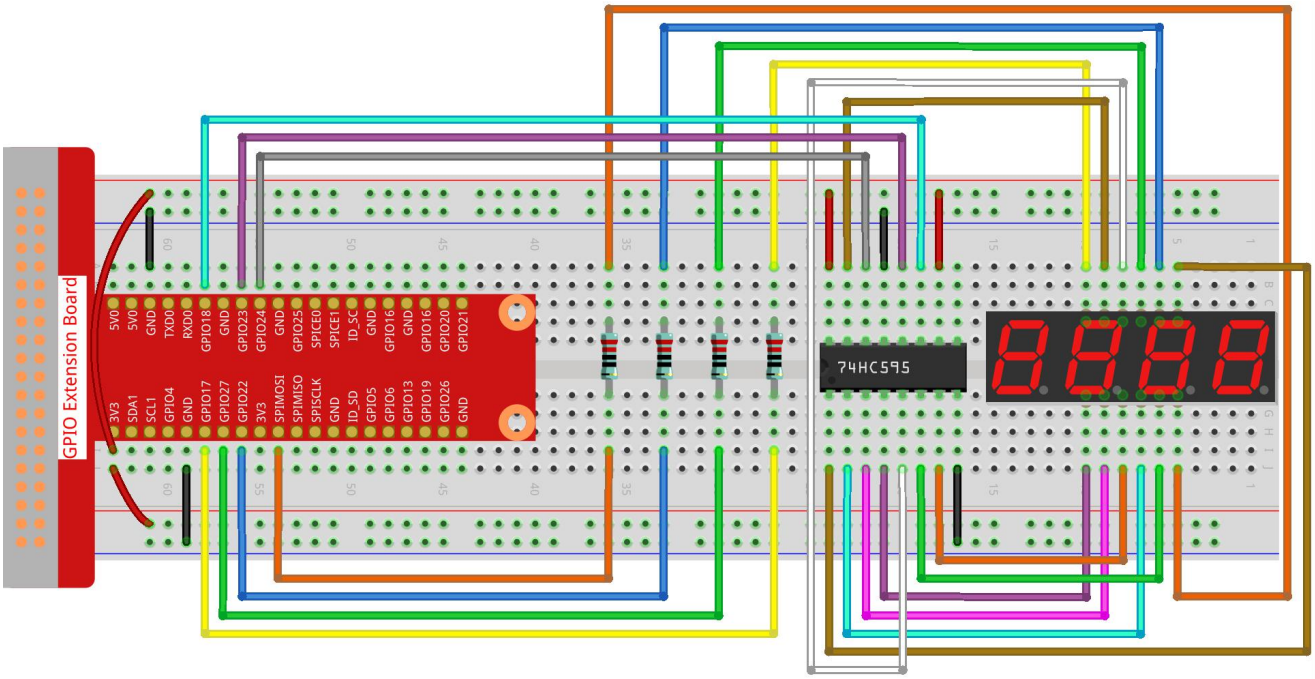
Schematic Diagram

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
GPIO18	Pin 12	1	18
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
#define RCLK 4
#define SRCLK 1

const int placePin[] = {12, 3, 2, 0};
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

int counter = 0;

void pickDigit(int digit)
{
    for (int i = 0; i < 4; i++)
    {
        digitalWrite(placePin[i], 0);
    }
    digitalWrite(placePin[digit], 1);
}

void hc595_shift(int8_t data)
{
```

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

void clearDisplay()
{
    int i;
    for (i = 0; i < 8; i++)
    {
        digitalWrite(SDI, 1);
        digitalWrite(SRCLK, 1);
        delayMicroseconds(1);
        digitalWrite(SRCLK, 0);
    }
    digitalWrite(RCLK, 1);
    delayMicroseconds(1);
    digitalWrite(RCLK, 0);
}

void loop()
{
    while(1){
    clearDisplay();
    pickDigit(0);
    hc595_shift(number[counter % 10]);

    clearDisplay();
    pickDigit(1);
    hc595_shift(number[counter % 100 / 10]);

    clearDisplay();
```

```
        pickDigit(2);
        hc595_shift(number[counter % 1000 / 100]);

        clearDisplay();
        pickDigit(3);
        hc595_shift(number[counter % 10000 / 1000]);
        }
}

void timer(int timer1)
{
    if (timer1 == SIGALRM)
    {
        counter++;
        alarm(1);
        printf("%d\n", counter);
    }
}

void main(void)
{
    if (wiringPiSetup() == -1)
    {
        printf("setup wiringPi failed !");
        return;
    }
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);

    for (int i = 0; i < 4; i++)
    {
        pinMode(placePin[i], OUTPUT);
        digitalWrite(placePin[i], HIGH);
    }
    signal(SIGALRM, timer);
    alarm(1);
    loop();
}
```

Code Explanation

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

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 pickDigit(int digit)
{
    for (int i = 0; i < 4; i++)
    {
        digitalWrite(placePin[i], 0);
    }
    digitalWrite(placePin[digit], 1);
}
```

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

```
void loop()
{
    while(1){
    clearDisplay();
    pickDigit(0);
    hc595_shift(number[counter % 10]);

    clearDisplay();
    pickDigit(1);
    hc595_shift(number[counter % 100 / 10]);

    clearDisplay();
    pickDigit(2);
    hc595_shift(number[counter % 1000 / 100]);

    clearDisplay();
    pickDigit(3);
    hc595_shift(number[counter % 10000 / 1000]);
    }
}
```

The functionis used to set the number displayed on the 4-digit 7-segment display.

clearDisplay()： write in 11111111 to turn off these eight LEDs on 7-segment display so as to clear the displayed content.

pickDigit(0)： pick 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 timer1)
{
    if (timer1 == SIGALRM)
    {
        counter++;
        alarm(1);
        printf("%d\n", counter);
    }
}
```

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 python3 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

SDI = 24
RCLK = 23
SRCLK = 18

placePin = (10, 22, 27, 17)
number = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)

counter = 0
timer1 = 0

def clearDisplay():
    for i in range(8):
        GPIO.output(SDI, 1)
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    GPIO.output(RCLK, GPIO.LOW)

def hc595_shift(data):
    for i in range(8):
        GPIO.output(SDI, 0x80 & (data << i))
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    GPIO.output(RCLK, GPIO.LOW)
```

```
def pickDigit(digit):
    for i in placePin:
        GPIO.output(i,GPIO.LOW)
    GPIO.output(placePin[digit], GPIO.HIGH)

def timer():
    global counter
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter += 1
    print("%d" % counter)

def loop():
    global counter
    while True:
        clearDisplay()
        pickDigit(0)
        hc595_shift(number[counter % 10])

        clearDisplay()
        pickDigit(1)
        hc595_shift(number[counter % 100//10])

        clearDisplay()
        pickDigit(2)
        hc595_shift(number[counter % 1000//100])

        clearDisplay()
        pickDigit(3)
        hc595_shift(number[counter % 10000//1000])

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    for i in placePin:
        GPIO.setup(i, GPIO.OUT)
```

```
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()

def destroy():    # When "Ctrl+C" is pressed, the function is executed.
    global timer1
    GPIO.cleanup()
    timer1.cancel()   # cancel the timer

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

Code Explanation

```
placePin = (10, 22, 27, 17)
```

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

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

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

```
def clearDisplay():
    for i in range(8):
        GPIO.output(SDI, 1)
        GPIO.output(SRCLK, GPIO.HIGH)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    GPIO.output(RCLK, GPIO.LOW)
```

Write "1" for eight times in SDI., so that the eight LEDs on the 7-segment Dispaly will turn off so as to clear the displayed content.

```
def pickDigit(digit):
    for i in placePin:
        GPIO.output(i,GPIO.LOW)
    GPIO.output(placePin[digit], 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 high.

```
def loop():
    global counter
    while True:
        clearDisplay()
        pickDigit(0)
        hc595_shift(number[counter % 10])

        clearDisplay()
        pickDigit(1)
        hc595_shift(number[counter % 100//10])

        clearDisplay()
        pickDigit(2)
        hc595_shift(number[counter % 1000//100])

        clearDisplay()
        pickDigit(3)
        hc595_shift(number[counter % 10000//1000])
```

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

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
    timer1 = threading.Timer(1.0, timer)
    timer1.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={})
```

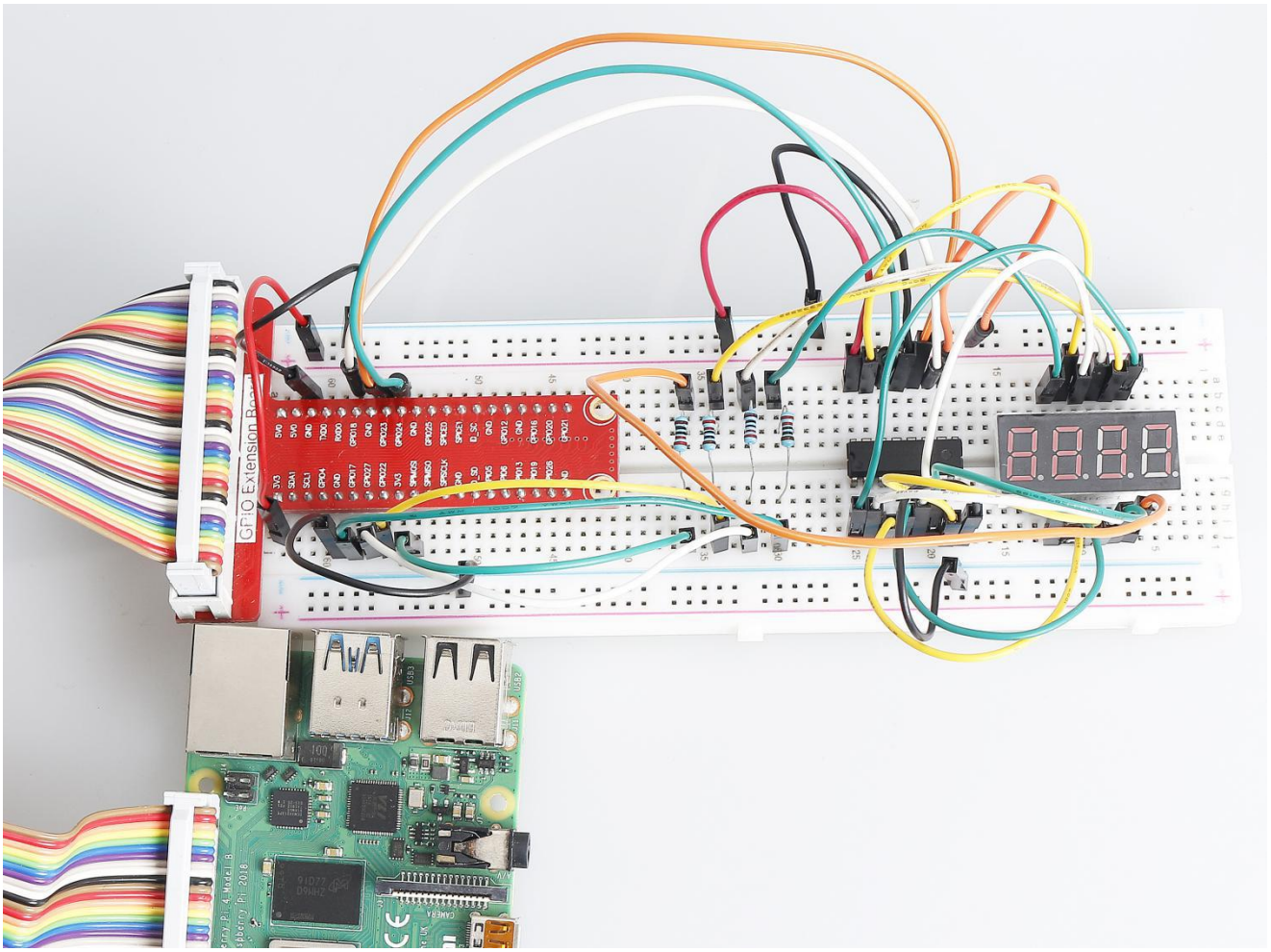
After the interval, the function will be run. Here, the interval is 1.0， and the function is timer().

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

```
def timer():
    global counter
    global timer1
    timer1 = threading.Timer(1.0, timer)
    timer1.start()
    counter += 1
    print("%d" % counter)
```

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

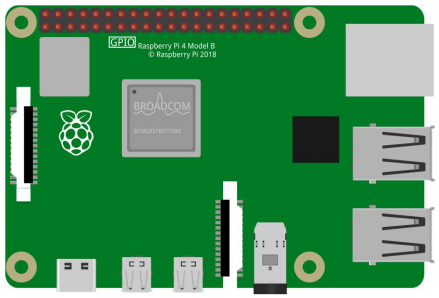
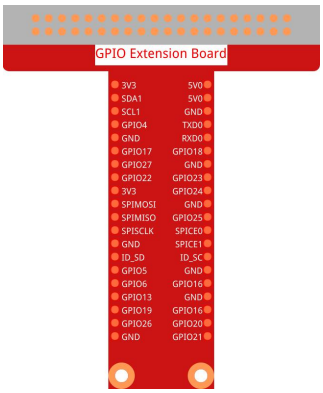
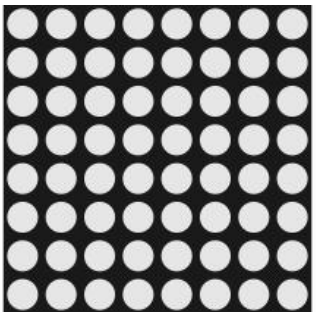
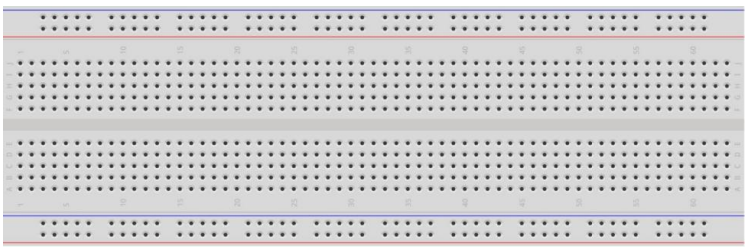


1.1.6 LED Dot Matrix

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

1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED Dot Matrix 
1 * 40-pin Cable 		Several Jumper Wires 
1 * Breadboard 		2 * 74HC595 

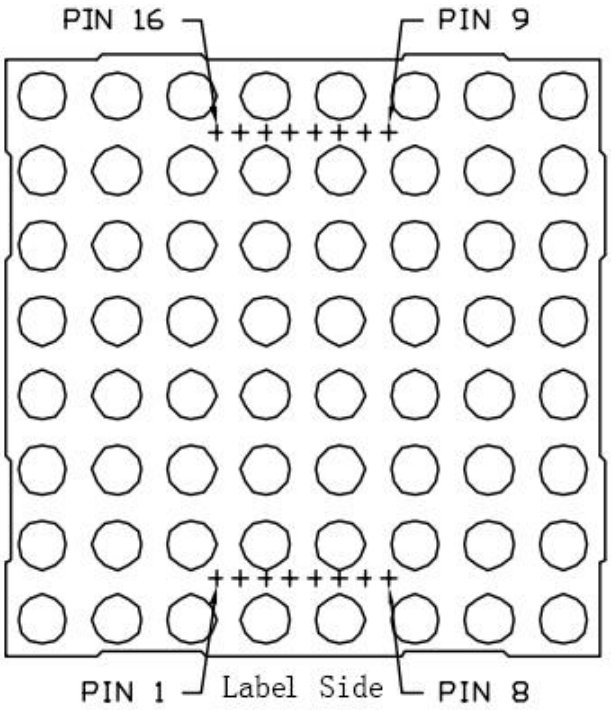
Principle

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.

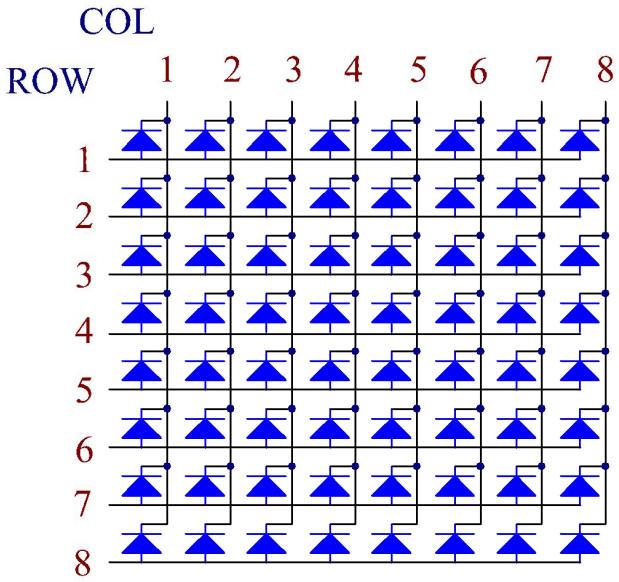
See the figure below. The pins are arranged at the two ends at the back. Take the label side for reference: pins on this end are pin 1-8, and oh the other are pin 9-16.

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:



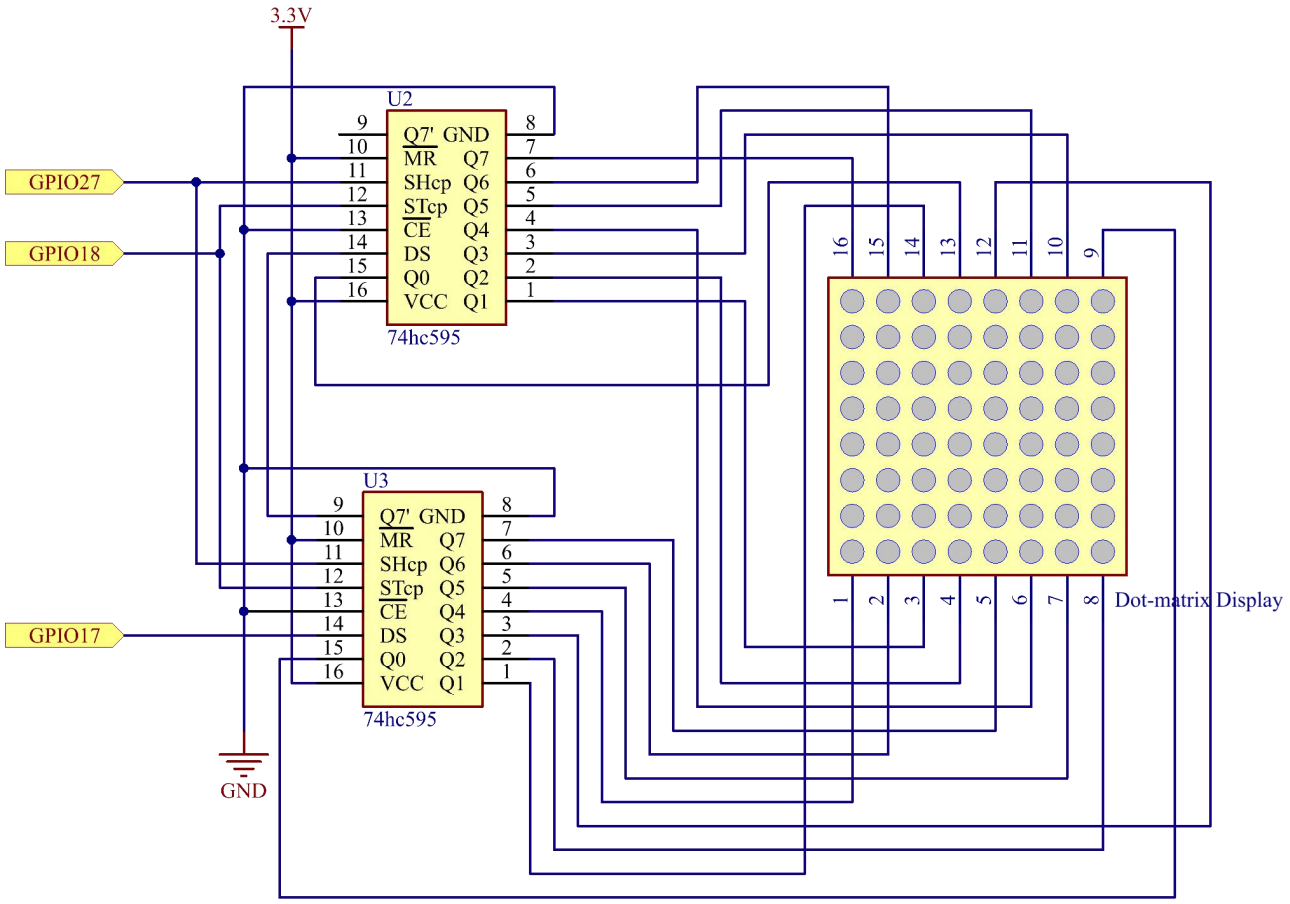
Pin numbering corresponding to the above rows and columns:

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

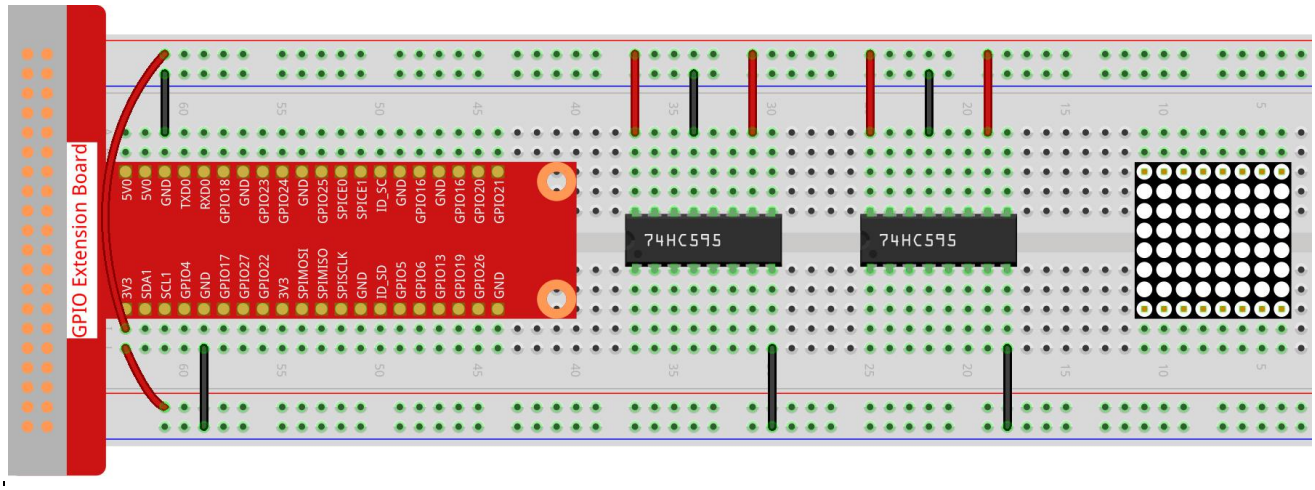
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



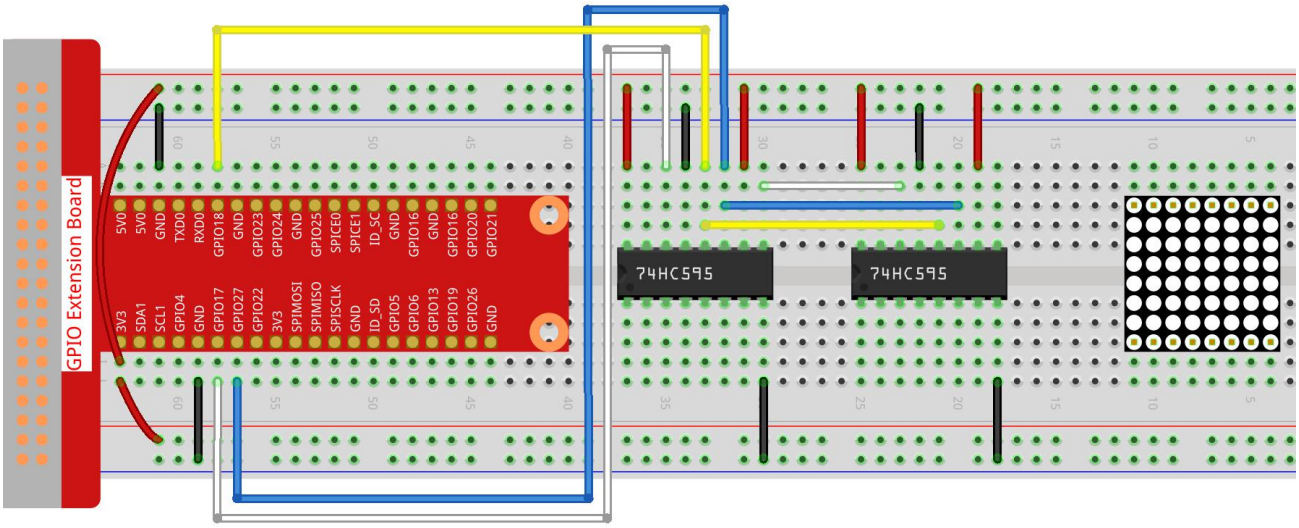
Experimental Procedures

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.

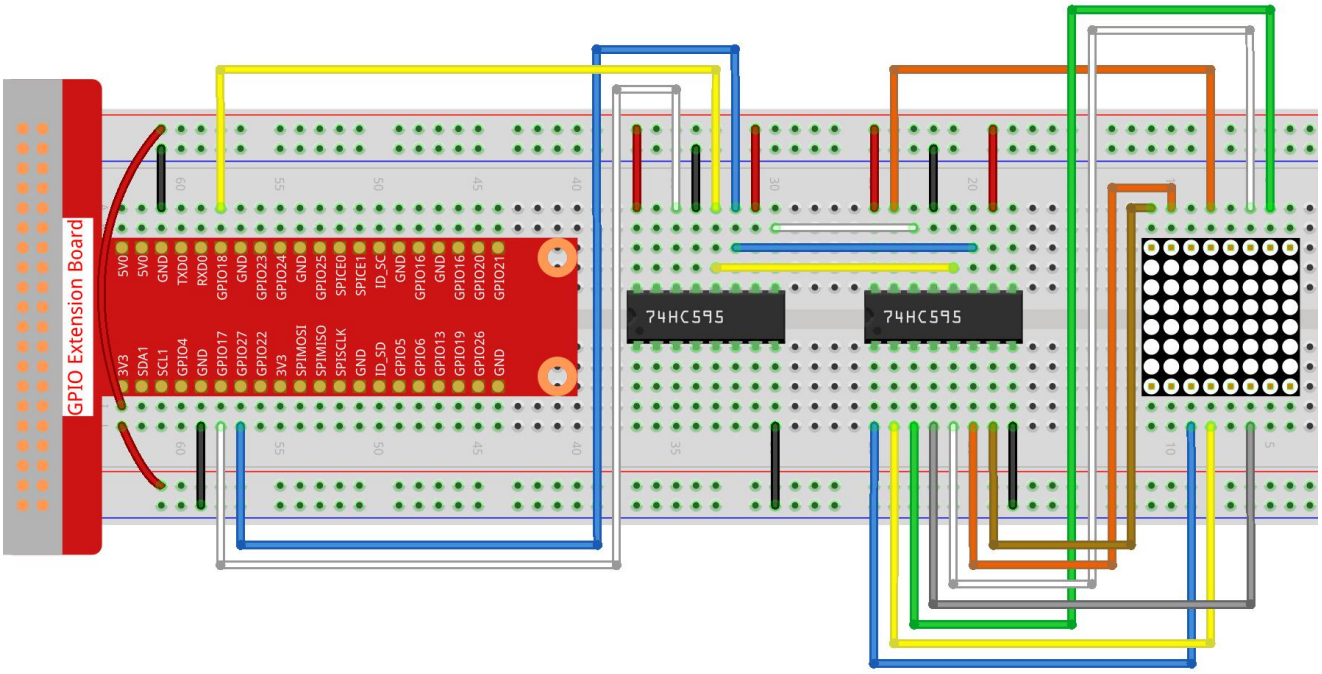


Step 2: Connect pin 11 of the two 74HC595 together, and then to GPIO27; then pin 12 of the two chips, and to GPIO18; next, pin 14 of the 74HC595 on the left side to GPIO17 and pin 9 to pin 14 of the second 74HC595.



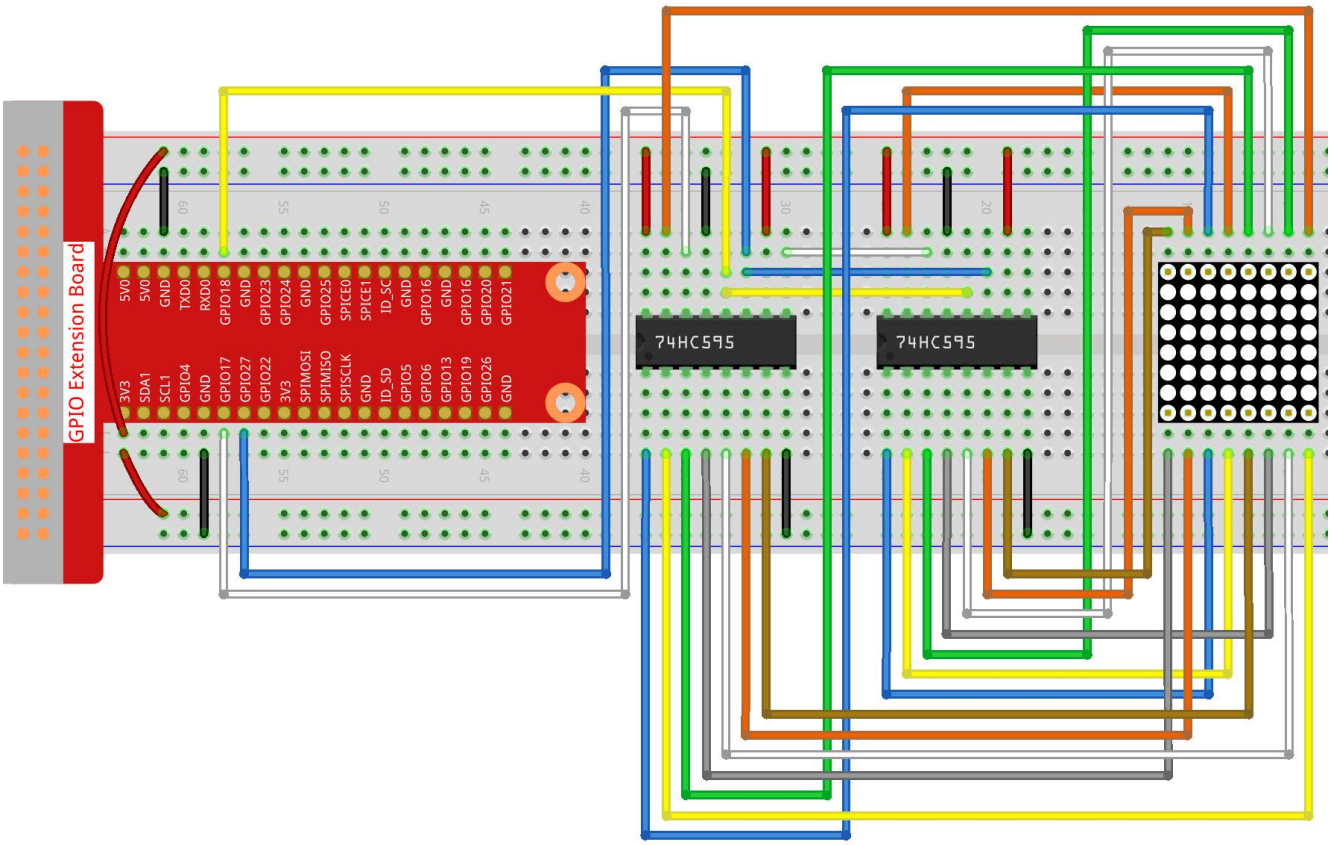
Step 3: The 74HC595 on the right side is to control columns of the LED dot matrix. See the table below for the mapping. Therefore, Q0-Q7 pins of the 74HC595 are mapped with pin 13, 3, 4, 10, 6, 11, 15, and 16 respectively.

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,0xff,0x
ff};
unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0x
bf,0x7f};

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,0xff,0xff};
unsigned char code_L[20] =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
```

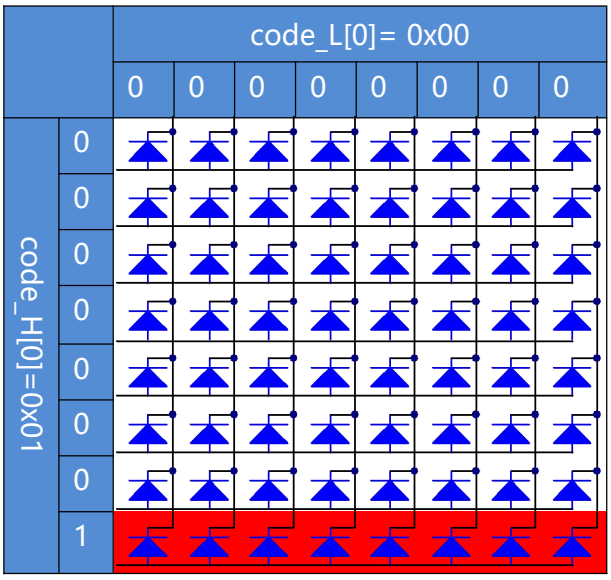
The array code_H represents the elements of the LED dot matrix row, and the array code_L refers to the elements of the column. When characters are displayed, an element in row and one in column are acquired and assigned to the two HC595 chips respectively. Thus a pattern is shown on the LED dot matrix.

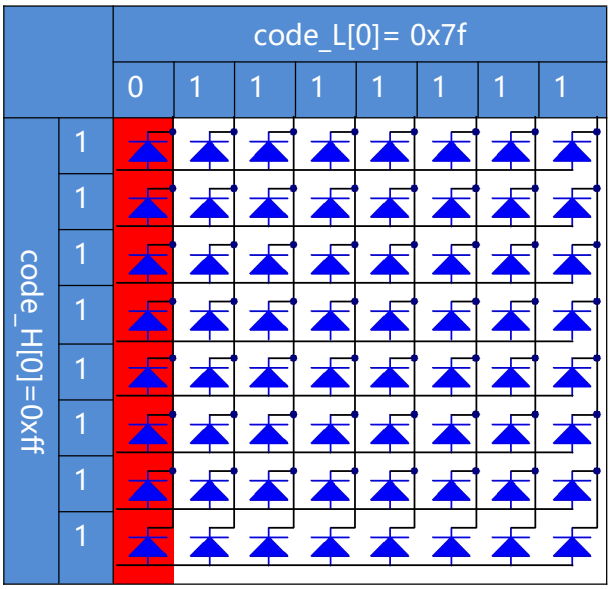
Take the first number of code_H, 0x01 and the first number of code_L, 0x00 as examples.

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 python3 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,0xff,0xff,0xff]
# COL   ----
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,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,0xff,0xff]
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f]
```

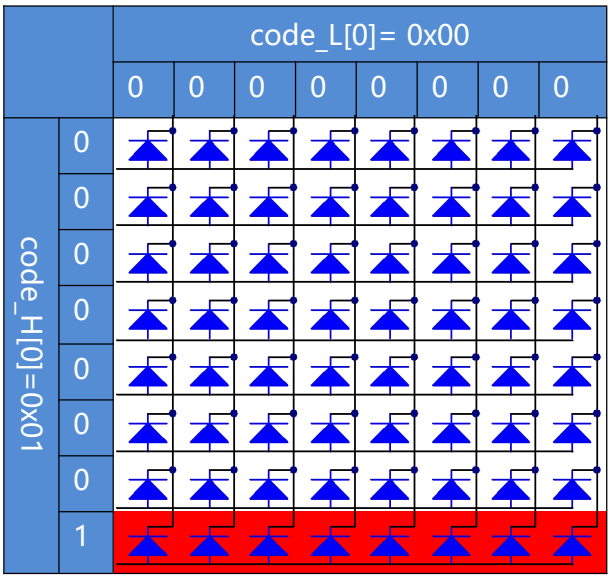
The array code_H represents the elements of the matix row, and the array code_L refers to the elements of the column. When characters are displayed, an element in row and one in column are acquired and assigned to the two HC595 chips respectively. Thus a pattern is shown on the LED dot matrix.

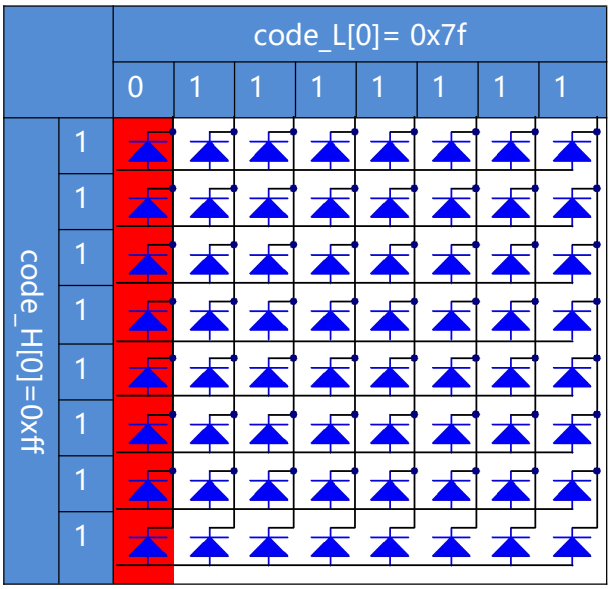
Take the first number of code_H, 0x01 and the first number of code_L, 0x00 as examples.

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.



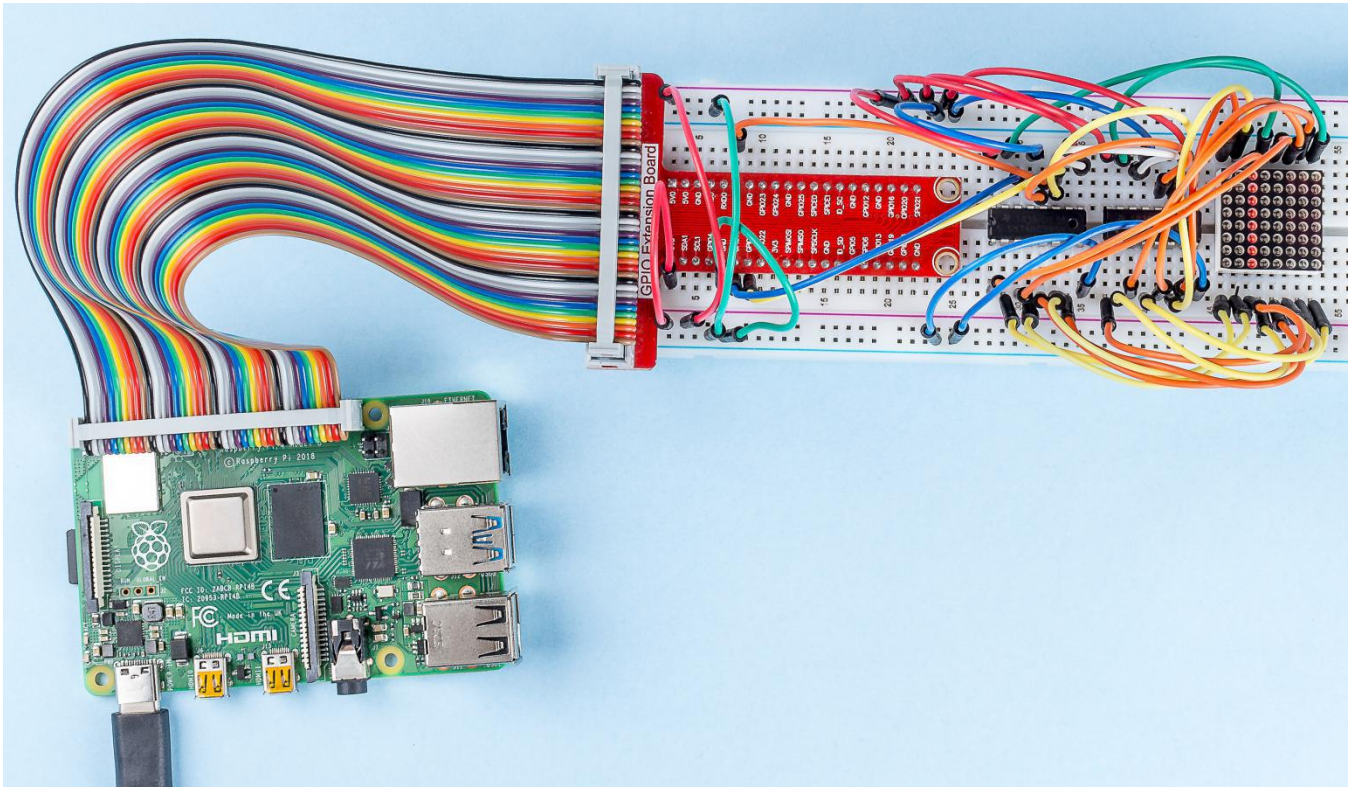


```
    for i in range(0, len(code_H)):
        hc595_shift(code_L[i])
        hc595_shift(code_H[i])
```

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

Phenomenon Picture

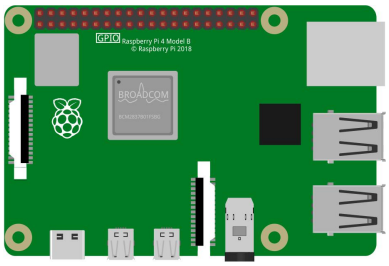
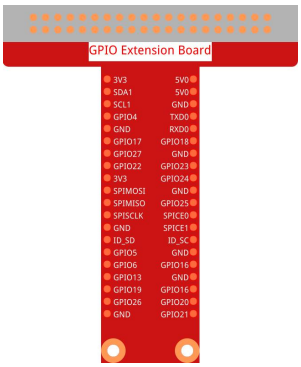
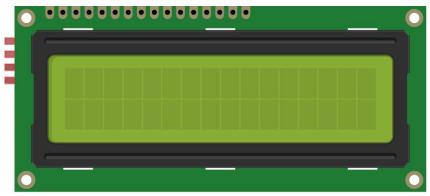
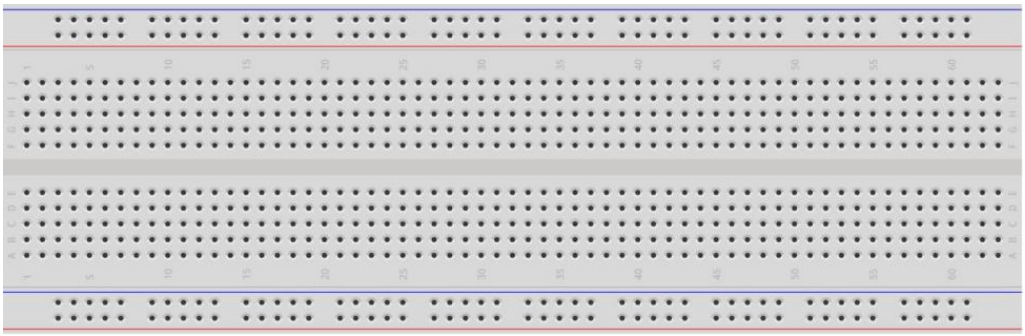


1.1.7 I2C LCD1602

Introduction

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

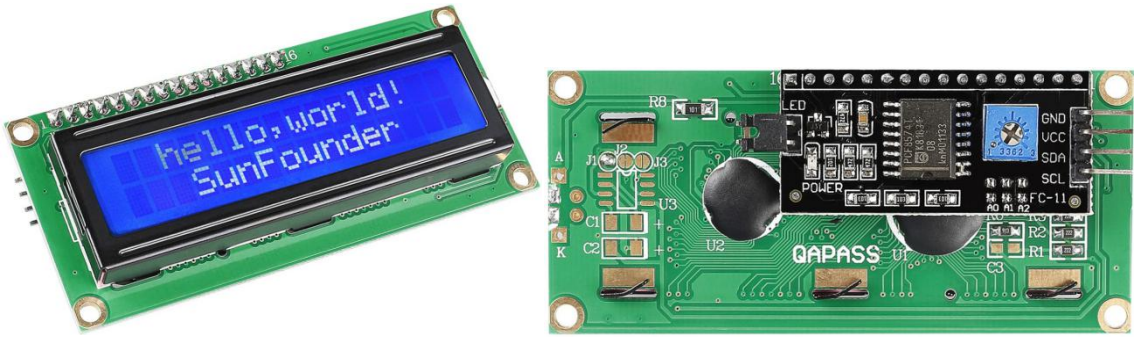
Components

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.



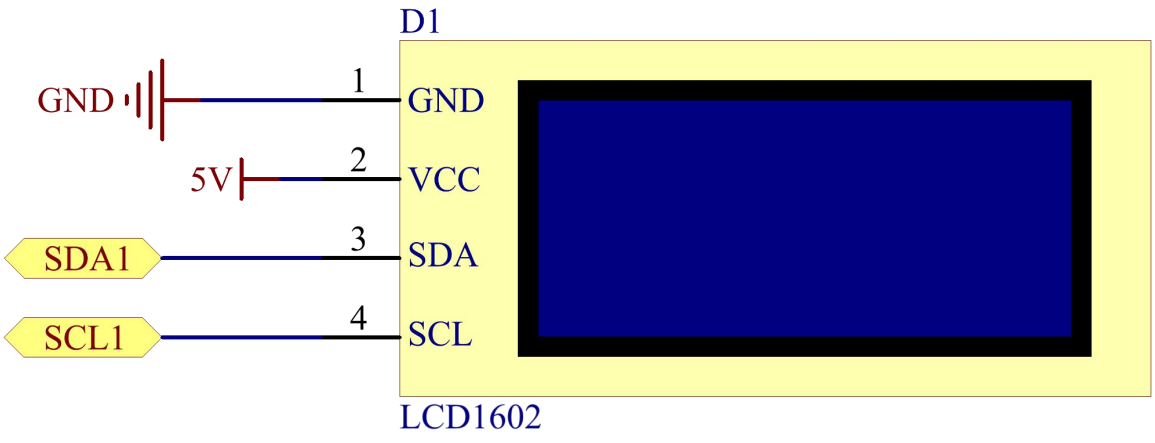
I2C communication

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).

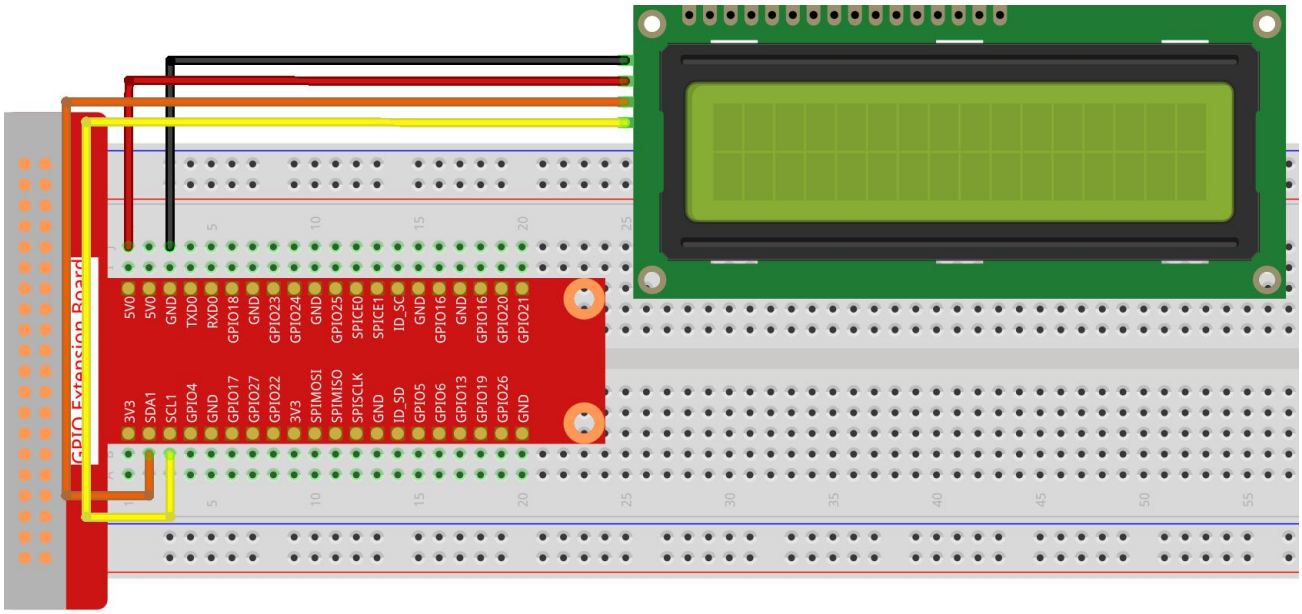
Schematic Diagram

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 = wiringPiI2CSetup(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 = wiringPiI2CSetup(LCDAddr);
```

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

```
int wiringPiI2CSetup(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 python3 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)
```

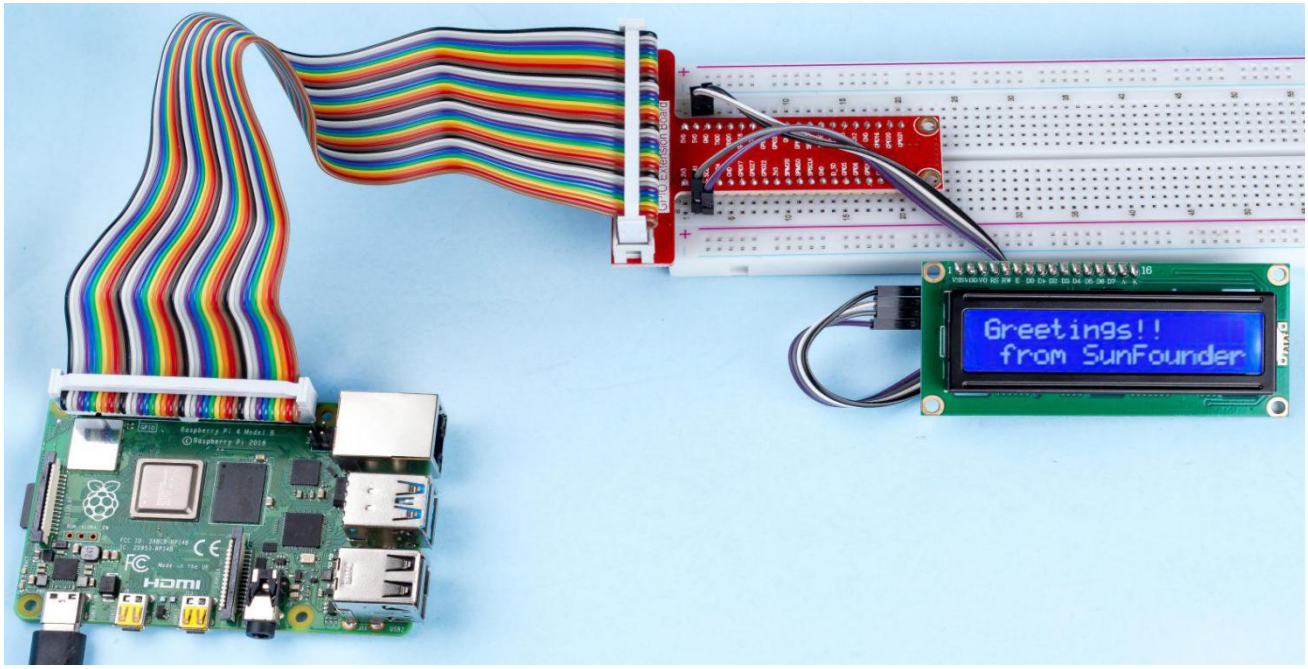
The function initializes the I2C system with the designated device symbol. The first parameter is the address of the I2C device, which can be detected through the i2cdetect command (see Appendix for details). The address of I2C LCD1602 is generally 0x27.

```
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



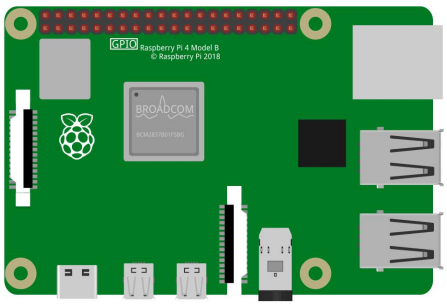
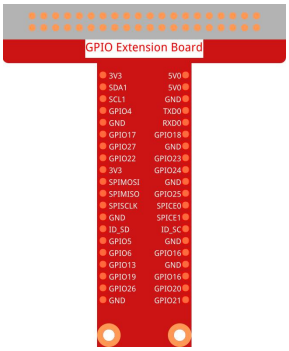
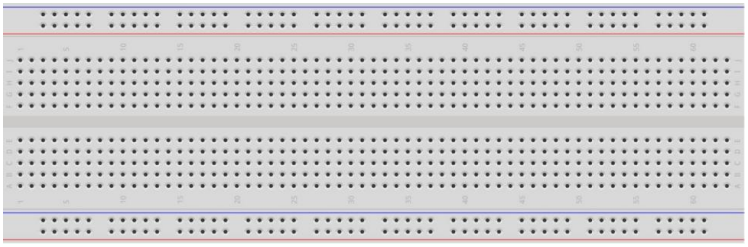
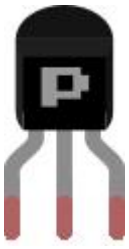
1.2 Sound

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

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
  </td>
<td>1 * Active Buzzer
  </td>
</tr>
<tr>
<td colspan="2">1 * 40-pin Cable
  </td>
<td>1 * Resistor(1kΩ)
  </td>
</tr>
<tr>
<td colspan="2">1 * Breadboard
  </td>
<td>Several Jumper Wires
  </td>
</tr>
<tr>
<td colspan="2"></td>
<td>1 * S8550 PNP Transistor
  </td>
</tr>
</table>

Principle

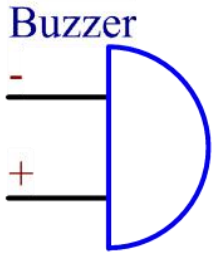
Buzzer

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.

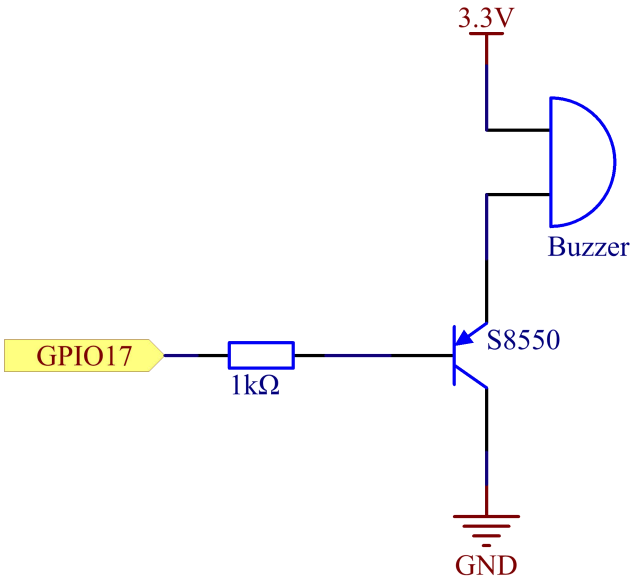


You can check the pins of the buzzer, the longer one is the anode and the shorter one is the cathode. Please don't mix them up when connecting, otherwise the buzzer will not make sound.

Schematic Diagram

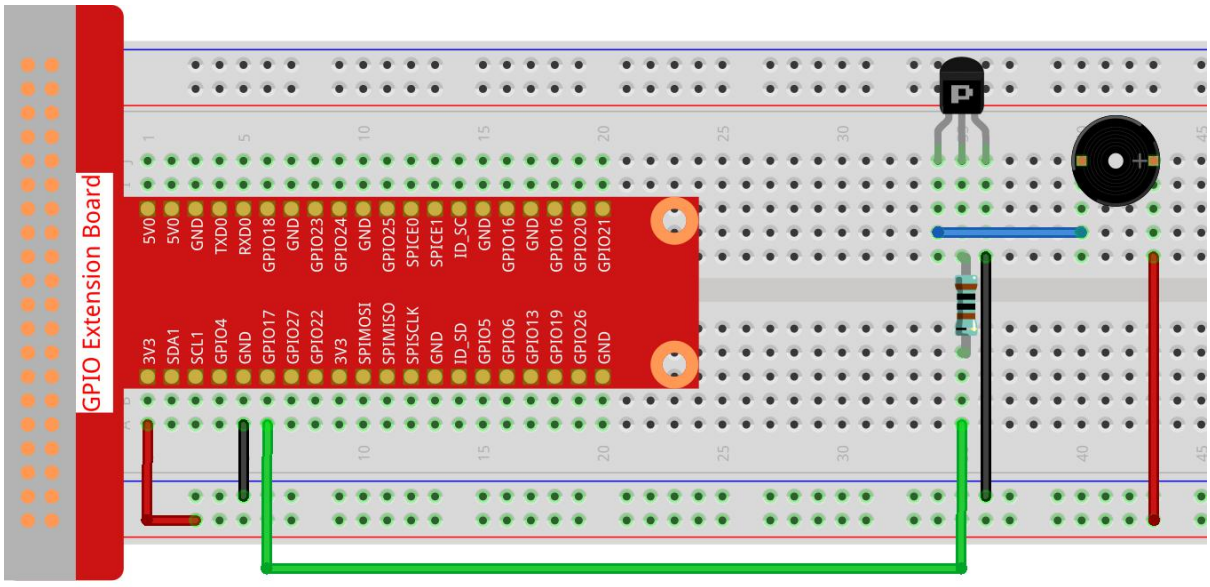
In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and 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.c -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 messageto 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 python3 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.

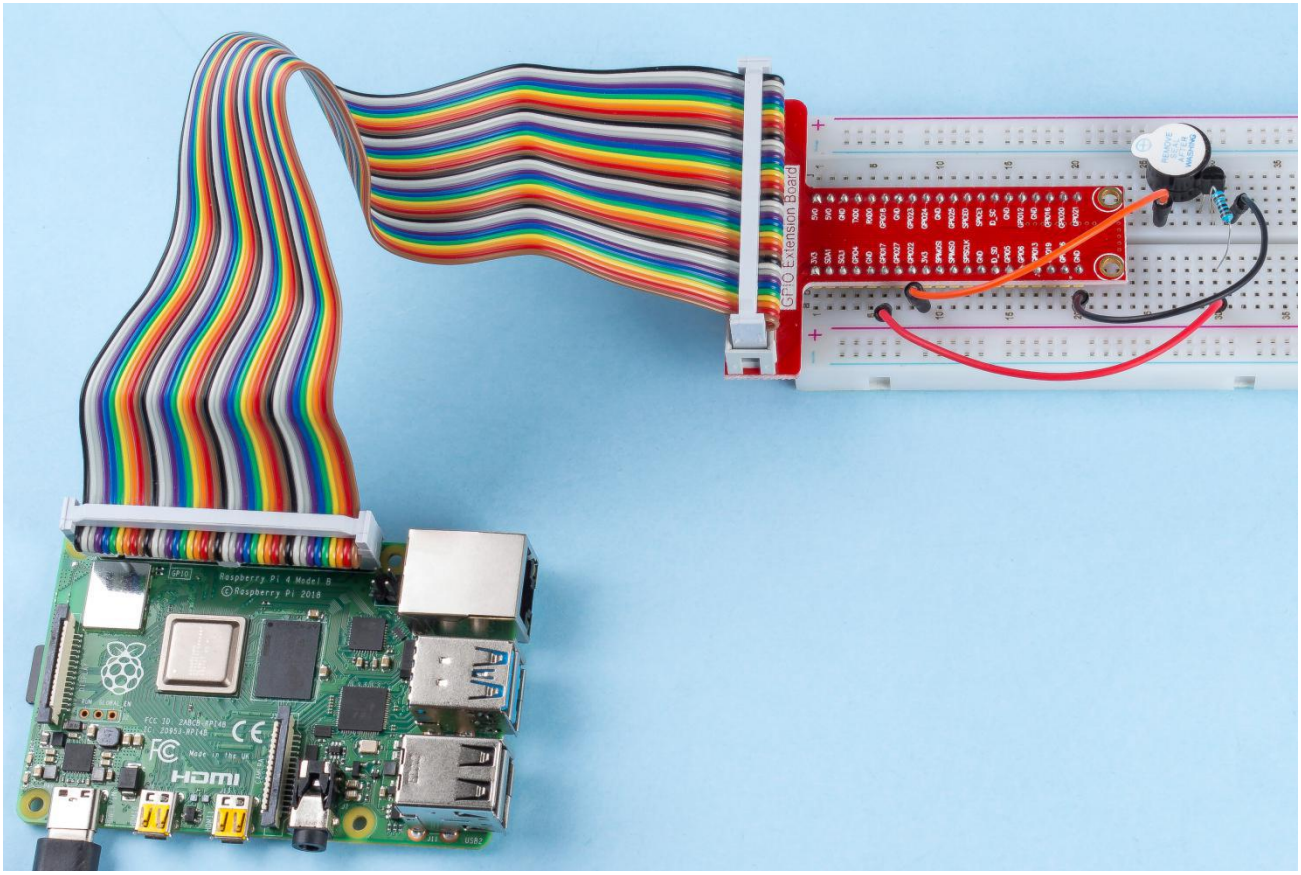
```
time.sleep(0.1)
```

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

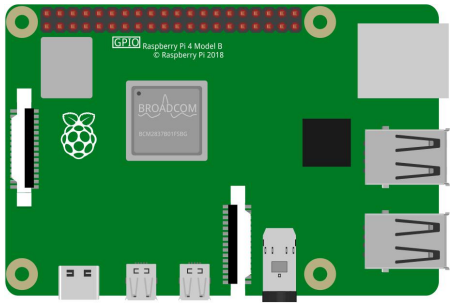
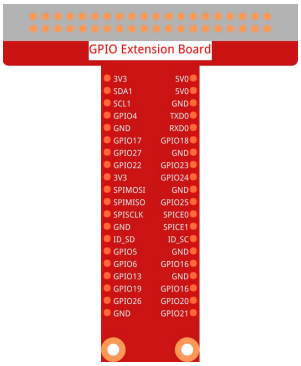
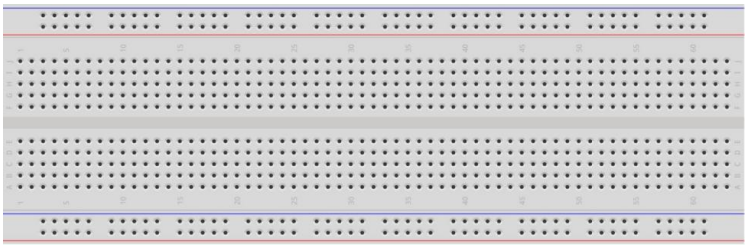


1.2.2 Passive Buzzer

Introduction

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

Components

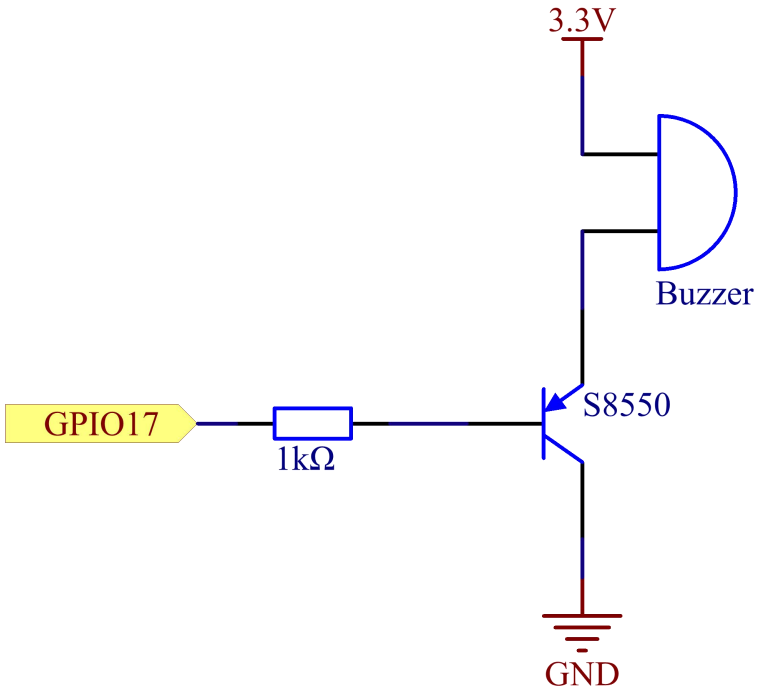
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Passive Buzzer
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>1 * Resistor(1kΩ)
 
Several Jumper Wires
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * S8550 PNP Transistor
  </td></tr>
</table>

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.

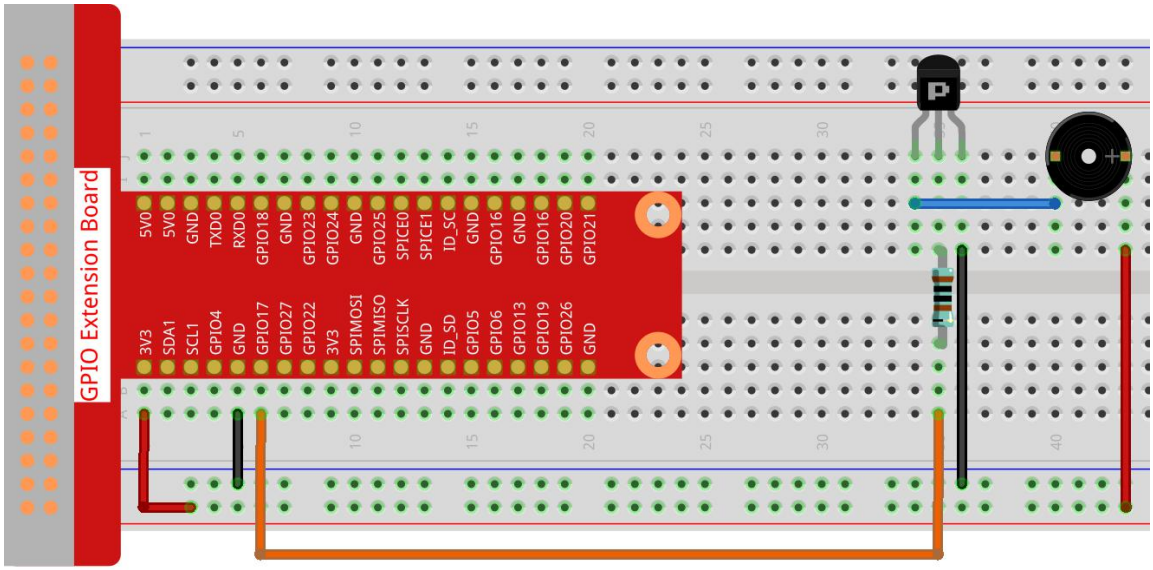
When GPIO17 is given different frequencies, the passive buzzer will emit different sounds; in this way, the buzzer plays music.

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,CL6,CM1,CM2,CM3,CM2,CL7,
                CL6,CM1,CL5};

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


int song_2[] = {CM1,CM1,CM1,CL5,CM3,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[] = {1,1,1,3,1,1,1,3,1,1,1,1,1,1,3,1,1,1,2,1,1,1,3,1,1,1,3,3,2,3};

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[] = {1,1,3,1,1,3,1,1,1,1,1,1,1,3,1,1,3,1,1,1,1,1,1,2,1,1,
                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 errnovariable 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 python3 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, 3, 1,
            1, 3, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1,
            1, 1, 3 ]
```

Every sound beat (each number) represents the ⅛ 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%.

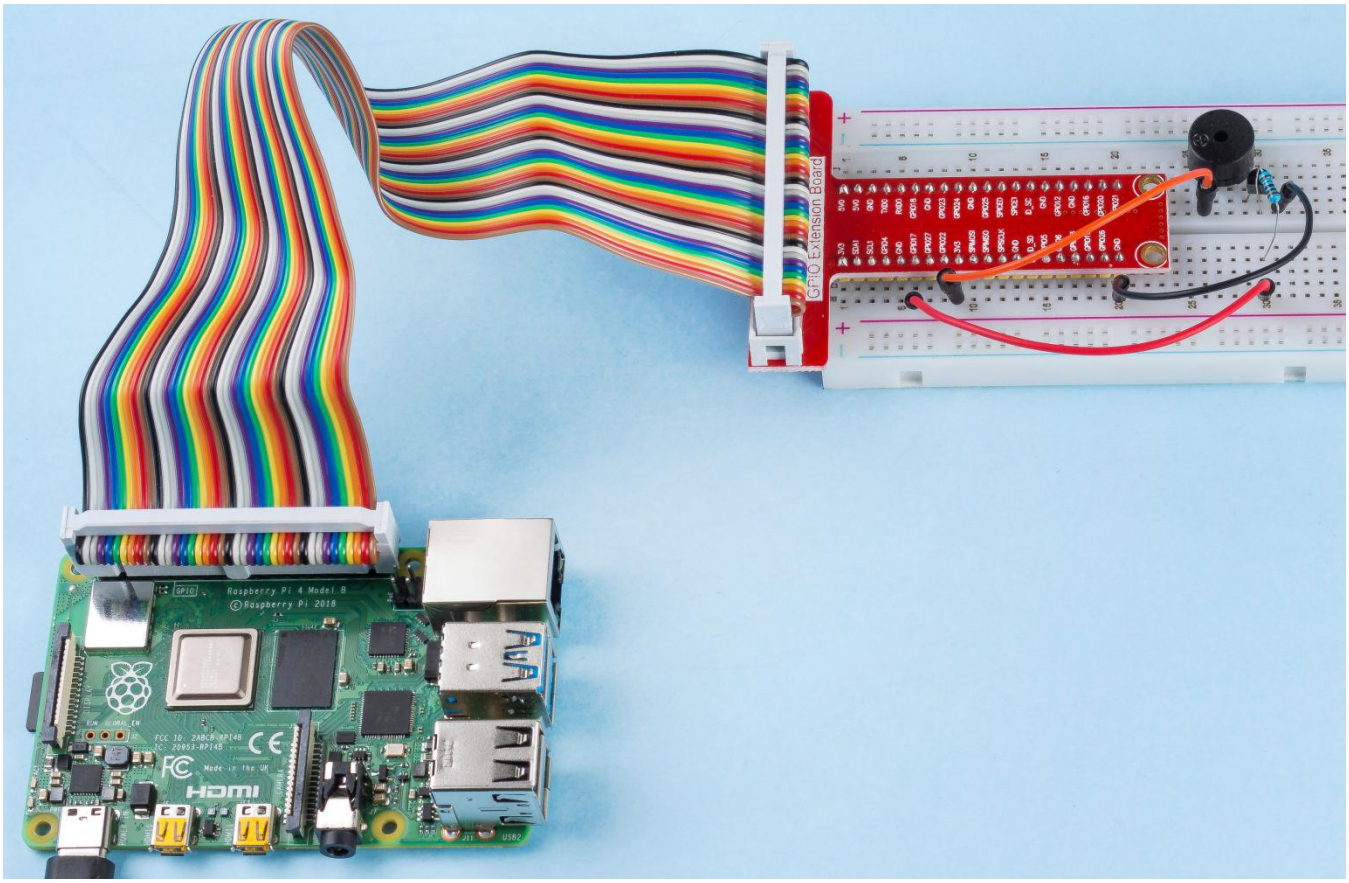
```
        for i in range(1, len(song_1)):
```

```
            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



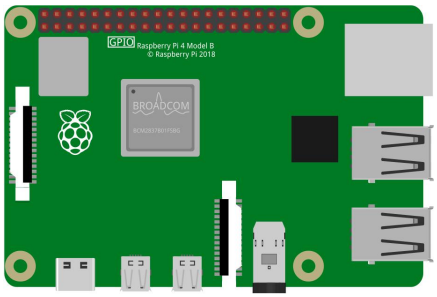
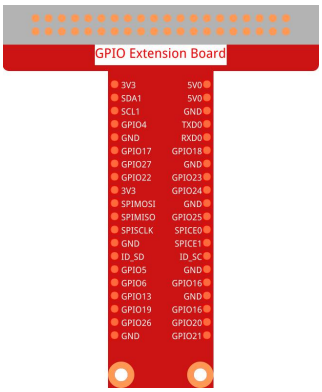
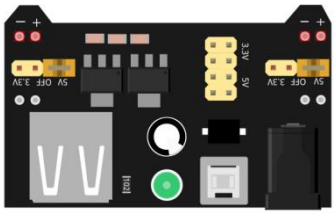
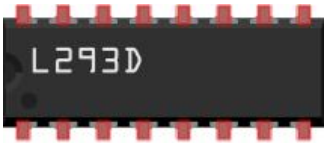
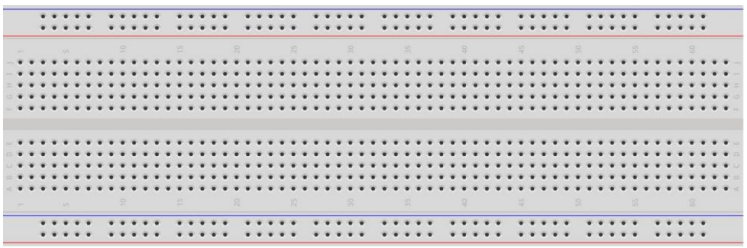
1.3 Drivers

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

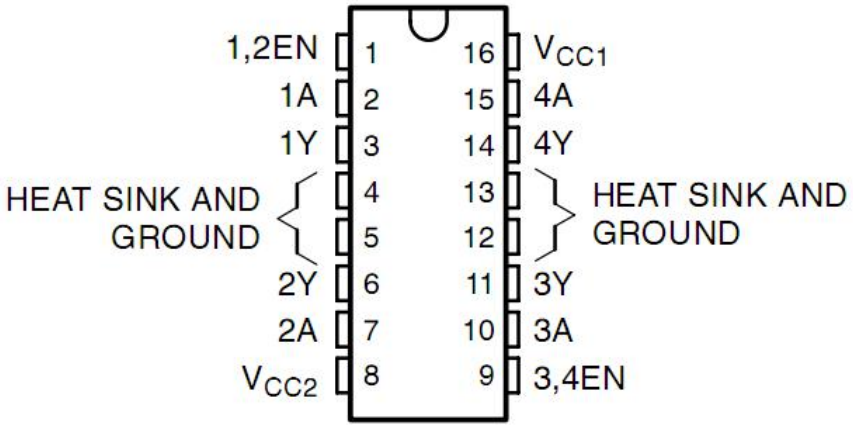
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Power Module (with 9V battery and buckle)
  </td></tr>
<tr><td rowspan="2">1 * 40-pin Cable
  </td><td>1 * L293D
 
</td></tr>
<tr><td>Several Jumper Wires
  </td></tr>
<tr><td>1 * Breadboard
  </td><td>1 * DC Motor
  </td></tr>
</table>

Principle

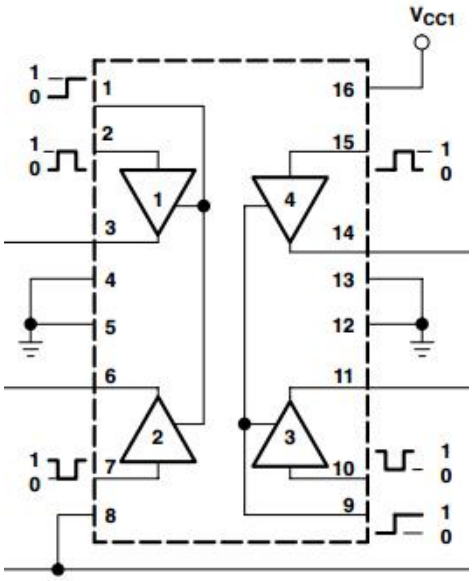
L293D

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): 70m

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

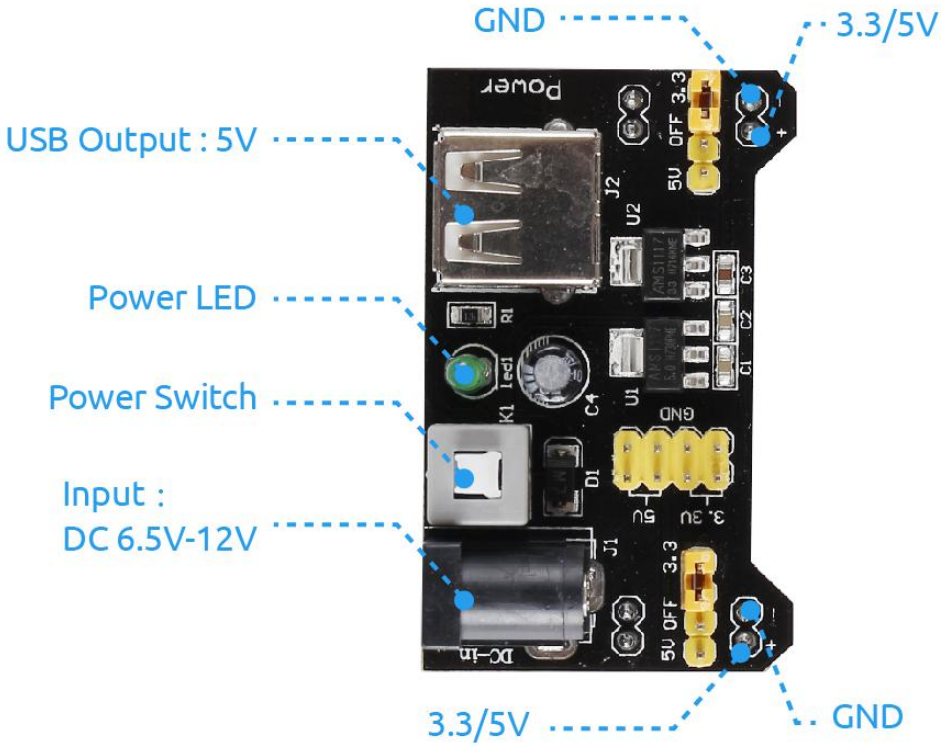
Stall current (3V): 800mA

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.

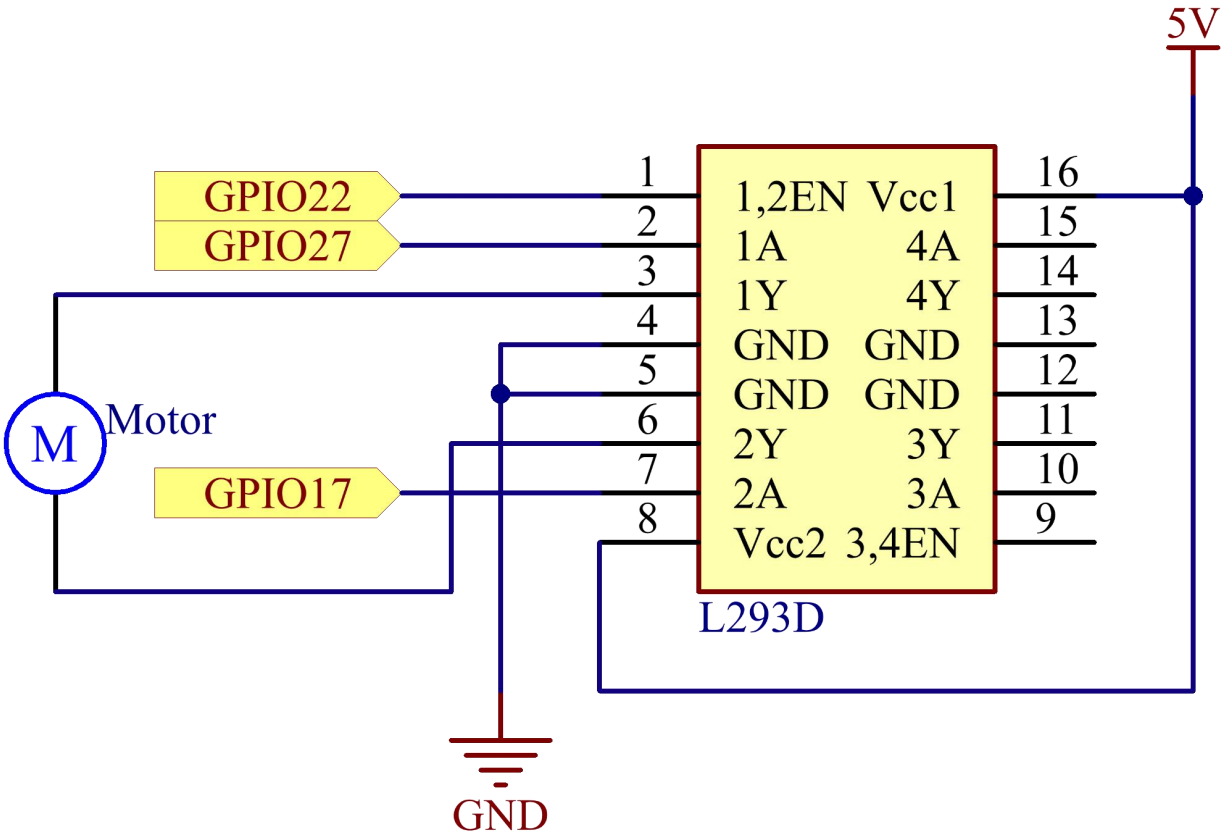
You can just plug it in the breadboard to supply power. It provides a voltage of 3.3V and 5V, and you can connect either via a jumper cap included.



Schematic Diagram

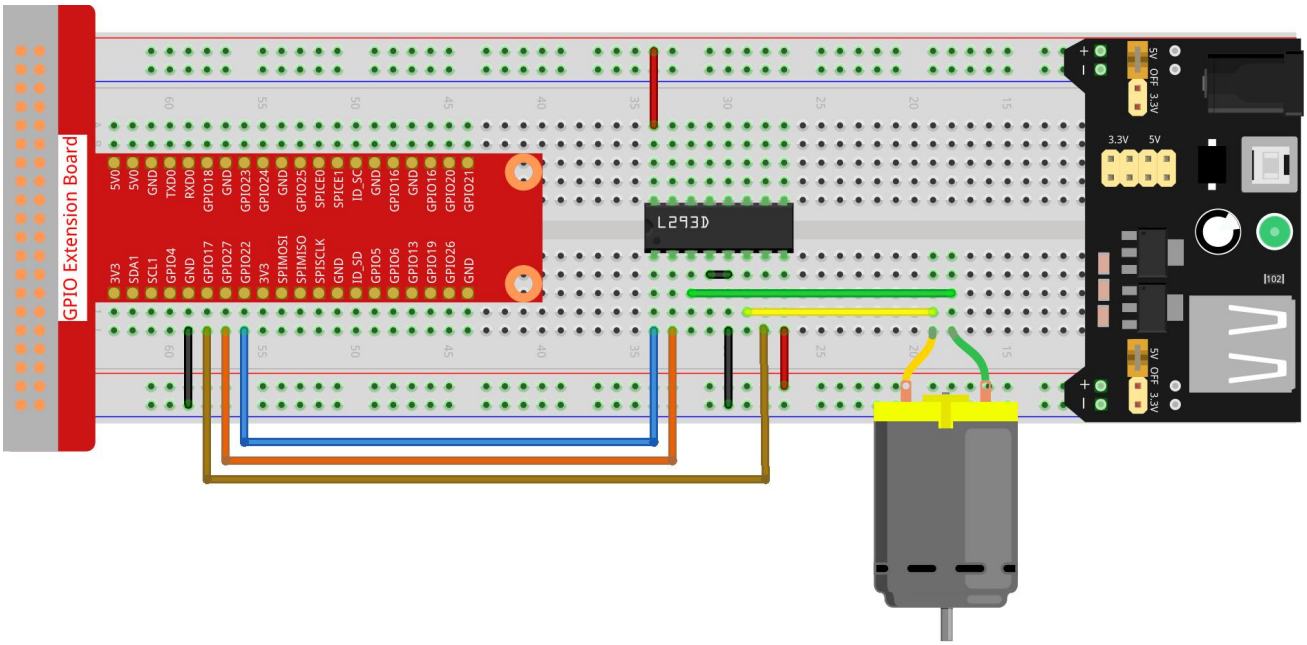
Plug the power supply module in breadboard, and insert the jumper cap to pin of 5V, then it will output voltage of 5V. Connect pin 1 of L293D to GPIO22, and set it as high level. Connect pin2 to GPIO27, and pin7 to GPIO17, then set one pin high, while the other low. Thus you can change the motor's rotation 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.c -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 messageto 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 python3 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'])
```

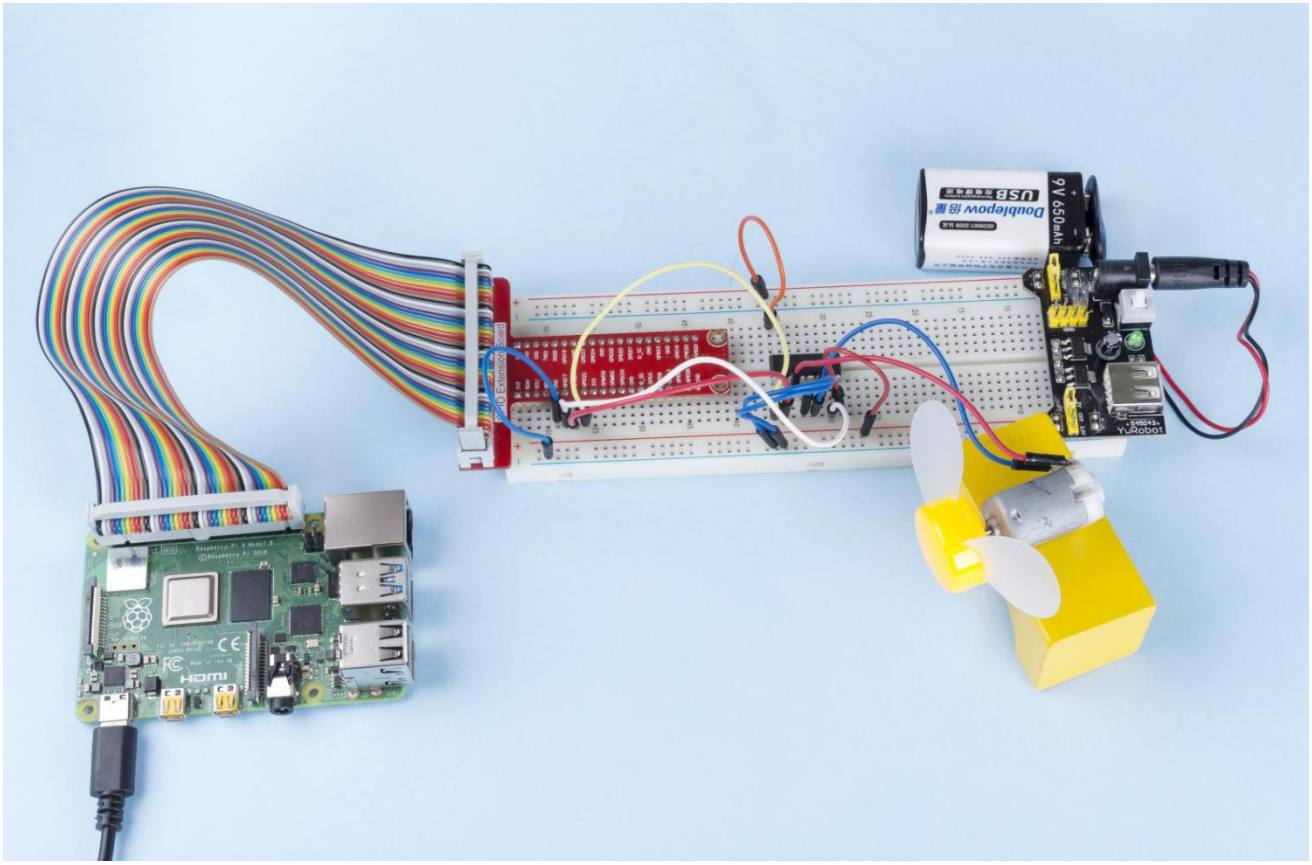
```
        time.sleep(5)
```

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

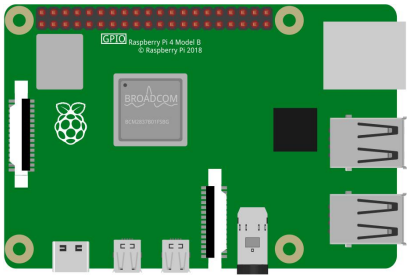
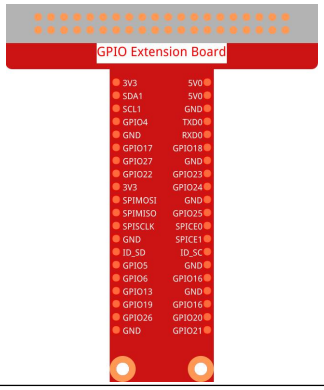


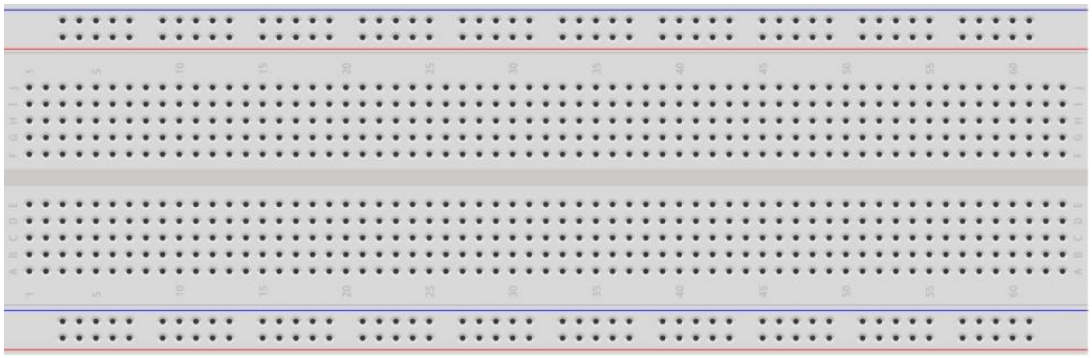
1.3.2 Servo

Introduction

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

Components

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
 
</td>
<td>1 * Servo
 

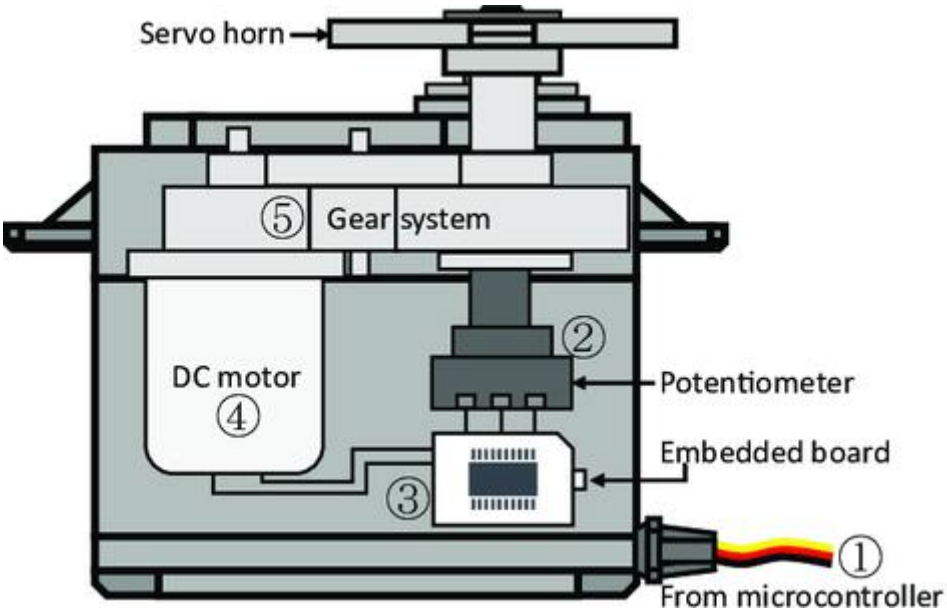
Several Jumper Wires
  </td>
</tr>
<tr>
<td colspan="3">1 * 40-pin Cable
  </td>
</tr>
<tr>
<td colspan="3">1 * Breadboard
  </td>
</tr>
</table>

Principle

Servo

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

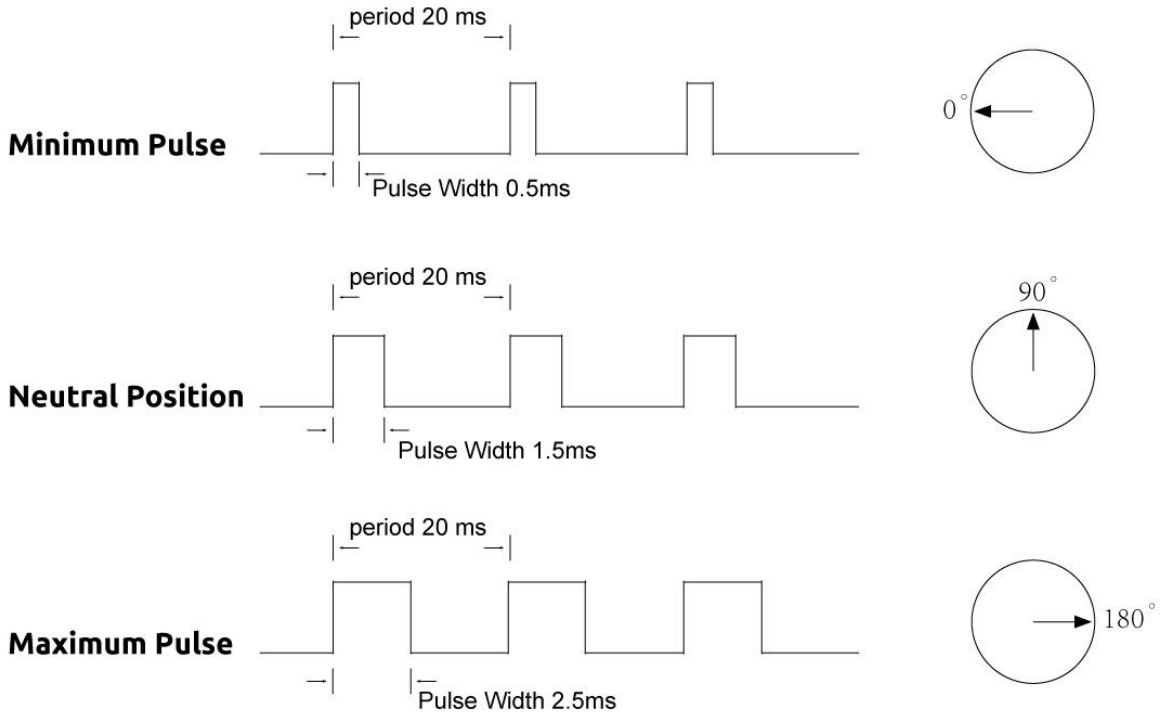
It works like this: The microcontroller sends out PWM signals to the servo, and then the embedded board in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear system and then motivates the shaft after deceleration. The shaft 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.

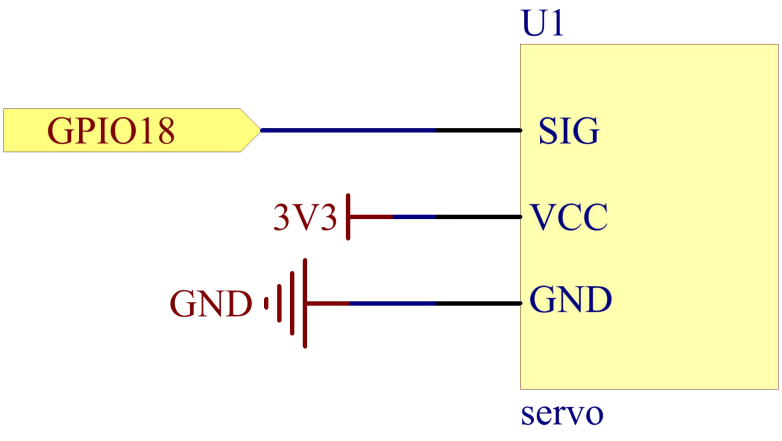
The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position).

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.



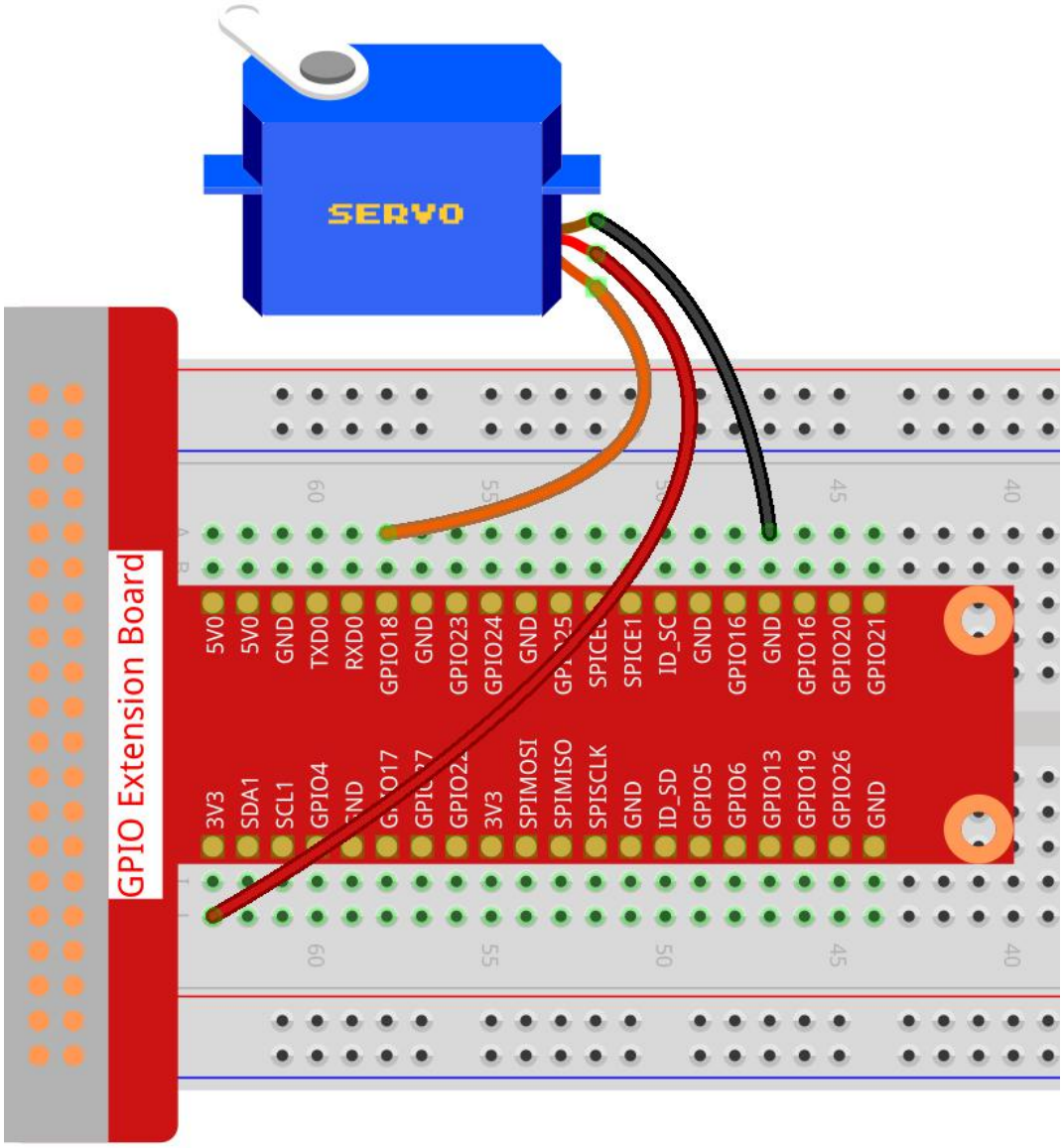
Schematic Diagram

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 servo to GPIO1
long Map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void setAngle(int pin, int angle){      //Create a funtion to control the 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++){       // Let servo rotate from 0 to 180.
         setAngle(ServoPin,i);
            delay(2);
        }
        delay(1000);
        for(i=181;i>-1;i--){           // Let servo rotate from 180 to 0.
         setAngle(ServoPin,i);
            delay(2);
        }
        delay(1000);
    }
    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 setAngle(int pin, int angle){      //Create a funtion to control the 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, setAngle() 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 ~ 180 °, the pulse width should change within the range of 0.5ms ~ 2.5ms when the period is 20ms; in the function, softPwmCreate(), we have set that the period is 200x100us=20ms, thus we need to map 0 ~ 180 to 5x100us ~ 25x100us.

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 100us.

Parameter pwmRange: the period of PWM is that pwmRange times 100us.

➢ 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 python3 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

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500
ServoPin = 18

def map(value, inMin, inMax, outMin, outMax):
    return (outMax - outMin) * (value - inMin) / (inMax - inMin) + outMin

def setup():
    global p
    GPIO.setmode(GPIO.BCM)          # Numbers GPIOs by BCM
    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 Frequecy to 50Hz
    p.start(0)                       # Duty Cycle = 0

def setAngle(angle):         # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it
def loop():
    while True:
        for i in range(0, 181, 5):    #make servo rotate from 0 to 180 deg
            setAngle(i)       # Write to servo
            time.sleep(0.002)
        time.sleep(1)
        for i in range(180, -1, -5): #make servo rotate from 180 to 0 deg
            setAngle(i)
            time.sleep(0.001)
        time.sleep(1)
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 Frequecy 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 setAngle(angle):        # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it
```

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

```
    angle = max(0, min(180, angle))
```

This code is used to limit the angle within the range 0-180°.

The min() function returns the minimum of the input values. If 180<angle, then return 180,if not, return angle.

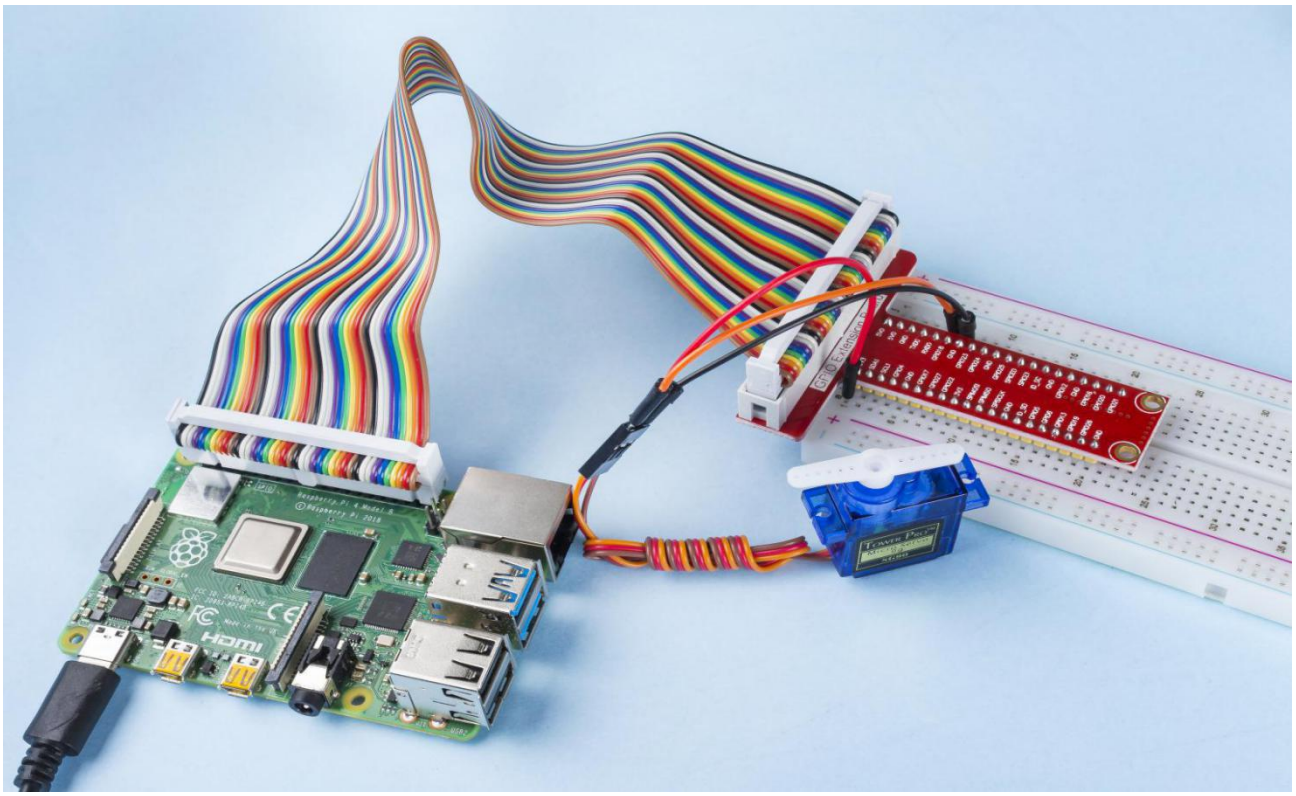
The max() method returns the maximum element in an iterable or largest of two or more parameters. If 0>angle, then return 0, if not, return angle.

```
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)
```

To render a range 0 ~ 180 ° to the servo, the pulse width of the servo is set to 0.5ms(500us)-2.5ms(2500us).

The period of PWM is 20ms(20000us), thus the duty cycle of PWM is (500/20000)%-(2500/20000)%, and the range 0 ~ 180 is mapped to 2.5 ~ 12.5.

Phenomenon Picture

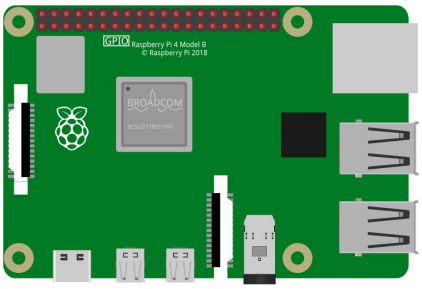
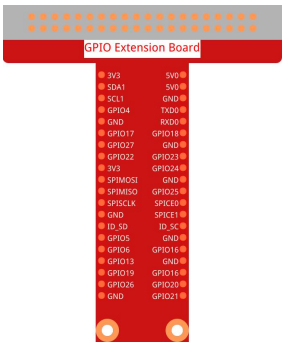
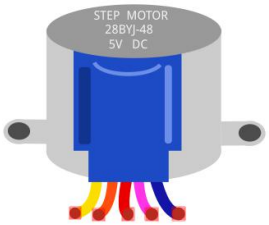
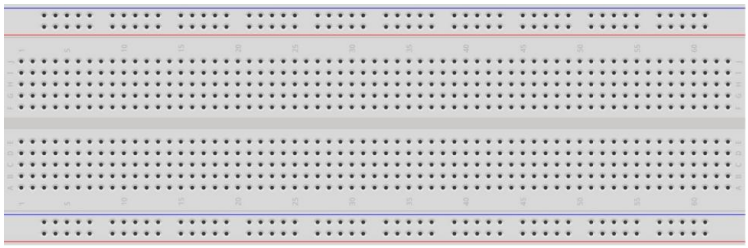
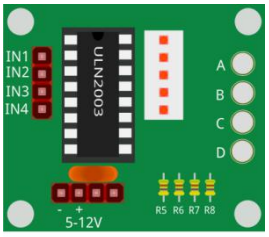


1.3.3 Stepper Motor

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

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Stepper Motor
  </td></tr>
<tr><td>1 * 40-pin Cable
  </td><td colspan="2">Several Jumper Wires
  </td></tr>
<tr><td>1 * Breadboard
  </td><td colspan="2">1 * ULN2003
  </td></tr>
</table>

Principle

Stepper Motor

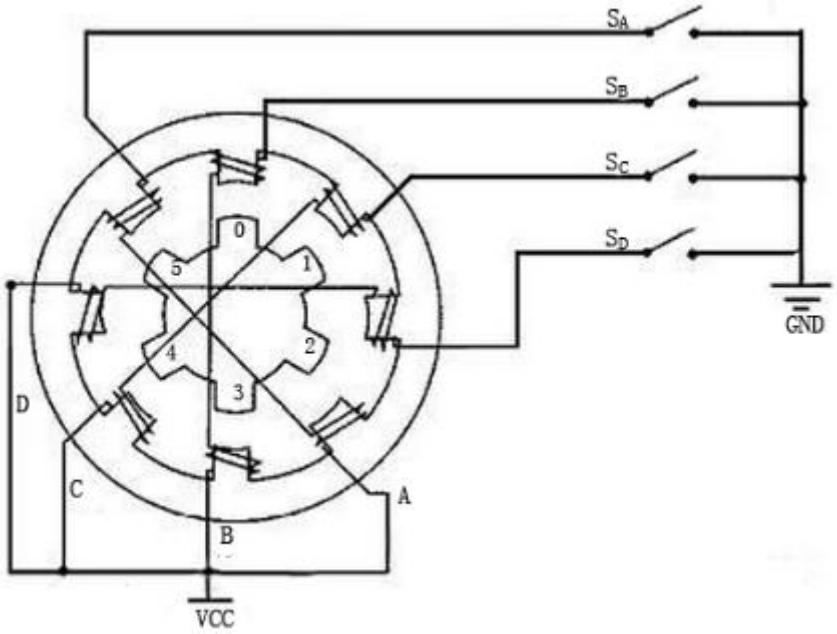
There are two types of steppers, unipolars and bipolars, and it is very important to know which type you are working with. In this experiment, we will use a unipolar stepper.

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:

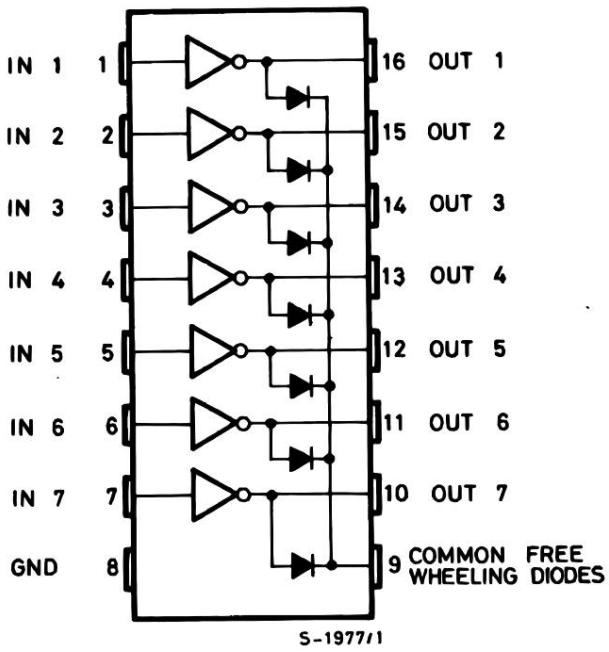
When switch SB is power on, switch SA, SC, and SD is power off, and B-phase magnetic poles align with tooth 0 and 3 of the rotor. At the same time, tooth 1 and 4 generate staggered teeth with C- and D-phase poles. Tooth 2 and 5 generate staggered teeth with D- and A-phase poles. When switch SC is power 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*64=2048 step.

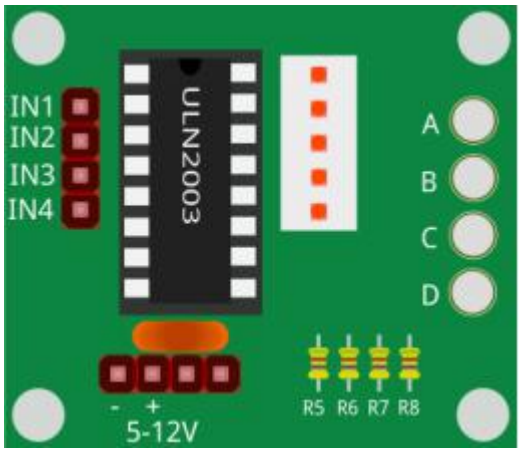
ULN2003



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.

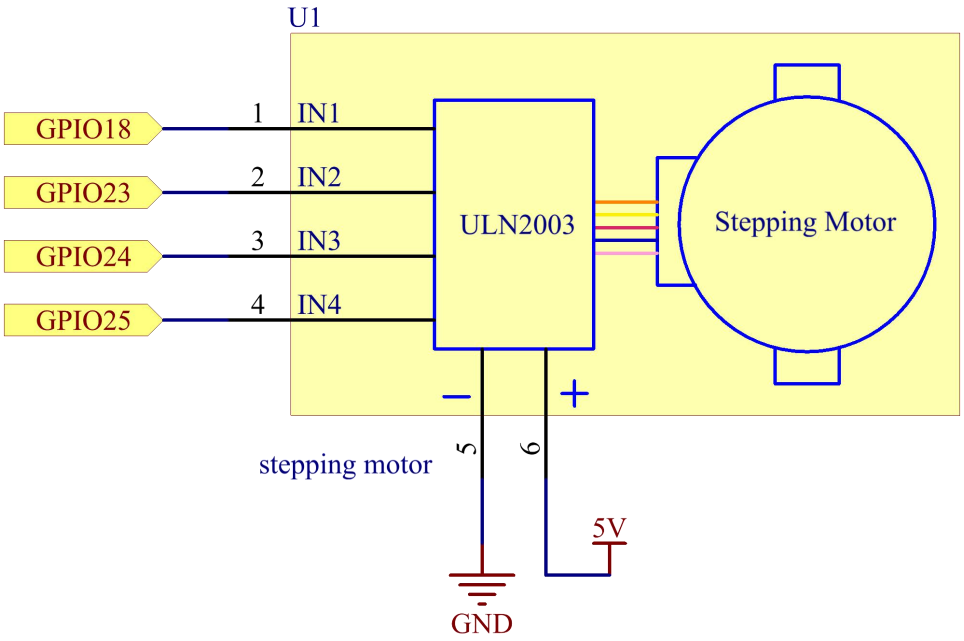
The internal structure of the chip is shown as below.

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.



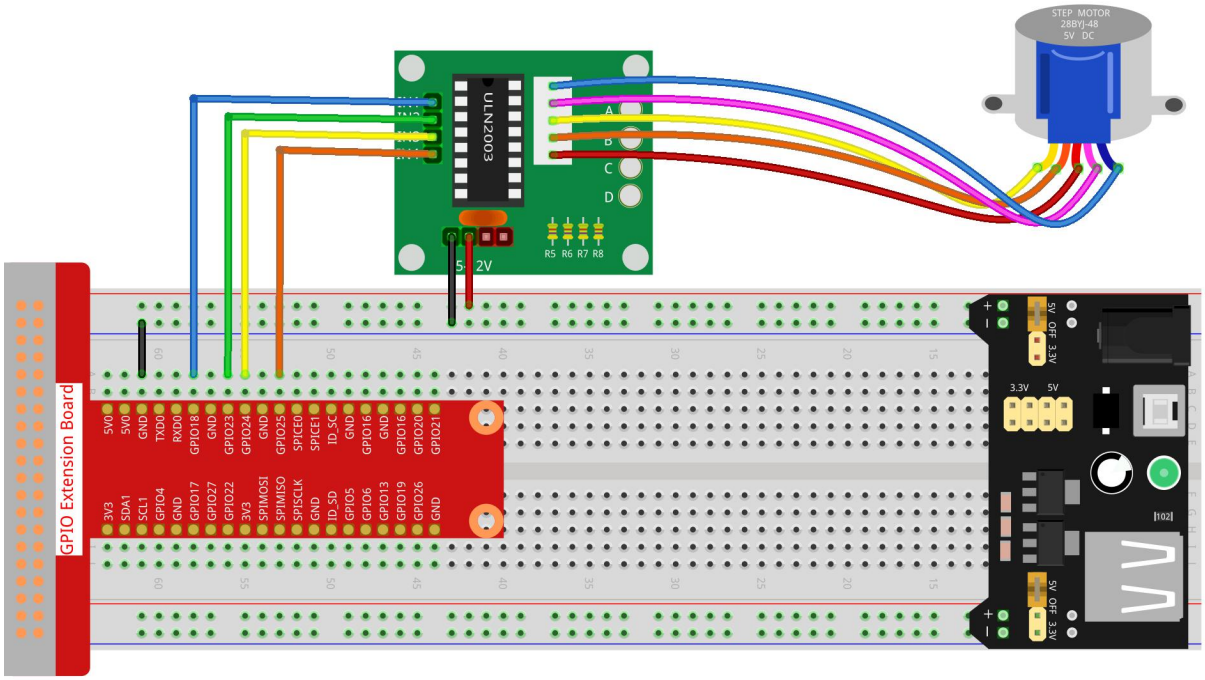
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
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_StepperMotor.c -lwiringPi
```

Step 4: Run the executable file.

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

As the code runs, the stepper motor will rotate clockwise or anticlockwise according to your input ‘a’ or ‘c’.

Code

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

const int motorPin[] = {1, 4, 5, 6};
int rolePerMinute = 15;
int stepsPerRevolution = 2048;
int stepSpeed = 0;
```

```
void rotary(char direction){
    if(direction == 'c'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}
            delayMicroseconds(stepSpeed);
        }
    }
    else if(direction =='a'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}
            delayMicroseconds(stepSpeed);
        }
    }
}

void loop()
{
    char direction = '0';
    while (1)
    {
        printf("select motor direction a=anticlockwise, c=clockwise: ");
        direction=getchar();
        if (direction == 'c')
        {
            printf("motor running clockwise\n");
            break;
        }
        else if (direction == 'a')
        {
            printf("motor running anti-clockwise\n");
            break;
        }
        else
        {
            printf("input error, please try again!\n");
        }
    }
    while(1)
```

```
    {
        rotary(direction);
    }
}

void main(void)
{
    if (wiringPiSetup() == -1)
    {
        printf("setup wiringPi failed !");
        return;
    }
    for (int i = 0; i < 4; i++)
    {
        pinMode(motorPin[i], OUTPUT);
    }
    stepSpeed = (60000000 / rolePerMinute) / stepsPerRevolution;
    loop();
}
```

Code Explanation

```
int rolePerMinute = 15;
int stepsPerRevolution = 2048;
int stepSpeed = 0;
```

rolePerMinute: revolutions per minute, the RPM of the stepper motor used in this kit should be 0~17.

stepPerRevolution: the number of steps for each turn, and the stepper motor used in this kit needs 2048 steps per revolution.

stepSpeed: the time used for each step, and in main(), we assign the values to them: 「(60000000 / rolePerMinute) / stepsPerRevolution」(60,000,000 us=1minute)

```
void loop()
{
    char direction = '0';
    while (1)
    {
        printf("select motor direction a=anticlockwise, c=clockwise: ");
```

```
        direction=getchar();
        if (direction == 'c')
        {
            printf("motor running clockwise\n");
            break;
        }
        else if (direction == 'a')
        {
            printf("motor running anti-clockwise\n");
            break;
        }
        else
        {
            printf("input error, please try again!\n");
        }
    }
    while(1)
    {
        rotary(direction);
    }
}
```

The loop() function is roughly divided into two parts (located between two while(1)) : The first part is to get the key value. When 'a' or 'c' is obtained, exit the loop and stop the input.
The second part calls rotary(direction) to make the stepper motor run.

```
void rotary(char direction){
    if(direction == 'c'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}
            delayMicroseconds(stepSpeed);
        }
    }
    else if(direction =='a'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}
            delayMicroseconds(stepSpeed);
        }
```

```
    }
}
```

To make stepper motor **rotate clockwise**, level status of motorPin should is shown in the table below:

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH
Step2	HIGH	HIGH	LOW	LOW
Step3	LOW	HIGH	HIGH	LOW
Step4	LOW	LOW	HIGH	HIGH
Step5(Step1)	HIGH	LOW	LOW	HIGH

Therefore, potential write of MotorPin is implemented by using a two-layer of for loop.

In Step1, j=0, i=0~4.

motorPin[0] will be written in the high level（10011001&00001000=1）

motorPin[1] will be written in the low level（10011001&00000100=0）

motorPin[2] will be written in the low level（10011001&00000010=0）

motorPin[3] will be written in the high level（10011001&00000001=1）

In Step2, j=1, i=0~4.

motorPin[0] will be written in the high level（01001100&00001000=1）

motorPin[1] will be written in the low level（01001100&00000100=1）

and so on.

And to make the stepper motor rotate **anti-clockwise**, the level status of motorPin is shown in the following table.

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH

Step2	**LOW**	**LOW**	**HIGH**	**HIGH**
Step3	**LOW**	**HIGH**	**HIGH**	**LOW**
Step4	**HIGH**	**HIGH**	**LOW**	**LOW**
Step5(1)	**HIGH**	**LOW**	**LOW**	**HIGH**

In Step1, j=0, i=0~4.

motorPin[0] will be written in the high level（10011001&10000000=1）

motorPin[1] will be written in the low level（10011001&01000000=0）

In Step2，j=1, i=0~4.

motorPin[0] will be written in the high level（00110010&10000000=0）

motorPin[1] will be written in the low level（00110010&01000000=0）

and so on.

➢ 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 python3 1.3.3_StepperMotor.py
```

As the code runs, the stepper motor will turn clockwise or anti-clockwise depending on your input 'a' or 'c'.

Code

```
import RPi.GPIO as GPIO
from time import sleep

motorPin = (18,23,24,25)
rolePerMinute =15
stepsPerRevolution = 2048
stepSpeed = (60/rolePerMinute)/stepsPerRevolution

def setup():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
```

```
    for i in motorPin:
        GPIO.setup(i, GPIO.OUT)

def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99>>j & (0x08>>i))
            sleep(stepSpeed)

    elif(direction == 'a'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
            sleep(stepSpeed)

def loop():
    while True:
        direction = input('select motor direction a=anticlockwise, c=clockwise: ')
        if(direction == 'c'):
            print('motor running clockwise\n')
            break
        elif(direction == 'a'):
            print('motor running anti-clockwise\n')
            break
        else:
            print('input error, please try again!')
    while True:
        rotary(direction)

def destroy():
    GPIO.cleanup()

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

Code Explanation

```
rolePerMinute =15
stepsPerRevolution = 2048
stepSpeed = (60/rolePerMinute)/stepsPerRevolution
```

rolePerMinute: revolutions per minute, the RPM of the stepper motor used in this kit should be 0~17.

stepPerRevolution: the number of steps for each turn, and the stepper motor used in this kit needs 2048 steps per revolution.

stepSpeed: the time used for each step, and we assign the values to them: 「(60 / rolePerMinute) / stepsPerRevolution」(60s=1minute).

```
def loop():
    while True:
        direction = input('select motor direction a=anticlockwise, c=clockwise: ')
        if(direction == 'c'):
            print('motor running clockwise\n')
            break
        elif(direction == 'a'):
            print('motor running anti-clockwise\n')
            break
        else:
            print('input error, please try again!')
    while True:
        rotary(direction)
```

The loop() function is roughly divided into two parts (located in two while(1)) :
The first part is to get the key value. When 'a' or 'c' is obtained, exit the loop and stop the input.
The second part calls rotary(direction) to make the stepper motor run.

```
def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99>>j & (0x08>>i))
            sleep(stepSpeed)

    elif(direction == 'a'):
```

```
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
            sleep(stepSpeed)
```

To make the stepper motor rotate clockwise, the level status of motorPin is shown in the following table:

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH
Step2	HIGH	HIGH	LOW	LOW
Step3	LOW	HIGH	HIGH	LOW
Step4	LOW	LOW	HIGH	HIGH
Step5(1)	HIGH	LOW	LOW	HIGH

Therefore, potential write of MotorPin is implemented by using a two-layer of for loop.

In Step1, j=0, i=0~4.

motorPin[0] will be written in the high level（10011001&00001000=1）

motorPin[1] will be written in the low level（10011001&00000100=0）

motorPin[2] will be written in the low level（10011001&00000010=0）

motorPin[3] will be written in the high level（10011001&00000001=1）

In Step2, j=1, i=0~4.

motorPin[0] will be written in the high level（01001100&00001000=1）

motorPin[1] will be written in the low level（01001100&00000100=1）

and so on

And to make the stepper motor rotate anti - clockwise, the level status of motorPin is shown in the following table.

	MotorPin A	MotorPin B	MotorPin C	MotorPin D
Step1	HIGH	LOW	LOW	HIGH

Step2	**LOW**	**LOW**	**HIGH**	**HIGH**
Step3	**LOW**	**HIGH**	**HIGH**	**LOW**
Step4	**HIGH**	**HIGH**	**LOW**	**LOW**
Step5(1)	**HIGH**	**LOW**	**LOW**	**HIGH**

In Step1, j=0, i=0~4.

motorPin[0] will be written in the high level（10011001&10000000=1）

motorPin[1] will be written in the low level（10011001&01000000=0）

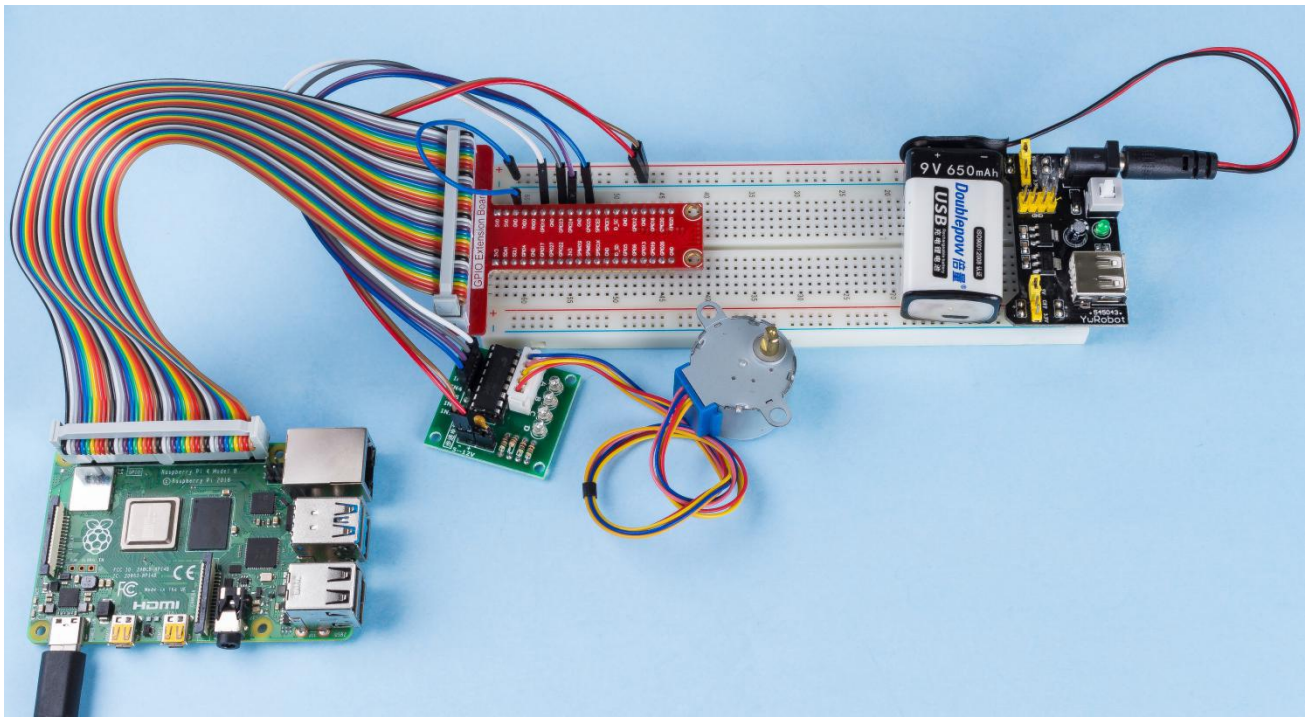
In Step2, j=1, i=0~4.

motorPin[0] will be written in the high level（00110010&10000000=0）

motorPin[1] will be written in the low level（00110010&01000000=0）

And so on.

Phenomenon Picture



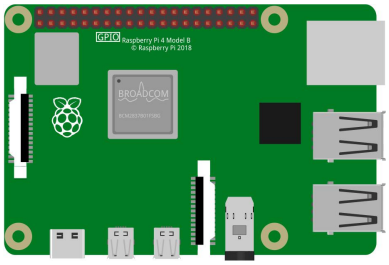
1.3.4 Relay

Introduction

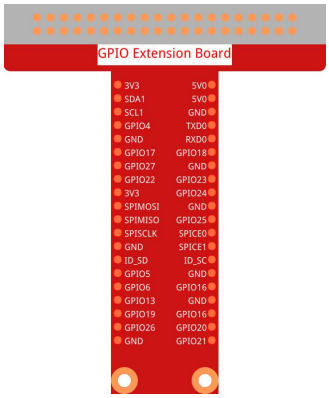
In this lesson, we will learn to use a relay. It is one of the commonly used components in automatic control system. When the voltage, current, temperature, pressure, etc., reaches, exceeds or is lower than the predetermined value, the relay will connect or interrupt the circuit, to control and protect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

Components

1 * Raspberry Pi



1 * T-Extension Board



1 * Relay

1 * 1N4007 Diode

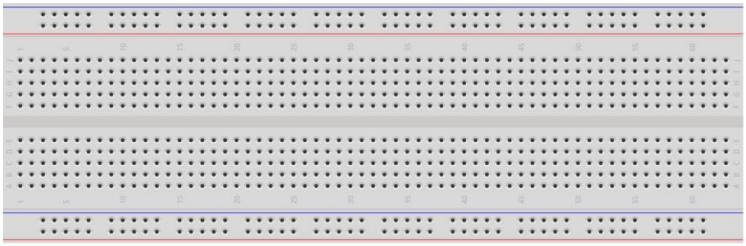
1 * LED

1 * S8050 NPN Transistor

1 * 40-pin Cable

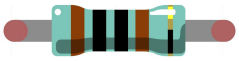
Several Jumper Wires

1 * Breadboard



1 * Resistor(220Ω)

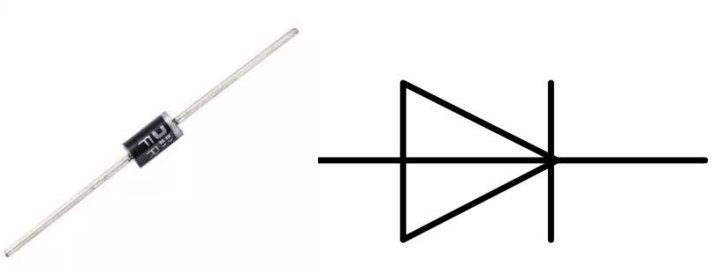
1 * Resistor 1KΩ



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 it 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.

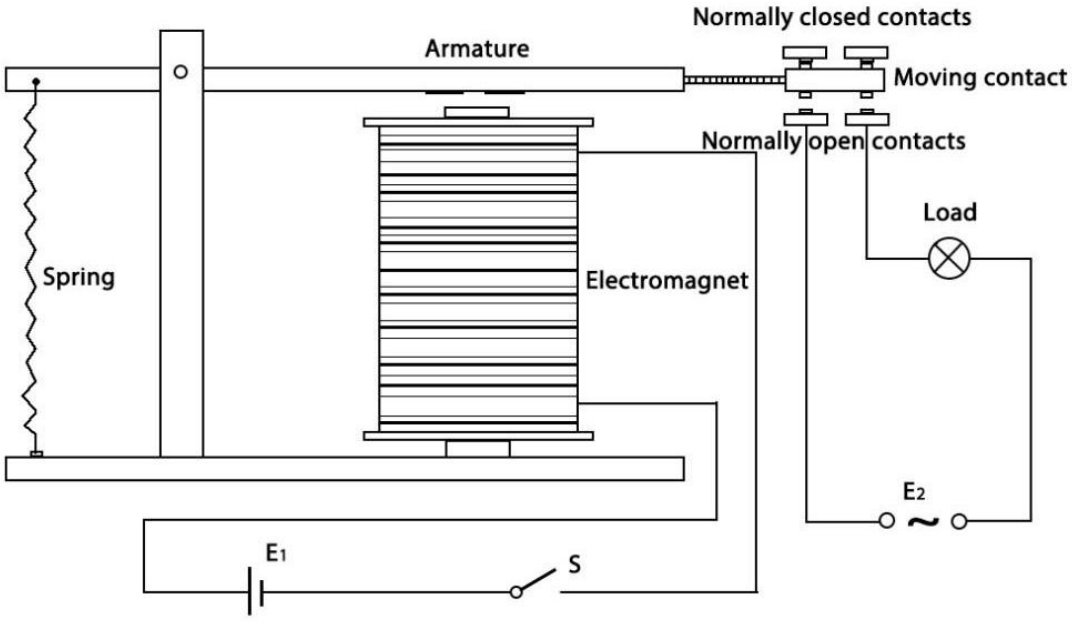
4. Set of electrical **contacts** - There are two contact points:

- 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.

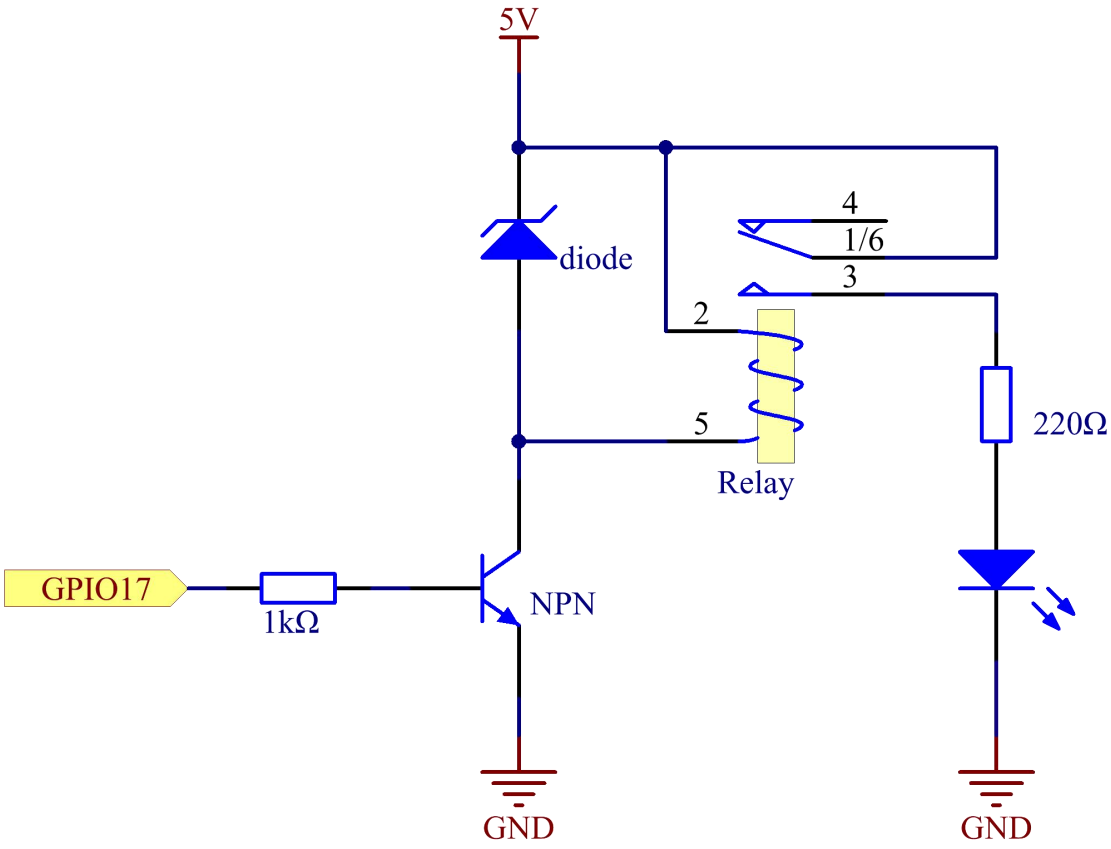
Working of Relay

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.



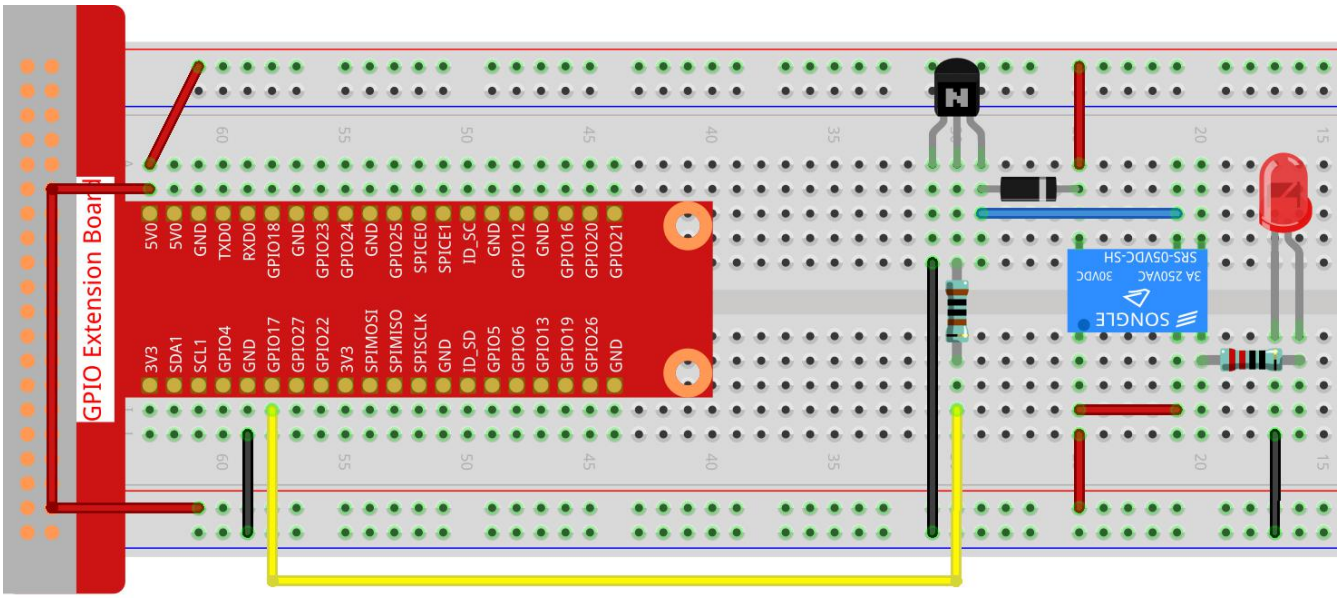
Schematic Diagram

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/1.3.4
```

Step 3: Compile the code.

```
gcc 1.3.4_Relay.c -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 python3 1.3.4_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

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

import RPi.GPIO as GPIO
import time

# Set GPIO17 as control 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.

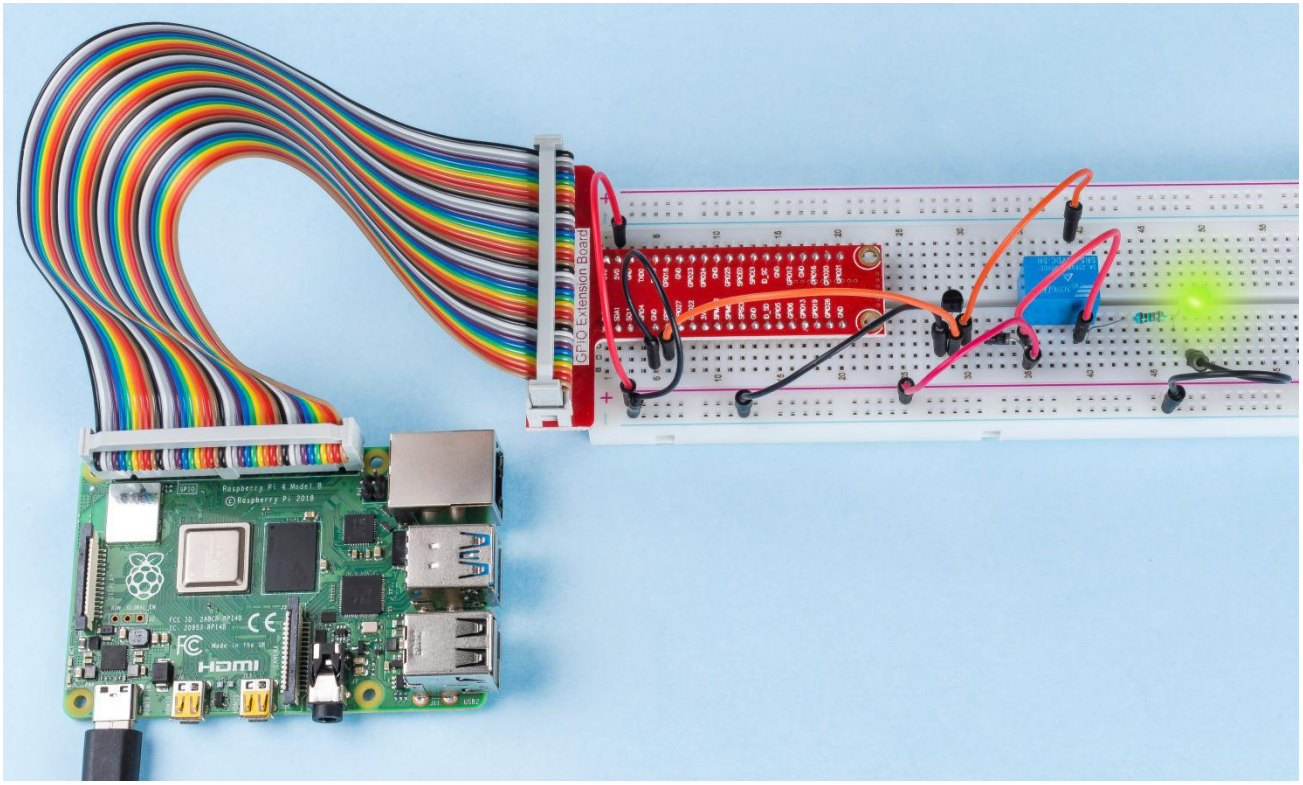
```
time.sleep(1)
```

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 Input

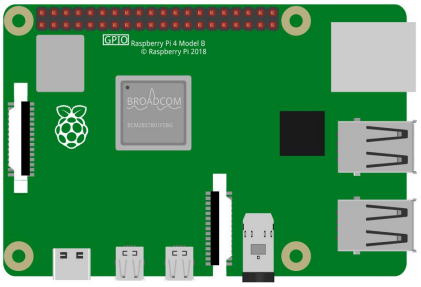
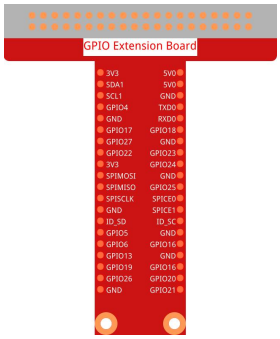
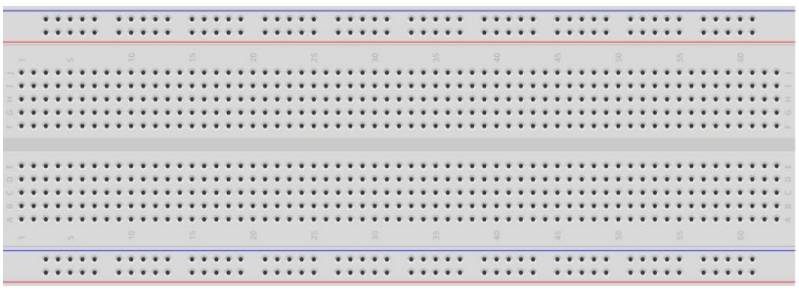
2.1 Controllers

2.1.1 Button

Introduction

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

Components

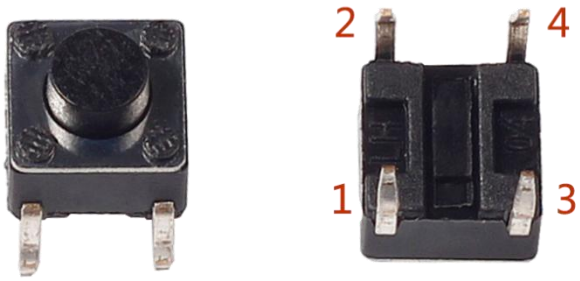
1 * Raspberry Pi 	1 * T-Extension Board 	1 * LED 
		1 * Resistor 10KΩ 
1 * 40-pin Cable 		1 * Resistor(220Ω) 
		Several Jumper Wires 
1 * Breadboard 		1 * Button 

Principle

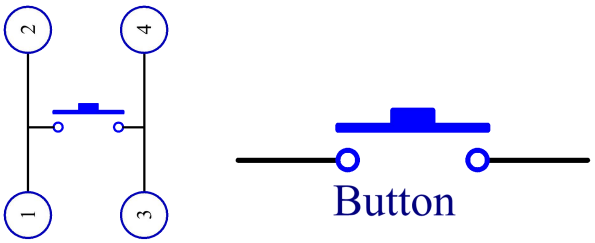
Button

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.



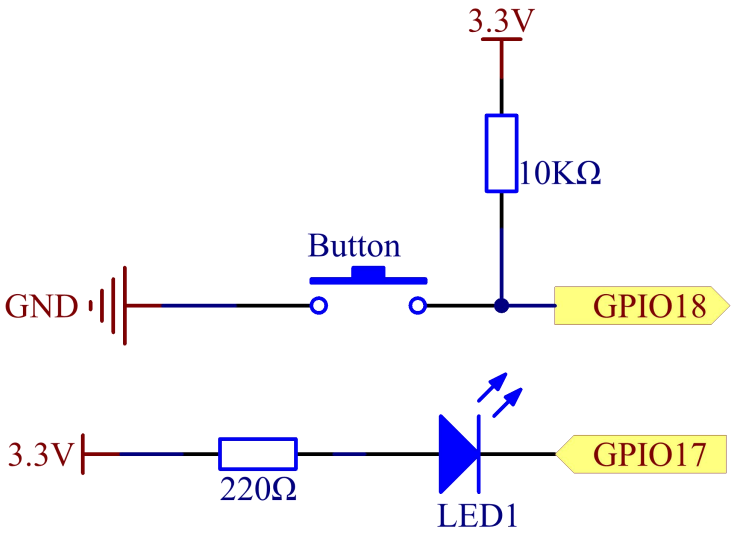
When the button is pressed, the 4 pins are connected, thus closing the circuit.

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.

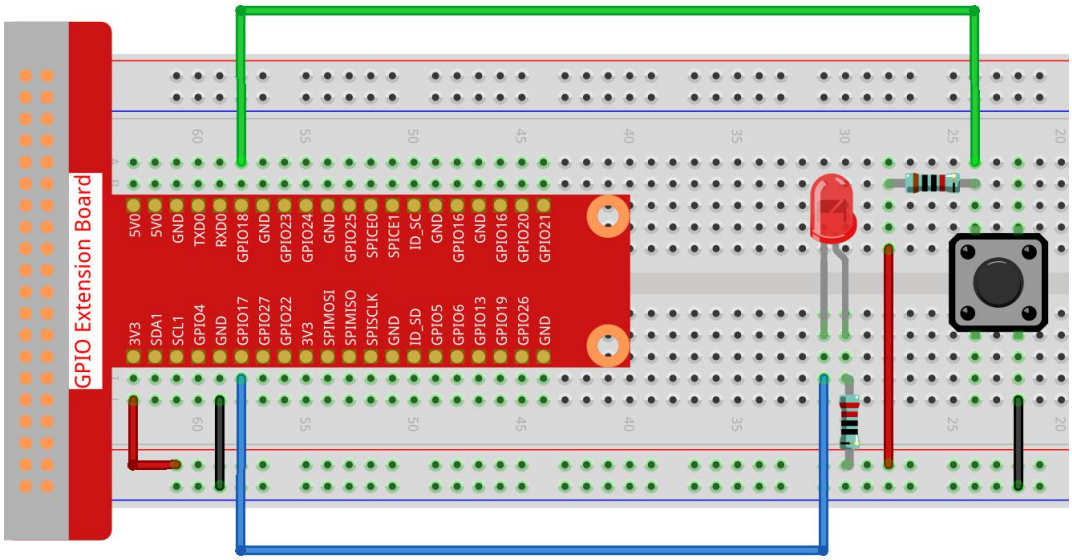
Note: The longer pin of the LED is the anode and the shorter one is the cathode.

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);
    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.

```
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 python3 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
LedPin = 17   # Set GPIO17 as LED pin
BtnPin = 18   # Set GPIO18 as button pin

# 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)


# 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():
    # Set up a falling detect on BtnPin,
    # and callback function to swLed
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
    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

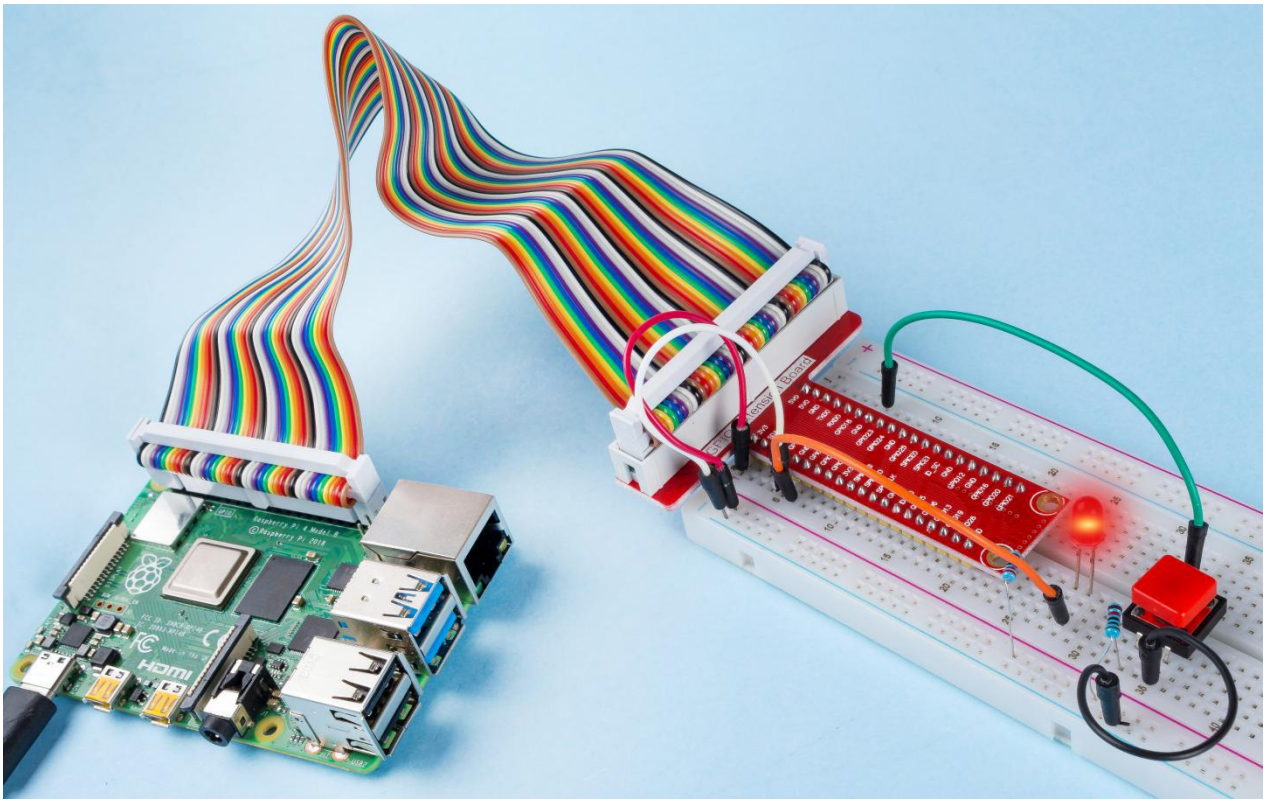
```
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

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



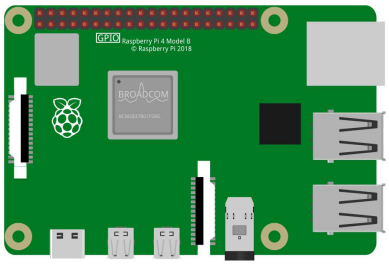
2.1.2 Slide Switch

Introduction

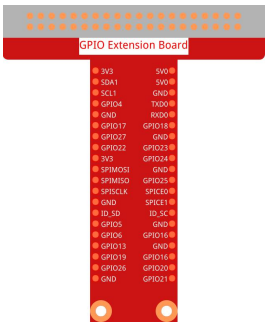
In this lesson, we will learn how to use a slide switch. Usually,the slide switch is soldered on PCB as a power switch, but here we need to insert it into the breadboard, thus it may not be tightened. And we use it on the breadboard to show its function.

Components

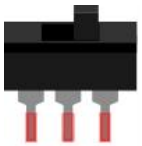
1 * Raspberry Pi



1 * T-Extension Board



1 * Slide Switch



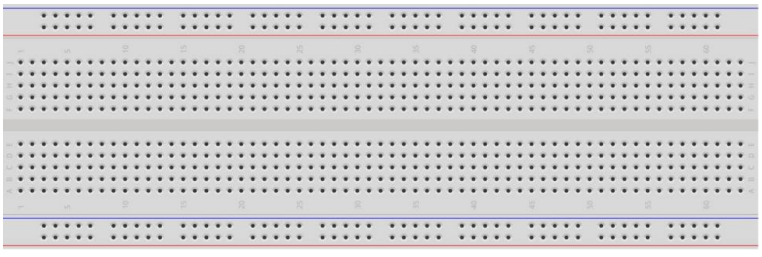
2 * LED

1 * 104 Capacitor

1 * 40-pin Cable

Several Jumper Wires

1 * Breadboard



2 * Resistor(220Ω)

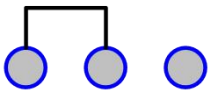
1 * Resistor 10KΩ

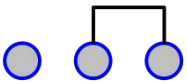
Principle

Slide Switch

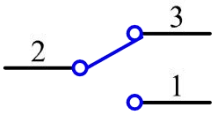
A slide switch, just as its name implies, is to slide the switch bar to connect or break the circuit, and further switch circuits. The common-used types are SPDT, SPTT, DPDT, DPTT etc. The slide switch is commonly used in low-voltage circuit. It has the features of flexibility and stability, and applies 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 (μF) 1μF = 1/1,000,000 = 0.000001 = 10^{-6} F

Nanofarad (nF) 1nF = 1/1,000,000,000 = 0.000000001 = 10^{-9}F

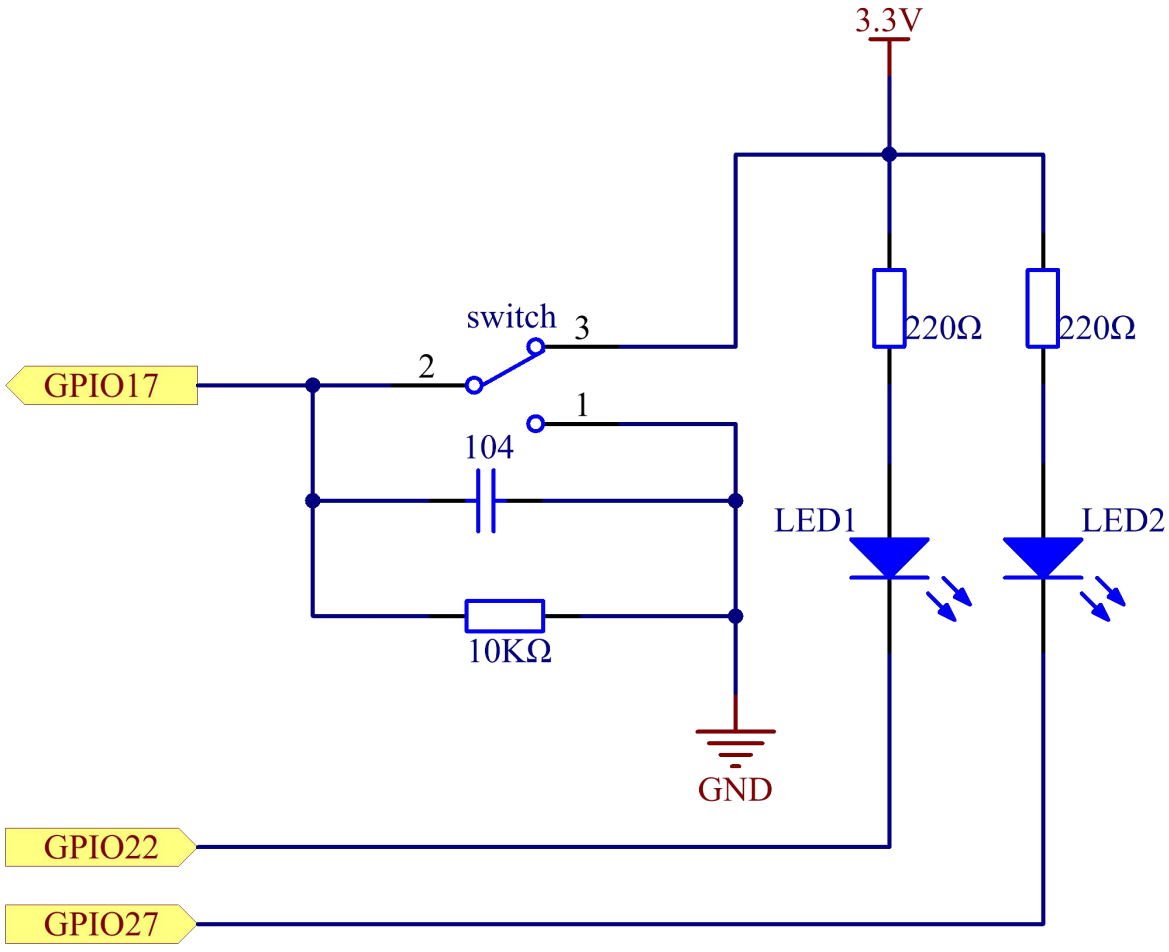
Picofarad (pF) 1pF = 1/1,000,000,000,000 = 0.000000000001 = 10^{-12}F

Note: Here we use **104 capacitor(10 x 10^4PF)**. 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 x 10 to 4 (in pF) equal to 100 nF.

Schematic Diagram

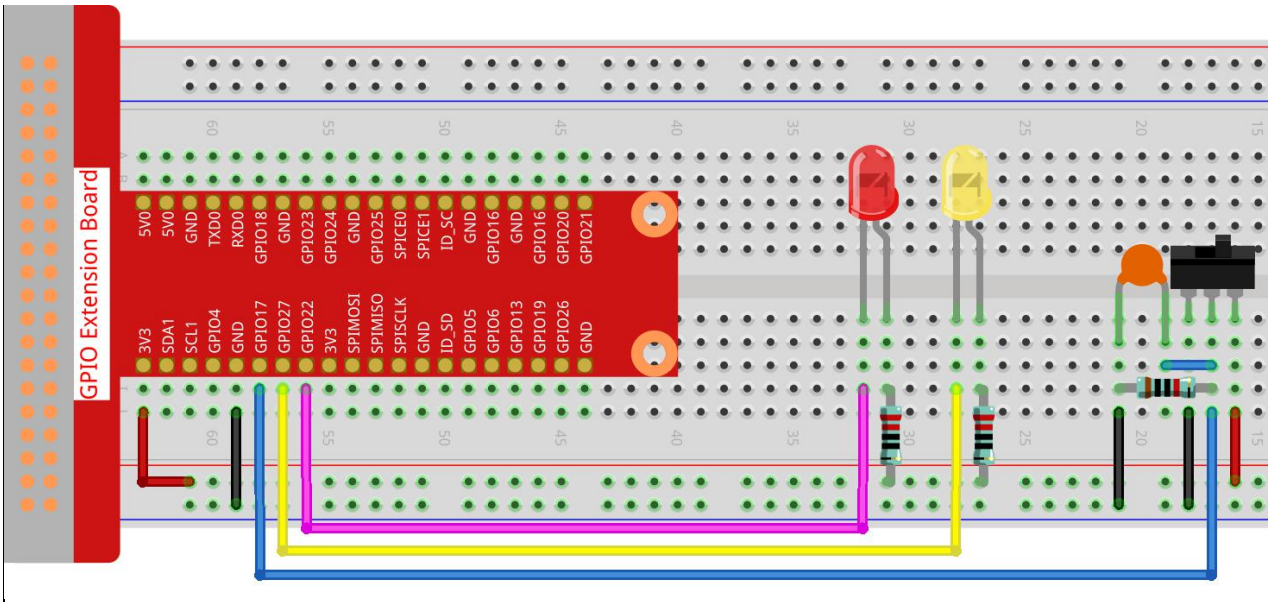
Connect the middle pin of the Slide Switch to GPIO17, and two LEDs to pin GPIO22 and GPIO27 respectively. Then when you pull the slide, you can see the two LEDs light up alternately.

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 python3 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)
```

```
            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

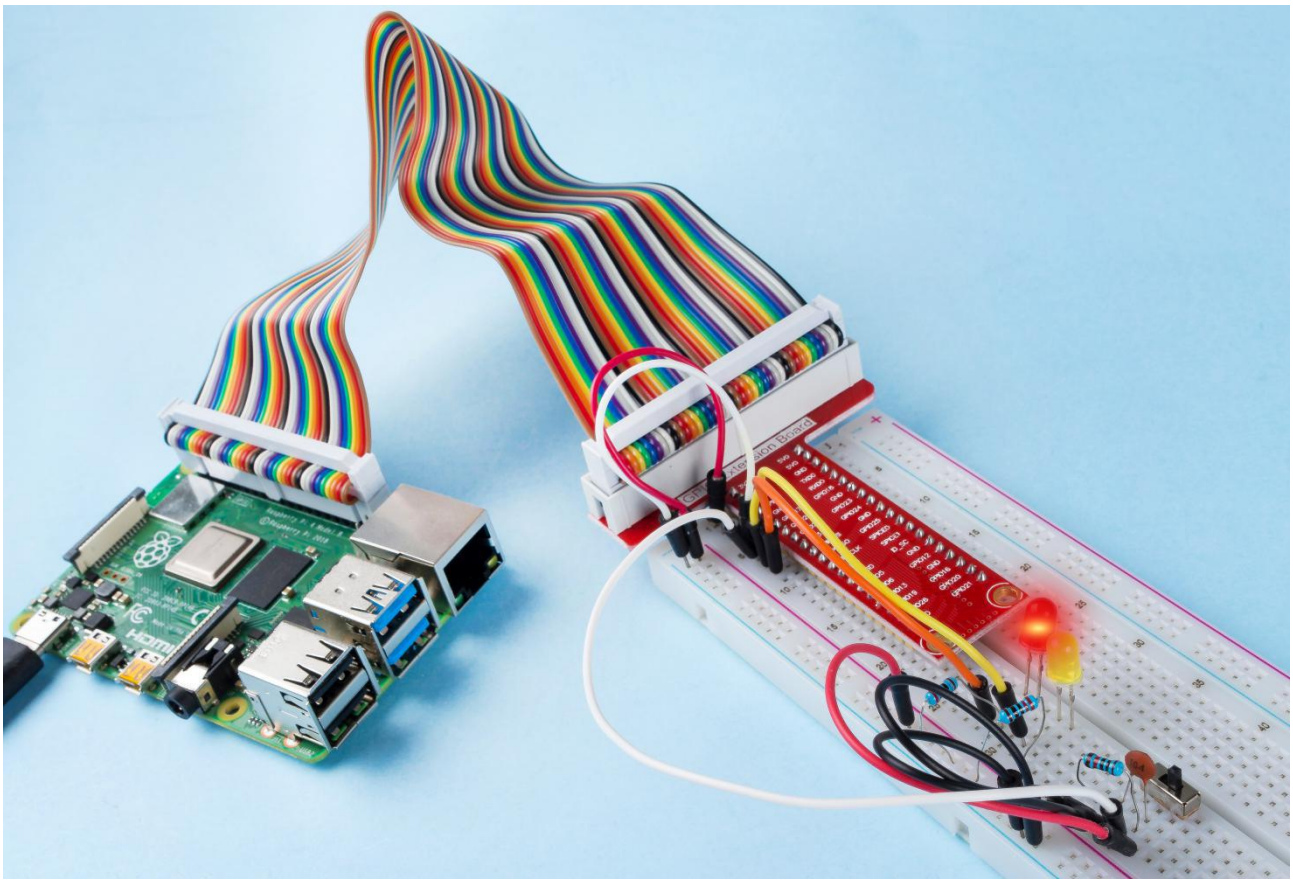
```
if GPIO.input(slidePin) == 1:
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)
```

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

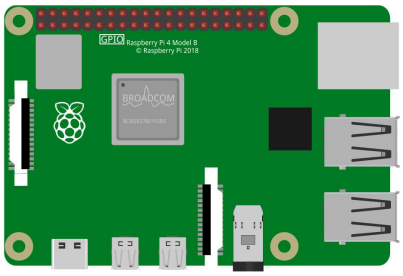
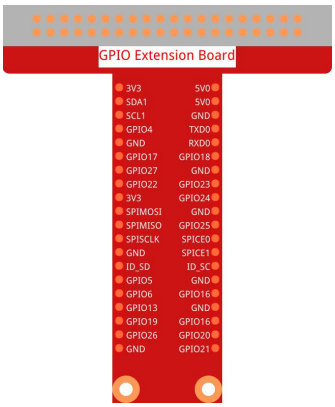
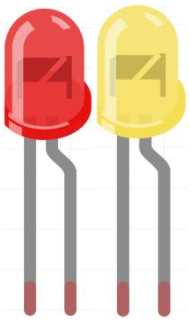
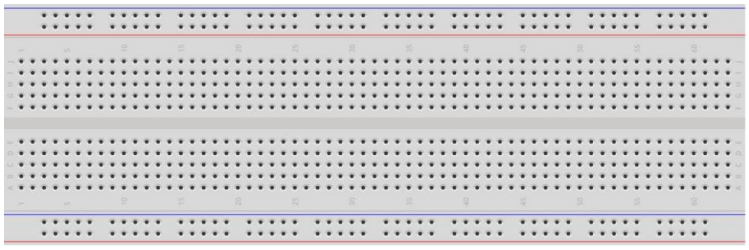


2.1.3 Tilt Switch

Introduction

This is a ball tilt-switch with a metal ball inside. It is used to detect inclinations of a small angle.

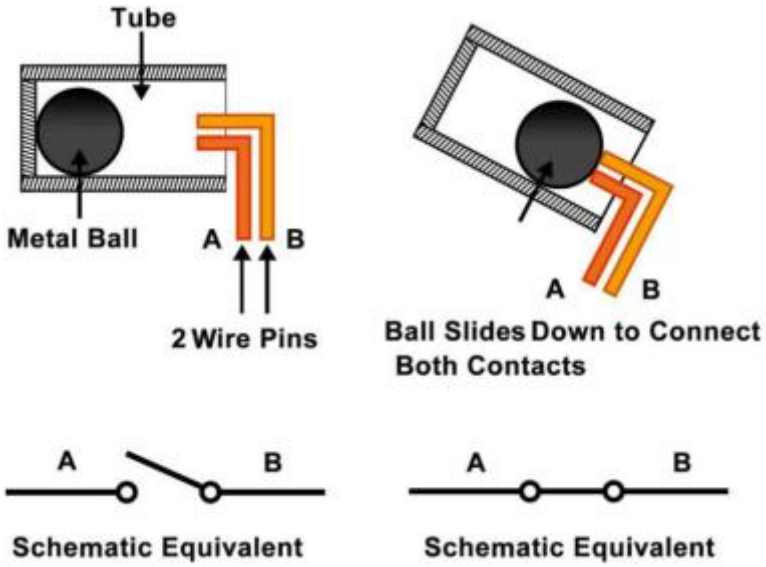
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Tilt Switch
 
</td><td>2 * LED
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td colspan="2">Several Jumper Wires
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td colspan="2">2 * Resistor(220Ω)
 
1 * Resistor 10KΩ
  </td></tr>
</table>

Principle

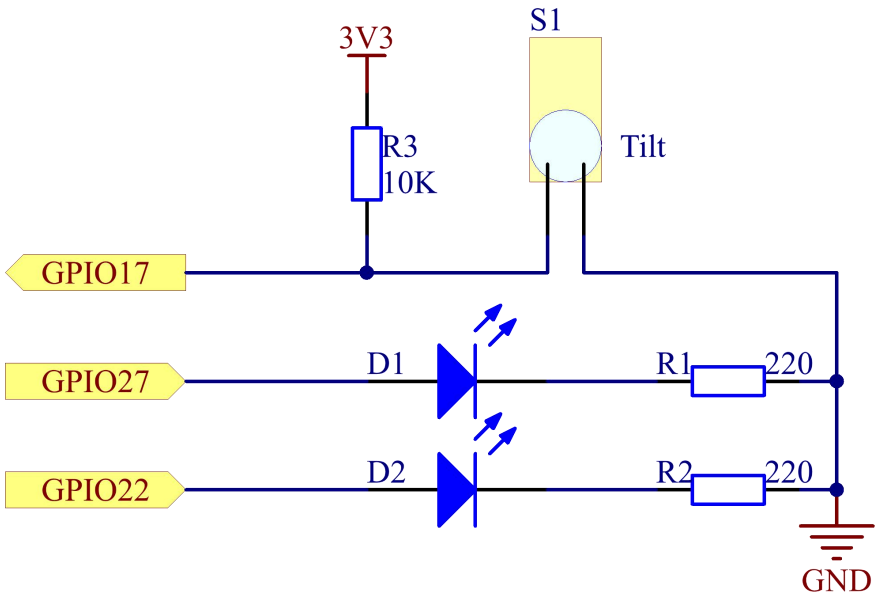
Tilt

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.



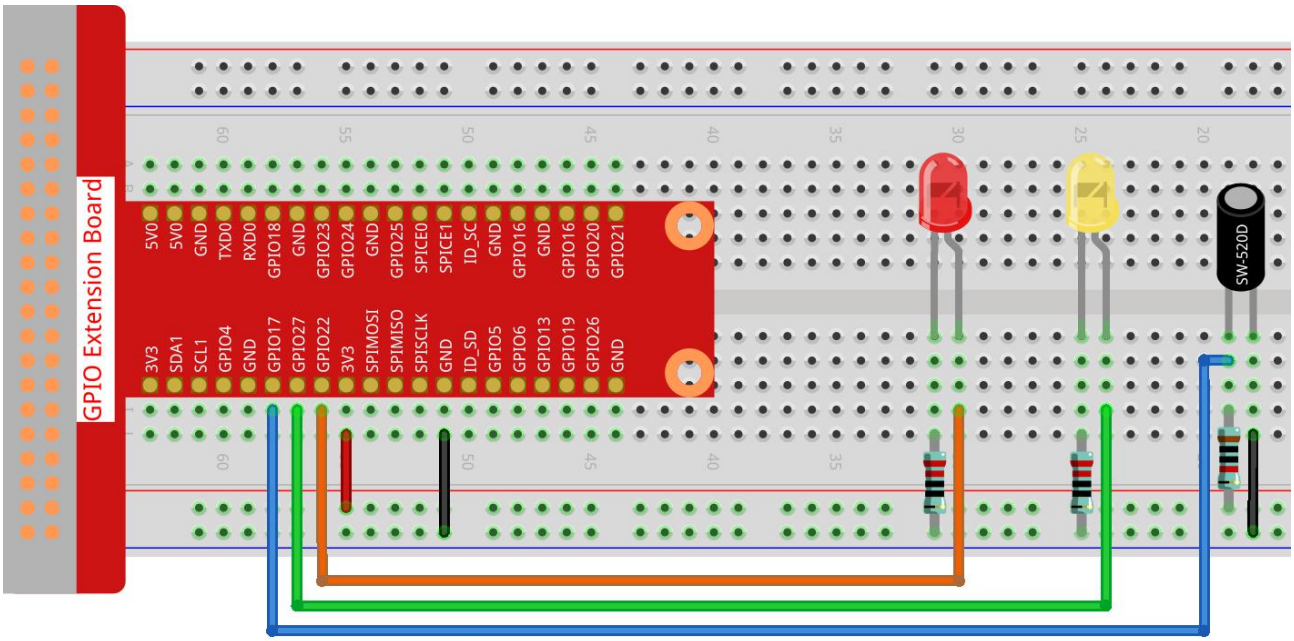
Schematic Diagram

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/2.1.3/
```

Step 3: Compile.

```
gcc 2.1.3_Tilt.c -lwiringPi
```

Step 4: Run.

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

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 lights on. Place it horizontally again, and the green LED will turns 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 python3 2.1.3_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.

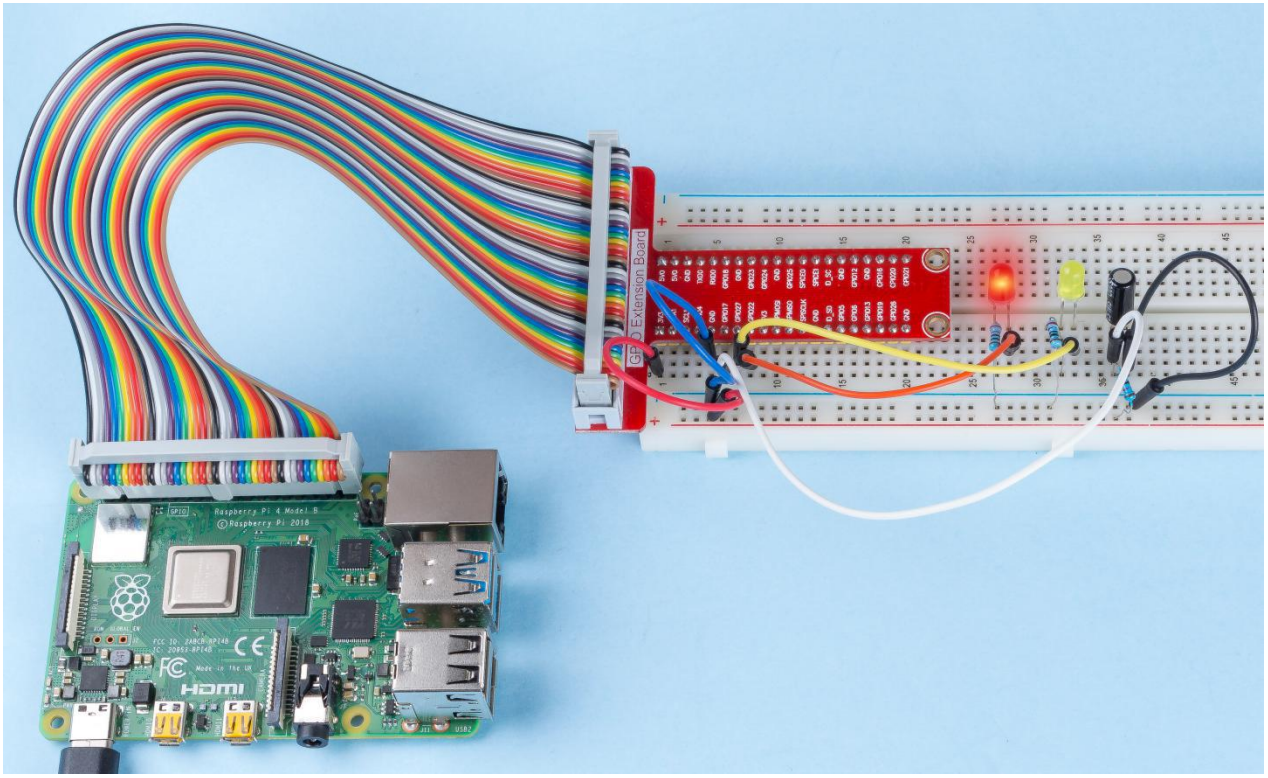
```
def Print(x):
    if x == 0:
        print ('     **************')
        print ('     *    Tilt!    *')
        print ('     **************')
```

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

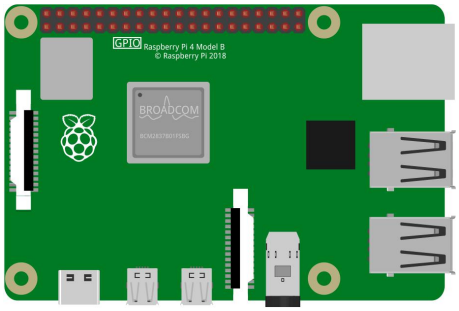
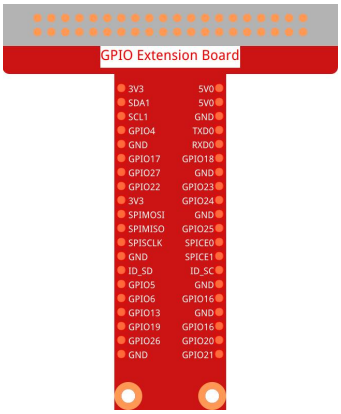
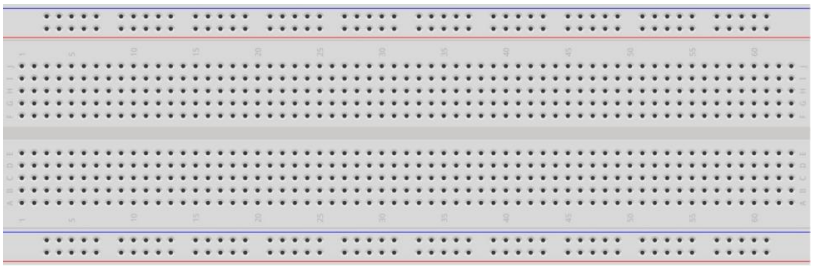


2.1.4 Potentiometer

Introduction

The ADC function can be used to convert analog signals to digital signals, and in this experiment, ADC0834 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.

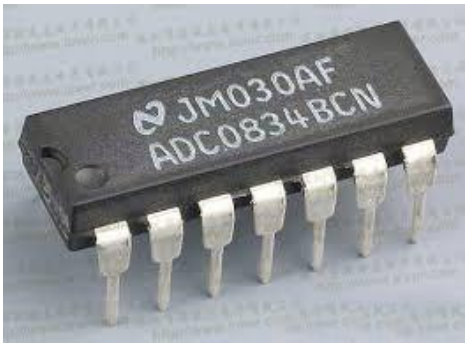
Components

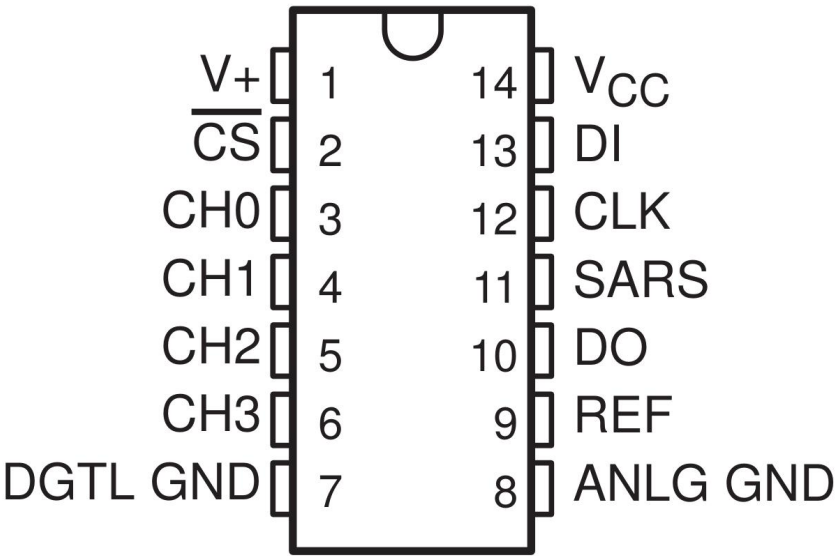
1 * Raspberry Pi 	1 * T-Extension Board 	1 * Potentiometer 
		1 * ADC0834 
1 * 40-pin Cable 		1 * Resistor(220Ω) 
		Several Jumper Wires 
1 * Breadboard 		1 * LED 

Principle

ADC0834

ADC0834 is an 8-bit successive approximation analog-to-digital converter that is equipped with an input-configurable multichannel multi-plexer and serial input/output. The serial input/output is configured to interface with standard shift registers or microprocessors.



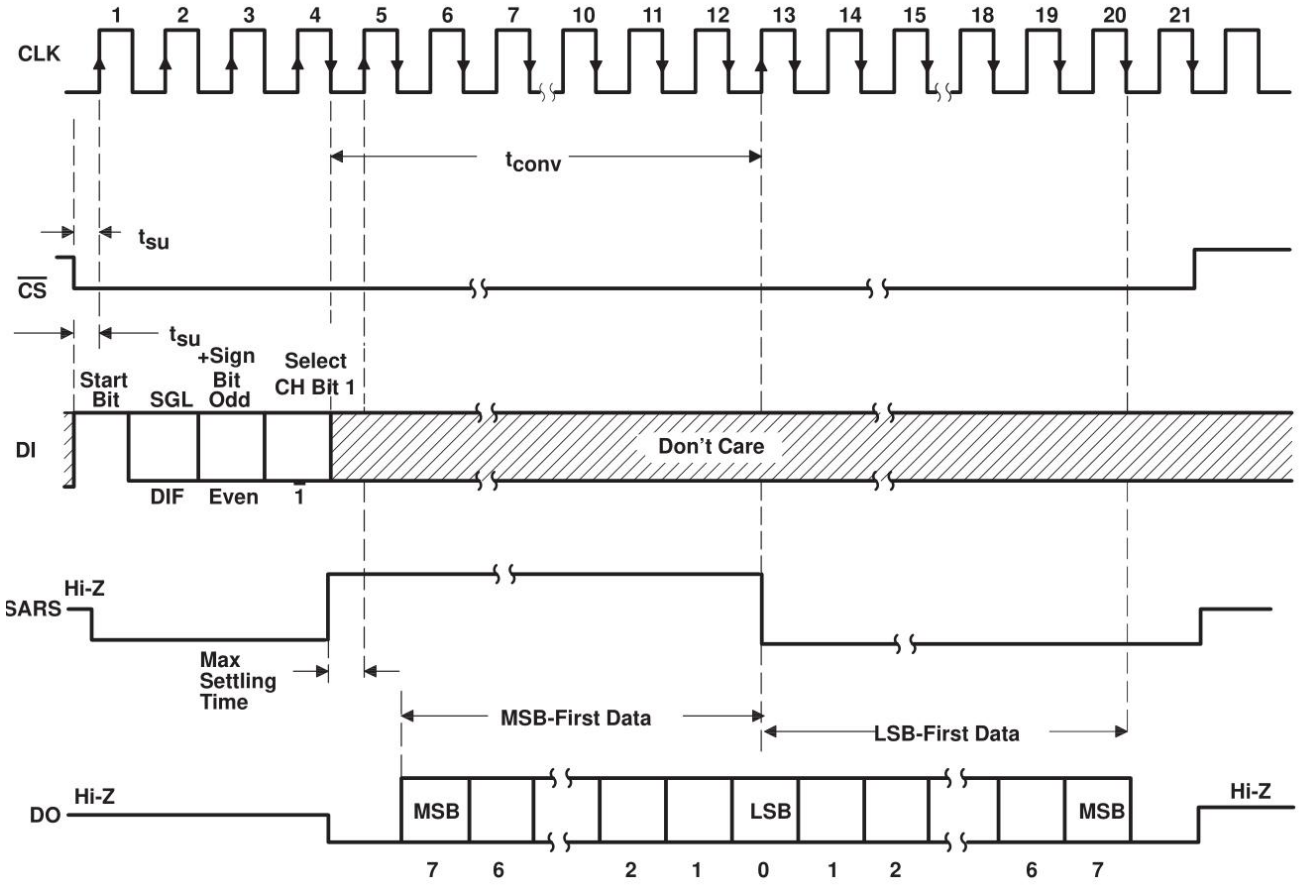


Sequence of Operation

A conversion is initiated by setting CS low, which enables all logic circuits. CS must be held low for the complete conversion process. A clock input is then received from the processor. On each low-to-high transition of the clock input, the data on DI is clocked into the multiplexer address shift register. The first logic high on the input is the start bit. A 3- to 4-bit assignment word follows the start bit. On each successive low-to-high transition of the clock input, the start bit and assignment word are shifted through the shift register. When the start bit is shifted into the start location of the multiplexer register, the input channel is selected and conversion starts. The SAR Statu output (SARS) goes high to indicate that a conversion is in progress, and DI to the multiplexer shift register is disabled the duration of the conversion.

An interval of one clock period is automatically inserted to allow the selected multiplexed channel to settle. The data output DO comes out of the high-impedance state and provides a leading low for this one clock period of multiplexer settling time. The SAR comparator compares successive outputs from the resistive ladder with the incoming analog signal. The comparator output indicates whether the analog input is greater than or less than the resistive ladder output. As the conversion proceeds, conversion data is simultaneously output from the DO output pin, with the most significant bit (MSB) first.

After eight clock periods, the conversion is complete and the SARS output goes low. Finally outputs the least-significant-bit-first data after the MSB-first data stream.



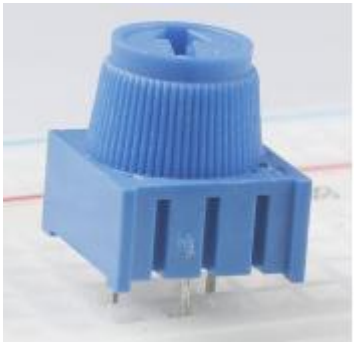
ADC0834 MUX ADDRESS CONTROL LOGIC TABLE

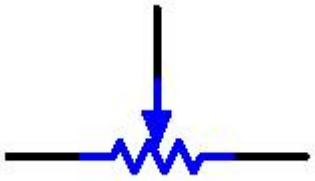
MUX ADDRESS			CHANNEL NUMBER			
SGL/$\overline{\text{DIF}}$	ODD/$\overline{\text{EVEN}}$	SELECT BIT 1	0	1	2	3
L	L	L	+	−		
L	L	H			+	−
L	H	L	−	+		
L	H	H			−	+
H	L	L	+			
H	L	H			+	
H	H	L		+		
H	H	H				+

H = high level, L = low level, – or + = polarity of selected input pin

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.





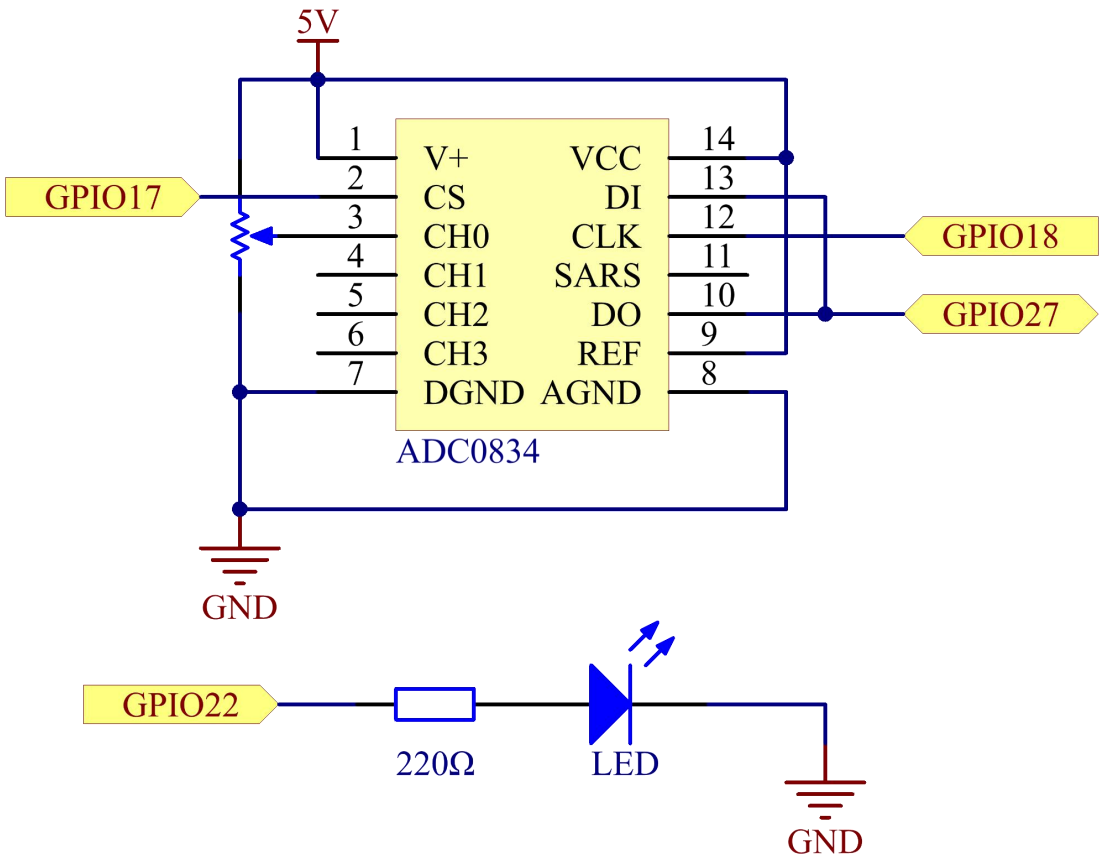
The functions of the potentiometer in the circuit are as follows:

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.

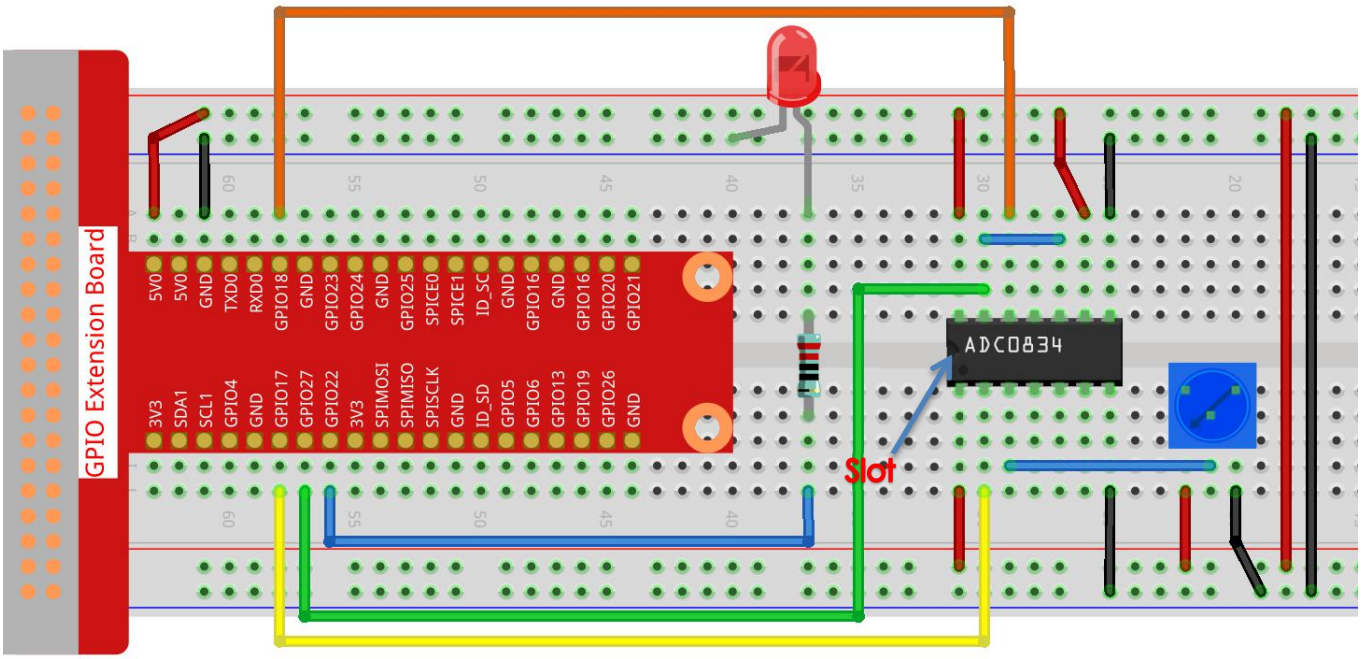
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22



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.

➢ For C Language Users

Step 2: Open the code file.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/2.1.4/
```

Step 3: Compile the code.

```
gcc 2.1.4_Potentiometer.c -lwiringPi
```

Step 4: Run.

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

After the code runs, rotate the knob on the potentiometer, the intensity of LED will change accordingly.

Code

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

typedef unsigned char uchar;
typedef unsigned int uint;

#define      ADC_CS     0
#define      ADC_CLK    1
#define      ADC_DIO    2
#define      LedPin     3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
  //Single End mode
```

```
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
```

```
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(LedPin,   0, 100);
    pinMode(ADC_CS,   OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

Code Explanation

```
#define     ADC_CS     0
#define     ADC_CLK    1
#define     ADC_DIO    2
#define     LedPin     3
```

Define CS, CLK, DIO of ADC0834, and connect them to GPIO0, GPIO1 and GPIO2 respectively. Then attach LED to GPIO3.

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
   //Single End mode
```

```
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}
```

There is a function of ADC0834 to get Analog to Digital Conversion. The specific workflow is as follows:

```
    digitalWrite(ADC_CS, 0);
```

Set CS to low level and start enabling AD conversion.

```
// Start bit
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
```

When the low-to-high transition of the clock input occurs at the first time, set DIO to 1 as Start bit. In the following three steps, there are 3 assignment words.

```
//Single End mode
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
```

As soon as the low-to-high transition of the clock input occurs for the second time, set DIO to 1 and choose SGL mode.

```
// ODD
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
```

Once occurs for the third time, the value of DIO is controlled by the variable **odd**.

```
//Select
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
digitalWrite(ADC_CLK,1);
```

The pulse of CLK converted from low level to high level for the forth time, the value of DIO is controlled by the variable **sel**.

Under the condition that channel=0, sel=0, odd=0, the operational formulas concerning **sel** and **odd** are as follows:

```
int sel = channel > 1 & 1;
int odd = channel & 1;
```

When the condition that channel=1, sel=0, odd=1 is met, please refer to the following address control logic table. Here CH1 is chosen, and the start bit is shifted into the start location of the multiplexer register and conversion starts.

MUX ADDRESS			CHANNEL NUMBER			
SGL/$\overline{DIF}$	ODD/$\overline{EVEN}$	SELECT BIT 1	0	1	2	3
L	L	L	+	−		
L	L	H			+	−
L	H	L	−	+		
L	H	H			−	+
H	L	L	+			
H	L	H			+	
H	H	L		+		
H	H	H				+

```
digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
digitalWrite(ADC_CLK,0);
digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
```

Here, set DIO to 1 twice, please ignore it.

```
for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }
```

In the first for() statement, as soon as the fifth pulse of CLK is converted from high level to low level, set DIO to input mode. Then the conversion starts and the converted value is stored in the variable dat1. After eight clock periods, the conversion is complete.

```
for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);
    }
```

In the second for() statement, output the converted values via DO after other eight clock periods and store them in the variable dat2.

```
digitalWrite(ADC_CS,1);
pinMode(ADC_DIO, OUTPUT);
```

```
    return(dat1==dat2) ? dat1 : 0;
```

return(dat1==dat2) ? dat1 : 0 is used to compare the value gotten during the conversion and the output value. If they are equal to each other, output the converting value dat1; otherwise, output 0. Here, the workflow of ADC0834 is complete.

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

The function is to use software to create a PWM pin, LedPin, then the initial pulse width is set to 0, and the period of PWM is 100 x 100us.

```
while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
```

In the main program, read the value of channel 0 that has been connected with a potentiometer. And store the value in the variable analogVal then write it in LedPin. Now you can see the brightness of LED changing with the value of the potentiometer.

➢ For Python Users

Step 2: Open the code file

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

Step 3: Run.

```
sudo python3 2.1.4_Potentiometer.py
```

After the code runs, rotate the knob on the potentiometer, the intensity of LED will change accordingly.

Code

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

import RPi.GPIO as GPIO
import ADC0834
import time

LedPin = 22
```

```
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)

    # Set all begin with value 0
    led_val.start(0)

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

def destroy():
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()

def loop():
    while True:
        res = ADC0834.getResult()
        print ('res = %d' % res)
        R_val = MAP(res, 0, 255, 0, 100)
        led_val.ChangeDutyCycle(R_val)
        time.sleep(0.2)

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

Code Explanation

```
import ADC0834
```

import ADC0834 library. You can check the content of the library by calling the command nano ADC0834.py.

```
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)

    # Set all begin with value 0
        led_val.start(0)
```

In setup(), define the naming method as BCM, set LedPin as PWM channel and render it a frequency of 2Khz.

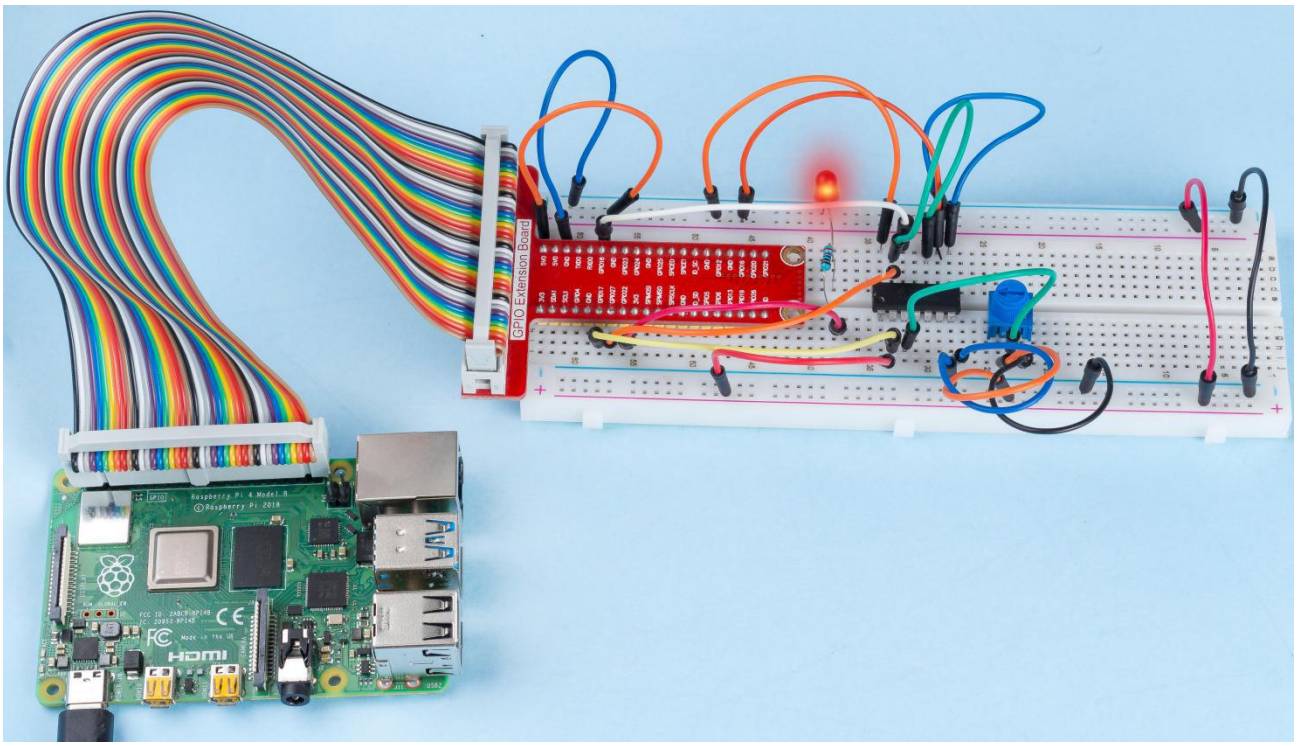
ADC0834.setup(): Initialize ADC0834, and connect the defined CS, CLK, DIO of ADC0834 to GPIO17, GPIO18 and GPIO27 respectively.

```
def loop():
    while True:
        res = ADC0834.getResult()
        print ('res = %d' % res)
        R_val = MAP(res, 0, 255, 0, 100)
        led_val.ChangeDutyCycle(R_val)
        time.sleep(0.2)
```

The function getResult() is used to read the analog values of the four channels of ADC0834. By default, the function reads the value of CH0, and if you want to read other channels, please input the channel number in **( )**, ex. getResult(1).

The function loop() first reads the value of CH0, then assign the value to the variable res. After that, call the function MAP to map the read value of potentiometer to 0~100. This step is used to control the duty cycle of LedPin. Now, you may see that the brightness of LED is changing with the value of potentiometer.

Phenomenon Picture

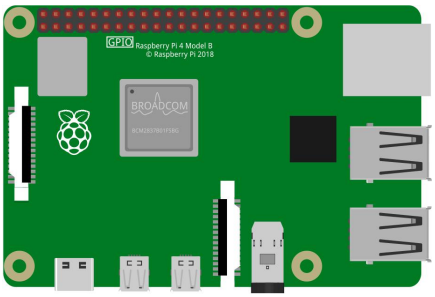
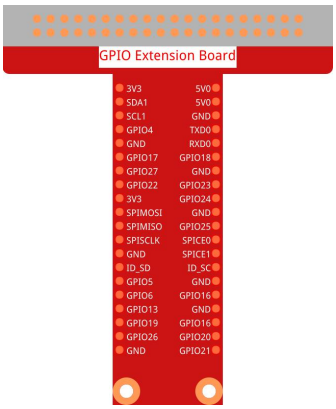
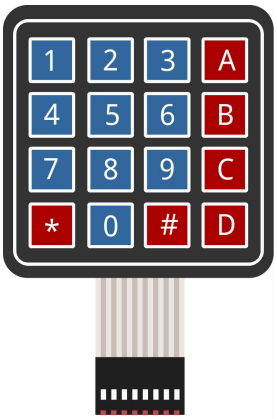
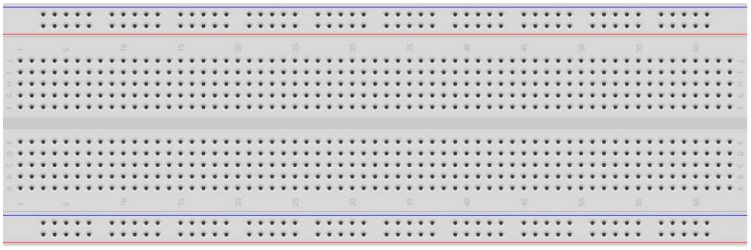


2.1.5 Keypad

Introduction

A keypad is a rectangular array of buttons. In this project, We will use it input characters.

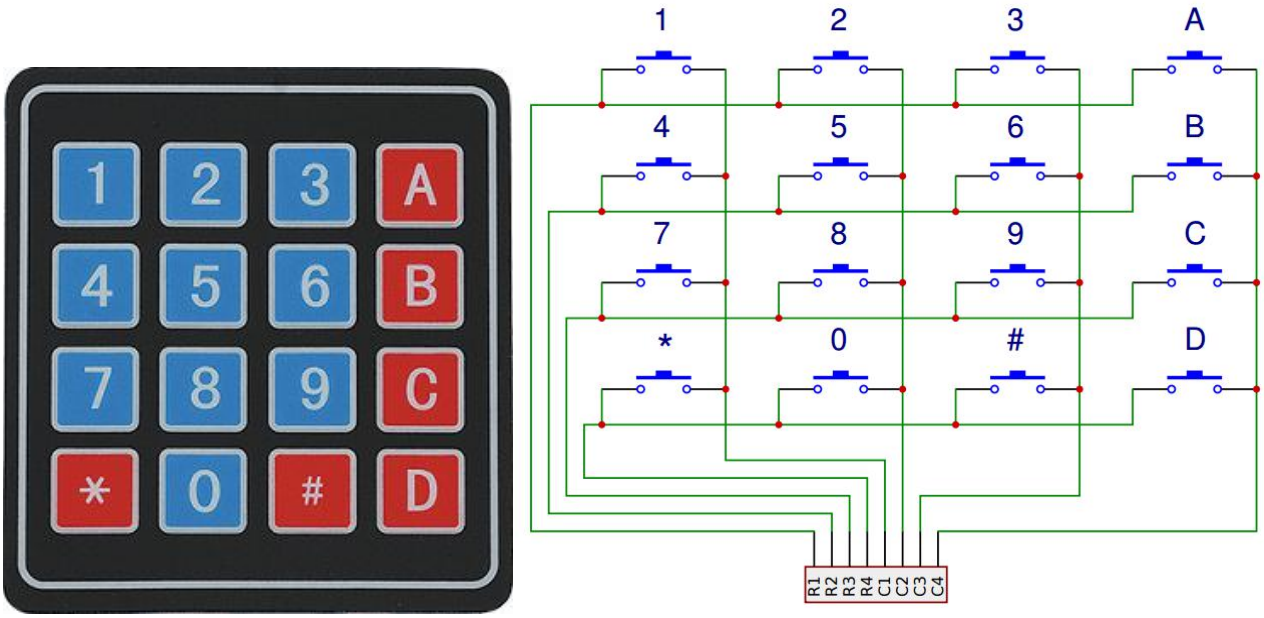
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Keypad
  </td></tr>
<tr><td>1 * Breadboard
  </td><td colspan="2">8 * Resistor 10KΩ
 
Several Jumper Wires
</td></tr>
<tr><td colspan="3">1 * 40-pin Cable
  </td></tr>
</table>

Principle

Keypad

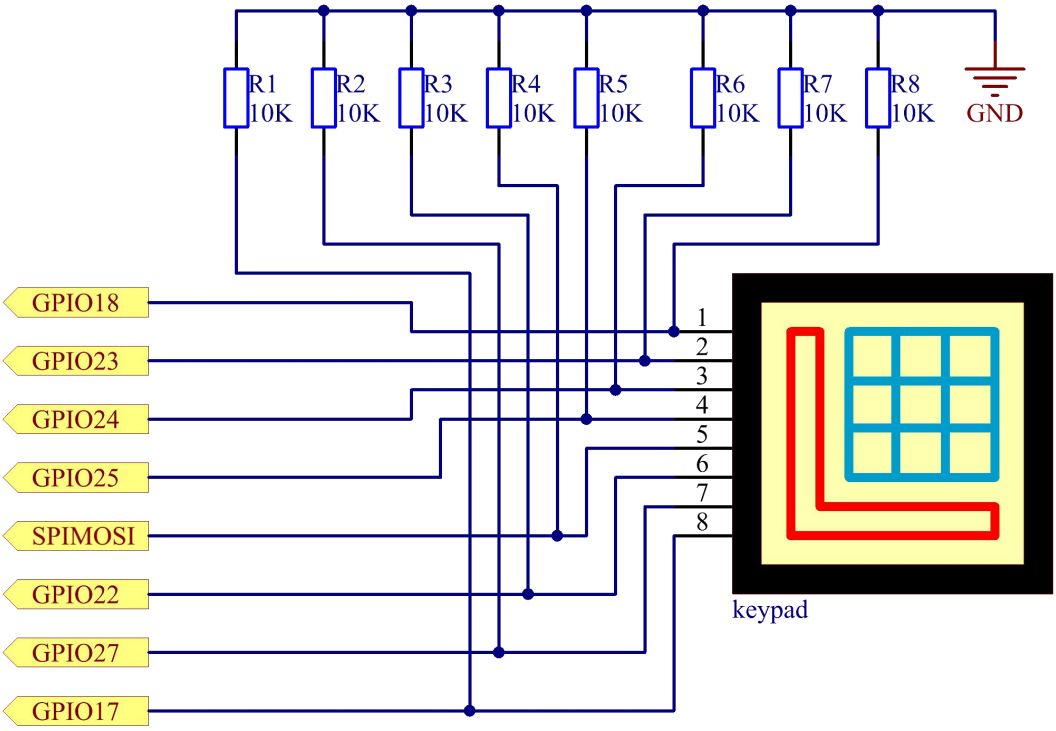
A keypad is a rectangular array of 12 or 16 OFF-(ON) buttons. Their contacts are accessed via a header suitable for connection with a ribbon cable or insertion into a printed circuit board. In some keypads, each button connects with a separate contact in the header, while all the buttons share a common 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.

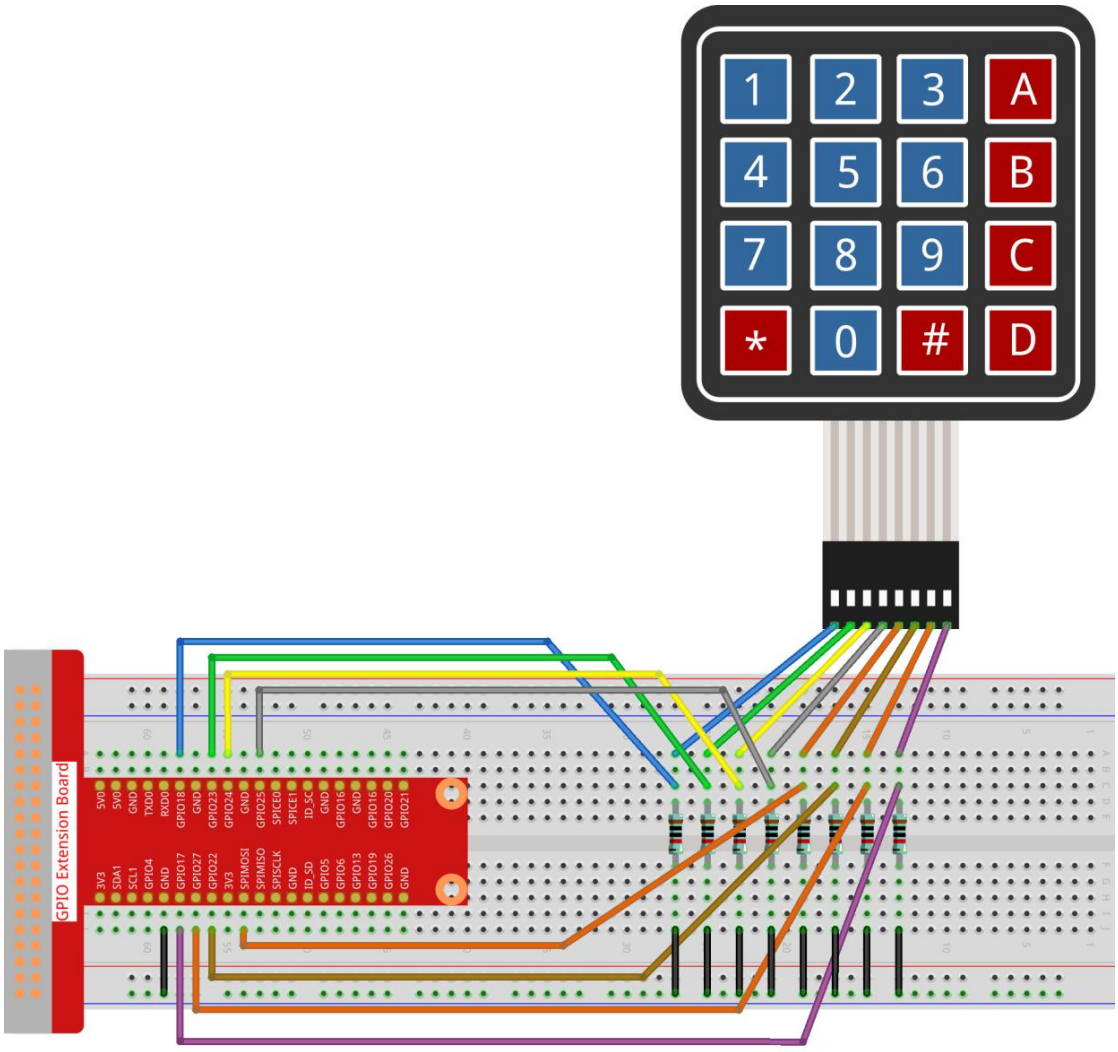
Schematic Diagram

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 -lwiringPi
```

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

#define ROWS  4
#define COLS  4
#define BUTTON_NUM (ROWS * COLS)

unsigned char KEYS[BUTTON_NUM] {
  '1','2','3','A',
  '4','5','6','B',
  '7','8','9','C',
  '*','0','#','D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};

void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);

void init(void) {
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
```

```
        pinMode(colPins[i], INPUT);
    }
}

int main(void){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            keyPrint(pressed_keys);
            keyCopy(last_key_pressed, pressed_keys);
        }
        delay(100);
    }
    return 0;
}

void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
    for(int i=0 ; i<ROWS ; i++ ){
        digitalWrite(rowPins[i], HIGH);
        for(int j =0 ; j < COLS ; j++){
            index = i * ROWS + j;
            if(digitalRead(colPins[j]) == 1){
                result[count]=KEYS[index];
                count += 1;
            }
        }
        delay(1);
        digitalWrite(rowPins[i], LOW);
```

```
    }
}

bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}

void keyPrint(unsigned char* a){
    if (a[0] != 0){
        printf("%c",a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c",a[i]);
        }
    }
    printf("\n");
}

void keyClear(unsigned char* a){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = 0;
    }
}

int keyIndexOf(const char value){
    for (int i=0; i<BUTTON_NUM; i++){
        if ((const char)KEYS[i] == value){
            return i;
```

```
            }
        }
        return -1;
}
```

Code Explanation

```
unsigned char KEYS[BUTTON_NUM] {
  '1','2','3','A',
  '4','5','6','B',
  '7','8','9','C',
  '*','0','#','D'};

unsigned char rowPins[ROWS] = {1, 4, 5, 6};
unsigned char colPins[COLS] = {12, 3, 2, 0};
```

Declare each key of the matrix keyboard to the array keys[] and define the pins on each row and column.

```
 while(1){
            keyRead(pressed_keys);
            bool comp = keyCompare(pressed_keys, last_key_pressed);
            if (!comp){
                keyPrint(pressed_keys);
                keyCopy(last_key_pressed, pressed_keys);
            }
            delay(100);
        }
```

This is the part of the main function that reads and prints the button value.

The function keyRead() will read the state of every button.

KeyCompare() and keyCopy() are used to judge whether the state of a button has changed (that is, a button has been pressed or released).

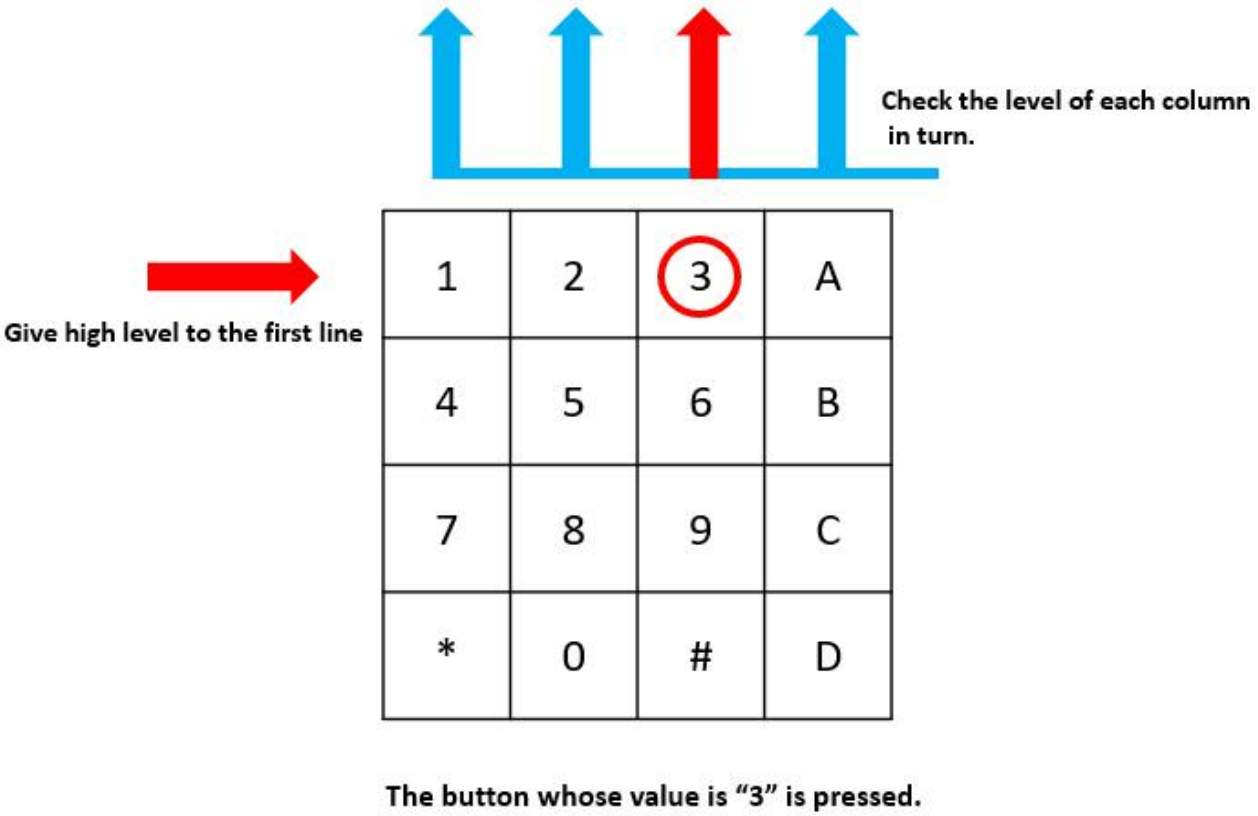
keyPrint() will print the button value of the button whose current level is high level (the button is pressed).

```
void keyRead(unsigned char* result){
    int index;
    int count = 0;
    keyClear(result);
```

```
    for(int i=0 ; i<ROWS ; i++ ){
        digitalWrite(rowPins[i], HIGH);
        for(int j =0 ; j < COLS ; j++){
            index = i * ROWS + j;
            if(digitalRead(colPins[j]) == 1){
                result[count]=KEYS[index];
                count += 1;
            }
        }
        delay(1);
        digitalWrite(rowPins[i], LOW);
    }
}
```

This function assigns a high level to each row in turn, and when the key in the column is pressed, the column in which the key is located gets a high level. After the two-layer loop judgment, the key state compilation will generate an array (reasult[]).

When pressing button 3:



The button whose value is "3" is pressed.

RowPin [0] writes in the high level, and colPin[2] gets the high level. ColPin [0], colPin[1], colPin[3] get the low level.

This gives us 0,0,1,0. When rowPin[1], rowPin[2] and rowPin[3] are written in high level, colPin[0]~colPin[4] will get low level.

After the loop judgment is completed, an array will be generated:

```
result[BUTTON_NUM] {
  0, 0, 1, 0,
  0, 0, 0, 0,
  0, 0, 0, 0,
  0, 0, 0, 0};
```

```
bool keyCompare(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        if (a[i] != b[i]){
            return false;
        }
    }
    return true;
}

void keyCopy(unsigned char* a, unsigned char* b){
    for (int i=0; i<BUTTON_NUM; i++){
        a[i] = b[i];
    }
}
```

These two functions are used to judge whether the key state has changed, for example when you release your hand when pressing '3' or pressing '2', keyCompare() returns false.

KeyCopy() is used to re-write the current button value for the a array (last_key_pressed[BUTTON_NUM]) after each comparison. So we can compare them next time.

```
void keyPrint(unsigned char* a){
  //printf("{");
    if (a[0] != 0){
        printf("%c",a[0]);
    }
    for (int i=1; i<BUTTON_NUM; i++){
        if (a[i] != 0){
            printf(", %c",a[i]);
        }
    }
```

```
    printf("\n");
}
```

This function is used to print the value of the button currently pressed. If the button '1' is pressed, the '1' will be printed. If the button '1' is pressed and the button '3' is pressed, the '1, 3' will be printed.

➢ For Python Language Users

Step 2: Open the code file.

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

Step 3: Run.

```
sudo python3 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 Keypad():

    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
```

```
                GPIO.output(row, GPIO.LOW)
        return pressed_keys

def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
    colsPins = [10,22,27,17]
    keys = ["1","2","3","A",
            "4","5","6","B",
            "7","8","9","C",
            "*","0","#","D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []

def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Release resource
    GPIO.cleanup()

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

Code Explanation

```
def setup():
    global keypad, last_key_pressed
    rowsPins = [18,23,24,25]
    colsPins = [10,22,27,17]
    keys = ["1","2","3","A",
            "4","5","6","B",
            "7","8","9","C",
            "*","0","#","D"]
    keypad = Keypad(rowsPins, colsPins, keys)
    last_key_pressed = []
```

Declare each key of the matrix keyboard to the array keys[] and define the pins on each row and column.

```
def loop():
    global keypad, last_key_pressed
    pressed_keys = keypad.read()
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)
```

This is the part of the main function that reads and prints the button value.

The function keyRead() will read the state of every button.

The statement if len(pressed_keys) != 0 and last_key_pressed != pressed_keys is used to judge

whether a key is pressed and the state of the pressed button. (If you press '3' when you press '1', the judgement is tenable.)

Prints the value of the currently pressed key when the condition is tenable.

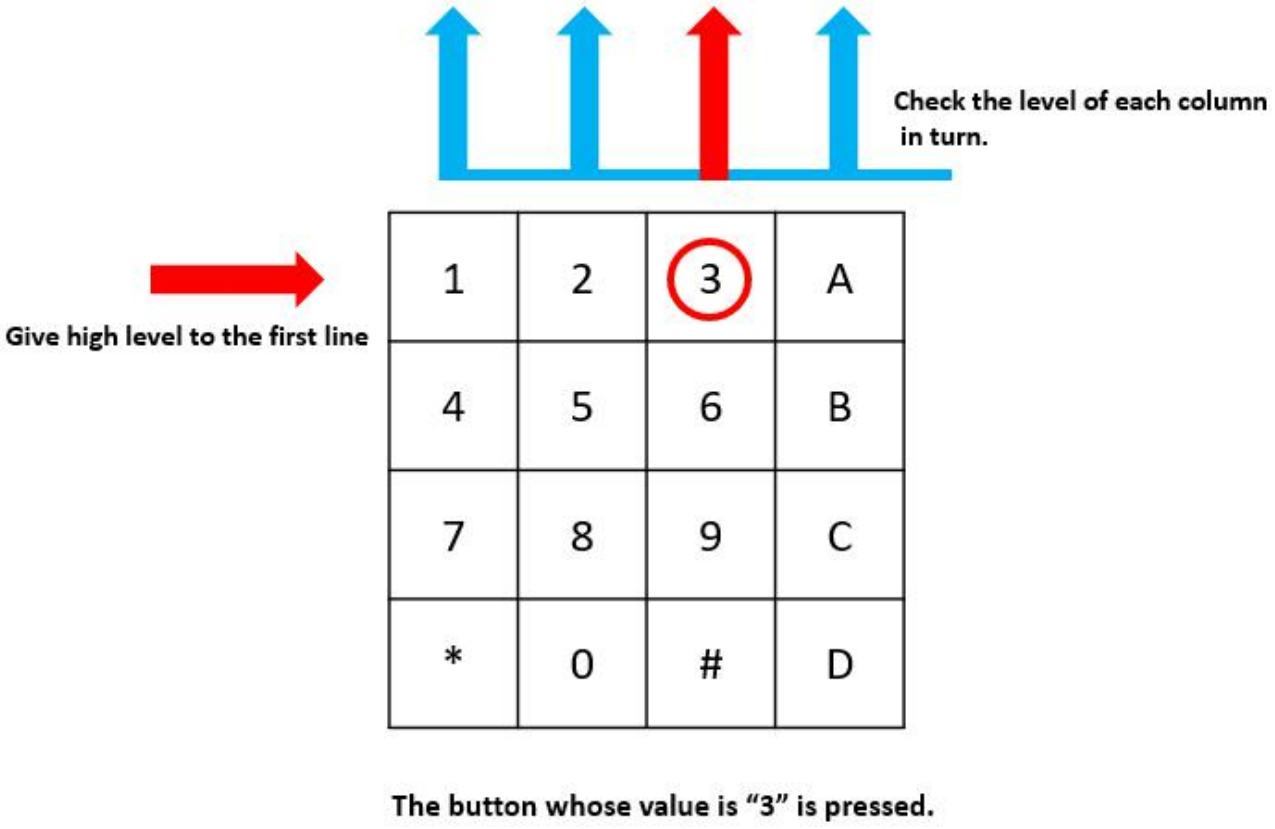
The statement last_key_pressed = pressed_keys assigns the state of each judgment to an array last_key_pressed to facilitate the next round of conditional judgment.

```
    def read(self):
        pressed_keys = []
        for i, row in enumerate(self.rowsPins):
            GPIO.output(row, GPIO.HIGH)
            for j, col in enumerate(self.colsPins):
```

```
                index = i * len(self.colsPins) + j
                if (GPIO.input(col) == 1):
                    pressed_keys.append(self.keys[index])
            GPIO.output(row, GPIO.LOW)
        return pressed_keys
```

This function assigns a high level to each row in turn, and when the button in the column is pressed, the column in which the key is located gets a high level. After the two-layer loop is judged, the value of the button whose state is 1 is stored in the array pressed_keys.

If you press the key '3':



The button whose value is "3" is pressed.

rowPins[0] is written in high level, and colPins[2] gets high level.

colPins[0]、colPins[1]、colPins[3] get low level.

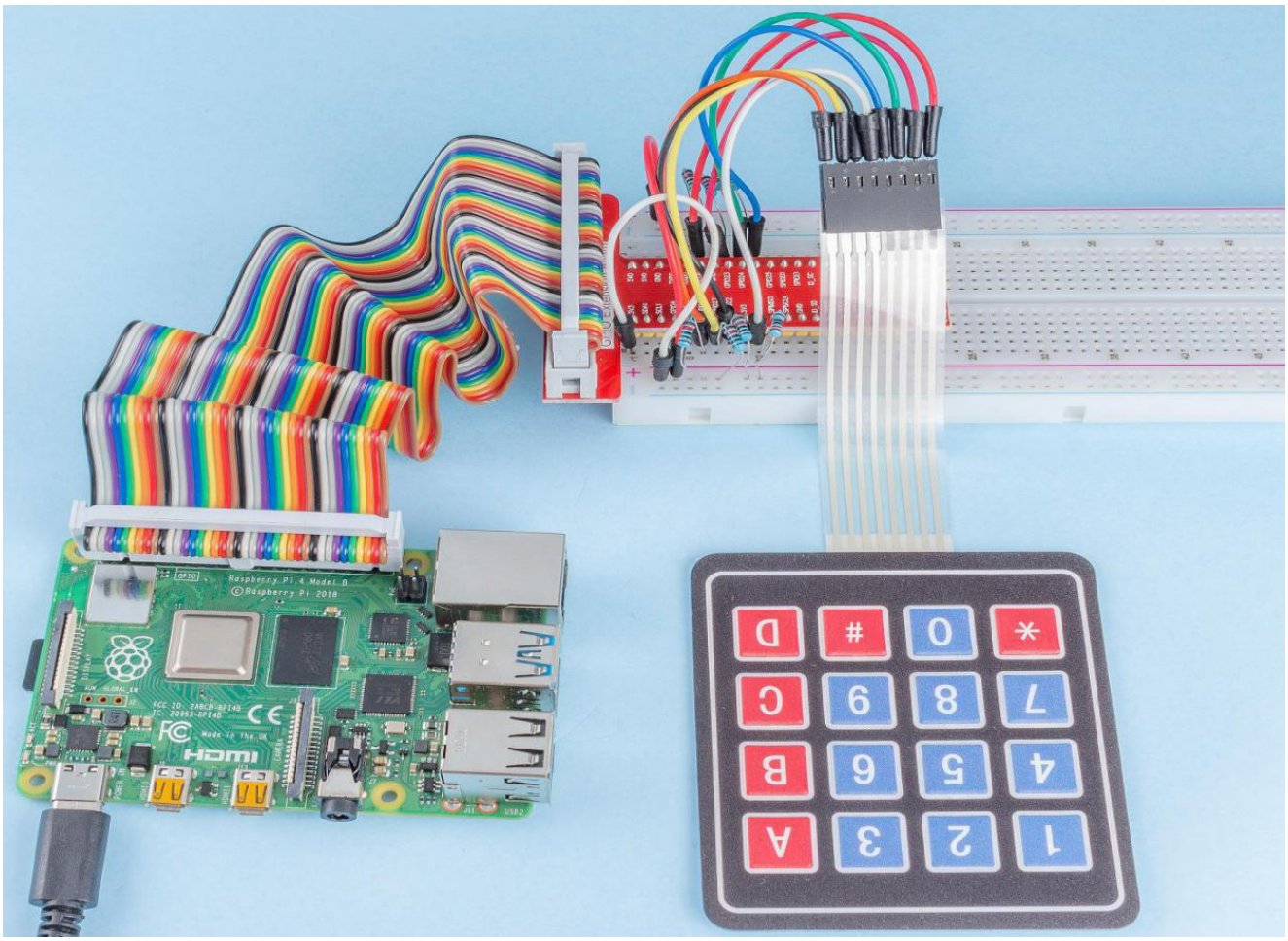
There are four states:0, 0, 1, 0; and we write '3' into pressed_keys.

When rowPins[1] , rowPins[2] , rowPins[3] are written into high level, colPins[0] ~ colPins[4] get low level.

The loop stopped, there returns pressed_keys = '3'.

If you press the buttons '1' and '3', there will return pressed_keys = ['1','3'].

Phenomenon Picture



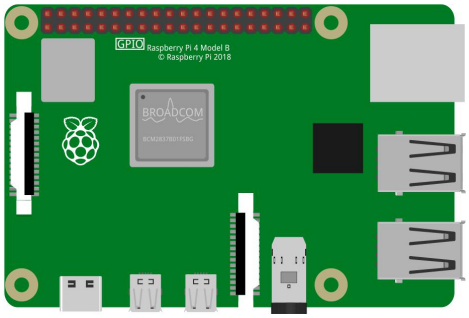
2.1.6 Joystick

Introduction

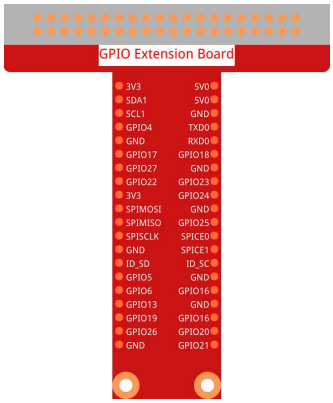
In this project, We're going to learn how joystick works. We manipulate the Joystick and display the results on the screen.

Components

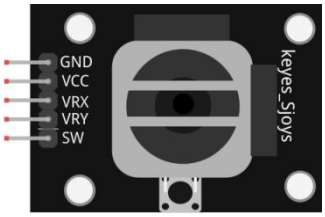
1 * Raspberry Pi



1 * T-Extension Board



1 * Joystick



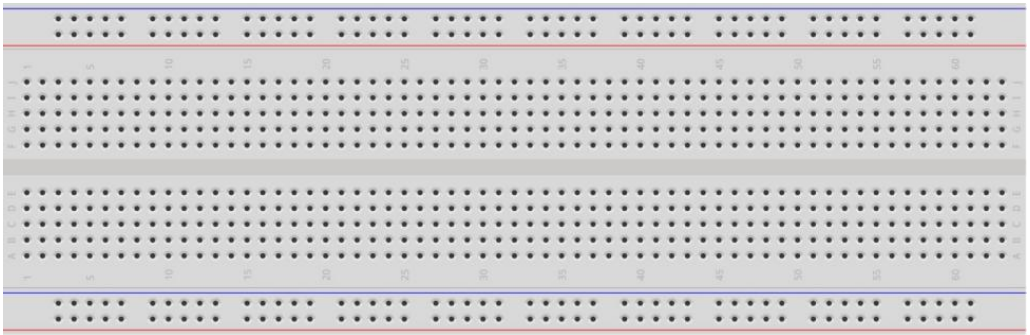
1 * Resistor 10KΩ

1 * 40-pin Cable

1 * ADC0834

Several Jumper Wires

1 * Breadboard



Principle

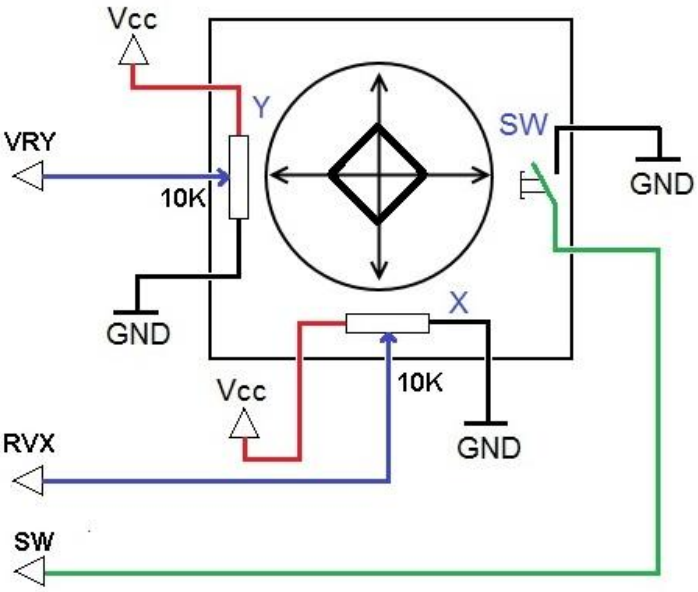
Joystick

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.

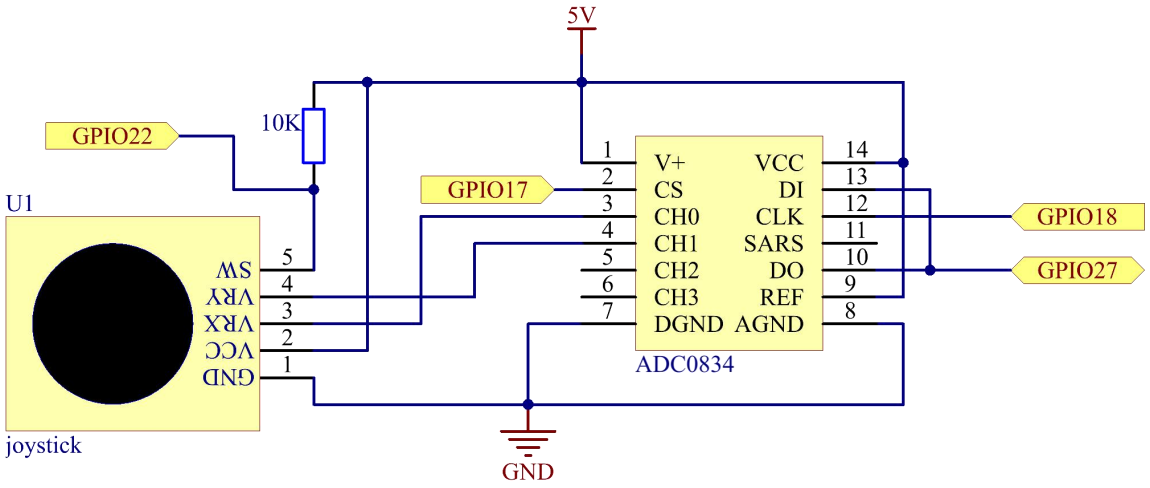
The joystick also has a digital input that is actuated when the joystick is pressed down.



Schematic Diagram

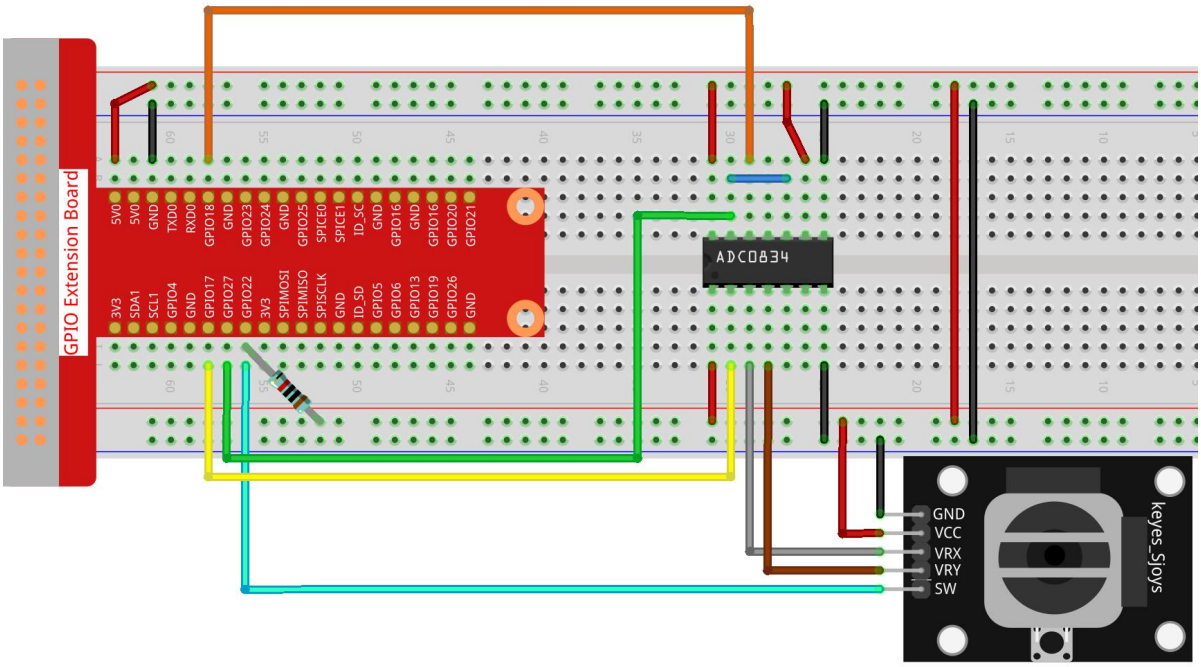
When the data of joystick is read, there are some differents between axis: data of X and Y axis is analog, which need to use ADC0834 to convert the analog value to digital value. Data of Z axis is digital, so you can directly use the GPIO to read, or you can also use ADC to read.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	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/2.1.6/
```

Step 3: Compile the code.

```
gcc 2.1.6_Joystick.c -lwiringPi
```

Step 4: 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 <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>

typedef unsigned char uchar;
typedef unsigned int uint;

#define      ADC_CS      0
#define      ADC_CLK     1
#define      ADC_DIO     2
#define      BtnPin      3

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
   //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
```

```
    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
        pinMode(ADC_DIO, INPUT);
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }
    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);    delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);    delayMicroseconds(2);
    }
    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}
int main(void)
{
    uchar x_val;
    uchar y_val;
    uchar btn_val;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(BtnPin,   INPUT);
    pullUpDnControl(BtnPin, PUD_UP);
    pinMode(ADC_CS,   OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        x_val = get_ADC_Result(0);
        y_val = get_ADC_Result(1);
        btn_val = digitalRead(BtnPin);
        printf("x = %d, y = %d, btn = %d\n", x_val, y_val, btn_val);
        delay(100);
    }
    return 0;
}
```

Code Explanation

```
uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;
    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
   //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    ......
```

The working process of the function is detailed in 2.1.4 Potentiometer.

```
while(1){
        x_val = get_ADC_Result(0);
        y_val = get_ADC_Result(1);
        btn_val = digitalRead(BtnPin);
        printf("x = %d, y = %d, btn = %d\n", x_val, y_val, btn_val);
        delay(100);
    }
```

VRX and VRY of Joystick are connected to CH0, CH1 of ADC0834 respectively. So the function getResult() is called to read the values of CH0 and CH1. Then the read values should be stored in the variables x_val and y_val. In addition, read the value of SW of joystick and store it into the variable Btn_val. Finally, the values of x_val, y_val and Btn_val shall be printed with print() function.

➢ 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.

```
sudo python3 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

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

import RPi.GPIO as GPIO
import ADC0834
import time

BtnPin = 22

def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    ADC0834.setup()

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

def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d   Y: %d   Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)

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

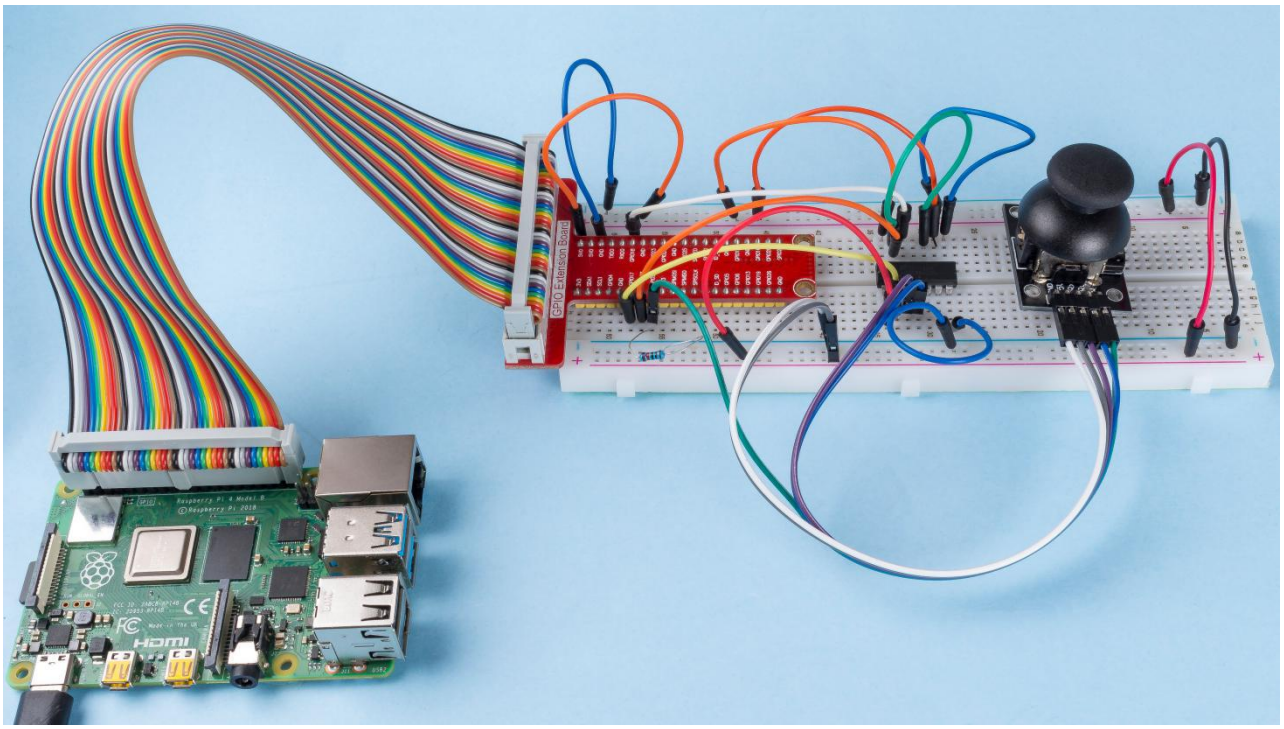
```
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be executed.
        destroy()
```

Code Explanation

```
def loop():
    while True:
        x_val = ADC0834.getResult(0)
        y_val = ADC0834.getResult(1)
        Btn_val = GPIO.input(BtnPin)
        print ('X: %d   Y: %d   Btn: %d' % (x_val, y_val, Btn_val))
        time.sleep(0.2)
```

VRX and VRY of Joystick are connected to CH0, CH1 of ADC0834 respectively. So the function getResult() is called to read the values of CH0 and CH1. Then the read values should be stored in the variables x_val and y_val. In addition, read the value of SW of joystick and store it into the variable Btn_val. Finally, the values of x_val, y_val and Btn_val shall be printed with print() function.

Phenomenon Picture



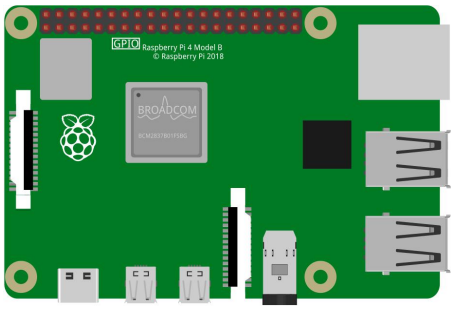
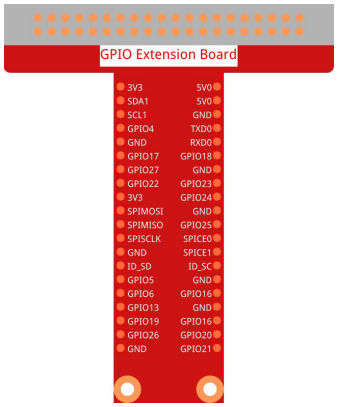
2.2 Sensors

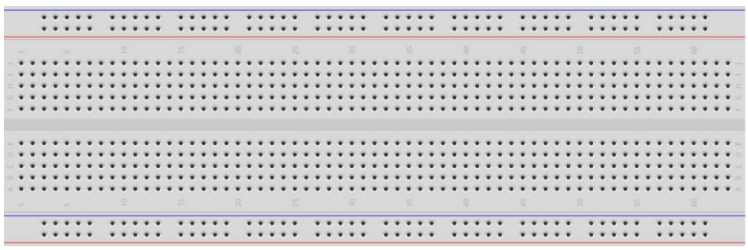
2.2.1 Photoresistor

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

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
 
</td>
<td rowspan="2">1 * Photoresistor
 
1 * ADC0834
 

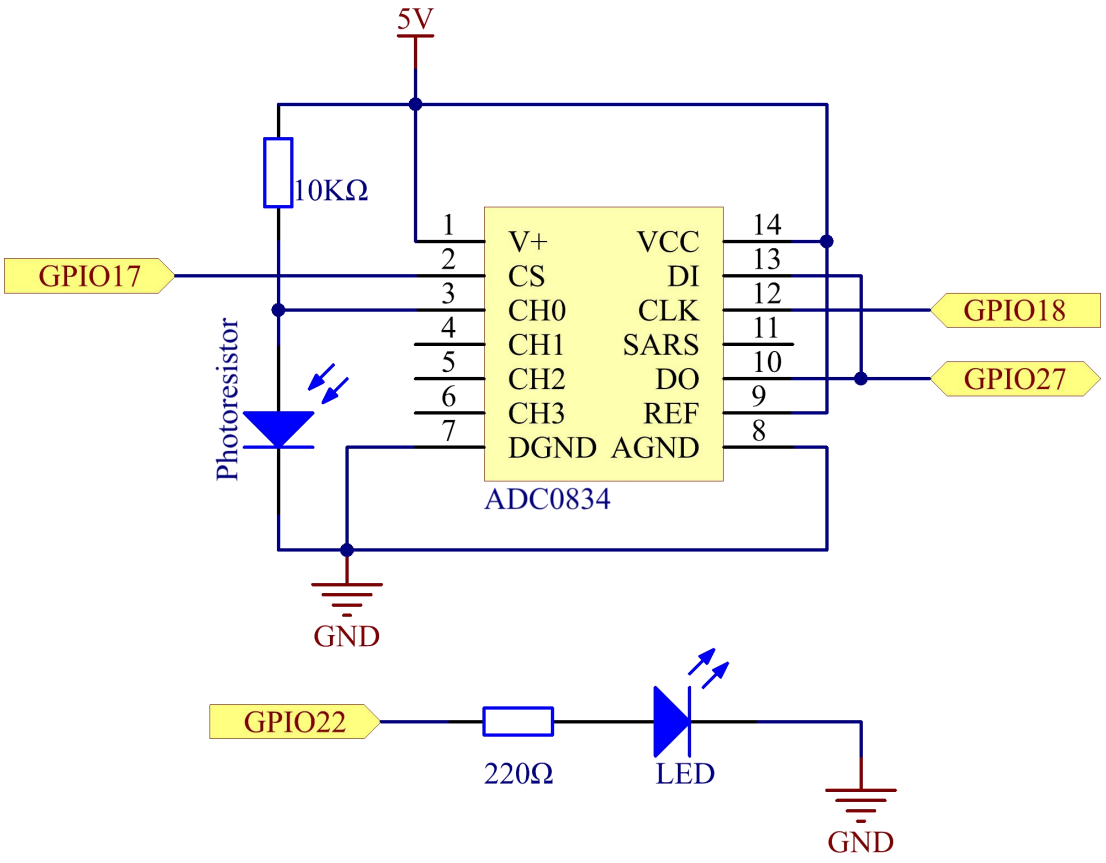
1 * LED
  </td>
</tr>
<tr>
<td colspan="2">1 * 40-pin Cable
  </td>
</tr>
<tr>
<td colspan="2">1 * Breadboard
  </td>
<td>Several Jumper Wires
 
1 * Resistor(220Ω)
 
1 * Resistor 10KΩ
  </td>
</tr>
</table>

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.

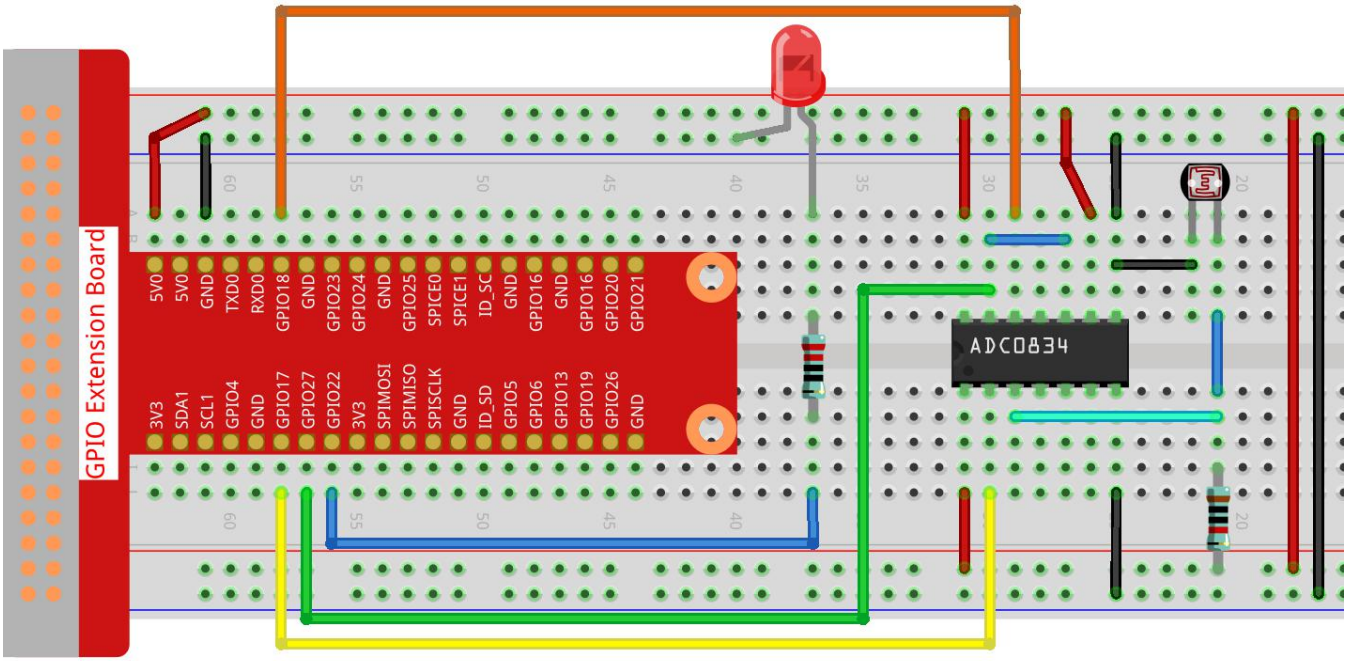
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin14	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/2.2.1/
```

Step 3: Compile the code.

```
gcc 2.2.1_Photoresistor.c -lwiringPi
```

Step 4: 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 <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>

typedef unsigned char uchar;
typedef unsigned int uint;

#define      ADC_CS     0
#define      ADC_CLK    1
```

```
#define       ADC_DIO    2
#define       LedPin     3

uchar get_ADC_Result(uint channel)
{
      uchar i;
      uchar dat1=0, dat2=0;
      int sel = channel > 1 & 1;
      int odd = channel & 1;

      pinMode(ADC_DIO, OUTPUT);
      digitalWrite(ADC_CS, 0);
      // Start bit
      digitalWrite(ADC_CLK,0);
      digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
      digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    //Single End mode
      digitalWrite(ADC_CLK,0);
      digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
      digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
      // ODD
      digitalWrite(ADC_CLK,0);
      digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
      digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
      //Select
      digitalWrite(ADC_CLK,0);
      digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
      digitalWrite(ADC_CLK,1);

      digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
      digitalWrite(ADC_CLK,0);
      digitalWrite(ADC_DIO,1);      delayMicroseconds(2);

      for(i=0;i<8;i++)
      {
            digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
            digitalWrite(ADC_CLK,0);      delayMicroseconds(2);

            pinMode(ADC_DIO, INPUT);
            dat1=dat1<<1 | digitalRead(ADC_DIO);
```

```
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(LedPin,   0, 100);
    pinMode(ADC_CS,   OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        printf("Current analogVal : %d\n", analogVal);
        softPwmWrite(LedPin, analogVal);
        delay(100);
    }
    return 0;
}
```

Code Explanation

The codes here are the same as that in 2.1.4 Potentiometer. If you have any other questions, please check the code explanation of **2.1.4 Potentiometer.c** for details.

➢ 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 python3 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

```
#!/usr/bin/env python3
import RPi.GPIO as GPIO
import ADC0834
import time
LedPin = 22
def setup():
    global led_val
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set all LedPin's mode to output and initial level to High(3.3v)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.HIGH)
    ADC0834.setup()
    # Set led as pwm channel and frequece to 2KHz
    led_val = GPIO.PWM(LedPin, 2000)
    # Set all begin with value 0
    led_val.start(0)
def destroy():
    # Stop all pwm channel
    led_val.stop()
    # Release resource
    GPIO.cleanup()
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
if __name__ == '__main__':
    setup()
```

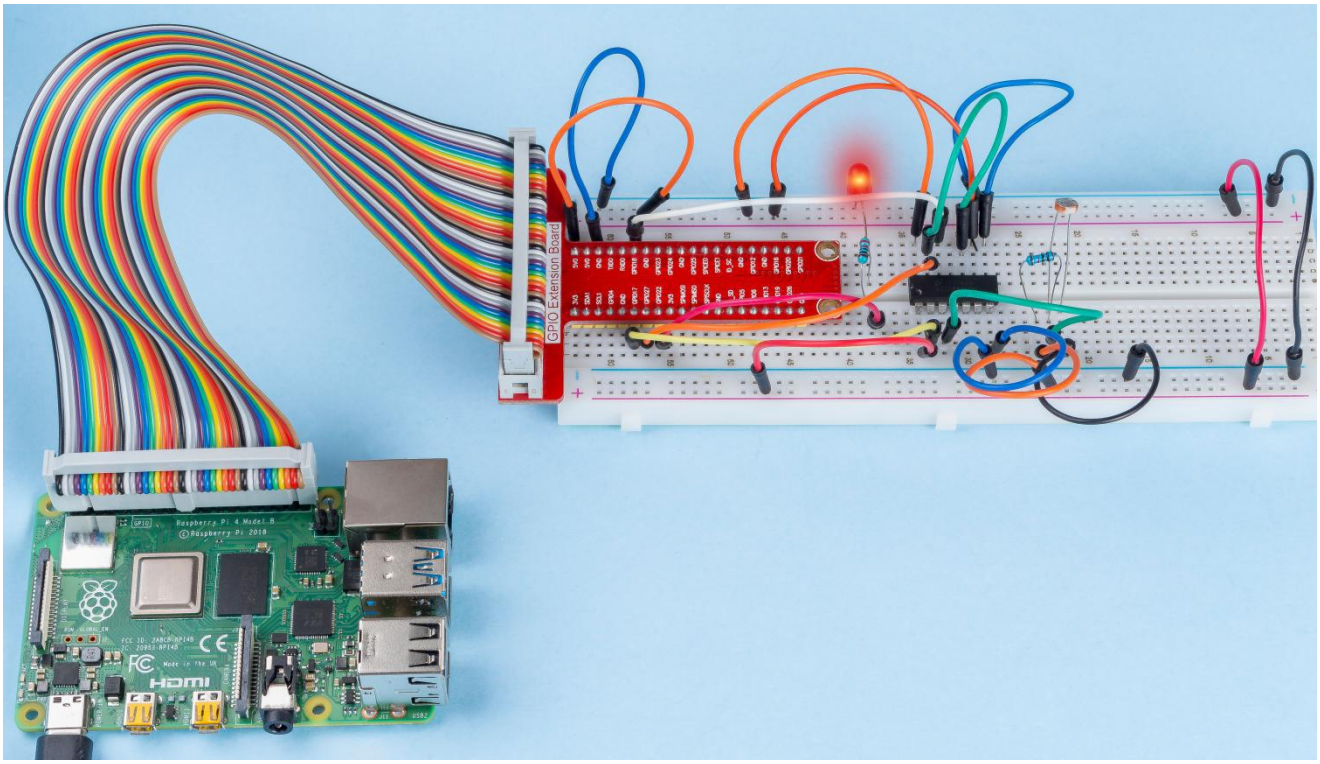
```
    try:
        loop()
    except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program destroy() will be executed.
        destroy()
```

Code Explanation

```
def loop():
    while True:
        analogVal = ADC0834.getResult()
        print ('analog value = %d' % analogVal)
        led_val.ChangeDutyCycle(analogVal*100/255)
        time.sleep(0.2)
```

Read the analog value of CH0 of ADC0834. By default, the function getResult() is used to read the value of CH0, so if you want to read other channels, please input 1, 2, or 3 into () of the function getResult(). Next, what you need is to print the value via the print function. Because the changing element is the duty cycle of LedPin, the computational formula, analogVal*100/255 is needed to convert analogVal into percentage. Finally, ChangeDutyCycle() is called to write the percentage into LedPin.

Phenomenon Picture

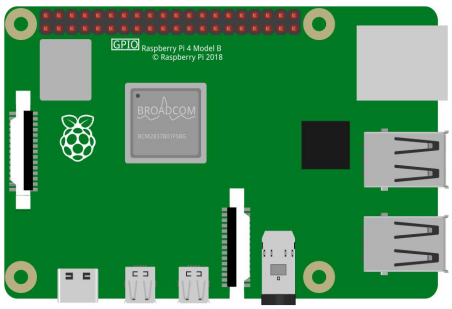
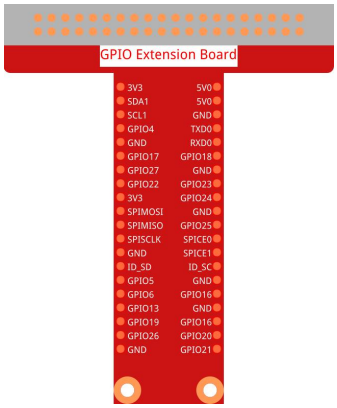
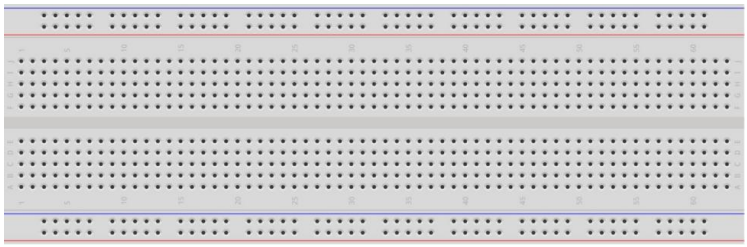


2.2.2 Thermistor

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 
		1 * ADC0834 
1 * 40-pin Cable 		Several Jumper Wires 
1 * Breadboard 		1 * 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:

$$R_T = R_N \exp^{B(1/TK - 1/TN)}$$

$\mathbf{R_T}$ 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 $\mathbf{T_N}$. 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.

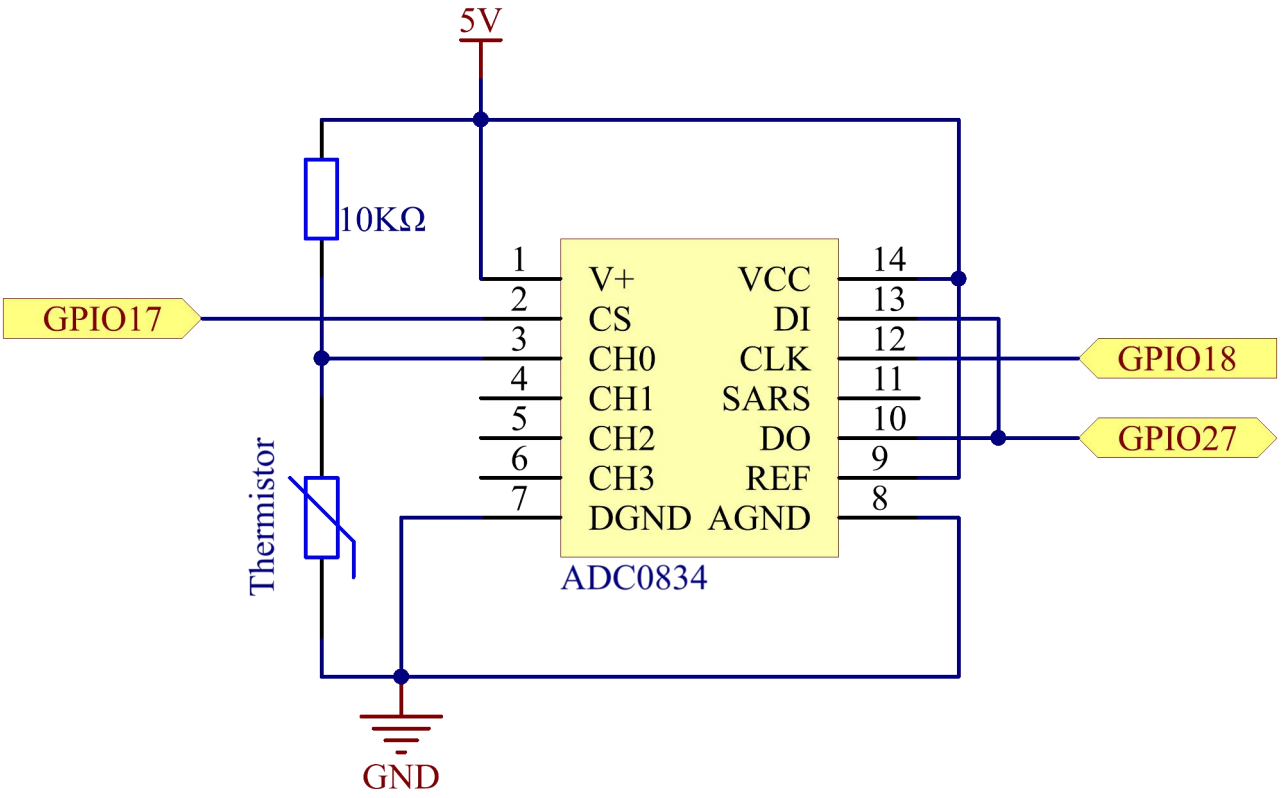
exp is the abbreviation of exponential, and the base number e is a natural number and equals 2.7 approximately.

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.

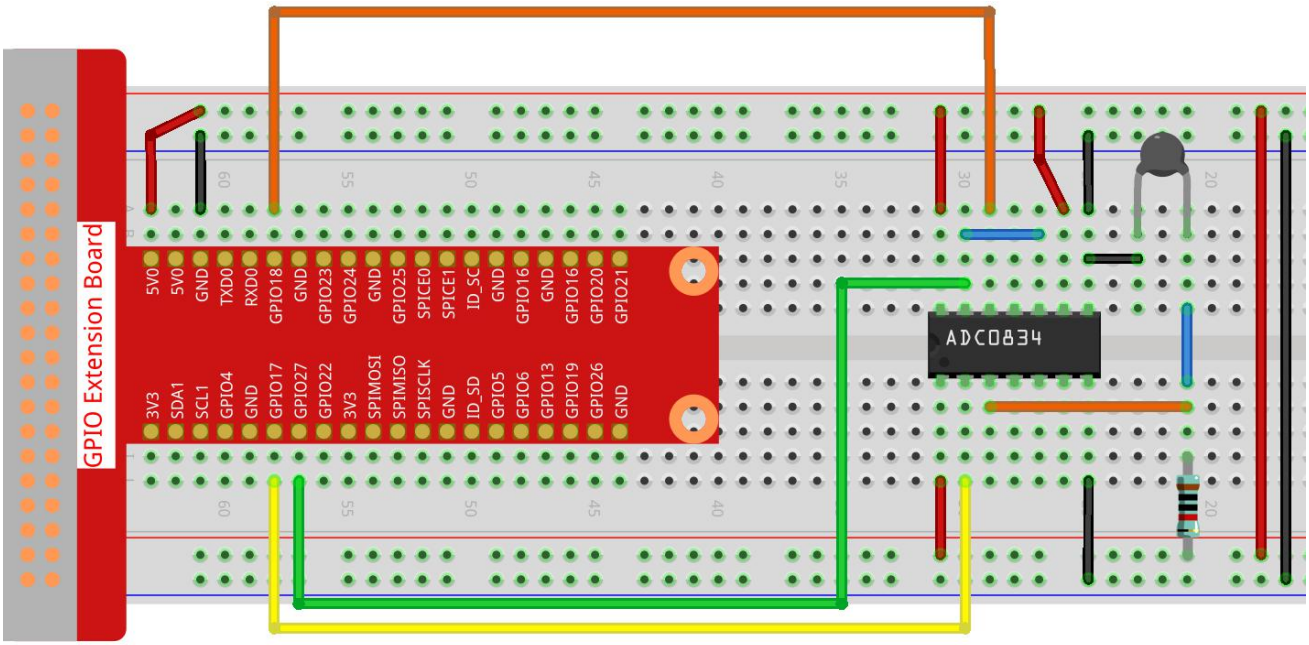
Schematic Diagram

T-Board Name	physical	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/2.2.2/
```

Step 3: 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 4: 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 <wiringPi.h>
#include <stdio.h>
#include <math.h>

typedef unsigned char uchar;
typedef unsigned int uint;
```

```
#define      ADC_CS     0
#define      ADC_CLK    1
#define      ADC_DIO    2

uchar get_ADC_Result(uint channel)
{
    uchar i;
    uchar dat1=0, dat2=0;
    int sel = channel > 1 & 1;
    int odd = channel & 1;

    pinMode(ADC_DIO, OUTPUT);
    digitalWrite(ADC_CS, 0);
    // Start bit
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
   //Single End mode
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    // ODD
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,odd);   delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
    //Select
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,sel);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,1);

    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);
    digitalWrite(ADC_CLK,0);
    digitalWrite(ADC_DIO,1);      delayMicroseconds(2);

    for(i=0;i<8;i++)
    {
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);

        pinMode(ADC_DIO, INPUT);
```

```
        dat1=dat1<<1 | digitalRead(ADC_DIO);
    }

    for(i=0;i<8;i++)
    {
        dat2 = dat2 | ((uchar)(digitalRead(ADC_DIO))<<i);
        digitalWrite(ADC_CLK,1);      delayMicroseconds(2);
        digitalWrite(ADC_CLK,0);      delayMicroseconds(2);
    }

    digitalWrite(ADC_CS,1);
    pinMode(ADC_DIO, OUTPUT);
    return(dat1==dat2) ? dat1 : 0;
}

int main(void)
{
    unsigned char analogVal;
   double Vr, Rt, temp, cel, Fah;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ADC_CS,    OUTPUT);
    pinMode(ADC_CLK, OUTPUT);

    while(1){
        analogVal = get_ADC_Result(0);
        Vr = 5 * (double)(analogVal) / 255;
        Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
        temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
        cel = temp - 273.15;
        Fah = cel * 1.8 +32;
        printf("Celsius: %.2f C    Fahrenheit: %.2f F\n", cel, Fah);
        delay(100);
    }
    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 = get_ADC_Result(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)));
        cel = temp - 273.15;
        Fah = cel * 1.8 +32;
        printf("Celsius: %.2f C    Fahrenheit: %.2f F\n", cel, Fah);
```

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.

```
Fah = cel * 1.8 +32;
```

Convert degree Celsius into Fahrenheit.

```
printf("Celsius: %.2f C    Fahrenheit: %.2f F\n", cel, Fah);
```

Print centigrade degree, Fahrenheit degree and their units on the display.

➢ 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 python3 2.2.2_Thermistor.py
```

With the code run, the thermistor detects ambient temperature which will be printed on the screen once it finishes the program calculation.

Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import RPi.GPIO as GPIO
import ADC0834
import time
import math

def init():
    ADC0834.setup()

def loop():
    while True:
        analogVal = ADC0834.getResult()
        Vr = 5 * float(analogVal) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
        Cel = temp - 273.15
        Fah = Cel * 1.8 + 32
        print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
        time.sleep(0.2)

if __name__ == '__main__':
    init()
    try:
        loop()
    except KeyboardInterrupt:
        ADC0834.destroy()
```

Code Explanation

```
import math
```

There is a numerics library which declares a set of functions to compute common mathematical operations and transformations.

```
analogVal = ADC0834.getResult()
```

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)))
        Cel = temp - 273.15
        Fah = Cel * 1.8 + 32
        print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
```

These calculations convert the thermistor values into centigrade degree and Fahrenheit degree.

```
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 centigrade degree.

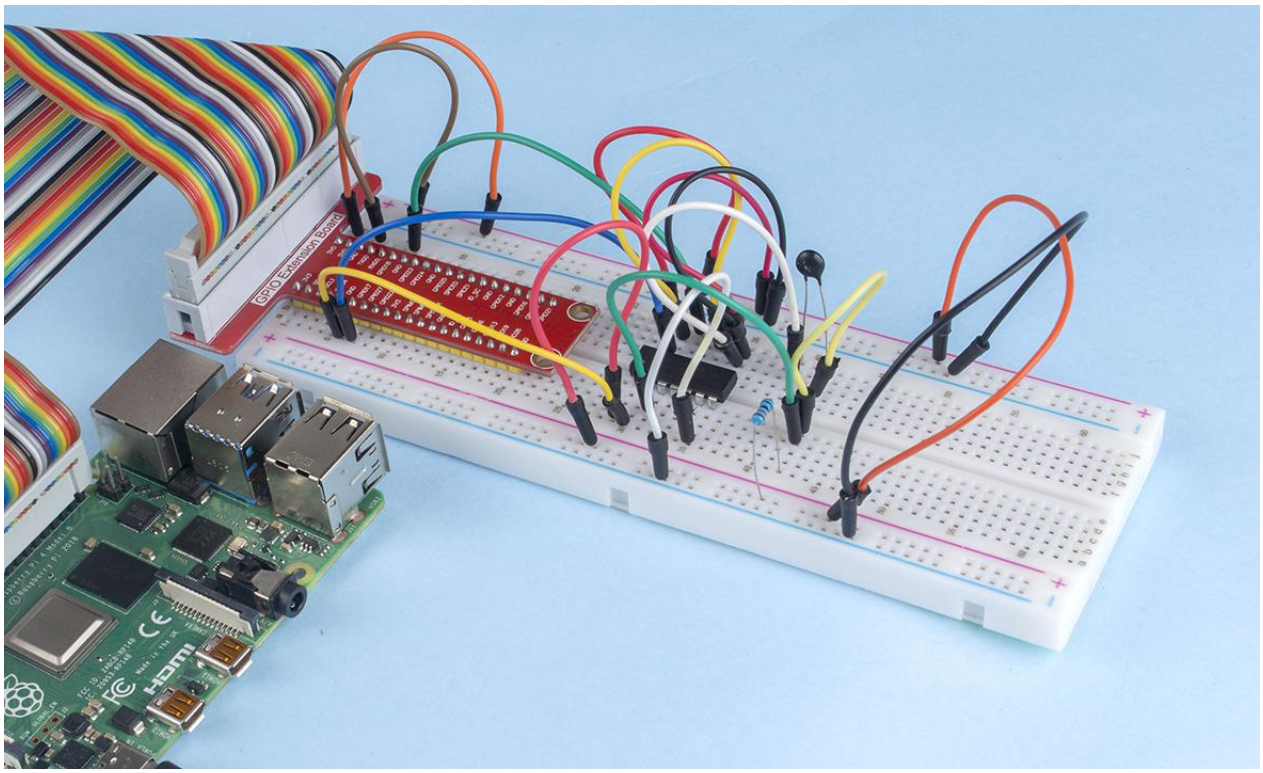
```
Fah = Cel * 1.8 + 32
```

Convert the centigrade degree into Fahrenheit degree.

```
print ('Celsius: %.2f °C  Fahrenheit: %.2f °F' % (Cel, Fah))
```

Print centigrade degree, Fahrenheit degree and their units on the display.

Phenomenon Picture



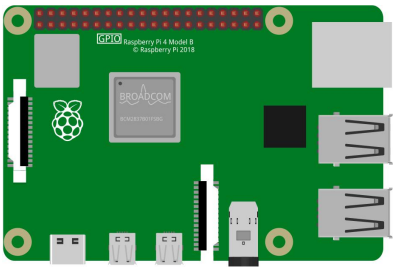
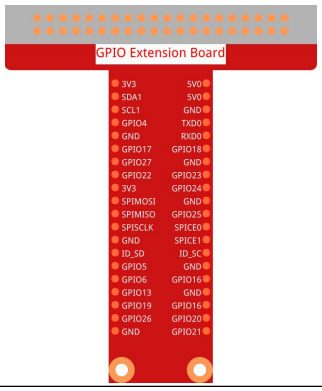
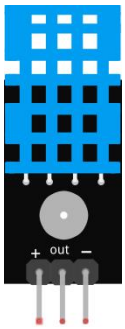
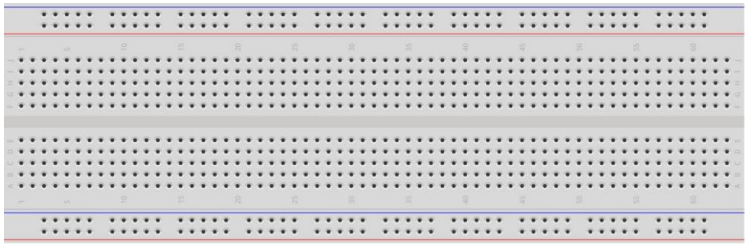
2.2.3 DHT-11

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.

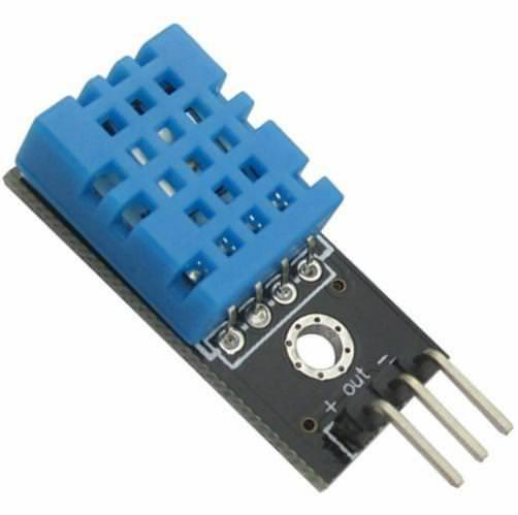
The sensors include a wet element resistive sensor and a NTC temperature sensor and they are connected to a high performance 8-bit microcontroller.

Components

1 * Raspberry Pi 	1 * T-Extension Board 	1 * DHT-11 
1 * 40-pin Cable 		Several Jumper Wires 
1 * Breadboard 		1 * Resistor 10KΩ 

Principle

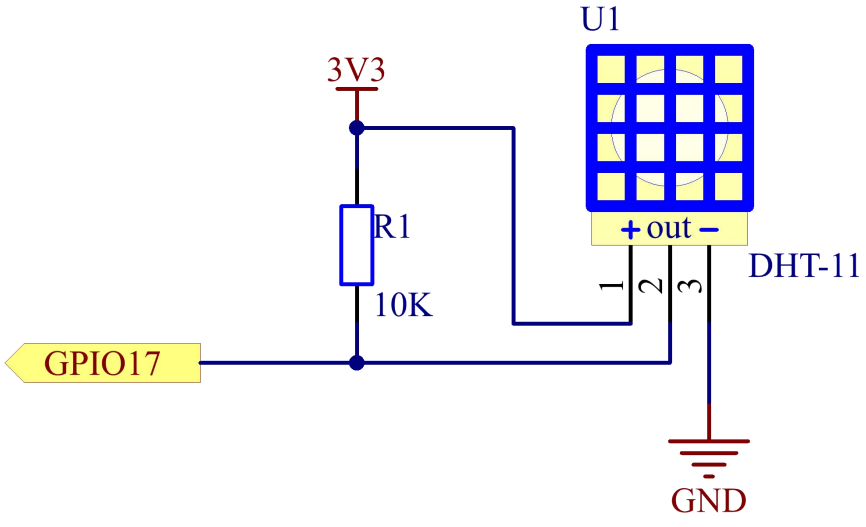
The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins are needed).



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.

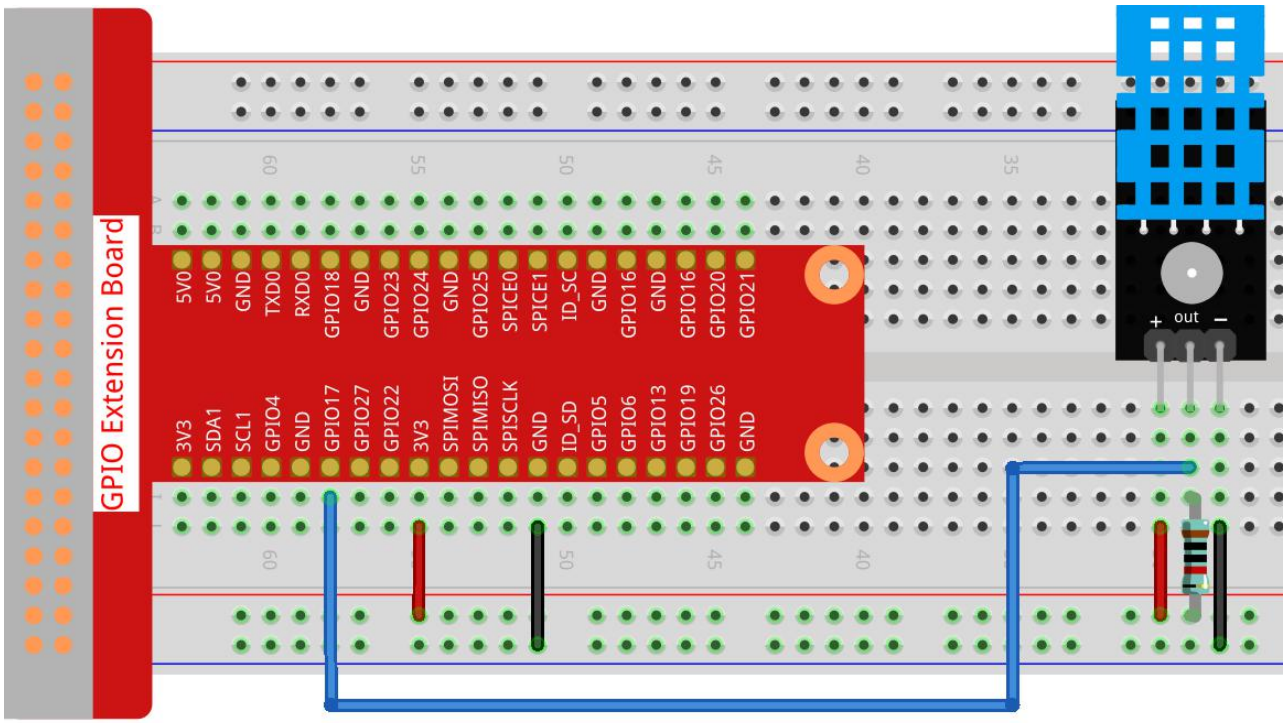
Schematic Diagram

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.c -lwiringPi
```

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

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

#define maxTim 85
#define dhtPin 0
```

```
int dht11_dat[5] = {0,0,0,0,0};

void readDht11() {
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit
    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // pull pin down for 18 milliseconds
    pinMode(dhtPin, OUTPUT);
    digitalWrite(dhtPin, LOW);
    delay(18);
    // then pull it up for 40 microseconds
    digitalWrite(dhtPin, HIGH);
    delayMicroseconds(40);
    // prepare to read the pin
    pinMode(dhtPin, INPUT);

    // detect change and read data
    for ( i=0; i< maxTim; i++) {
        counter = 0;
        while (digitalRead(dhtPin) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 255) {
                break;
            }
        }
        laststate = digitalRead(dhtPin);

        if (counter == 255) break;
        // ignore first 3 transitions
        if ((i >= 4) && (i%2 == 0)) {
            // shove each bit into the storage bytes
            dht11_dat[j/8] <<= 1;
            if (counter > 50)
                dht11_dat[j/8] |= 1;
            j++;
        }
```

```
    }
    // check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
    // print it out if data is good
    if ((j >= 40) &&
            (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] + dht11_dat[3])
& 0xFF)) ) {
        Fah = dht11_dat[2] * 9. / 5. + 32;
        printf("Humidity = %d.%d %% Temperature = %d.%d *C (%.1f *F)\n",
                dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], Fah);
    }
}

int main (void) {
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    while (1) {
        readDht11();
        delay(500); // wait 1sec to refresh
    }
    return 0 ;
}
```

Code Explanation

```
void readDht11() {
    uint8_t laststate = HIGH;
    uint8_t counter = 0;
    uint8_t j = 0, i;
    float Fah; // fahrenheit
    dht11_dat[0] = dht11_dat[1] = dht11_dat[2] = dht11_dat[3] = dht11_dat[4] = 0;
    // ...
}
```

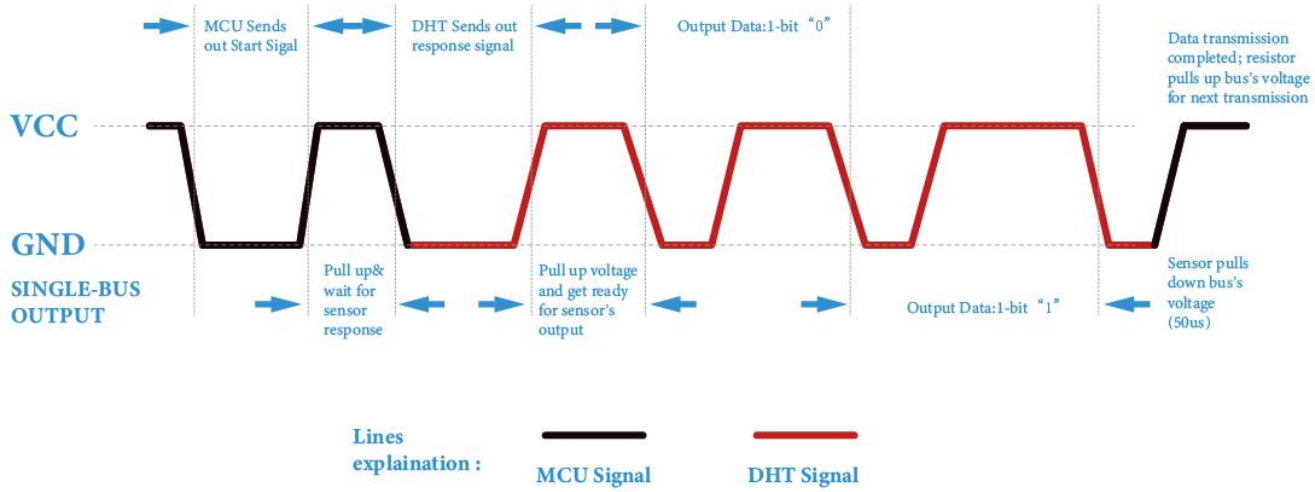
This function is used to realize the function of DHT11.

It generally can be divided into 3 parts:

1. prepare to read the pin:

```
    // pull pin down for 18 milliseconds
    pinMode(dhtPin, OUTPUT);
    digitalWrite(dhtPin, LOW);
    delay(18);
    // then pull it up for 40 microseconds
    digitalWrite(dhtPin, HIGH);
    delayMicroseconds(40);
    // prepare to read the pin
    pinMode(dhtPin, INPUT);
```

Its communication flow is determined by work timing.



When DHT11 starts up, MCU will send a low level signal and then keep the signal at high level for 40us. After that, the detection of the condition of external environment will start.

2. read data:

```
// detect change and read data
for ( i=0; i< maxTim; i++) {
        counter = 0;
        while (digitalRead(dhtPin) == laststate) {
            counter++;
            delayMicroseconds(1);
            if (counter == 255) {
                break;
            }
        }
        laststate = digitalRead(dhtPin);
        if (counter == 255) break;
```

```
        // ignore first 3 transitions
        if ((i >= 4) && (i%2 == 0)) {
            // shove each bit into the storage bytes
            dht11_dat[j/8] <<= 1;
            if (counter > 50)
                dht11_dat[j/8] |= 1;
            j++;
        }
    }
```

The loop stores the detected data in the dht11_dat[] array. DHT11 transmits data of 40 bits at a time. The first 16 bits are related to humidity, the middle 16 bits are related to temperature, and the last eight bits are used for verification. The data format is:

8bit humidity integer data + **8bit humidity decimal data** + **8bit temperature integer data** + **8bit temperature decimal data** + **8bit check bit**.

3. Print Humidity & Temperature.

```
    // check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
    // print it out if data is good
    if ((j >= 40) &&
            (dht11_dat[4] == ((dht11_dat[0] + dht11_dat[1] + dht11_dat[2] + dht11_dat[3])
& 0xFF)) ) {
        Fah = dht11_dat[2] * 9. / 5. + 32;
        printf("Humidity = %d.%d %% Temperature = %d.%d *C (%.1f *F)\n",
                dht11_dat[0], dht11_dat[1], dht11_dat[2], dht11_dat[3], Fah);
    }
```

When the data storage is up to 40 bits, check the validity of the data through the **check bit (dht11_dat[4])**, and then print the temperature and humidity.

For example, if the received data is 00101011(8-bit value of humidity integer) 00000000 (8-bit value of humidity decimal) 00111100 (8-bit value of temperature integer) 00000000 (8-bit value of temperature decimal) 01100111 (check bit)

Calculation:

00101011+00000000+00111100+00000000=01100111.

The final result is equal to the check bit data, then the received data is correct:

Humidity =43%， Temperature =60*C.

If it is not equal to the check bit data, the data transmission is not normal and the data is received again.

➢ 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 python3 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 = 17

GPIO.setmode(GPIO.BCM)

MAX_UNCHANGE_COUNT = 100

STATE_INIT_PULL_DOWN = 1
STATE_INIT_PULL_UP = 2
STATE_DATA_FIRST_PULL_DOWN = 3
STATE_DATA_PULL_UP = 4
STATE_DATA_PULL_DOWN = 5

def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)

    unchanged_count = 0
```

```
    last = -1
    data = []
    while True:
        current = GPIO.input(dhtPin)
        data.append(current)
        if last != current:
            unchanged_count = 0
            last = current
        else:
            unchanged_count += 1
            if unchanged_count > MAX_UNCHANGE_COUNT:
                break

    state = STATE_INIT_PULL_DOWN

    lengths = []
    current_length = 0

    for current in data:
        current_length += 1

        if state == STATE_INIT_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_INIT_PULL_UP
            else:
                continue
        if state == STATE_INIT_PULL_UP:
            if current == GPIO.HIGH:
                state = STATE_DATA_FIRST_PULL_DOWN
            else:
                continue
        if state == STATE_DATA_FIRST_PULL_DOWN:
            if current == GPIO.LOW:
                state = STATE_DATA_PULL_UP
            else:
                continue
        if state == STATE_DATA_PULL_UP:
            if current == GPIO.HIGH:
                current_length = 0
                state = STATE_DATA_PULL_DOWN
```

```
            else:
                continue
        if state == STATE_DATA_PULL_DOWN:
            if current == GPIO.LOW:
                lengths.append(current_length)
                state = STATE_DATA_PULL_UP
            else:
                continue
    if len(lengths) != 40:
        #print ("Data not good, skip")
        return False

    shortest_pull_up = min(lengths)
    longest_pull_up = max(lengths)
    halfway = (longest_pull_up + shortest_pull_up) / 2
    bits = []
    the_bytes = []
    byte = 0

    for length in lengths:
        bit = 0
        if length > halfway:
            bit = 1
        bits.append(bit)
    #print ("bits: %s, length: %d" % (bits, len(bits)))
    for i in range(0, len(bits)):
        byte = byte << 1
        if (bits[i]):
            byte = byte | 1
        else:
            byte = byte | 0
        if ((i + 1) % 8 == 0):
            the_bytes.append(byte)
            byte = 0
    #print (the_bytes)
    checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False
```

```
    return the_bytes[0], the_bytes[2]

def main():

    while True:
        result = readDht11()
        if result:
            humidity, temperature = result
            print ("humidity: %s %%,  Temperature: %s C`" % (humidity, temperature))
        time.sleep(1)

def destroy():
    GPIO.cleanup()

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

Code Explanation

```
def readDht11():
    GPIO.setup(dhtPin, GPIO.OUT)
    GPIO.output(dhtPin, GPIO.HIGH)
    time.sleep(0.05)
    GPIO.output(dhtPin, GPIO.LOW)
    time.sleep(0.02)
    GPIO.setup(dhtPin, GPIO.IN, GPIO.PUD_UP)
    unchanged_count = 0
    last = -1
    data = []
    #...
```

This function is used to implement the functions of DHT11. It stores the detected data in the the_bytes[] array. DHT11 transmits data of 40 bits at a time. The first 16 bits are related to humidity, the middle 16 bits are related to temperature, and the last eight bits are used for verification. The data format is:

8bit humidity integer data +8bit humidity decimal data +8bit temperature integer data + 8bit temperature decimal data + 8bit check bit.

When the validity is detected via the check bit, the function returns two results: 1. error; 2. humidity and temperature.

```
    checksum = (the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]) & 0xFF
    if the_bytes[4] != checksum:
        #print ("Data not good, skip")
        return False

    return the_bytes[0], the_bytes[2]
```

For example, if the received date is 00101011(8-bit value of humidity integer) 00000000 (8-bit value of humidity decimal) 00111100 (8-bit value of temperature integer) 00000000 (8-bit value of temperature decimal) 01100111 (check bit)

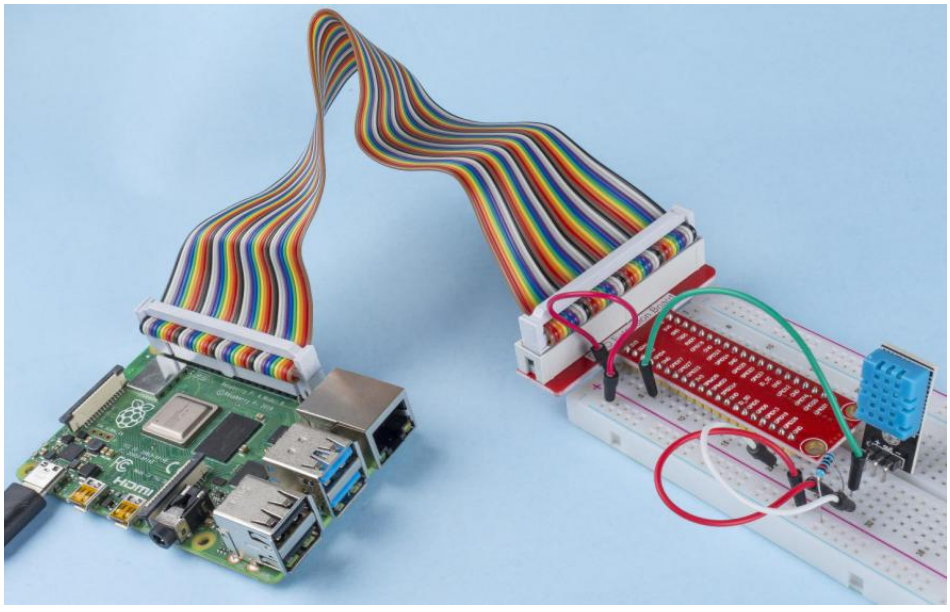
Calculation:

00101011+00000000+00111100+00000000=01100111.

If the final result is equal to the check bit data, the data transmission is abnormal: return False.

If the final result is equal to the check bit data, the received data is correct, then there will return the_bytes[0] and the_bytes[2] and output “Humidity =43%， Temperature =60C”.

Phenomenon Picture

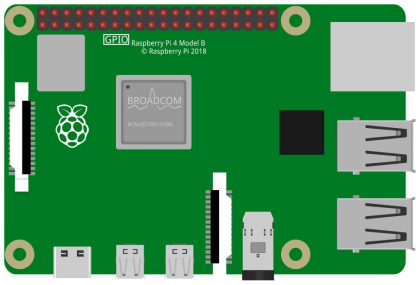
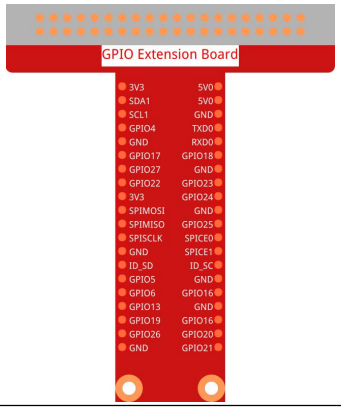
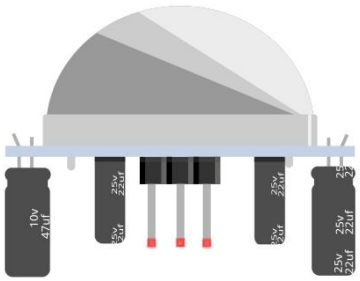
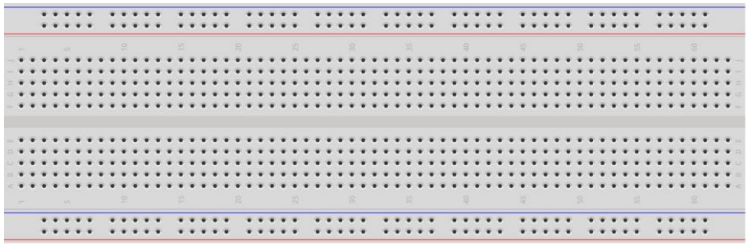


2.2.4 PIR

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

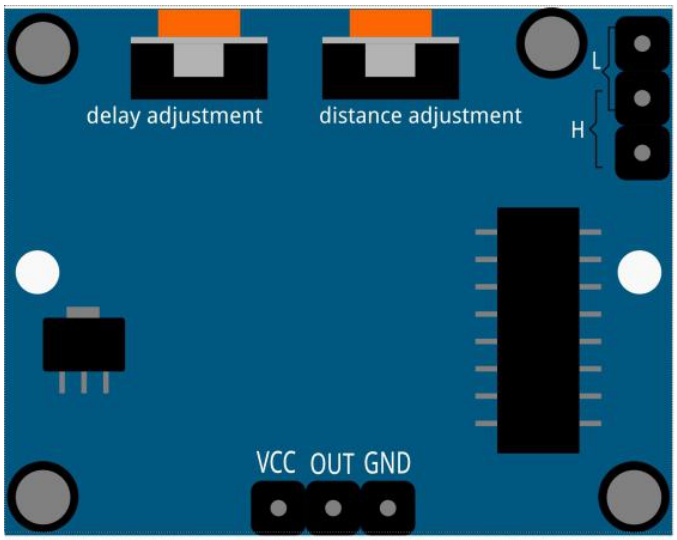
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * PIR Sensor Module
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>Several Jumper Wires
</td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>3 * 220 Resistor
  </td><td>1 * RGB LED
  </td></tr>
</table>

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.

After the sensing module is wired, there is a one-minute initialization. During the initialization, module will output for 0~3 times at intervals. Then the module will be in the standby mode. Please keep the interference of light source and other sources away from the surface of the module so as to avoid the misoperation caused by the interfering signal. Even you'd better use the module without too much wind, because the wind can also interfere with the sensor.



Distance Adjustment

Turning the knob of the distance adjustment potentiometer clockwise, the range of sensing distance increases, and the maximum sensing distance range is about 0-7 meters. If turn it anticlockwise, the range of sensing distance is reduced, and the minimum sensing distance range is about 0-3 meters.

Delay adjustment

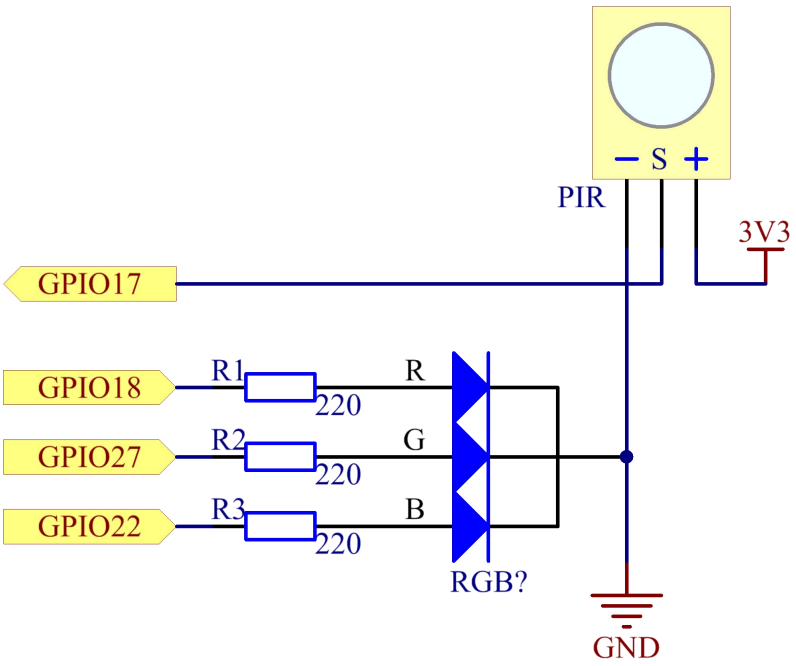
Rotate the knob of the delay adjustment potentiometer clockwise, you can also see the sensing delay increasing. The maximum of the sensing delay can reach up to 300s. On the contrary, if rotate it anticlockwise, you can shorten the delay with a minimum of 5s.

Two trigger modes: (choosing different modes by using the jumper cap).

- ✧ **H: Repeatable trigger mode**, after sensing the human body, the module outputs high level. During the subsequent delay period, if somebody enters the sensing range,the output will keep being the high level.
- ✧ **L**: **Non-repeatable trigger mode**, outputs high level when it senses the human body. After the delay, the output will change from high level into low level automatically.

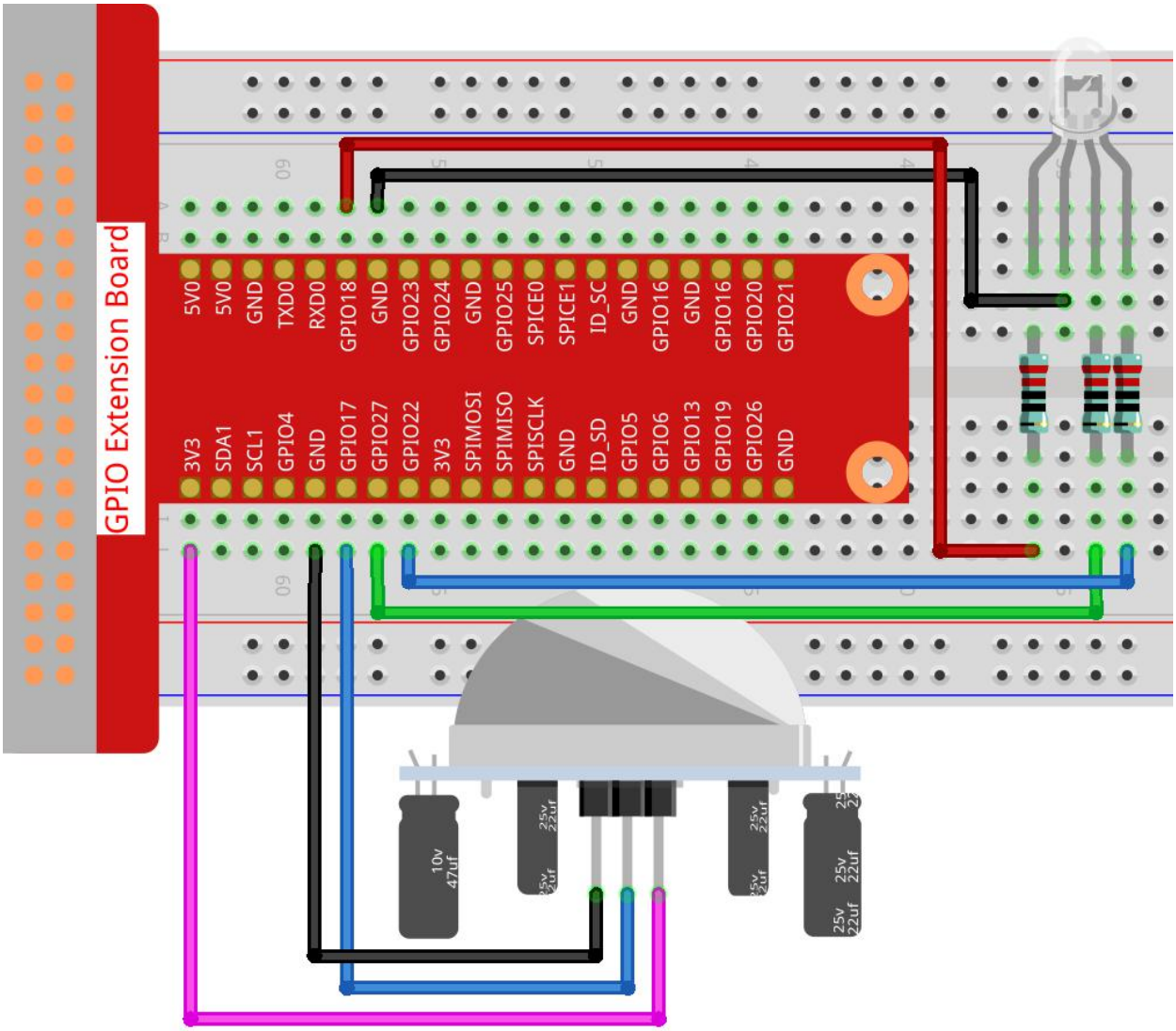
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin12	1	18
GPIO27	Pin13	2	27
GPIO22	Pin15	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/2.2.4/
```

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 let RGB LED glow yellow 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 <softPwm.h>
#include <stdio.h>
#define uchar unsigned char

#define pirPin    0     //the pir connect to GPIO0
#define redPin    1
#define greenPin  2
#define bluePin   3

void ledInit(void){
    softPwmCreate(redPin,  0, 100);
    softPwmCreate(greenPin,0, 100);
    softPwmCreate(bluePin, 0, 100);
}
void ledColorSet(uchar r_val, uchar g_val, uchar b_val){
    softPwmWrite(redPin,   r_val);
    softPwmWrite(greenPin, g_val);
    softPwmWrite(bluePin,  b_val);
}
int main(void)
{
    int pir_val;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to
screen
        printf("setup wiringPi failed !");
        return 1;
    }
    ledInit();
    pinMode(pirPin, INPUT);
    while(1){
         pir_val = digitalRead(pirPin);
        if(pir_val== 1){ //if read pir is HIGH level
            ledColorSet(0xff,0xff,0x00);
        }
        else {
          ledColorSet(0x00,0x00,0xff);
        }
    }
    return 0;
}
```

Code Explanation

```
void ledInit(void);
void ledColorSet(uchar r_val, uchar g_val, uchar b_val);
```

These codes are used to set the color of the RGB LED, and please refer to **1.1.2-RGB LED** for more details.

```
int main(void)
{
    int pir_val;
    //……
    pinMode(pirPin, INPUT);
    while(1){
         pir_val = digitalRead(pirPin);
        if(pir_val== 1){ //if read pir is HIGH level
            ledColorSet(0xff,0xff,0x00);
        }
        else {
          ledColorSet(0x00,0x00,0xff);
        }
    }
    return 0;
}
```

When PIR detects the human infrared spectrum, RGB LED emits the yellow light; if not, emits the blue light.

➢ 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 python3 2.2.4_PIR.py
```

After the code runs, PIR detects surroundings and let RGB LED glow yellow 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
import time

rgbPins = {'Red':18, 'Green':27, 'Blue':22}
pirPin = 17    # the pir connect to pin17

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)      # Set the GPIO modes to BCM Numbering
    GPIO.setup(pirPin, GPIO.IN)    # Set pirPin to input
    # Set all LedPin's mode to output and initial level to High(3.3v)
    for i in rgbPins:
        GPIO.setup(rgbPins[i], GPIO.OUT, initial=GPIO.HIGH)

    # Set all led as pwm channel and frequece to 2KHz
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)

    # Set all begin with value 0
    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 .
    # Devide colors from 'color' veriable
    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)

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

```
    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 loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)

def destroy():
    p_R.stop()
    p_G.stop()
    p_B.stop()
    GPIO.cleanup()                     # Release resource

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

Code Explanation

```
rgbPins = {'Red':18, 'Green':27, 'Blue':22}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    # ……
    for i in rgbPins:
        GPIO.setup(rgbPins[i], GPIO.OUT, initial=GPIO.HIGH)
    p_R = GPIO.PWM(rgbPins['Red'], 2000)
    p_G = GPIO.PWM(rgbPins['Green'], 2000)
    p_B = GPIO.PWM(rgbPins['Blue'], 2000)
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)
```

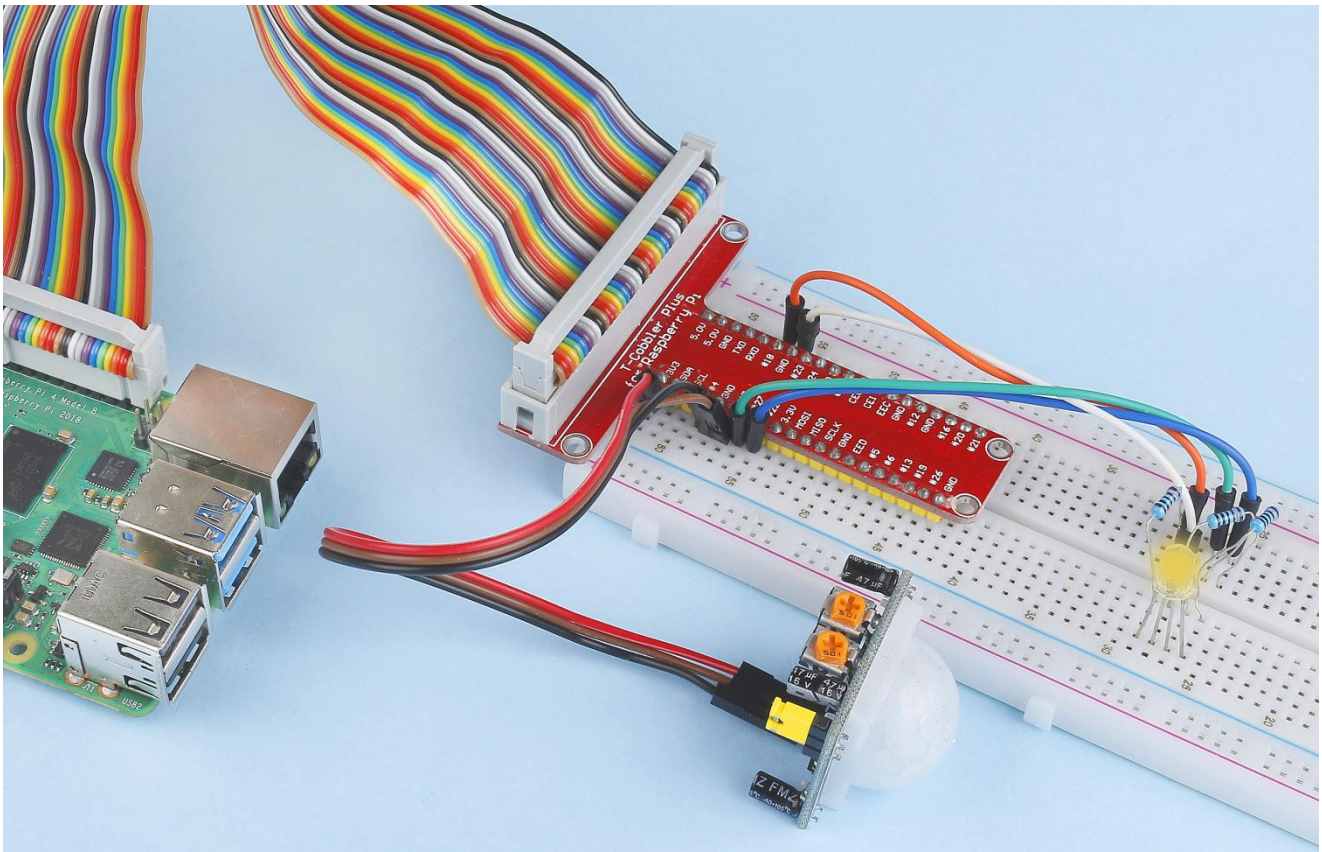
```
def MAP(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

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

These codes are used to set the color of the RGB LED, and please refer to **1.1.2-RGB LED** for more details.

```
def loop():
    while True:
        pir_val = GPIO.input(pirPin)
        if pir_val==GPIO.HIGH:
            setColor(0xFFFF00)
        else :
            setColor(0x0000FF)
```

When PIR detects the human infrared spectrum, RGB LED emits the yellow light; if not, emits the blue light.

Phenomenon Picture

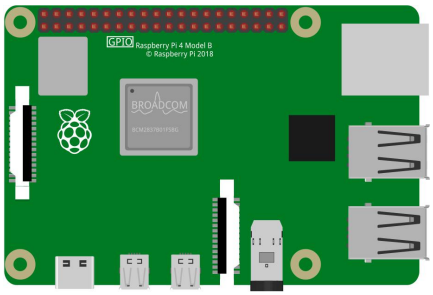
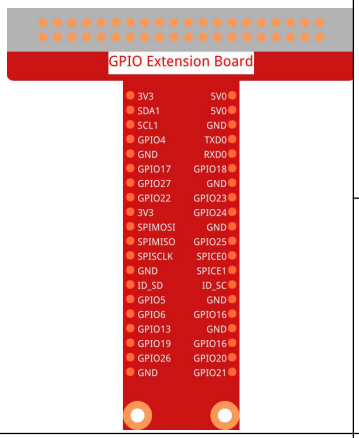
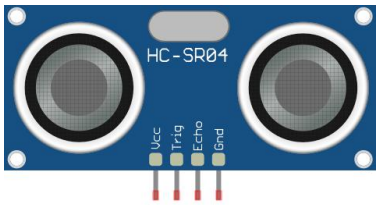
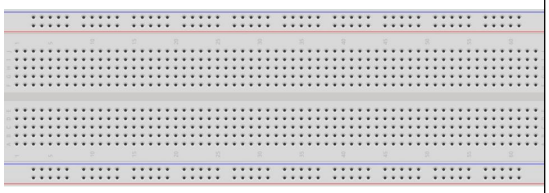


2.2.5 Ultrasonic Sensor Module

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

<table>
<tr>
<td>1 * Raspberry Pi
  </td>
<td>1 * T-Extension Board
 
</td>
<td>1 * HC SR04
 
</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Several Jumper Wires
  </td>
</tr>
<tr>
<td>1 * 40-pin Cable
  </td>
<td></td>
<td>1 * Breadboard
  </td>
</tr>
</table>

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;

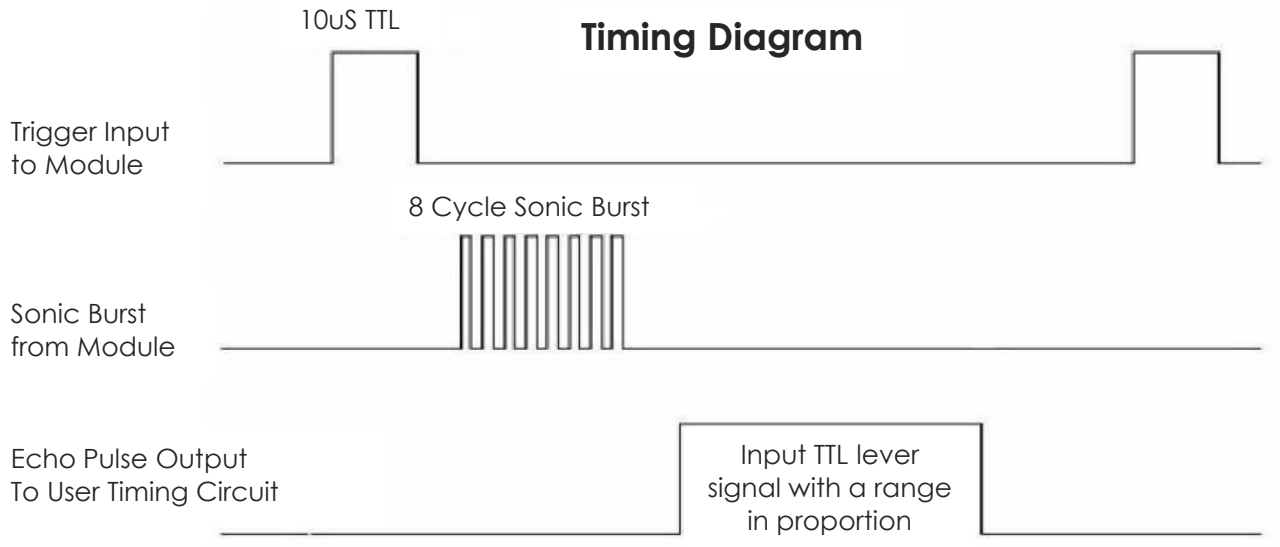
(2) The module automatically sends eight 40khz and detects if there is a pulse signal return.

(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.

TRIG	Trigger Pulse Input
ECHO	Echo Pulse Output
GND	Ground
VCC	Supply

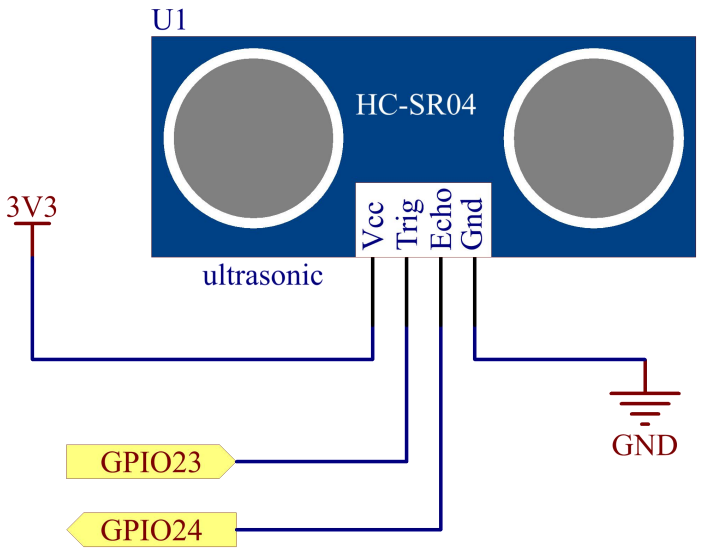
The timing diagram is shown below. You only need to supply a short 10us pulse for the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. You can calculate the range through the time interval between sending trigger signal and receiving echo signal.

Formula: us / 58 = centimeters or us / 148 =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.



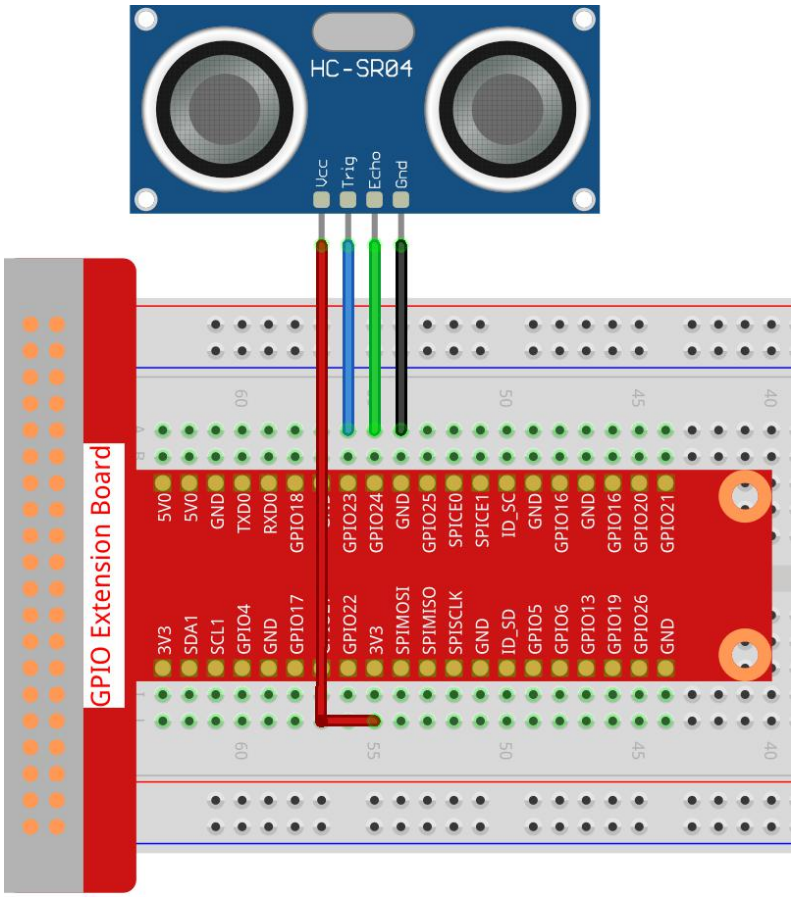
Schematic Diagram

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/c/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 python3 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 ('Distance: %.2f' % dis )
        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.

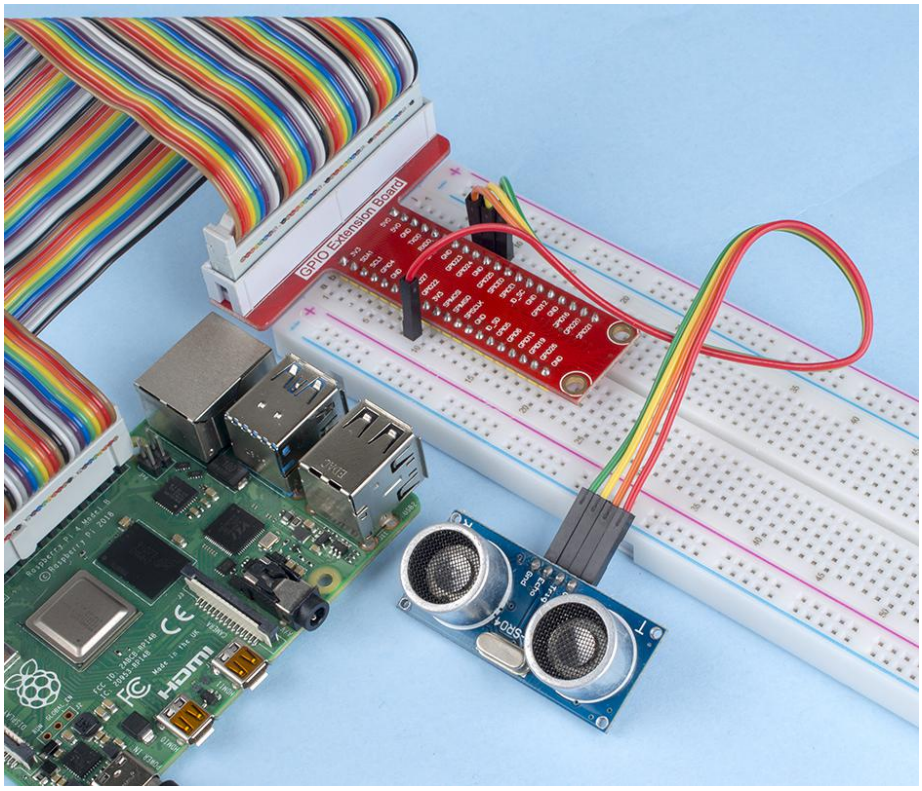
```
    during = time2 - time1
```

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

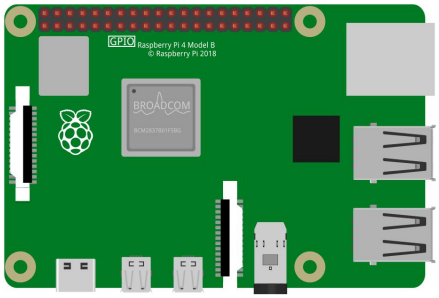
Introduction

The MPU-6050 is the world's first and only 6-axis motion tracking devices (3-axis Gyroscope and 3-axis Accelerometer) designed for smartphones, tablets and wearable sensors that have these features, including the low power, low cost, and high performance requirements.

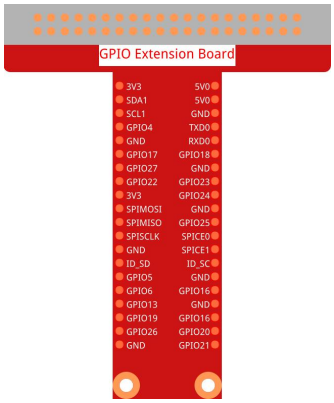
In this experiment, use I2C to obtain the values of the three-axis acceleration sensor and three-axis gyroscope for MPU6050 and display them on the screen.

Components

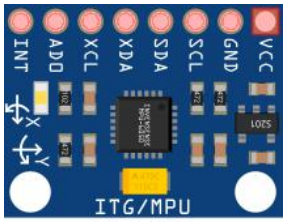
1 * Raspberry Pi



1 * T-Extension Board



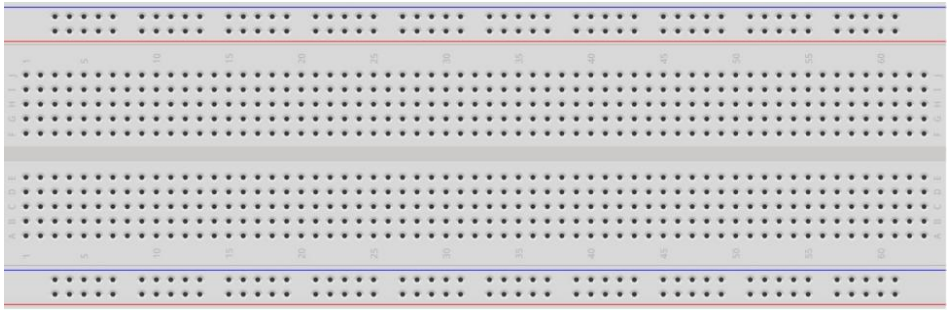
1 * MPU6050



Several Jumper Wires

1 * 40-pin Cable

1 * Breadboard



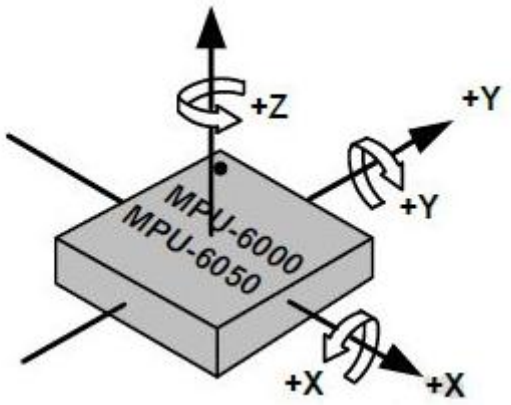
Principle

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:

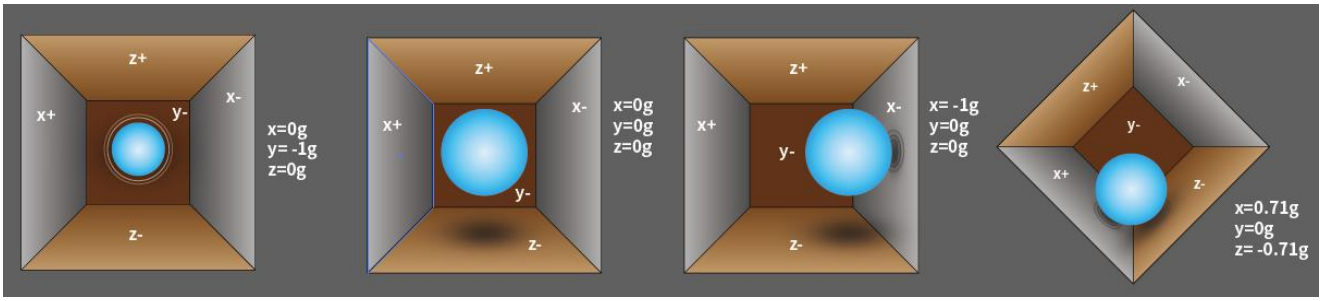
Put MPU6050 flat on the table, assure that the face with label is upward and a dot on this surface is on the top left corner. Then the upright direction upward is the z-axis of the chip. The direction from left to right is regarded as the X-axis. Accordingly the direction from back to front is defined as the Y-axis.



3-axis Accelerometer

The accelerometer works on the principle of piezo electric effect, the ability of certain materials to generate an electric charge in response to applied mechanical stress.

Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude.



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.

Acceleration = (Accelerometer axis raw data / 65536 * full scale Acceleration range) g

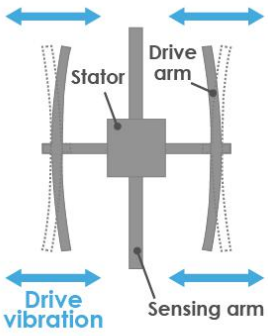
Take the X-axis as an example, when Accelerometer X axis raw data is 16384 and the range is selected as +/-2g:

Acceleration along the X axis = (16384 / 65536 * 4) g

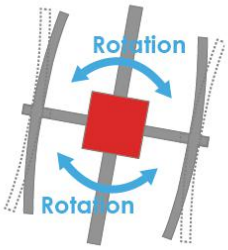
=1g

4-axis Gyroscope

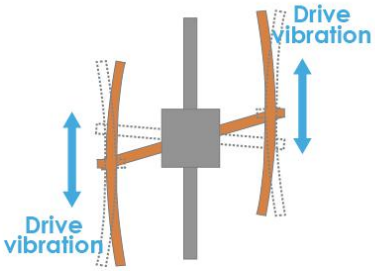
Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified.



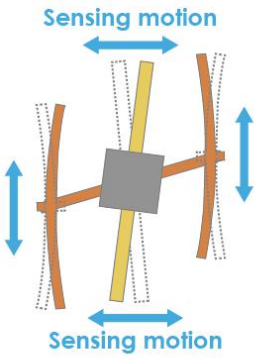
1.Normally, a drive arm vibrates in a certain direction.



2.Direction of rotation



3.When the gyro is rotated,the Coriolis force acts on the drive arms, producing vertical vibration.



4.The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

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:

Angular velocity = (Gyroscope axis raw data / 65536 * full scale Gyroscope range) °/s

The X axis, for example, the Accelerometer X axis raw data is 16384 and ranges + / - 250°/ s:

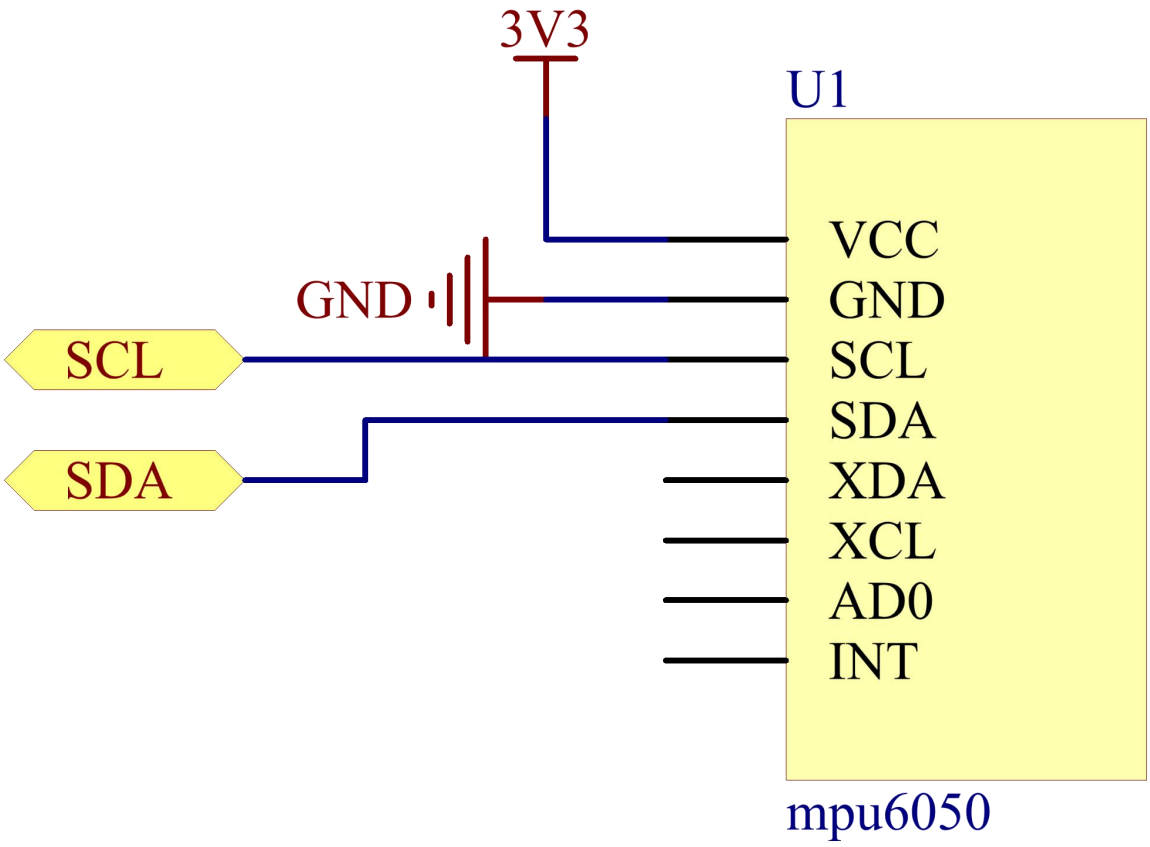
Angular velocity along the X axis = (16384 / 65536 * 500)°/s

=125°/s

Schematic Diagram

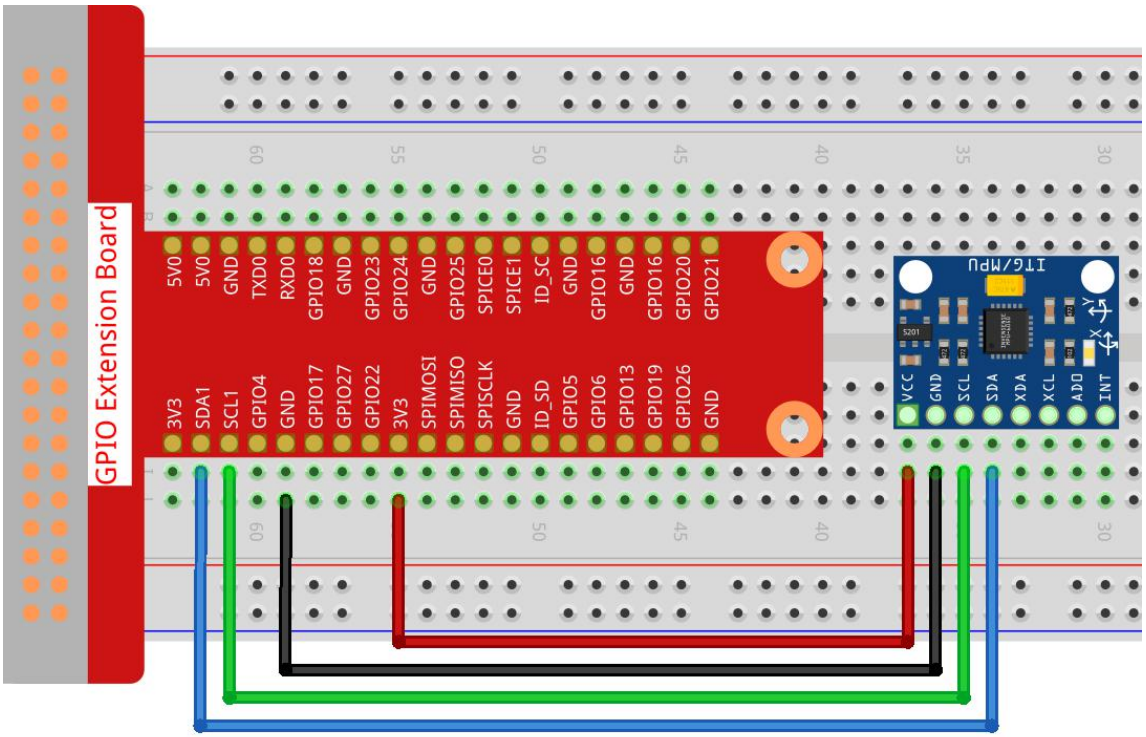
MPU6050 communicates with the microcontroller through the I2C bus interface. The SDA1 and SCL1 need to be connected to the corresponding pin.

T-Board Name	physical
SDA1	Pin 3
SCL1	Pin 5



Experimental Procedures

Step 1: Build the circuit.



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/2.2.6/
```

Step 4: Compile the code.

```
gcc 2.2.6_mpu6050.c -lwiringPi -lm
```

Step 5: Run the executable file.

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

With the code run, deflection angle of x axis, y axis and the acceleration, angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

Code

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

```
int fd;
int acclX, acclY, acclZ;
int gyroX, gyroY, gyroZ;
double acclX_scaled, acclY_scaled, acclZ_scaled;
double gyroX_scaled, gyroY_scaled, gyroZ_scaled;

int read_word_2c(int addr)
{
  int val;
  val = wiringPiI2CReadReg8(fd, addr);
  val = val << 8;
  val += wiringPiI2CReadReg8(fd, addr+1);
  if (val >= 0x8000)
    val = -(65536 - val);
  return val;
}

double dist(double a, double b)
{
  return sqrt((a*a) + (b*b));
}

double get_y_rotation(double x, double y, double z)
{
  double radians;
  radians = atan2(x, dist(y, z));
  return -(radians * (180.0 / M_PI));
}

double get_x_rotation(double x, double y, double z)
{
  double radians;
  radians = atan2(y, dist(x, z));
  return (radians * (180.0 / M_PI));
}

int main()
{
  fd = wiringPiI2CSetup (0x68);
  wiringPiI2CWriteReg8 (fd,0x6B,0x00);//disable sleep mode
```

```
    printf("set 0x6B=%X\n",wiringPiI2CReadReg8 (fd,0x6B));

    while(1) {

        gyroX = read_word_2c(0x43);
        gyroY = read_word_2c(0x45);
        gyroZ = read_word_2c(0x47);

        gyroX_scaled = gyroX / 131.0;
        gyroY_scaled = gyroY / 131.0;
        gyroZ_scaled = gyroZ / 131.0;

        //Print values for the X, Y, and Z axes of the gyroscope sensor.
        printf("My gyroX_scaled: %f\n", gyroY X_scaled);
        printf("My gyroY_scaled: %f\n", gyroY Y_scaled);
        printf("My gyroZ_scaled: %f\n", gyroY Z_scaled);

        acclX = read_word_2c(0x3B);
        acclY = read_word_2c(0x3D);
        acclZ = read_word_2c(0x3F);

        acclX_scaled = acclX / 16384.0;
        acclY_scaled = acclY / 16384.0;
        acclZ_scaled = acclZ / 16384.0;

        //Print the X, Y, and Z values of the acceleration sensor.
        printf("My acclX_scaled: %f\n", acclX_scaled);
        printf("My acclY_scaled: %f\n", acclY_scaled);
        printf("My acclZ_scaled: %f\n", acclZ_scaled);

        printf("My X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
        printf("My Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));

        delay(100);
    }
    return 0;
}
```

Code Explanation

```
int read_word_2c(int addr)
{
  int val;
  val = wiringPiI2CReadReg8(fd, addr);
  val = val << 8;
  val += wiringPiI2CReadReg8(fd, addr+1);
  if (val >= 0x8000)
    val = -(65536 - val);
  return val;
}
```

Read sensor data sent from MPU6050.

```
double get_y_rotation(double x, double y, double z)
{
  double radians;
  radians = atan2(x, dist(y, z));
  return -(radians * (180.0 / M_PI));
}
```

We get the deflection angle on the Y-axis.

```
double get_x_rotation(double x, double y, double z)
{
  double radians;
  radians = atan2(y, dist(x, z));
  return (radians * (180.0 / M_PI));
}
```

Calculate the deflection angle of the x-axis.

```
    gyroX = read_word_2c(0x43);
    gyroY = read_word_2c(0x45);
    gyroZ = read_word_2c(0x47);

    gyroX_scaled = gyroX / 131.0;
    gyroY_scaled = gyroY / 131.0;
    gyroZ_scaled = gyroZ / 131.0;

    //Print values for the X, Y, and Z axes of the gyroscope sensor.
```

```
    printf("My gyroX_scaled: %f\n", gyroY X_scaled);
    printf("My gyroY_scaled: %f\n", gyroY Y_scaled);
    printf("My gyroZ_scaled: %f\n", gyroY Z_scaled);
```

Read the values of the x axis, y axis and z axis on the gyroscope sensor, convert the metadata to angular velocity values, and then print them.

```
    acclX = read_word_2c(0x3B);
    acclY = read_word_2c(0x3D);
    acclZ = read_word_2c(0x3F);

    acclX_scaled = acclX / 16384.0;
    acclY_scaled = acclY / 16384.0;
    acclZ_scaled = acclZ / 16384.0;

    //Print the X, Y, and Z values of the acceleration sensor.
    printf("My acclX_scaled: %f\n", acclX_scaled);
    printf("My acclY_scaled: %f\n", acclY_scaled);
    printf("My acclZ_scaled: %f\n", acclZ_scaled);
```

Read the values of the x axis, y axis and z axis on the acceleration sensor, convert the metadata to accelerated speed values (gravity unit), and then print them.

```
    printf("My X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
    printf("My Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
```

Print the deflection angles of the x-axis and y-axis.

➢ 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 python3 2.2.6_mpu6050.py
```

With the code run, the angle of deflection of the x-axis and y-axis and the acceleration, angular velocity on each axis read by MPU6050 will be printed on the screen after being calculating.

Code

```
import smbus
import math
import time

# Power management registers
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)


bus = smbus.SMBus(1) # or bus = smbus.SMBus(1) for Revision 2 boards
address = 0x68       # This is the address value read via the i2cdetect command
```

```
# Now wake the 6050 up as it starts in sleep mode
bus.write_byte_data(address, power_mgmt_1, 0)

while True:
    time.sleep(0.1)
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))

    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
    accel_zout = read_word_2c(0x3f)

    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
    print ("accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled)
    print ("accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled)

    print ("x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
    print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))

    time.sleep(0.5)
```

Code Explanation

```
def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val
```

Read sensor data sent from MPU6050.

```
def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)
```

Calculate the deflection angle of the y-axis.

```
def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)
```

Calculate the deflection angle of the x-axis.

```
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print ("gyro_xout : ", gyro_xout, " scaled: ", (gyro_xout / 131))
    print ("gyro_yout : ", gyro_yout, " scaled: ", (gyro_yout / 131))
    print ("gyro_zout : ", gyro_zout, " scaled: ", (gyro_zout / 131))
```

Read the values of the x axis, y axis and z axis on the gyroscope sensor, convert the metadata to angular velocity values, and then print them.

```
    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
```

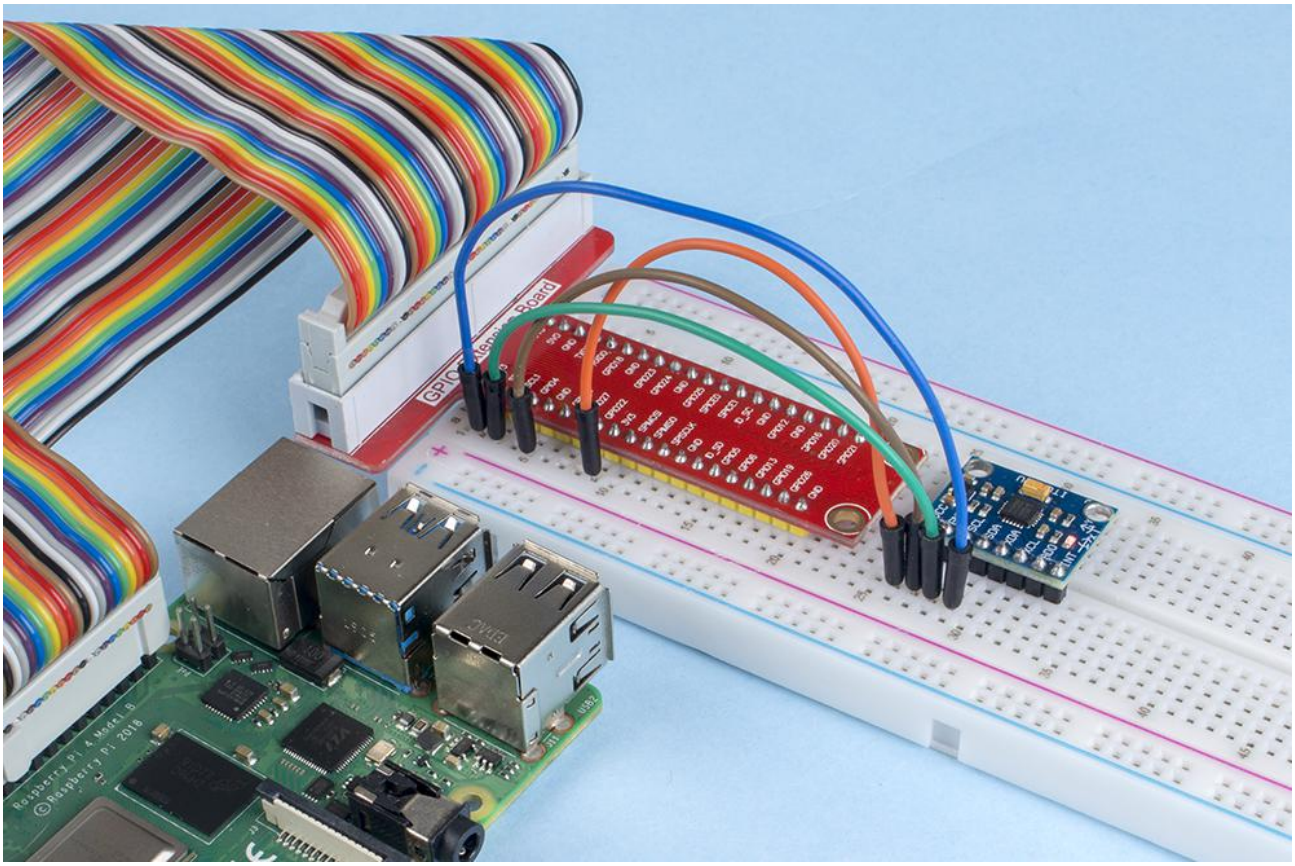
```
    accel_zout = read_word_2c(0x3f)

    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0

    print ("accel_xout: ", accel_xout, " scaled: ", accel_xout_scaled)
    print ("accel_yout: ", accel_yout, " scaled: ", accel_yout_scaled)
    print ("accel_zout: ", accel_zout, " scaled: ", accel_zout_scaled)
```

Read the values of the x axis, y axis and z axis on the acceleration sensor, convert the elements to accelerated speed value (gravity unit), and print them.

```
    print ("x rotation: " , get_x_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
    print ("y rotation: " , get_y_rotation(accel_xout_scaled, accel_yout_scaled,
accel_zout_scaled))
```

Print the deflection angles of the x-axis and y-axis.

Phenomenon Picture



2.2.7 MFRC522 RFID Module

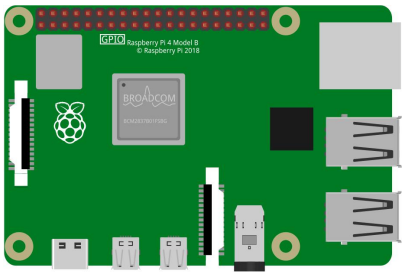
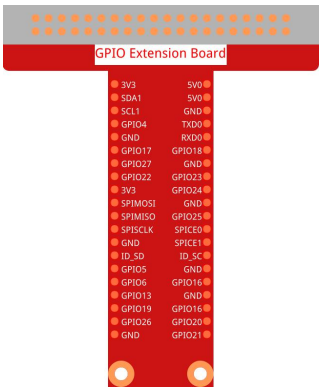
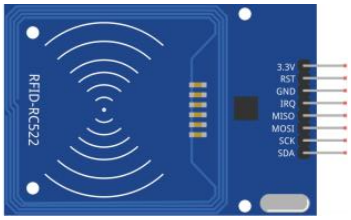
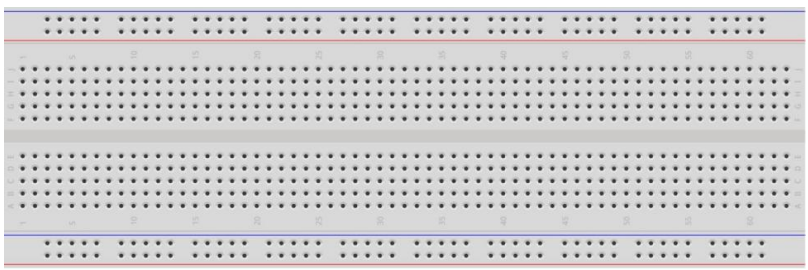
Introduction

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

<table>
<tr>
<td rowspan="2">1 * Raspberry Pi
  </td>
<td rowspan="2">1 * T-Extension Board
 
</td>
<td>1 * RFID RC522 (with white card and key tag)
 
</td>
</tr>
<tr>
<td>Several Jumper Wires
  </td>
</tr>
<tr>
<td colspan="3">1 * 40-pin Cable
  </td>
</tr>
<tr>
<td colspan="3">1 * Breadboard
  </td>
</tr>
</table>

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

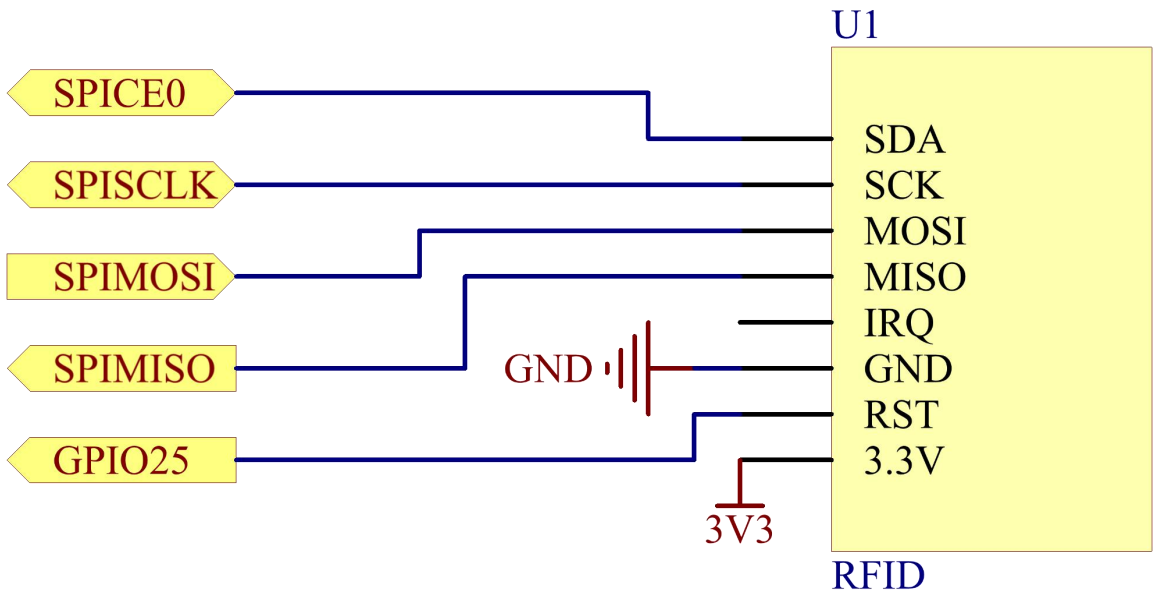
MFRC522 is a kind of integrated read and write card chip. It is commonly used in the radio at 13.56MHz. Launched by the NXP Company, it is a low-voltage, low-cost, and small-sized non-contact card chip, a best choice of intelligent instrument and portable handheld device.

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.

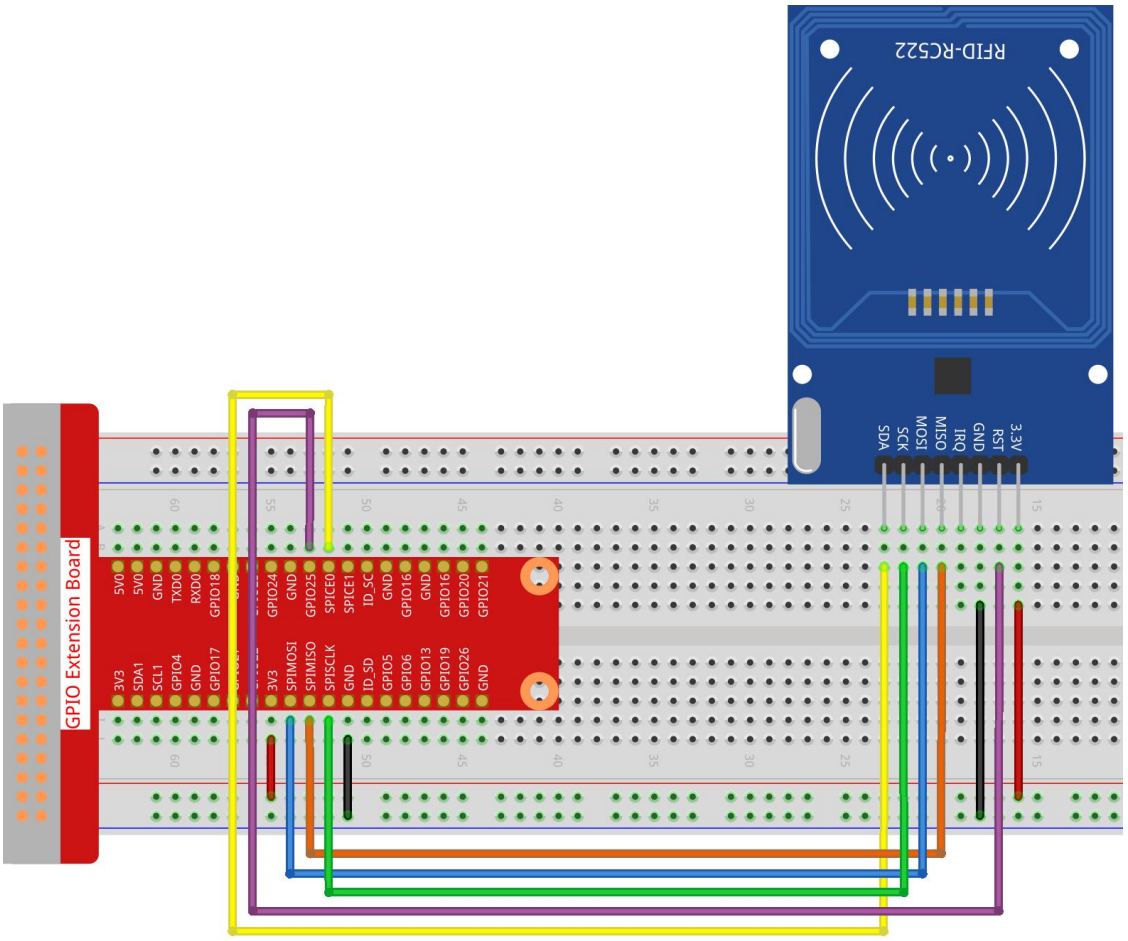
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
SPICE0	Pin 24	10	8
SPISCLK	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.

```
make read
make write
```

Note: There are two examples for you to read or write the card ID, and you can choose one of them according to your need.

Step 5: Run the executable file.

```
sudo ./read
sudo ./write
```

Code Explanation

```
InitRc522();
```

This function is used to initialize the RFID RC522 module.

```
uint8_t read_card_data();
```

This function is used to read the data of the card, and if the read is successful, it will return "1".

```
uint8_t write_card_data(uint8_t *data);
```

This function is used to write the data of card and returns "1" if the write is successful. *data is the information that will be written to the card.

➢ 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 python3 2.2.7_read.py
sudo python3 2.2.7_write.py
```

Note: There are two examples for you to read or write the card ID, and you can choose one of them according to your need.

Code Explanation

```
RC522()
```

Instantiate rc522 class.

```
RC522.Pcd_start()
```

Initialize RFID

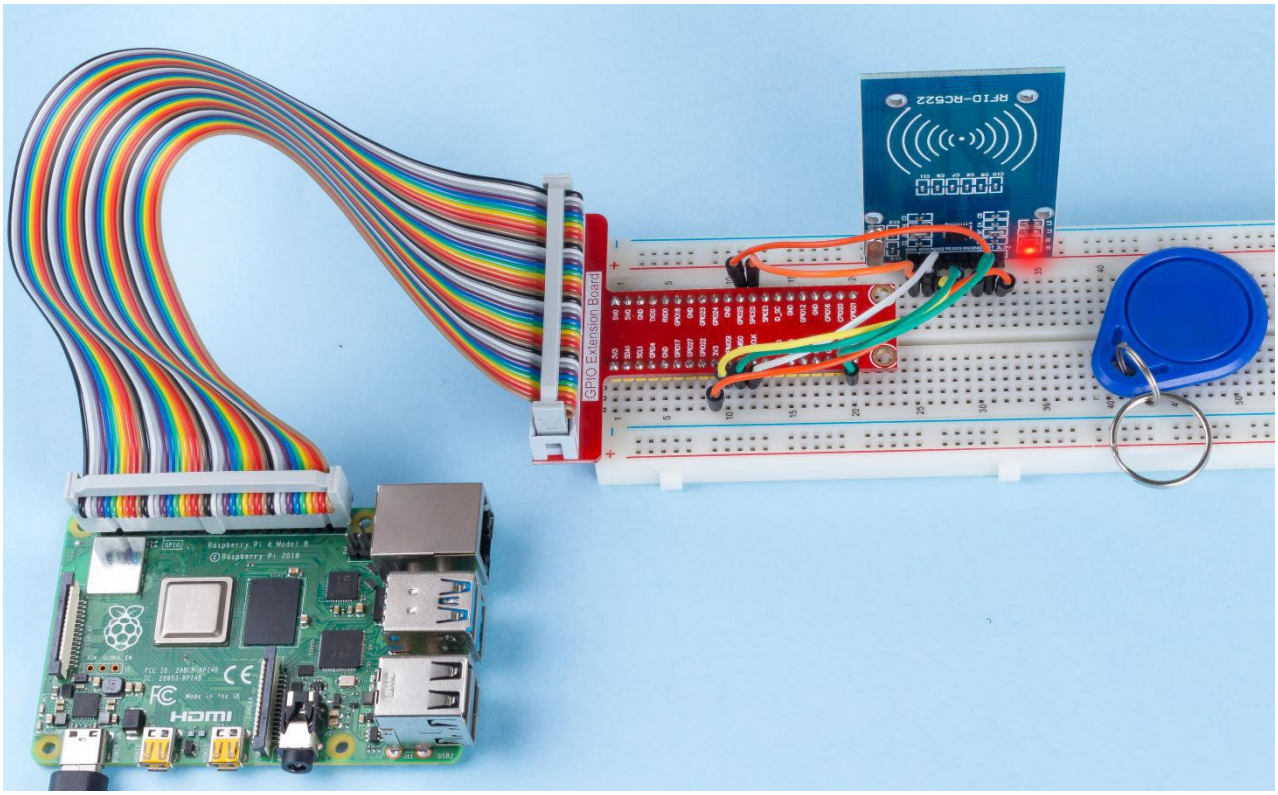
```
RC522.read_card_data(addr)
```

This function is used to read the card data. If the reading is successful, it will return "1". addr is the address of the card.

```
RC522.write_card_data(addr, data)
```

This function is used to write card data, and if the write is successful, there will return "1". addr is the address of the card and data is the information to be written to the card.

Phenomenon Picture



3 Extension

This chapter shows some very funny expansion experiments. The following points need to be noted:

1) The code and wiring will be much more complicated, and you need more patience to complete these experiments.

2) The complete code won't be given on the document, so you can go to the code folder to see the complete code.

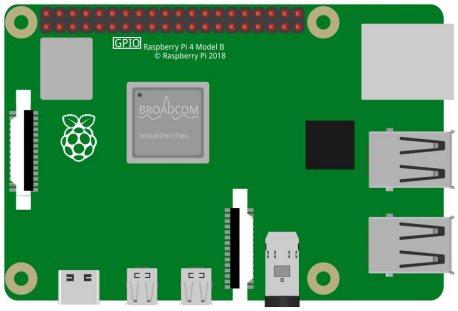
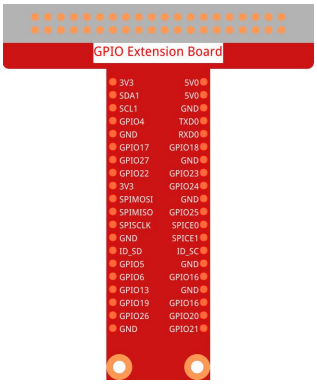
3.1 Application

3.1.1 Counting Device

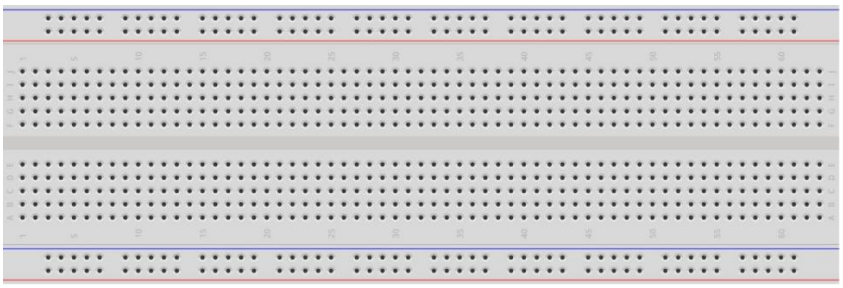
Introduction

Here we will make a number-displaying counter system, consisting of a PIR sensor and a 4-digit segment display. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1. You can use this counter to count the number of people walking through the passageway.

Components

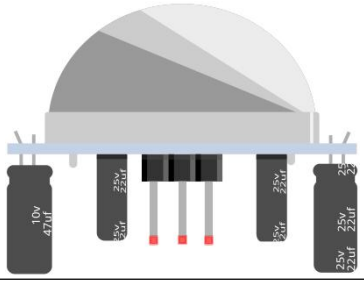
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * 4-Digit 7-Segment Display
  </td></tr>
<tr><td colspan="2"></td><td>4 * Resistor(220Ω)
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable</td><td>1 * 74HC595
  </td></tr>
</table>

1 * Breadboard



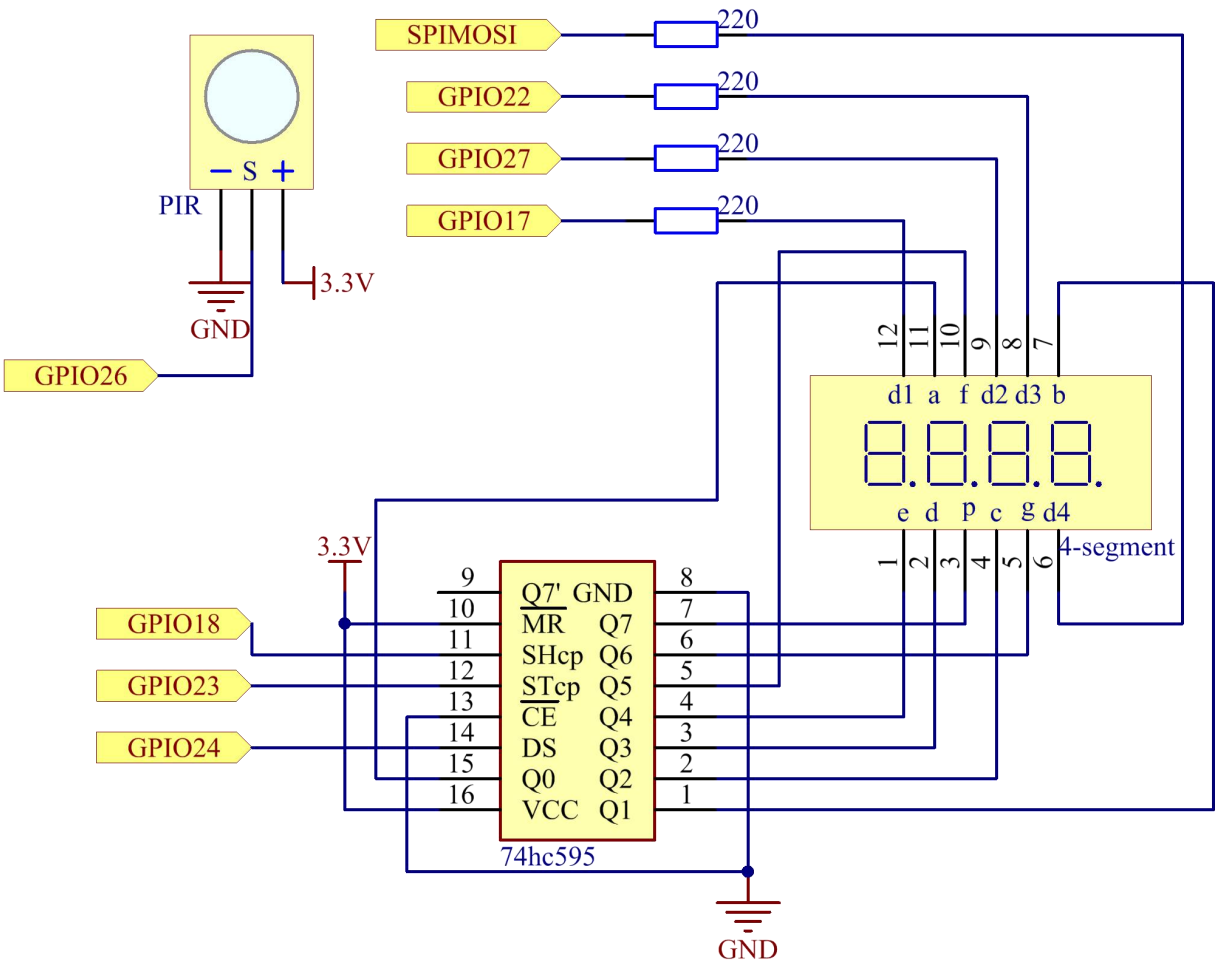
Several Jumper Wires

1 * PIR Sensor Module



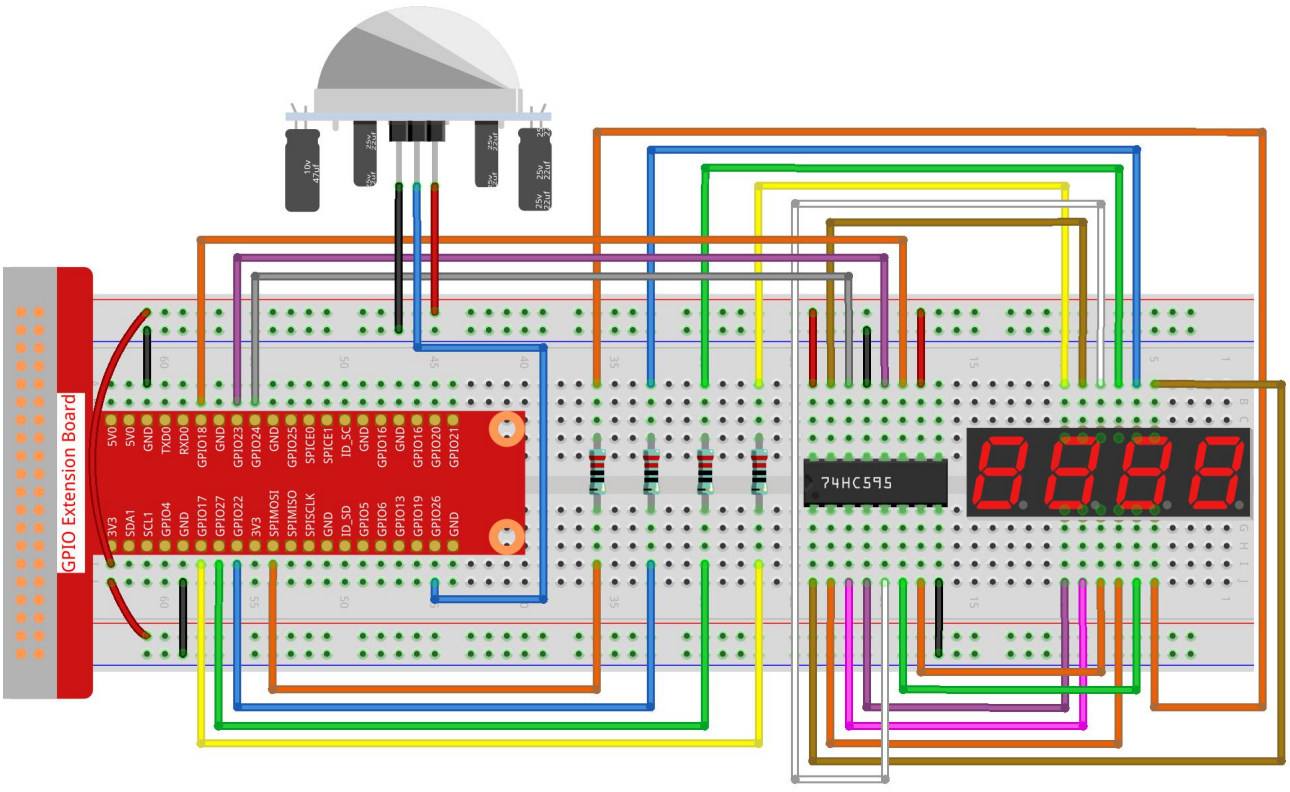
Schematic Diagram

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
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO26	Pin 37	25	26



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/3.1.1/
```

Step 3: Compile the code.

```
gcc 3.1.1_CountingDevice.c -lwiringPi
```

Step 4: Run the executable file.

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

After the code runs, when the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

Code Explanation

```
void display()
{
    clearDisplay();
    pickDigit(0);
    hc595_shift(number[counter % 10]);

    clearDisplay();
    pickDigit(1);
    hc595_shift(number[counter % 100 / 10]);

    clearDisplay();
    pickDigit(2);
    hc595_shift(number[counter % 1000 / 100]);

    clearDisplay();
    pickDigit(3);
    hc595_shift(number[counter % 10000 / 1000]);
}
```

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
void loop(){
    int currentState =0;
```

```
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            counter +=1;
        }
        lastState=currentState;
    }
}
```

This is the main function: display the number on the 4-digit segment display and read the PIR value. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

➢ 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 python3 3.1.1_CountingDevice.py
```

After the code runs, when the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

Code Explanation

Based on **1.1.5 4-Digit 7-Segment Display**, this lesson adds **PIR module** to change the automatic counting of lesson 1.1.5 into count detecting. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

```
def display():
    global counter
    clearDisplay()
    pickDigit(0)
    hc595_shift(number[counter % 10])

    clearDisplay()
    pickDigit(1)
    hc595_shift(number[counter % 100//10])

    clearDisplay()
```

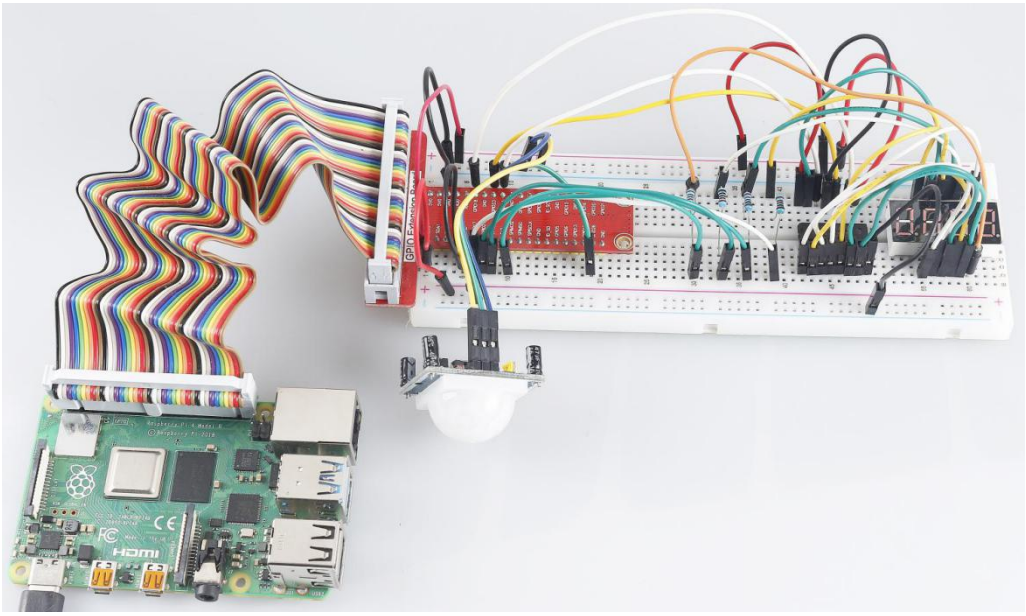
```
    pickDigit(2)
    hc595_shift(number[counter % 1000//100])

    clearDisplay()
    pickDigit(3)
    hc595_shift(number[counter % 10000//1000])
```

First, start the fourth segment display, write the single-digit number. Then start the third segment display, and type in the tens digit; after that, start the second and the first segment display respectively, and write the hundreds and thousands digits respectively. Because the refreshing speed is very fast, we see a complete four-digit display.

```
def loop():
global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            counter +=1
        lastState=currentState
```

This is the main function: display the number on the 4-digit segment display and read the PIR value. When the PIR detects that someone is passing by, the number on the 4-digit segment display will add 1.

Phenomenon Picture

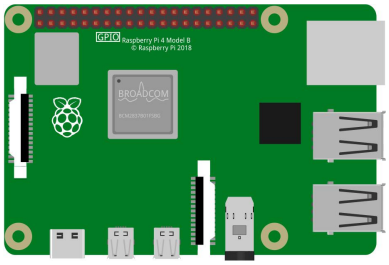
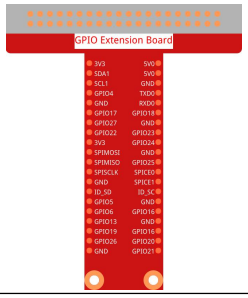
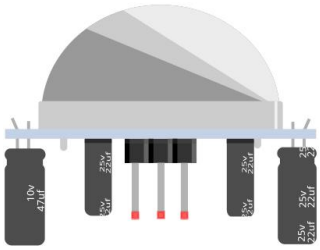
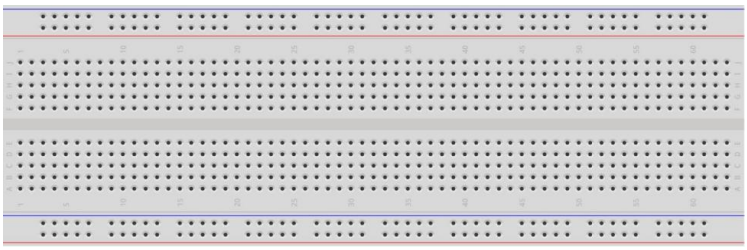


3.1.2 Welcome

Introduction

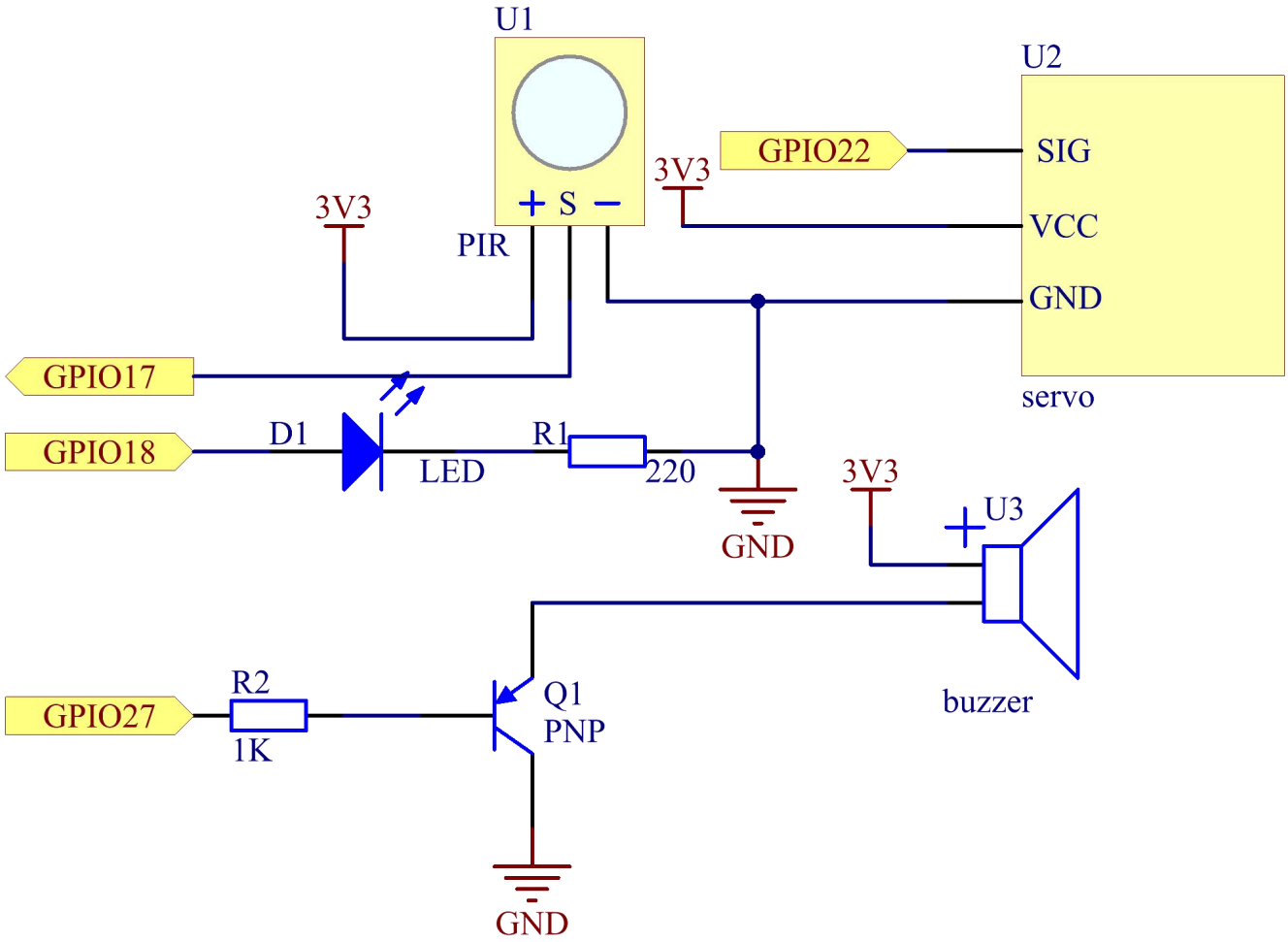
In this project, we will use PIR to sense the movement of pedestrians, and use servos, LED, buzzer to simulate the work of the sensor door of the convenience store. When the pedestrian appears within the sensing range of the PIR, the indicator light will be on, the door will be opened, and the buzzer will play the opening bell.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td colspan="2">1 * PIR
  </td></tr>
<tr><td>1 * Servo
  </td><td>1 * Passive Buzzer
  </td><td>1 * S8550 PNP Transistor
  </td><td>1 * LED
  </td></tr>
<tr><td>1 * Resistor 1kΩ
  </td><td>1 * Resistor 220Ω
  </td><td colspan="2">Several Jumper Wires
  </td></tr>
<tr><td colspan="4">1 * 40-pin Cable
  </td></tr>
<tr><td colspan="4">1 * Breadboard
  </td></tr>
</table>

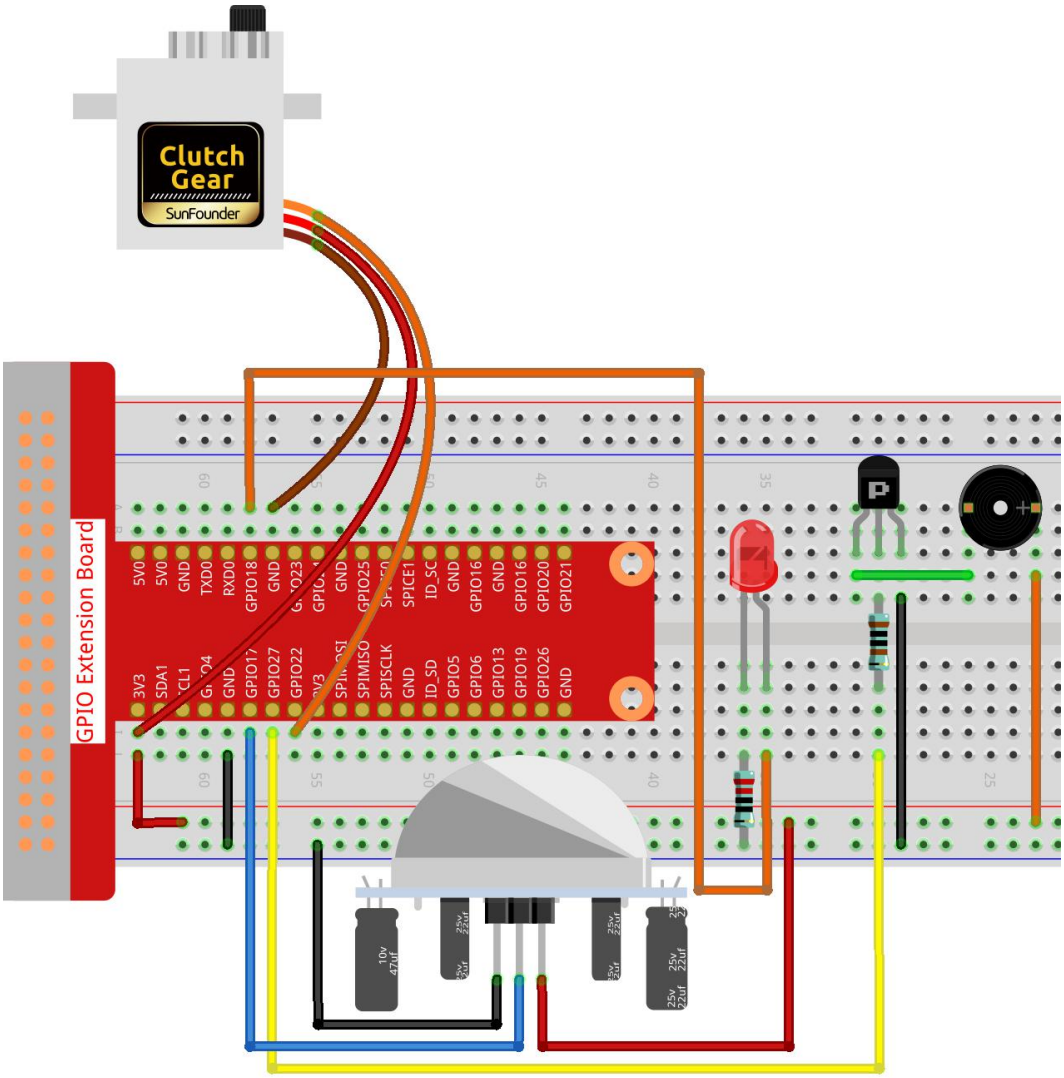
Schematic Diagram

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/3.1.2/
```

Step 3: Compile.

```
gcc 3.1.2_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 Explanation

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

Create a function, setAngle 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
        setAngle(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
        setAngle(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 python3 3.1.2_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 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(pirPin, GPIO.IN)     # Set sensorPin's mode is input
    GPIO.setup(buzPin, GPIO.OUT)    # Set pins' mode is output
    Buzz = GPIO.PWM(buzPin, 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 Frequece to 50Hz
    p.start(0)                              # Duty Cycle = 0
```

These statements are used to initialize the pins of each component.

```
def setAngle(angle):        # make the servo rotate to specific angle (0-180 degrees)
      angle = max(0, min(180, angle))
      pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
      pwm = map(pulse_width, 0, 20000, 0, 100)
      p.ChangeDutyCycle(pwm)#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)
    Buzz.ChangeFrequency(1)
    for i in range(180, -1, -1): #make servo rotate from 180 to 0 deg
          setAngle(i)
          time.sleep(0.001)
```

Close the door and turn off the indicator light.

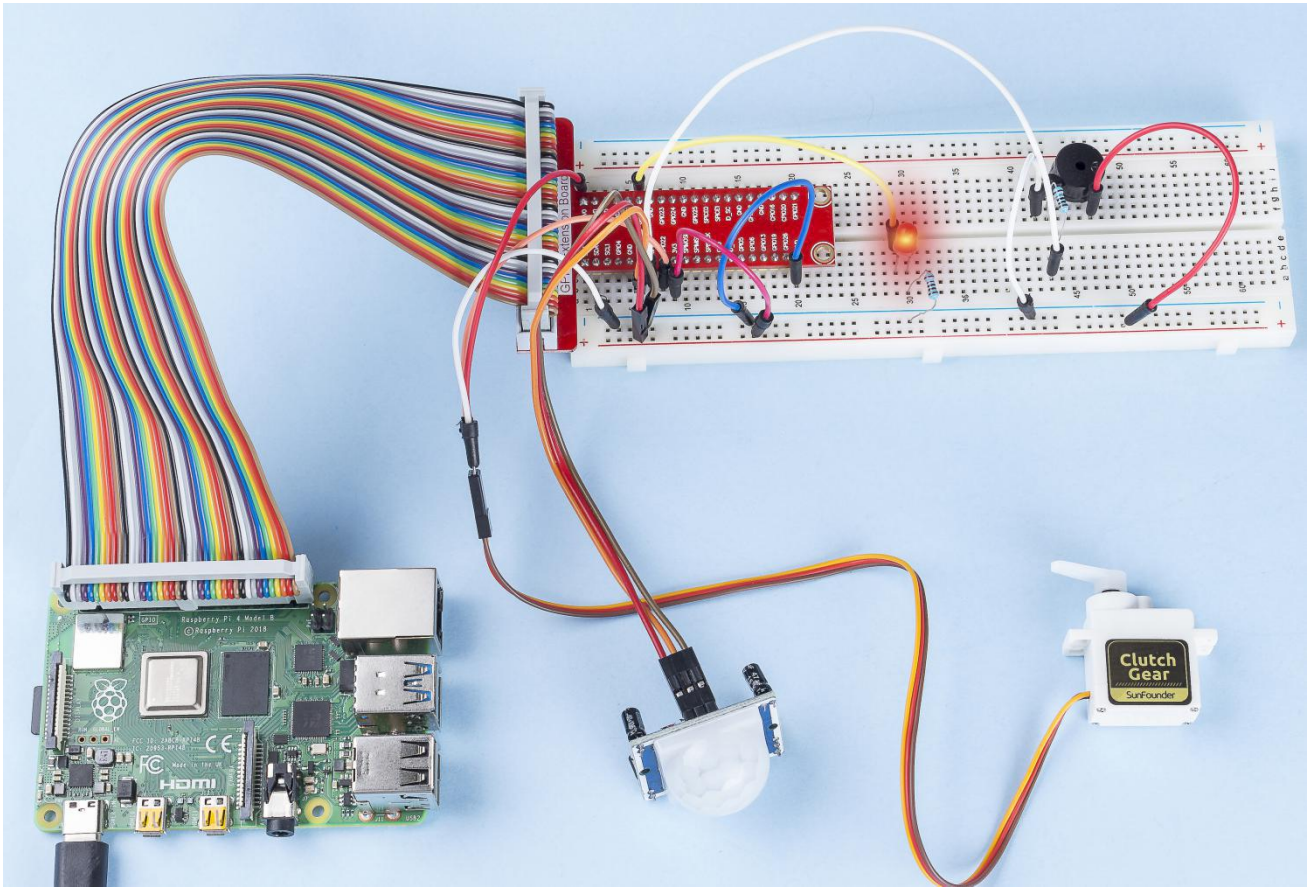
```
def opendoor():
    GPIO.output(ledPin, GPIO.LOW)
    for i in range(0, 181, 1):    #make servo rotate from 0 to 180 deg
          setAngle(i)       # Write to servo
          time.sleep(0.001)
    doorbell()
    closedoor()
```

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

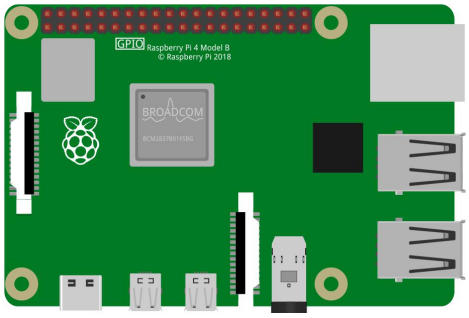
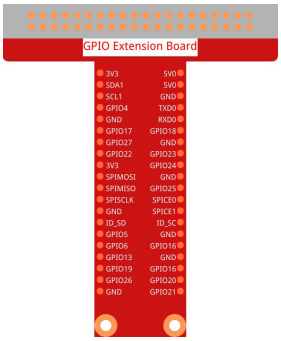
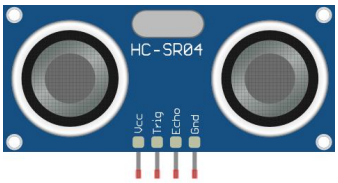
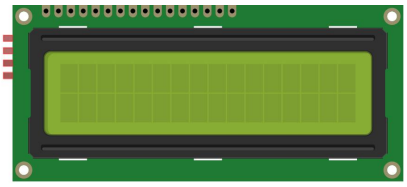


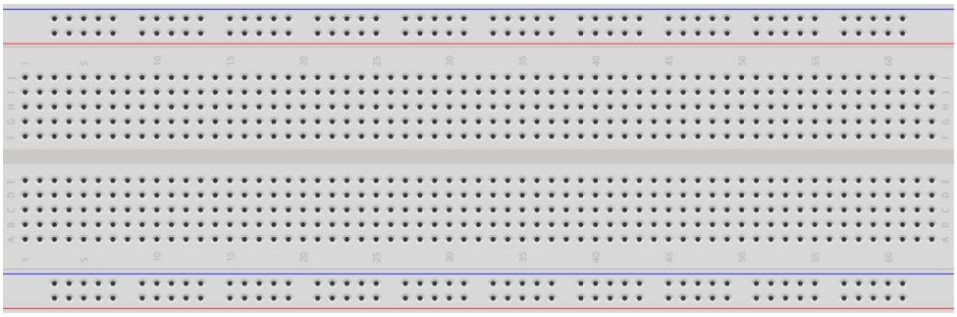
3.1.3 Reversing Alarm

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

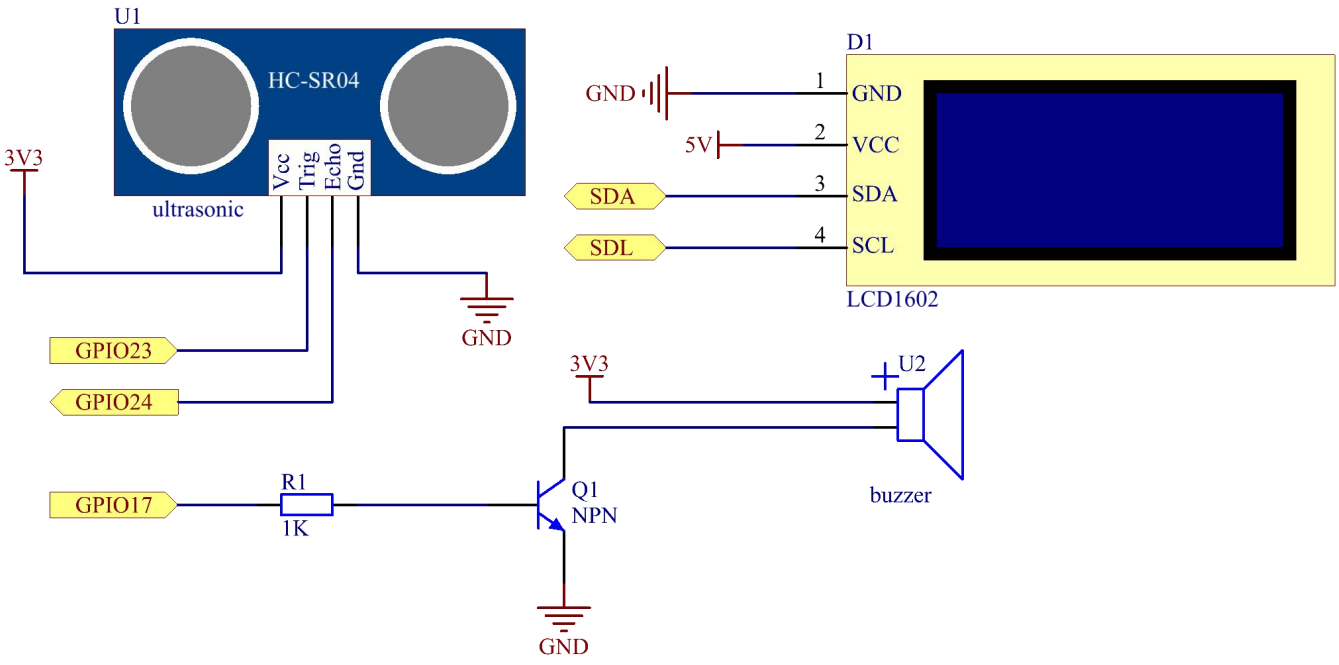
<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Active Buzzer
 
1 * S8050 NPN Transistor
  </td></tr>
<tr><td>1 * HC SR04
  </td><td>1 * I2C LCD1602
  </td><td>Several Jumper Wires

1 * Resistor(1kΩ)
  </td></tr>
<tr><td colspan="3">1 * 40-pin Cable
  </td></tr>
<tr><td colspan="3">1 * Breadboard
  </td></tr>
</table>

Schematic Diagram

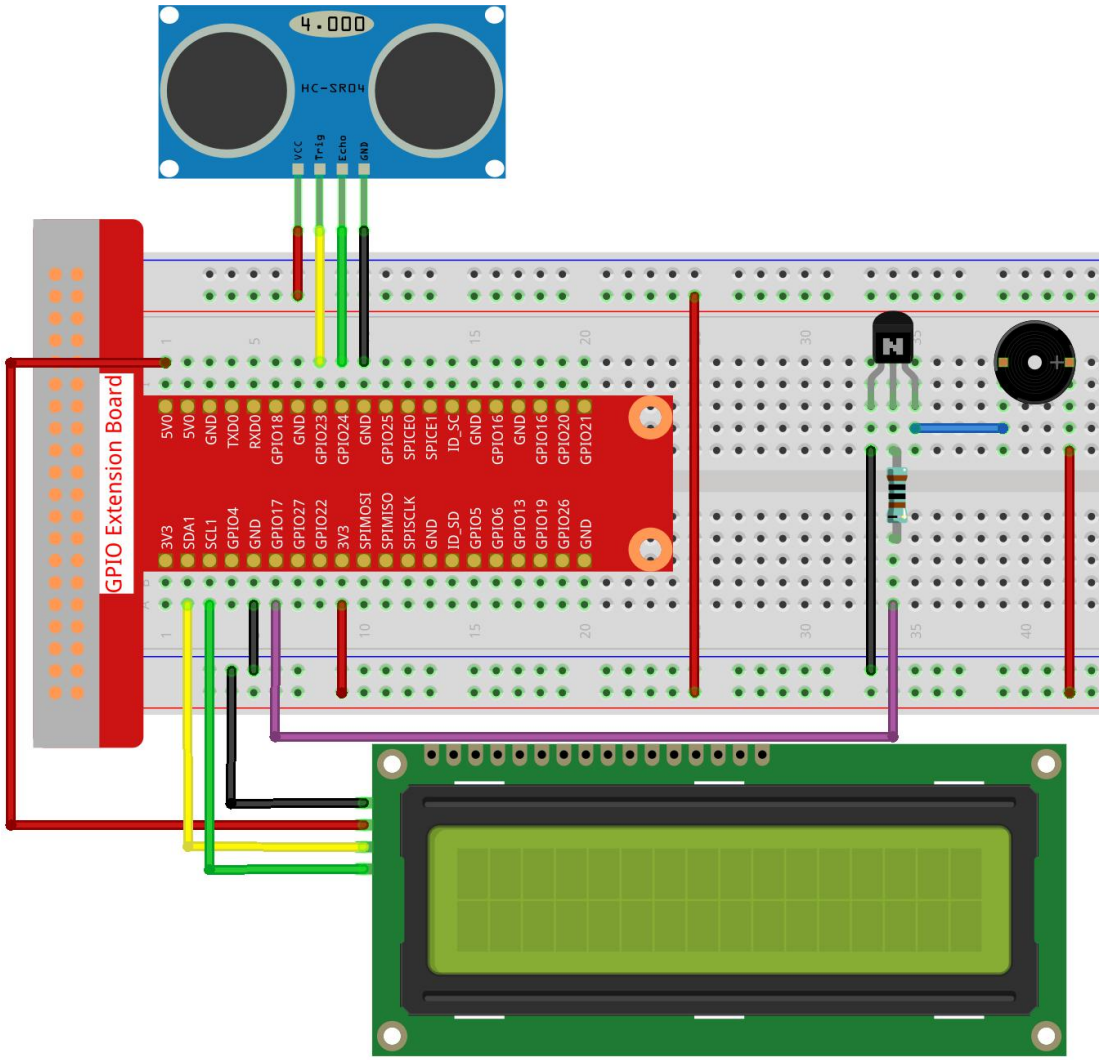
Ultrasonic sensor detects the distance between itself and the obstacle that will be displayed on the LCD in the form of code. At the same time, the ultrasonic sensor let the buzzer issue prompt sound of different frequency according to different distance value.

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/3.1.3/
```

Step 3: Compile.

```
gcc 3.1.3_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 ultraInit(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 python3 3.1.3_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.

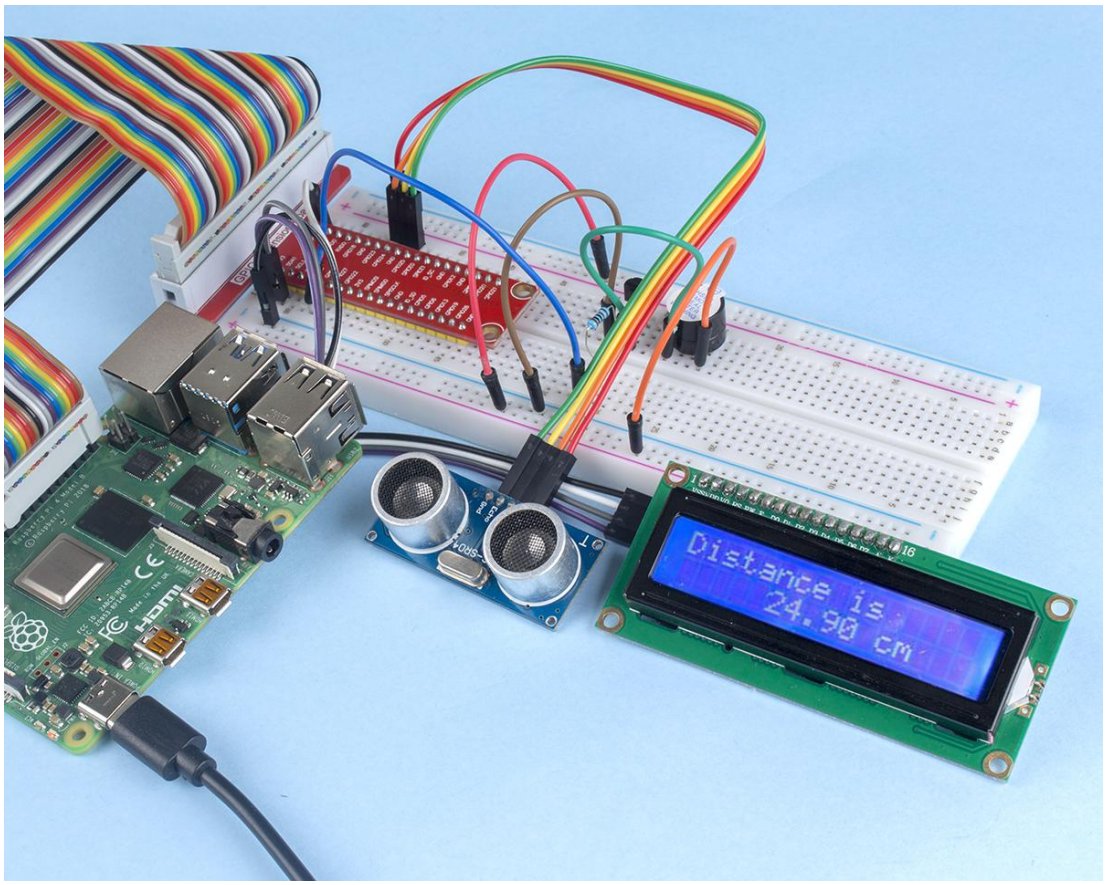
LCD1602.write(5, 1, str(round(dis,2)) +' cm')

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

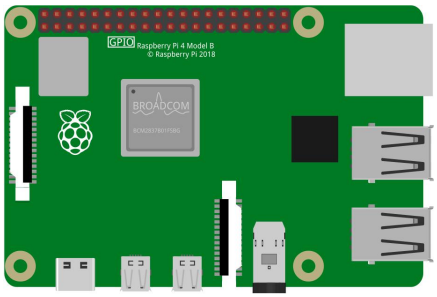
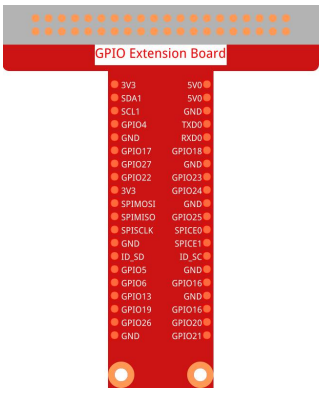
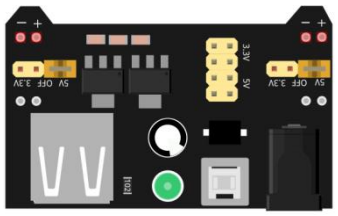
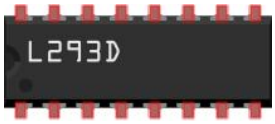
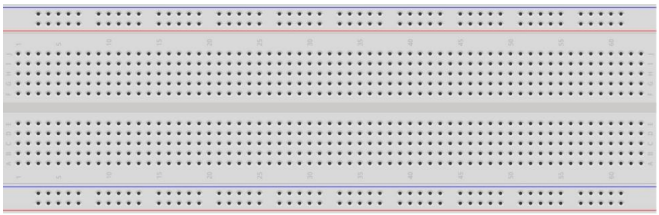


3.1.4 Smart Fan

Introduction

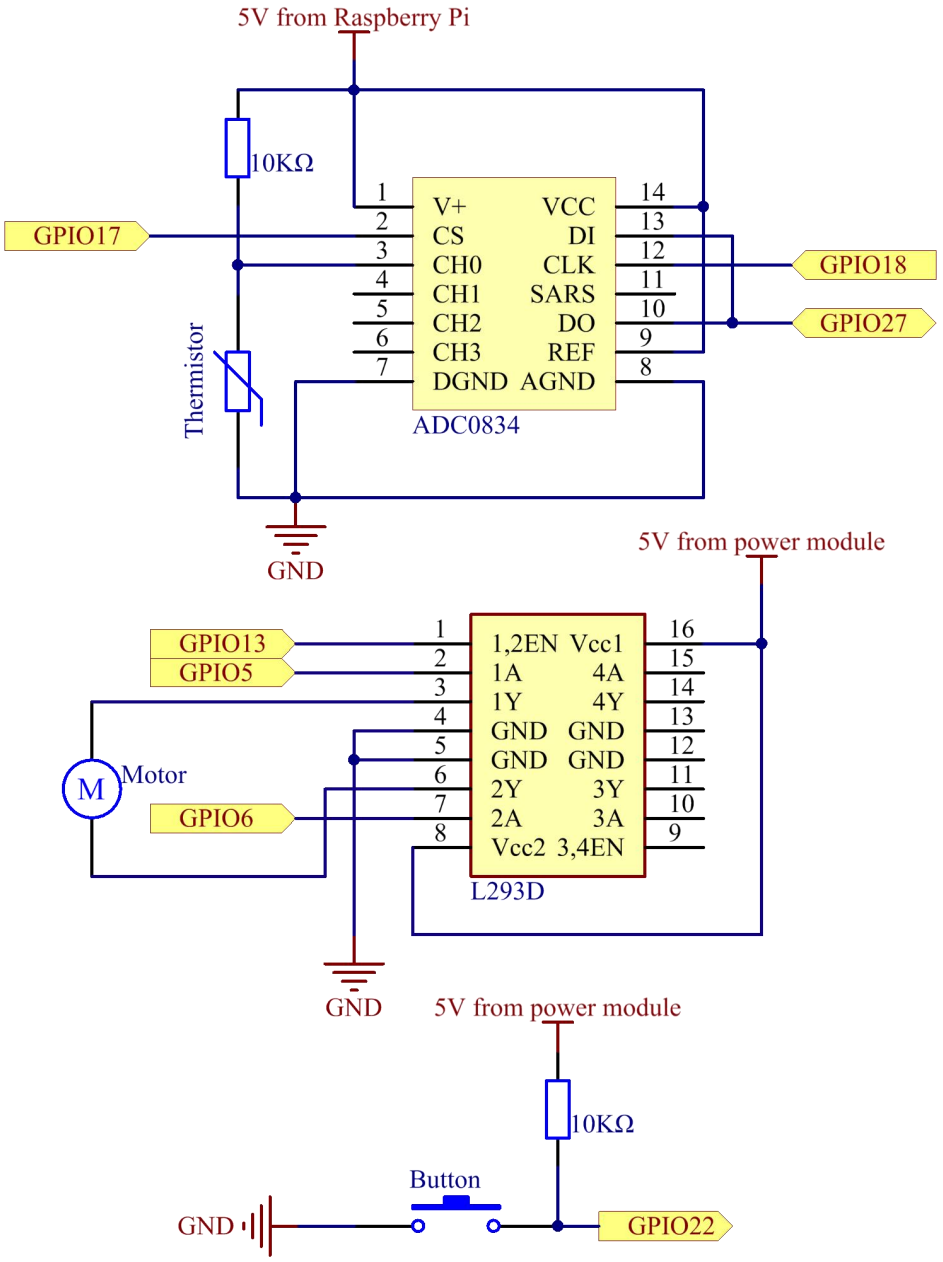
In this course, we will use motors, buttons and thermistors to make a manual + automatic smart fan whose wind speed is adjustable.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Power Module
 
2 * Resistor 10KΩ
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>1 * Thermistor
  </td><td>1 * L293D
 
1 * ADC0834
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * Button
  </td><td>1 * DC Motor
 
Several Jumper Wires
</td></tr>
</table>

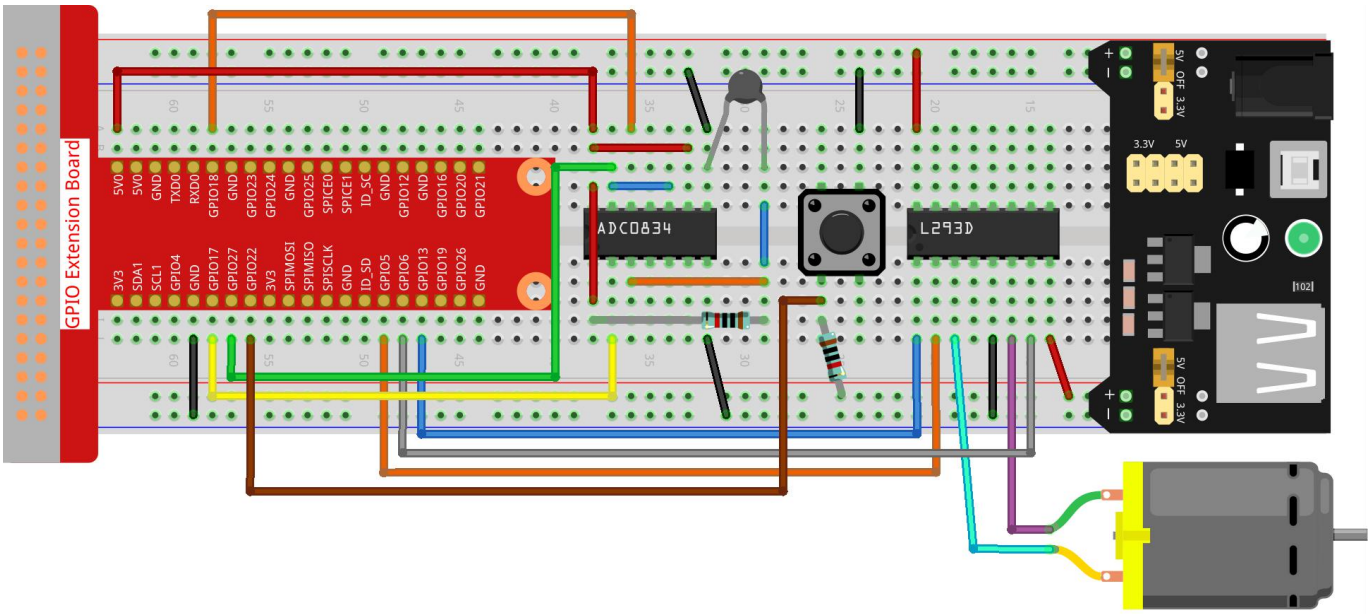
Schematic Diagram

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
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13



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/3.1.4/
```

Step 3: Compile.

```
gcc 3.1.4_SmartFan.c -lwiringPi -lm
```

Step 4: Run the executable file above.

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

As the code runs, start the fan by pressing the button. Every time you press, 1 speed grade is adjusted up or down. There are **5** kinds of speed grades: **0~4**. When set to the 4th speed grade and you press the button, the fan stops working with a **0** wind speed.

Once the temperature goes up or down for more than 2℃, the speed automatically gets 1-grade faster or slower.

Code Explanation

```
int temperture(){
    unsigned char analogVal;
    double Vr, Rt, temp, cel, Fah;
    analogVal = get_ADC_Result(0);
    Vr = 5 * (double)(analogVal) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    int t=cel;
    return t;
}
```

Temperture() works by converting thermistor values read by ADC0834 into temperature values. Refer to **2.2.2 Thermistor** for more details.

```
int motor(int level){
    if(level==0){
        digitalWrite(MotorEnable,LOW);
        return 0;
    }
    if (level>=4){
        level =4;
    }
    digitalWrite(MotorEnable,HIGH);
    softPwmWrite(MotorPin1, level*25);
    return level;
}
```

This function controls the rotating speed of the motor. The range of the **Level**: **0-4** (level **0** stops the working motor). One level adjustment stands for a **25%** change of the wind speed.

```
int main(void)
{
    setup();
    int currentState,lastState=0;
    int level = 0;
    int currentTemp,markTemp=0;
    while(1){
        currentState=digitalRead(BtnPin);
```

```
        currentTemp=temperture();
        if (currentTemp<=0){continue;}
        if (currentState==1&&lastState==0){
            level=(level+1)%5;
            markTemp=currentTemp;
            delay(500);
        }
        lastState=currentState;
        if (level!=0){
            if (currentTemp-markTemp<=-2){
                level=level-1;
                markTemp=currentTemp;
            }
            if (currentTemp-markTemp>=2){
                level=level+1;
                markTemp=currentTemp;
            }
        }
        level=motor(level);
    }
    return 0;
}
```

The function **main()** contains the whole program process as shown:

1) Constantly read the button state and the current temperature.

2) Every press makes level**+1** and at the same time, the temperature is updated. The **Level** ranges **1~4**.

3) As the fan works (the level is **not 0**), the temperature is under detection. A **2℃**$^{+}$ change causes the up and down of the level.

4) The motor changes the rotating speed with the **Level**.

➢ 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 python3 3.1.4_SmartFan.py
```

As the code runs, start the fan by pressing the button. Every time you press, 1 speed grade is adjusted up or down. There are **5** kinds of speed grades: **0~4**. When set to the 4th speed grade and you press the button, the fan stops working with a **0** wind speed.

Once the temperature goes up or down for more than 2℃, the speed automatically gets 1-grade faster or slower.

Code Explanation

```
def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return Cel
```

temperture() works by converting thermistor values read by **ADC0834** into temperature values. Refer to **2.2.2 Thermistor** for more details.

```
def motor(level):
    if level == 0:
        GPIO.output(MotorEnable, GPIO.LOW)
        return 0
    if level>=4:
        level = 4
    GPIO.output(MotorEnable, GPIO.HIGH)
    p_M1.ChangeDutyCycle(level*25)
    return level
```

This function controls the rotating speed of the motor. The range of the **Lever**: **0-4** (level **0** stops the working motor). One level adjustment stands for a **25%** change of the wind speed.

```
def main():
    lastState=0
    level=0
    markTemp = temperature()
    while True:
        currentState =GPIO.input(BtnPin)
        currentTemp=temperature()
        if currentState == 1 and lastState == 0:
```

```
            level=(level+1)%5
            markTemp = currentTemp
            time.sleep(0.5)
        lastState=currentState
        if level!=0:
            if currentTemp-markTemp <= -2:
                level = level -1
                markTemp=currentTemp
            if currentTemp-markTemp >= 2:
                level = level +1
                markTemp=currentTemp
        level = motor(level)
```

The function **main()** contains the whole program process as shown:

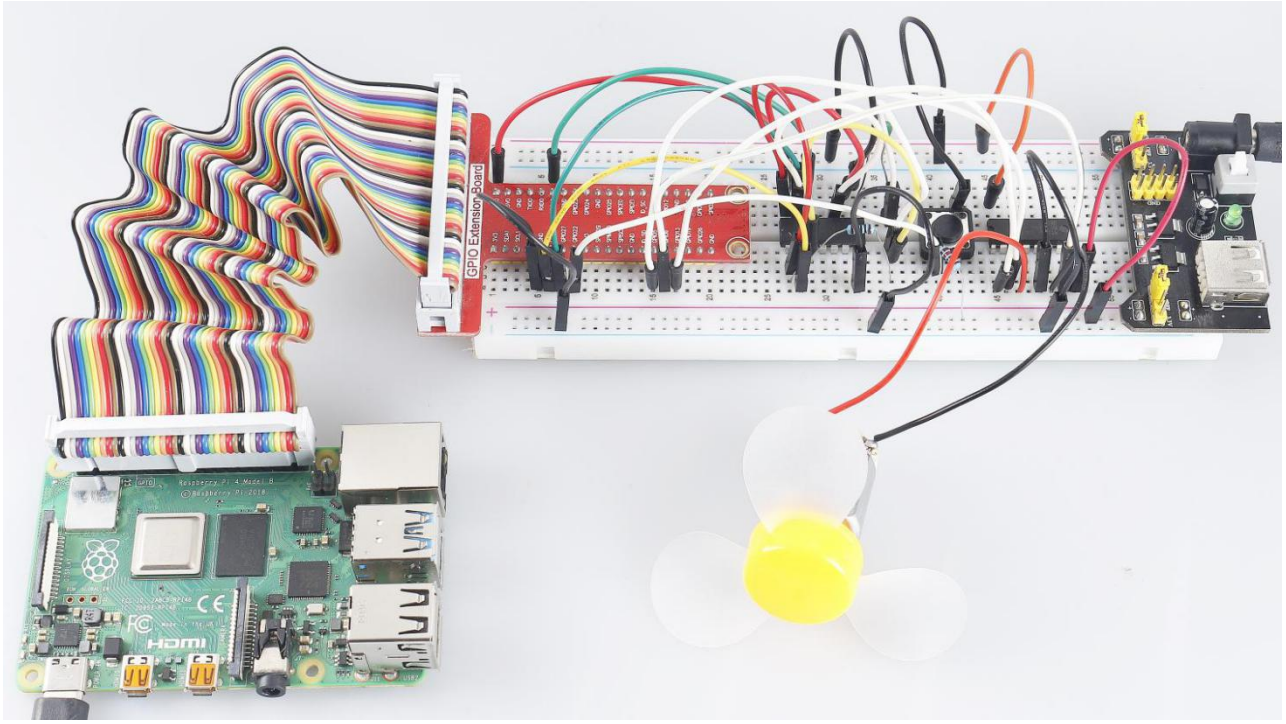
1) Constantly read the button state and the current temperature.

2) Every press makes level**+1** and at the same time, the temperature is updated. The **Level** ranges **1~4**.

3) As the fan works (the level is **not 0**), the temperature is under detection. A **2℃**⁺ change causes the up and down of the level.

4) The motor changes the rotating speed with the **Level**.

Phenomenon Picture



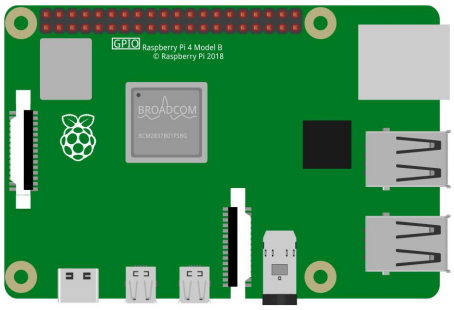
3.1.5 Battery Indicator

Introduction

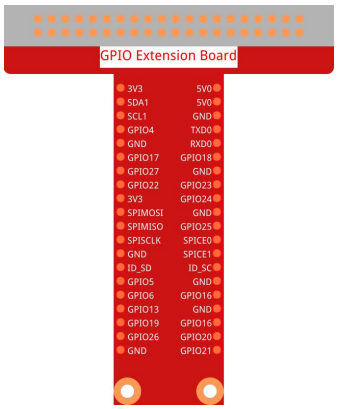
In this course, we will make a battery indicator device that can visually display the battery level on the LED Bargraph.

Components

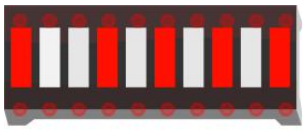
1 * Raspberry Pi



1 * T-Extension Board



1 * LED Bargraph



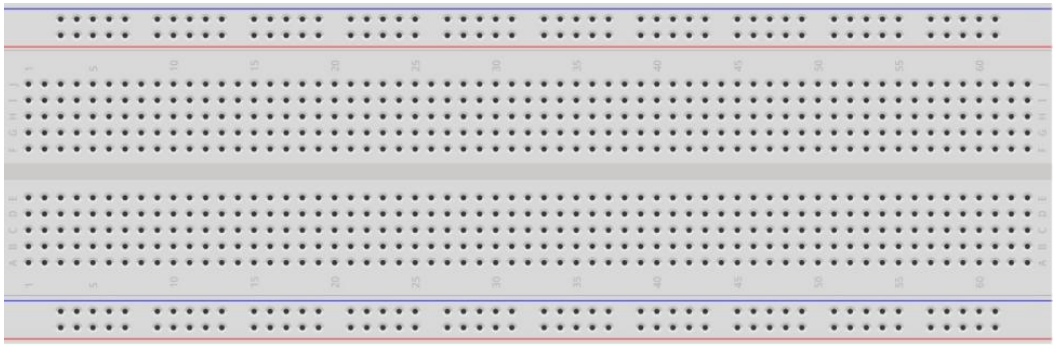
1 * ADC0834

10 * Resistor(220Ω)

1 * 40-pin Cable

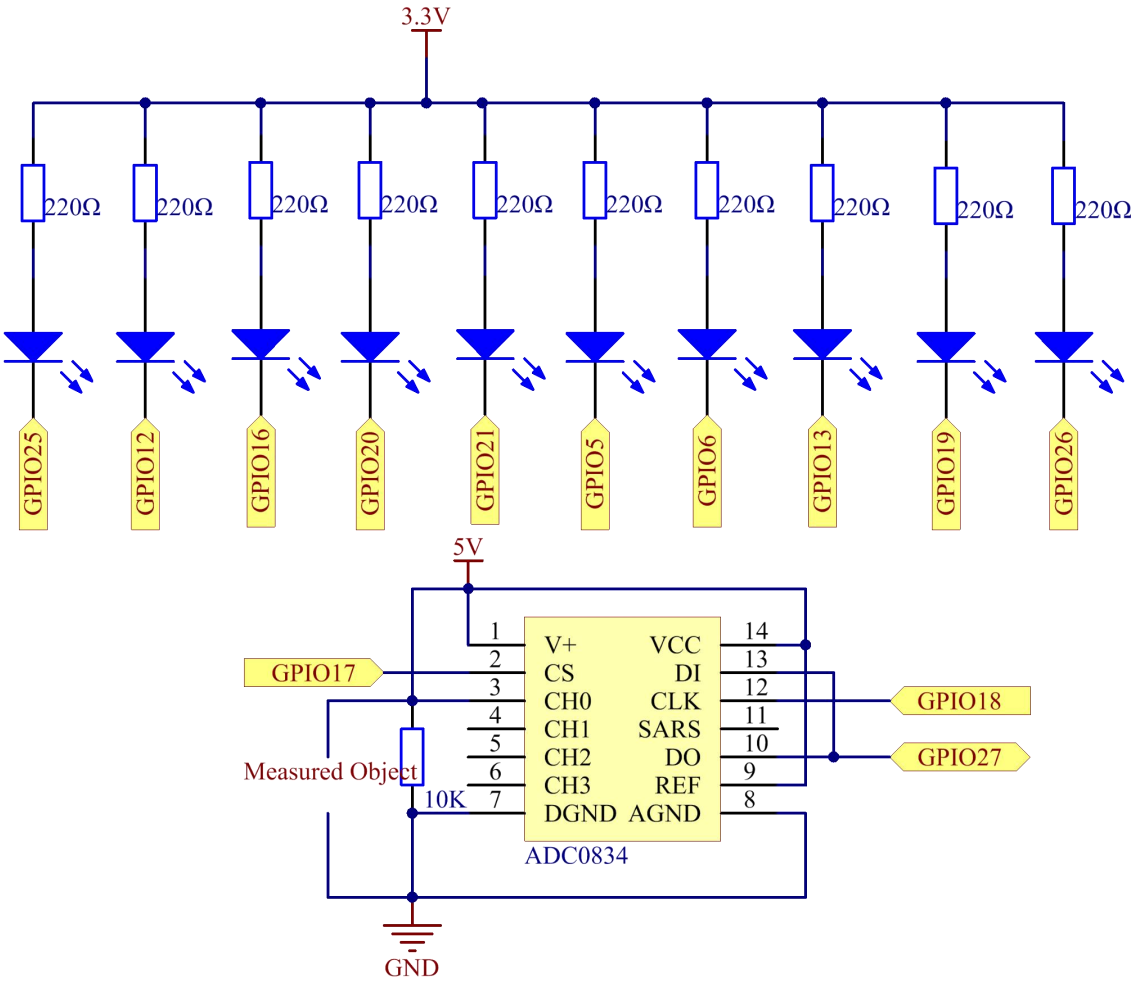
Several Jumper Wires

1 * Breadboard



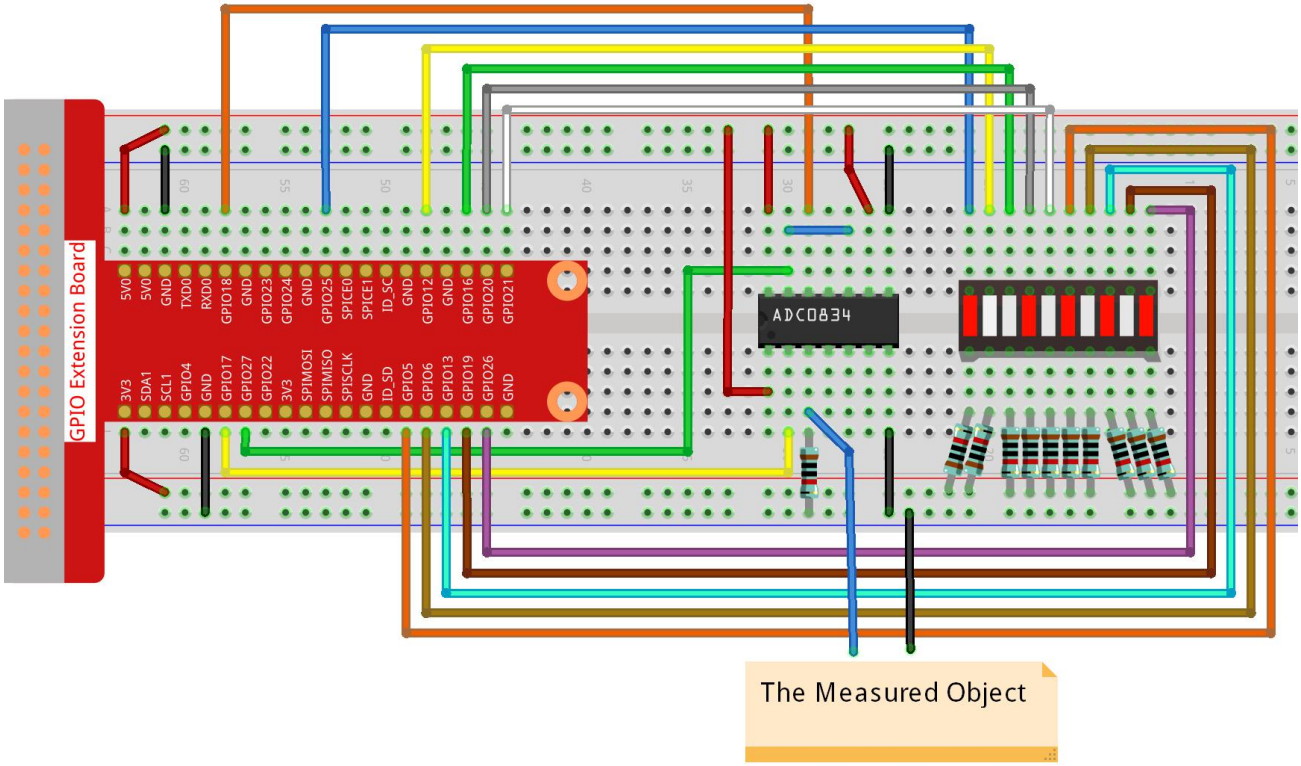
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO25	Pin 22	6	25
GPIO12	Pin 32	26	12
GPIO16	Pin 36	27	16
GPIO20	Pin 38	28	20
GPIO21	Pin 40	29	21
GPIO5	Pin 29	21	5
GPIO6	Pin 31	22	6
GPIO13	Pin 33	23	13
GPIO19	Pin 35	24	19
GPIO26	Pin 37	25	26



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/3.1.5/
```

Step 3: Compile the code.

```
gcc 3.1.5_BatteryIndicator.c -lwiringPi
```

Step 4: Run the executable file.

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

After the program runs, give the 3rd pin of ADC0834 and the GND a lead-out wire separately and then lead them to the two poles of a battery separately. You can see the corresponding LED on the LED Bargraph is lit up to display the power level (measuring range: 0-5V).

Code Explanation

```
void LedBarGraph(int value){
    for(int i=0;i<10;i++){
        digitalWrite(pins[i],HIGH);
    }
    for(int i=0;i<value;i++){
```

```
        digitalWrite(pins[i],LOW);
    }
}
```

This function works for controlling the turning on or off of the 10 LEDs on the LED Bargraph. We give these 10 LEDs high levels to let they are off at first, then decide how many LEDs are lit up by changing the received analog value.

```
int main(void)
{
    uchar analogVal;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(ADC_CS,  OUTPUT);
    pinMode(ADC_CLK, OUTPUT);
    for(int i=0;i<10;i++){      //make led pins' mode is output
        pinMode(pins[i], OUTPUT);
        digitalWrite(pins[i],HIGH);
    }
    while(1){
        analogVal = get_ADC_Result(0);
        LedBarGraph(analogVal/25);
        delay(100);
    }
    return 0;
}
```

analogVal produces values (**0-255**) with varying voltage values (**0-5V**), ex., if a 3V is detected on a battery, the corresponding value **152** is displayed on the voltmeter.

The **10** LEDs on the LED Bargraph are used to display the **analogVal** readings. 255/10=25, so every **25** the analog value increases, one more LED turns on, ex., if "analogVal=150 (about 3V), there are 6 LEDs turning on."

➢ 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 python3 3.1.5_BatteryIndicator.py
```

After the program runs, give the 3rd pin of ADC0834 and the GND a lead-out wire separately and then lead them to the two poles of a battery separately. You can see the corresponding LED on the LED Bargraph is lit up to display the power level (measuring range: 0-5V).

Code Explanation

```
def LedBarGraph(value):
    for i in ledPins:
        GPIO.output(i,GPIO.HIGH)
    for i in range(value):
        GPIO.output(ledPins[i],GPIO.LOW)
```

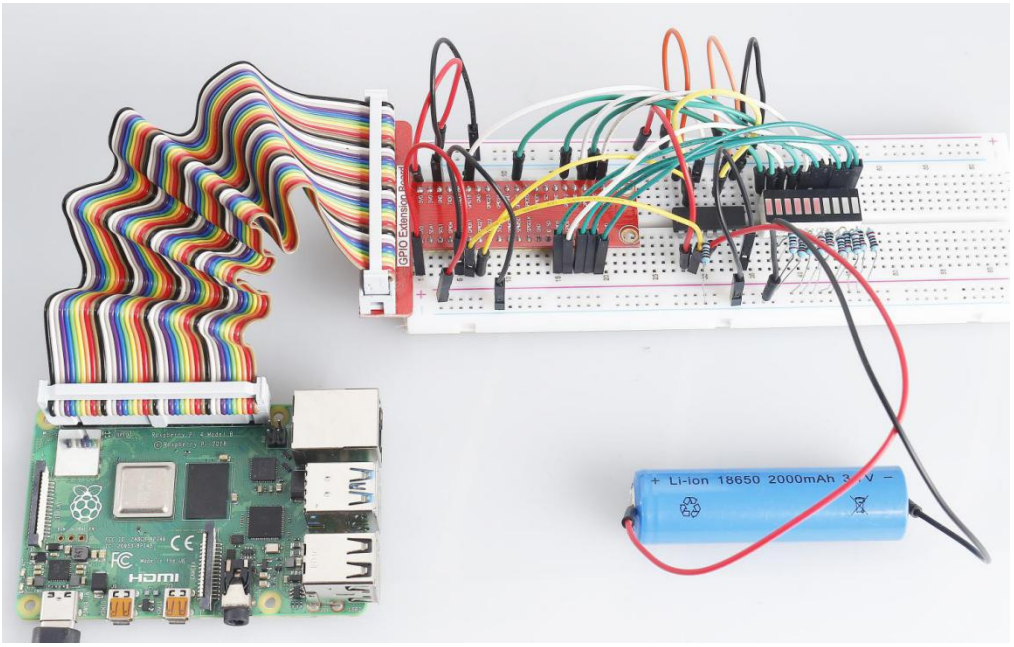
This function works for controlling the turning on or off of the **10** LEDs on the LED Bargraph. We give these **10** LEDs high levels to let they are **off** at first, then decide how many LEDs are lit up by changing the received analog value.

```
def loop():
    while True:
        analogVal = ADC0834.getResult()
        LedBarGraph(int(analogVal/25))
```

analogVal produces values (**0-255**) with varying voltage values (**0-5V**), ex., if a 3V is detected on a battery, the corresponding value **152** is displayed on the voltmeter.

The **10** LEDs on the LED Bargraph are used to display the **analogVal** readings. 255/10=25, so every **25** the analog value increases, one more LED turns on, ex., if "analogVal=150 (about 3V), there are 6 LEDs turning on."

Phenomenon Picture

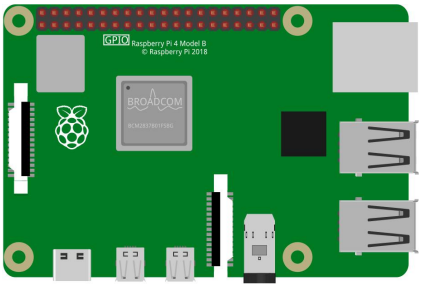
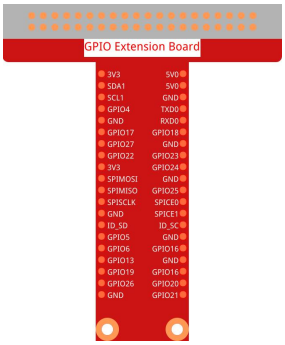
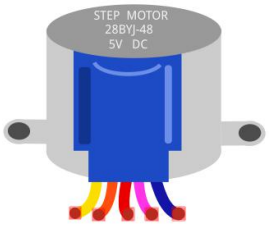
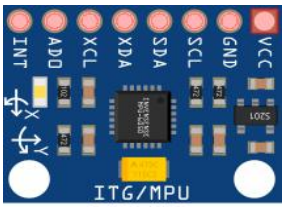
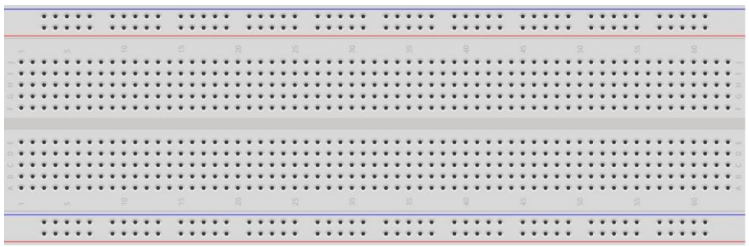
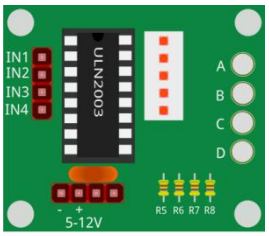


3.1.6 Motion Control

Introduction

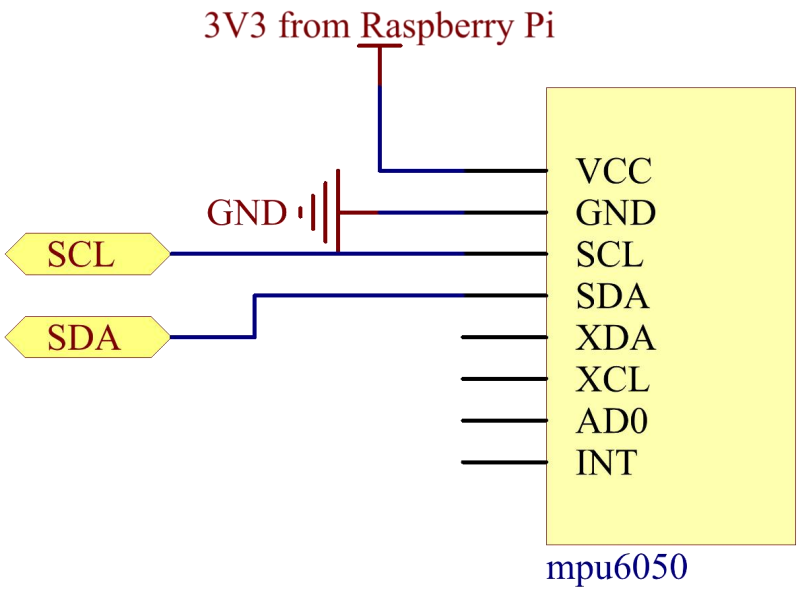
In this lesson, we will make a simple motion sensing and controlling device. The MPU6050 is used as a sensor and the stepper motor as a controlled device. With the MPU6050 mounted on the glove, you can control the stepper motor by rotating your wrist.

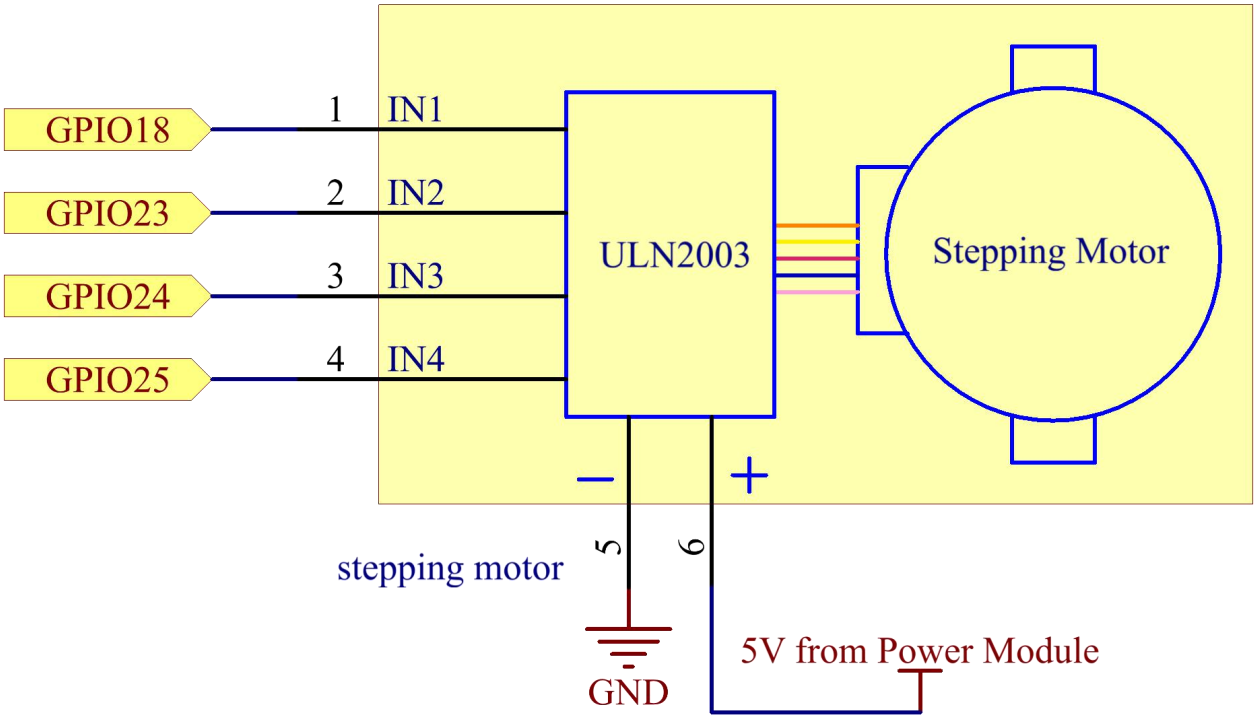
Components

1 * Raspberry Pi 	1 * T-Extension Board 	1 * Stepper Motor 
1 * 40-pin Cable 		Several Jumper Wires 
		1 * MPU6050 Module 
1 * Breadboard 		1 * ULN2003 Module 

Schematic Diagram

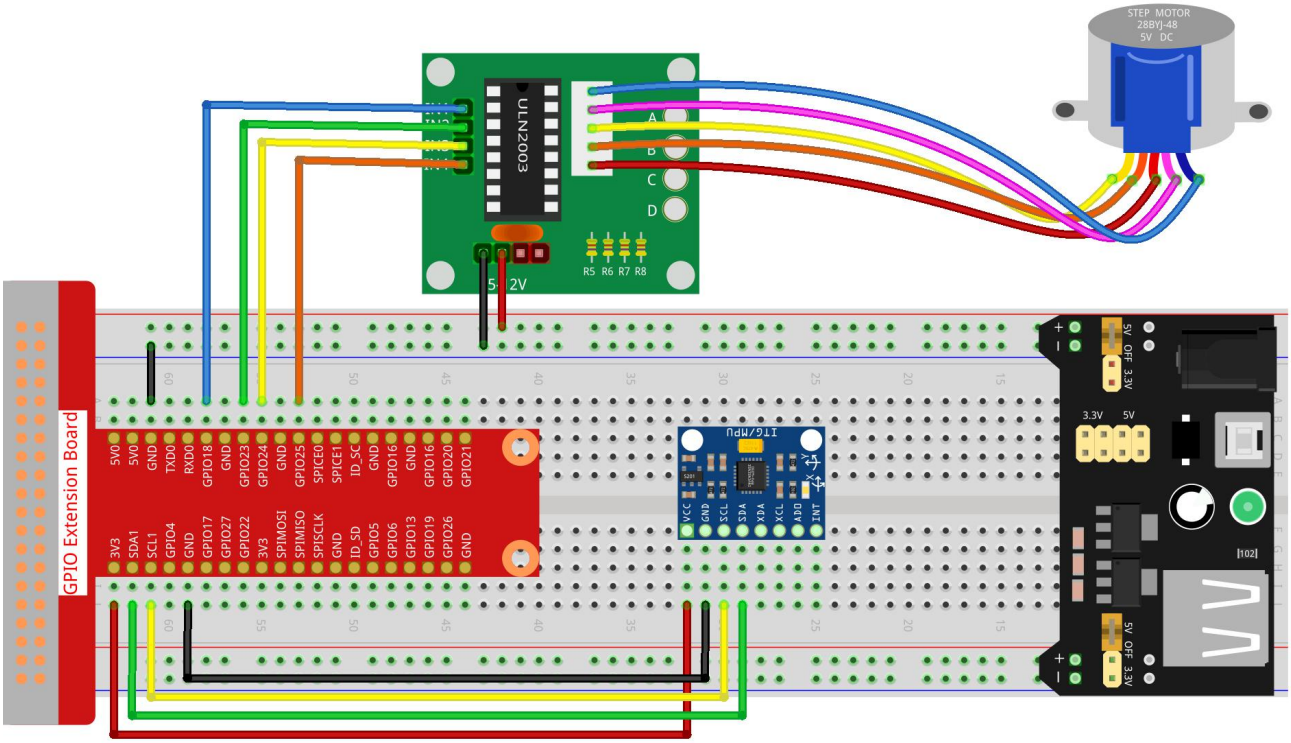
T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SDA1	Pin 3		
SCL1	Pin 5		





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/3.1.6/
```

Step 3: Compile the code.

```
gcc 3.1.6_MotionControl.c -lwiringPi
```

Step 4: Run the executable file.

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

As the code runs, if the tilt angle of **mpu6050** on the Y-axis is larger than **45** ℃, the stepper motor rotates anticlockwise; if less than **-45** ℃, the stepper motor rotates clockwise.

Code Explanation

```
double mpu6050(){
    acclX = read_word_2c(0x3B);
    acclY = read_word_2c(0x3D);
    acclZ = read_word_2c(0x3F);
    acclX_scaled = acclX / 16384.0;
    acclY_scaled = acclY / 16384.0;
```

```
    acclZ_scaled = acclZ / 16384.0;
    double angle=get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled);
    return angle;
}
```

mpu6050 gets the tilt angle in the direction of the Y-axis.

```
void rotary(char direction){
    if(direction == 'c'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99>>j & (0x08>>i));}
            delayMicroseconds(stepSpeed);
        }
    }
    else if(direction =='a'){
        for(int j=0;j<4;j++){
            for(int i=0;i<4;i++)
                {digitalWrite(motorPin[i],0x99<<j & (0x80>>i));}
            delayMicroseconds(stepSpeed);
        }
    }
}
```

If the received direction **key** is '**c**', the stepper motor rotates clockwise; if the **key** is '**a**', the motor rotates anticlockwise. Refer to **1.3.3 Stepper Motor** for more details about the calculation of the rotating direction of the stepper motor.

```
int main()
{
    setup();
    double angle;
    while(1) {
        angle = mpu6050();
        if (angle >=45){rotary('a');}
        else if (angle<=-45){rotary('c');}
    }
    return 0;
}
```

The tilt angle in the direction of the Y-axis is read from **mpu6050**, and if it's larger than **45**℃, the stepper motor rotates anticlockwise; if less than **-45** ℃, the stepper motor rotates clockwise.

➢ 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 python3 3.1.6_MotionControl.py
```

As the code runs, if the tilt angle of **mpu6050** on the **Y-axis** is larger than **45** ℃, the stepper motor rotates anticlockwise; if less than **-45** ℃, the stepper motor rotates clockwise.

Code Explanation

```
def mpu6050():
    accel_xout = read_word_2c(0x3b)
    accel_yout = read_word_2c(0x3d)
    accel_zout = read_word_2c(0x3f)
    accel_xout_scaled = accel_xout / 16384.0
    accel_yout_scaled = accel_yout / 16384.0
    accel_zout_scaled = accel_zout / 16384.0
    angle=get_y_rotation(accel_xout_scaled, accel_yout_scaled, accel_zout_scaled)
    return angle
```

mpu6050 gets the tilt angle in the direction of the **Y-axis**.

```
def rotary(direction):
    if(direction == 'c'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99>>j & (0x08>>i))
            time.sleep(stepSpeed)

    elif(direction == 'a'):
        for j in range(4):
            for i in range(4):
                GPIO.output(motorPin[i],0x99<<j & (0x80>>i))
            time.sleep(stepSpeed)
```

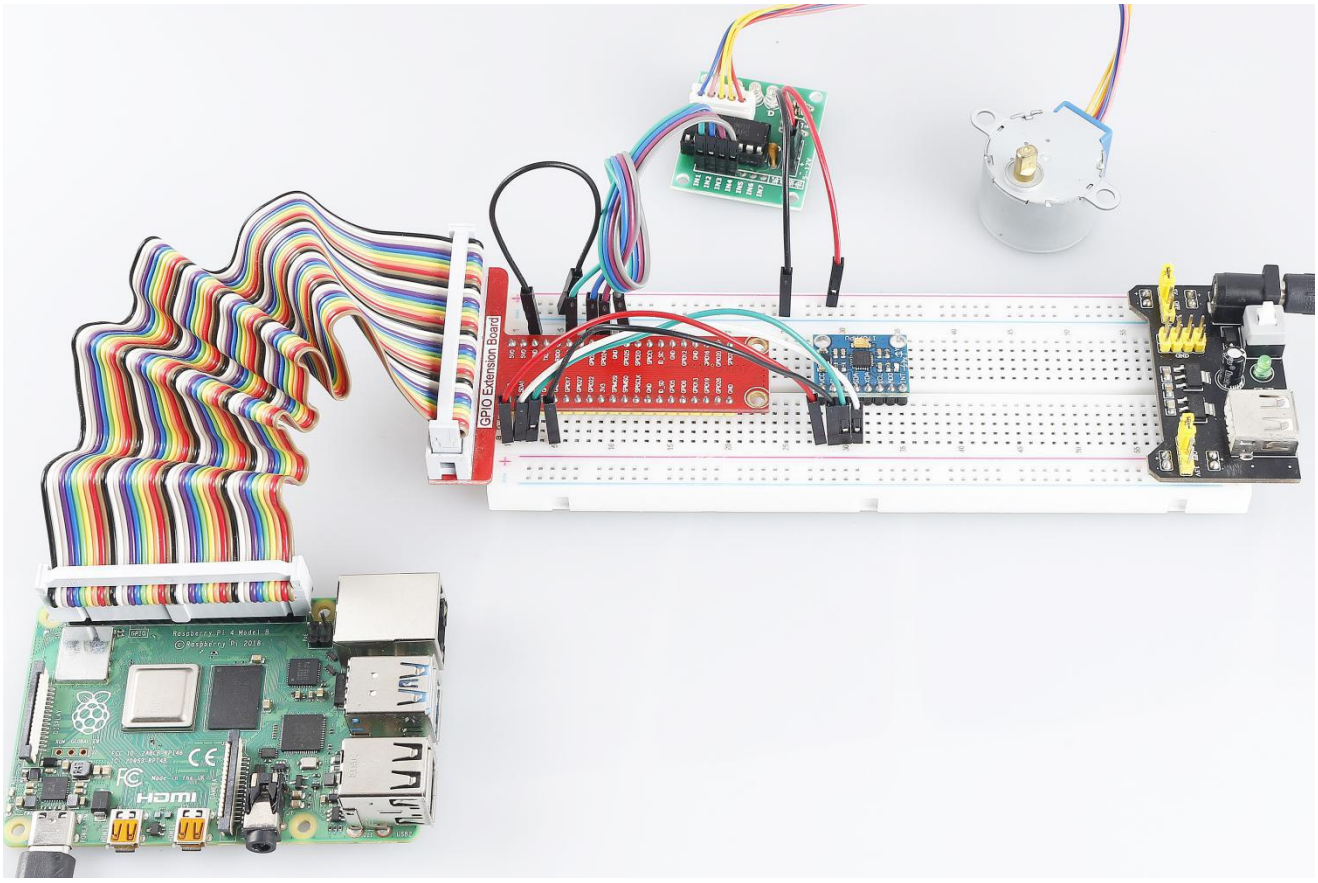
If the received direction **key** is '**c**', the stepper motor rotates clockwise; if the **key** is '**a**', the motor rotates anticlockwise. Refer to **1.3.3 Stepper Motor** for more details about the calculation of the rotating direction of the stepper motor.

```
def loop():
```

```
while True:
    angle=mpu6050()
    if angle >=45 :
        rotary('a')
    elif angle <=-45:
        rotary('c')
```

The tilt angle in the direction of the **Y-axis** is read from **mpu6050**, and if it's larger than **45**℃, rotary() is called to let the stepper motor rotate anticlockwise; if less than **-45** ℃, the stepper motor rotates clockwise.

Phenomenon Picture

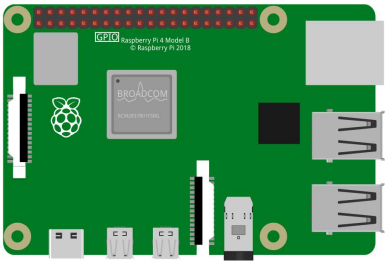
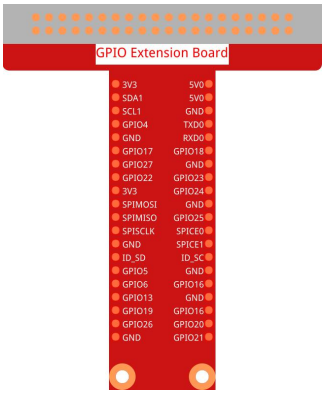
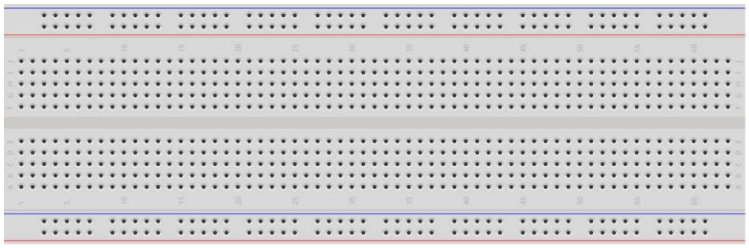


3.1.7 Traffic Light

Introduction

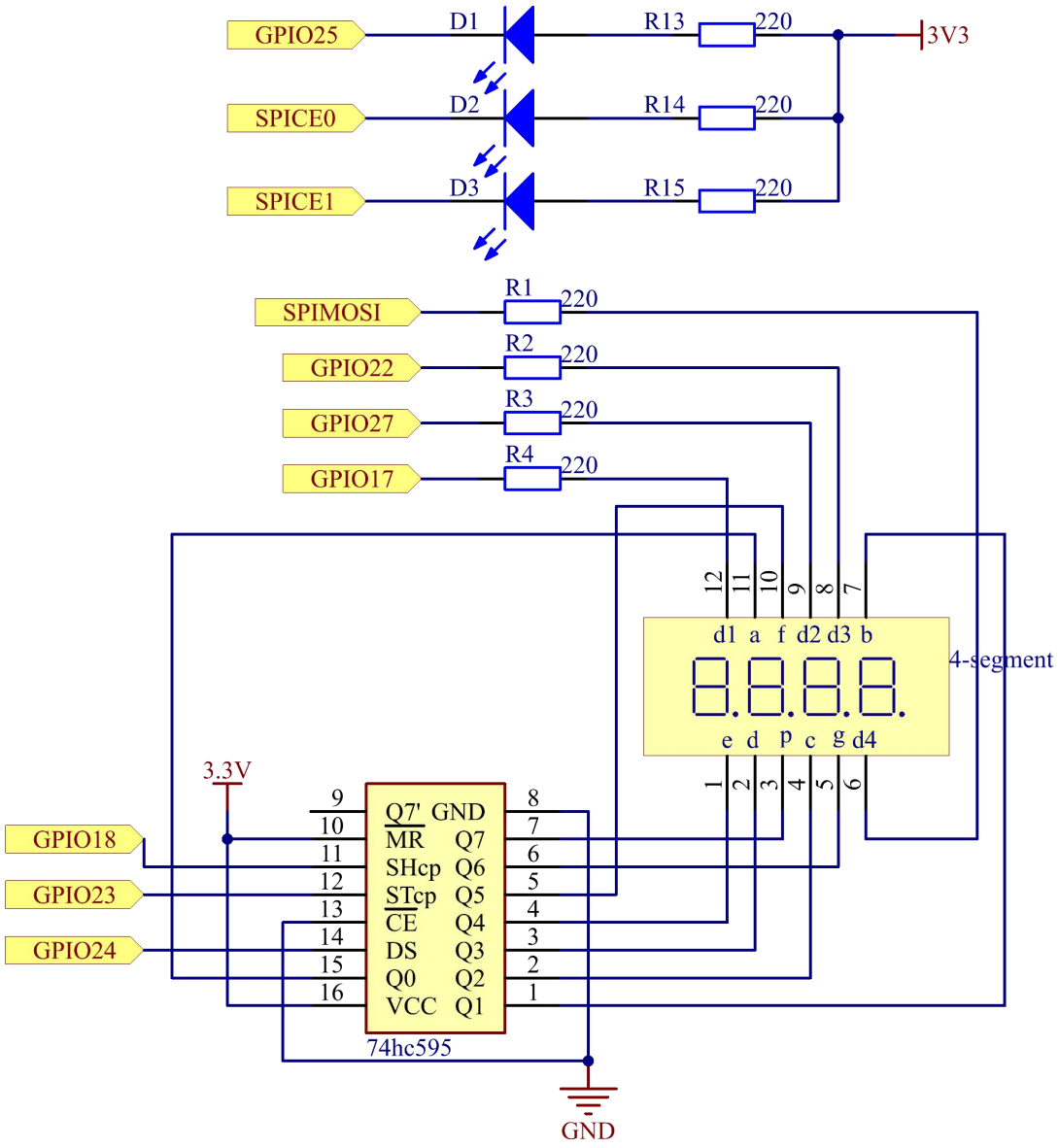
In this project, we will use LED lights of three colors to realize the change of traffic lights and a four-digit 7-segment display will be used to display the timing of each traffic state.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td colspan="2">1 * 4-Digit 7-segment display
  </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
  </td><td>4 * S8550 PNP Transistor
  </td><td>3 * LED
  </td></tr>
<tr><td colspan="2">1 * 74HC595
 
</td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td colspan="2">Several Jumper Wires
 
11 * Resistor(220Ω)
 
4 * Resistor 1KΩ
  </td></tr>
</table>

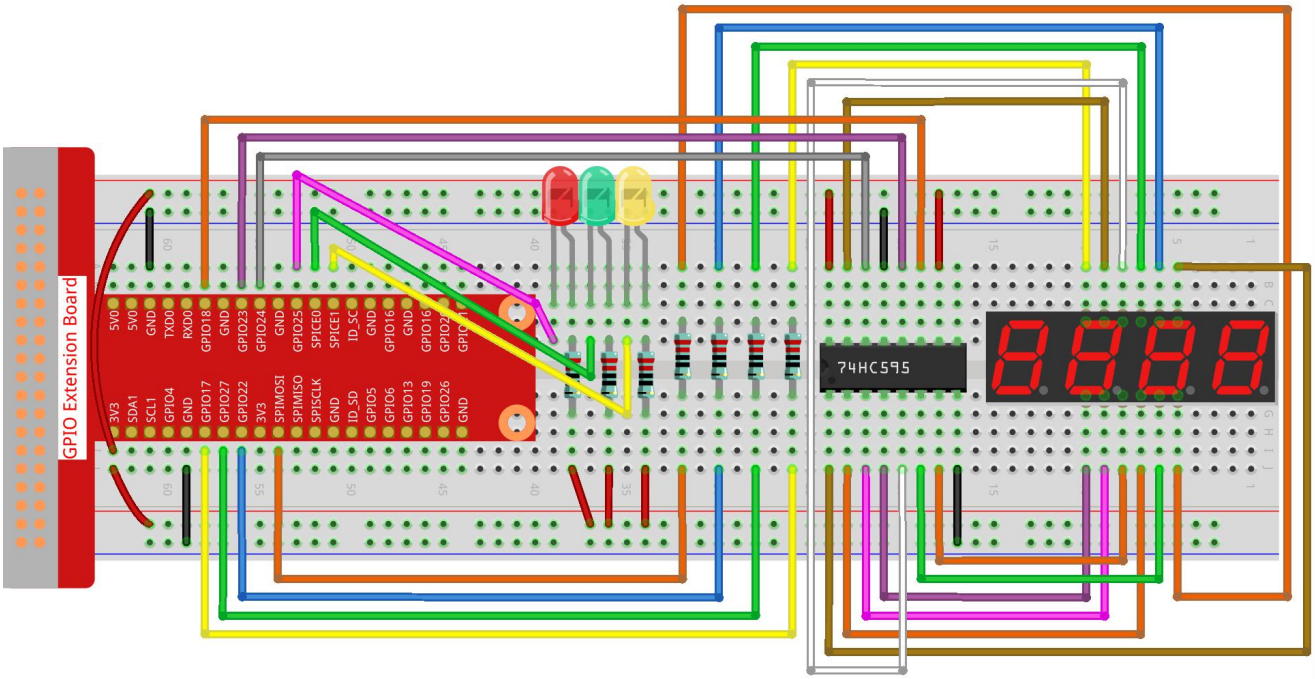
Schematic Diagram

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
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPICE0	Pin 24	10	8
SPICE1	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/3.1.7/
```

Step 3: Compile.

```
gcc 3.1.7_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 Explanation

```
#define     SDI     5
#define     RCLK    4
#define     SRCLK   1

const int placePin[] = {12, 3, 2, 0};
unsigned char number[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};

void pickDigit(int digit);
void hc595_shift(int8_t data);
void clearDisplay();
void display();
```

These codes are used to realize the function of number display of 4-Digit 7-Segment Displays. Refer to **chapter 1.1.5** of the document for more details. Here, we use the codes to display countdown of traffic light time.

```
const int ledPin[]={6,10,11};

int colorState = 0;

void lightup()
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],HIGH);
    }
    digitalWrite(ledPin[colorState],LOW);
}
```

The codes are used to switch the LED on and off.

```
int greenLight = 30;
int yellowLight = 5;
int redLight = 60;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 60;

void timer(int   timer1){          //Timer function
```

```
    if(timer1 == 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;
        }
        printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);
    }
}
```

The codes are used to switch the timer on and off. Refer to chapter 1.1.5 for more details. Here, when the timer returns to zero, colorState will be switched so as to switch LED, and the timer will be assigned to a new value.

```
void loop()
{
    while(1){
    display();
    lightup();
    }
}

int main(void)
{
    //...
    signal(SIGALRM,timer);
    alarm(1);
    loop();
    return 0;
}
```

The timer is started in the main() function. In loop() function, use **while(1)** loop and call the functions of 4-Digit 7-Segment and LED.

➢ For Python Language Users

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 3.1.7_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 Explanation

```
SDI    = 24         #serial data input(DS)
RCLK   = 23         #memory clock input(STCP)
SRCLK = 18          #shift register clock input(SHCP)
number = (0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90)
placePin = (17,27,22,10)

def clearDisplay():
def hc595_shift(data):
def pickDigit(digit):
def display():
```

These codes are used to realize the function of number display of 4-Digit 7-Segment. Refer to chapter 1.1.5 of the document for more details. Here, we use the codes to display countdown of traffic light time.

```
ledPin =(22,24,26)
colorState=0

def lightup():
    global colorState
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.HIGH)
    GPIO.output(ledPin[colorState], GPIO.LOW)
```

The codes are used to switch the LED on and off.

```
greenLight = 30
```

```
yellowLight = 5
redLight = 60
lightColor=("Red","Green","Yellow")

colorState=0
counter = 60
timer1 = 0

def timer():            #timer function
    global counter
    global colorState
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.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
    print ("counter : %d      color: %s "%(counter,lightColor[colorState]))
```

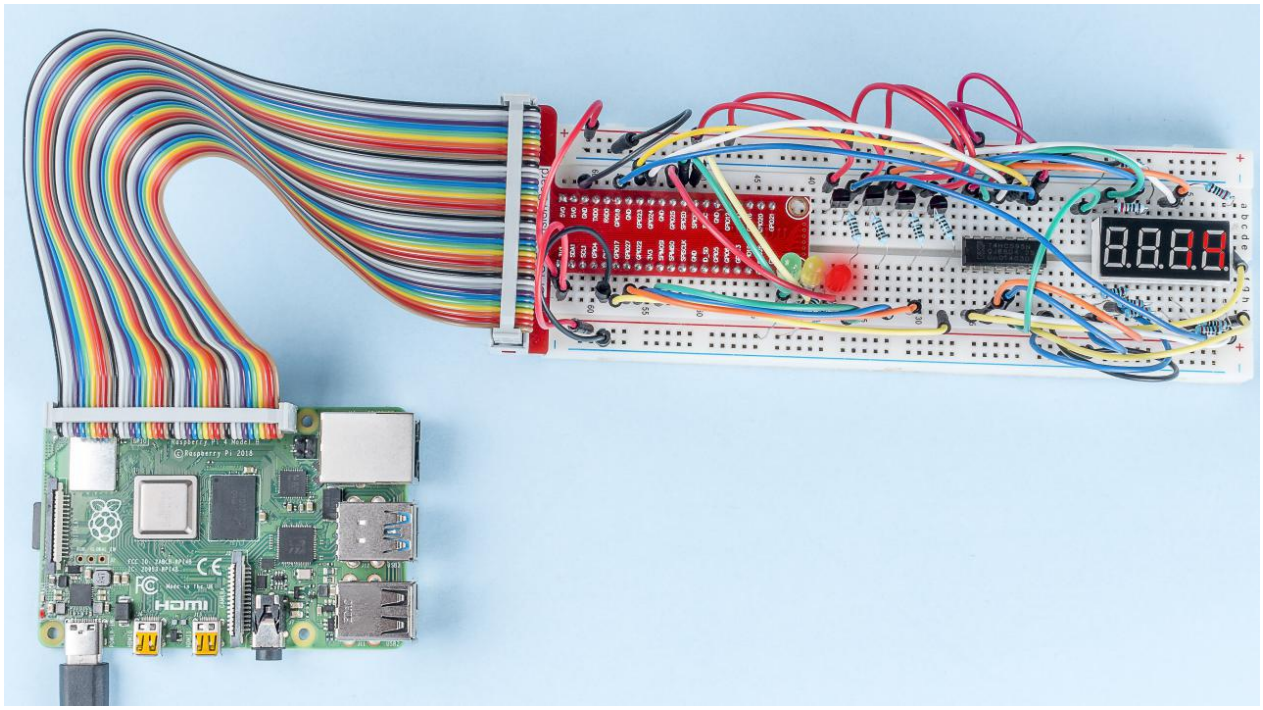
The codes are used to switch the timer on and off. Refer to chapter 1.1.5 for more details. Here, when the timer returns to zero, colorState will be switched so as to switch LED, and the timer will be assigned to a new value.

```
def setup():
    # ...
    global timer1
    timer1 = threading.Timer(1.0,timer)
    timer1.start()

def loop():
    while True:
        display()
        lightup()
```

```
def destroy():     # When "Ctrl+C" is pressed, the function is executed.
    global timer1
    GPIO.cleanup()
    timer1.cancel()         #cancel the timer

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

In setup() function, start the timer. In loop() function, a **while True** is used: call the relative functions of 4-Digit 7-Segment and LED circularly.

Phenomenon Picture

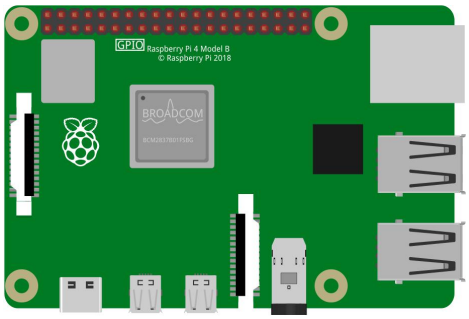
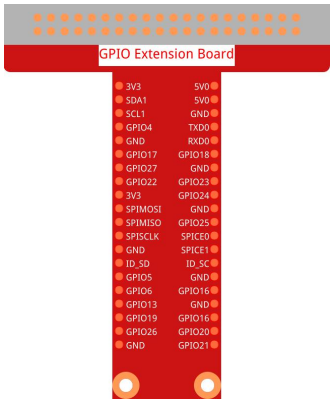
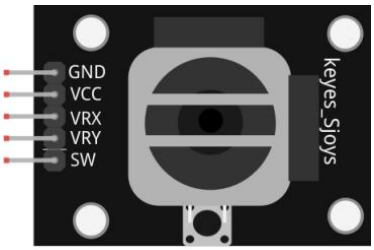
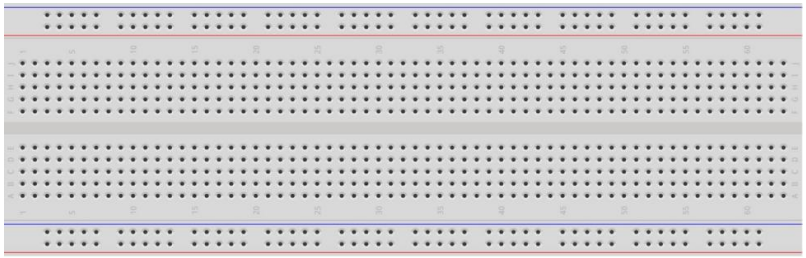


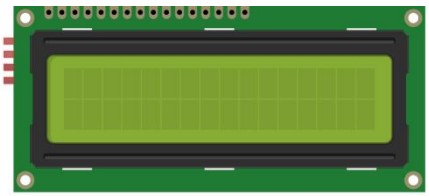
3.1.8 Overheat Monitor

Introduction

You may want to make an overheat monitoring device that applies to various situations, ex., in the factory, if we want to have an alarm and the timely automatic turning off of the machine when there is a circuit overheating. In this lesson, we will use thermistor, joystick, buzzer, LED and LCD to make an smart temperature monitoring device whose threshold is adjustable.

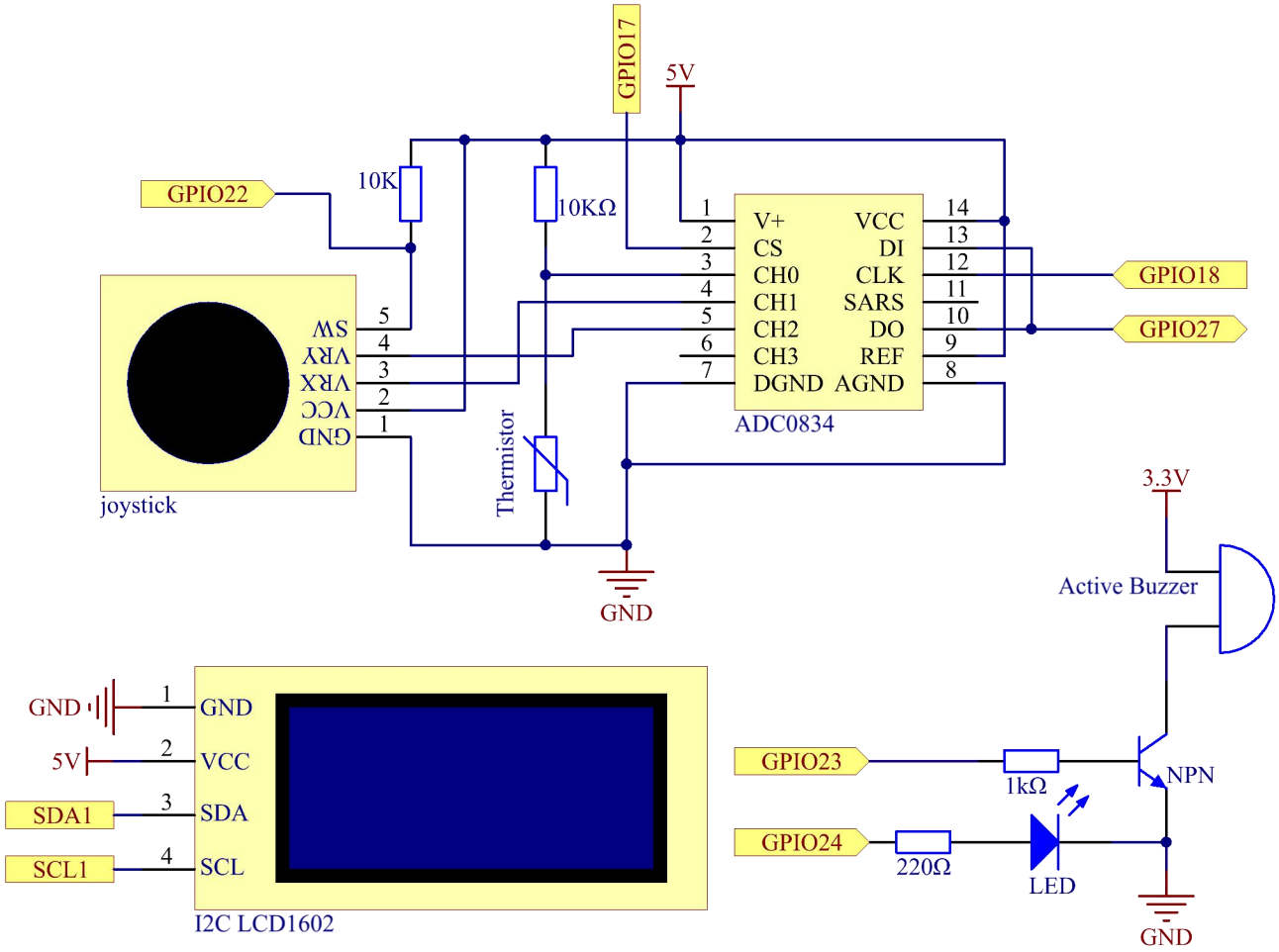
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Joystick
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>1 * ADC0834
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * LED
  </td><td>1 * S8050 NPN Transistor
  </td></tr>
<tr><td>1 * Resistor(220Ω)
  </td><td>1 * Resistor 1KΩ
  </td><td colspan="2">2 * Resistor 10KΩ
  </td></tr>
</table>

1 * I2C LCD1602	1 * Active Buzzer	Several Jumper Wires
		

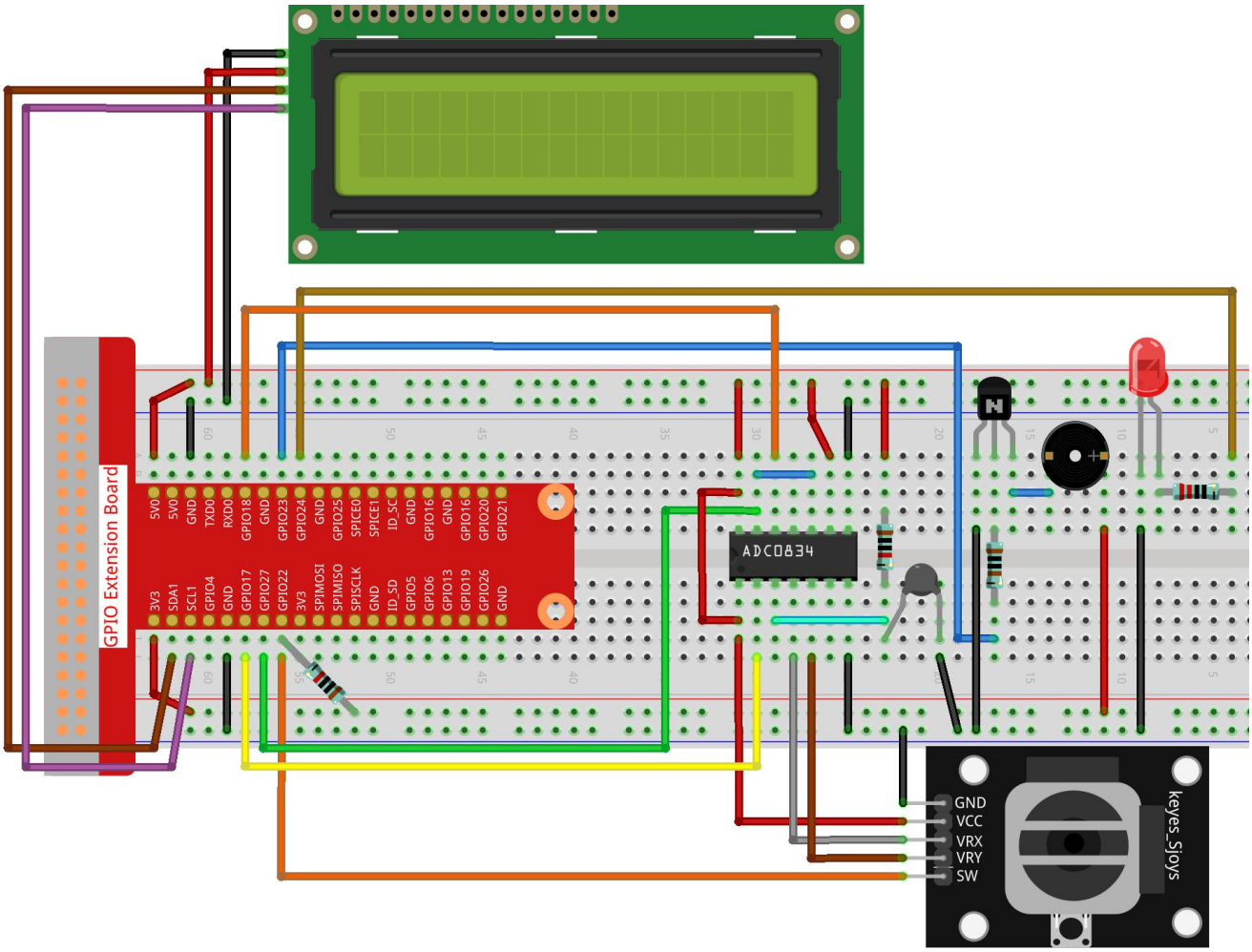
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO22	Pin15	3	22
GPIO23	Pin16	4	23
GPIO24	Pin18	5	24
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/3.1.8/
```

Step 3: Compile the code.

```
gcc 3.1.8_OverheatMonitor.c -lwiringPi
```

Step 4: Run the executable file.

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

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

Joystick here is for your pressing to adjust the high-temperature threshold. Toggling the **Joystick** in the direction of X-axis and Y-axis can adjust (turn up or down) the

current high-temperature threshold. Press the **Joystick** once again to reset the threshold to initial value.

Code Explanation

```
int get_joystick_value(){
    uchar x_val;
    uchar y_val;
    x_val = get_ADC_Result(1);
    y_val = get_ADC_Result(2);
    if (x_val > 200){
        return 1;
    }
    else if(x_val < 50){
        return -1;
    }
    else if(y_val > 200){
        return -10;
    }
    else if(y_val < 50){
        return 10;
    }
    else{
        return 0;
    }
}
```

This function reads values of X and Y. If **X>200**, there will return "**1**"; **X<50**, return "**-1**"; **y>200**, return "**-10**", and **y<50**, return "**10**".

```
void upper_tem_setting(){
    write(0, 0, "Upper Adjust:");
    int change = get_joystick_value();
    upperTem = upperTem + change;
    char str[6];
    snprintf(str,3,"%d",upperTem);
    write(0,1,str);
   int len;
   len = strlen(str);
   write(len,1,"              ");
    delay(100);
}
```

This function is for adjusting the threshold and displaying it on the I2C LCD1602.

```
double temperature(){
    unsigned char temp_value;
    double Vr, Rt, temp, cel, Fah;
    temp_value = get_ADC_Result(0);
    Vr = 5 * (double)(temp_value) / 255;
    Rt = 10000 * (double)(Vr) / (5 - (double)(Vr));
    temp = 1 / (((log(Rt/10000)) / 3950)+(1 / (273.15 + 25)));
    cel = temp - 273.15;
    Fah = cel * 1.8 +32;
    return cel;
}
```

Read the analog value of the **CH0** (thermistor) of **ADC0834** and then convert it to temperature value.

```
void monitoring_temp(){
    char str[6];
    double cel = temperature();
    snprintf(str,6,"%.2f",cel);
    write(0, 0, "Temp: ");
    write(6, 0, str);
    snprintf(str,3,"%d",upperTem);
    write(0, 1, "Upper: ");
    write(7, 1, str);
    delay(100);
    if(cel >= upperTem){
        digitalWrite(buzzPin, HIGH);
        digitalWrite(LedPin, HIGH);
    }
    else if(cel < upperTem){
        digitalWrite(buzzPin, LOW);
        digitalWrite(LedPin, LOW);
    }
}
```

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

```
int main(void)
{
```

```
    setup();
    int lastState =1;
    int stage=0;
    while (1)
    {
        int currentState = digitalRead(Joy_BtnPin);
        if(currentState==1 && lastState == 0){
            stage=(stage+1)%2;
            delay(100);
            lcd_clear();
        }
        lastState=currentState;
        if (stage==1){
            upper_tem_setting();
        }
        else{
            monitoring_temp();
        }
    }
    return 0;
}
```

The function main() contains the whole program process as shown:

1) When the program starts, the initial value of **stage** is **0**, and the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and the LED are started to alarm you.

2) Press the Joystick, and **stage** will be **1** and you can adjust the high-temperature threshold. Toggling the Joystick in the direction of X-axis and Y-axis can adjust (turn up or down) the current threshold. Press the Joystick once again to reset the threshold to initial value.

➢ 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 python3 3.1.8_OverheatMonitor.py
```

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

Joystick here is for your pressing to adjust the high-temperature threshold. Toggling the **Joystick** in the direction of X-axis and Y-axis can adjust (turn up or down) the current high-temperature threshold. Press the **Joystick** once again to reset the threshold to initial value.

Code Explanation

```
def get_joystick_value():
    x_val = ADC0834.getResult(1)
    y_val = ADC0834.getResult(2)
    if(x_val > 200):
        return 1
    elif(x_val < 50):
        return -1
    elif(y_val > 200):
        return -10
    elif(y_val < 50):
        return 10
    else:
        return 0
```

This function reads values of X and Y. If **X>200**, there will return "**1**"; **X<50**, return "**-1**"; **y>200**, return "**-10**", and **y<50**, return "**10**".

```
def upper_tem_setting():
    global upperTem
    LCD1602.write(0, 0, 'Upper Adjust: ')
    change = int(get_joystick_value())
    upperTem = upperTem + change
    LCD1602.write(0, 1, str(upperTem))
    LCD1602.write(len(strUpperTem),1, '                    ')
    time.sleep(0.1)
```

This function is for adjusting the threshold and displaying it on the I2C LCD1602.

```
def temperature():
    analogVal = ADC0834.getResult()
    Vr = 5 * float(analogVal) / 255
    Rt = 10000 * Vr / (5 - Vr)
```

```
    temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))
    Cel = temp - 273.15
    Fah = Cel * 1.8 + 32
    return round(Cel,2)
```

Read the analog value of the **CH0** (thermistor) of **ADC0834** and then convert it to temperature value.

```
def monitoring_temp():
    global upperTem
    Cel=temperature()
    LCD1602.write(0, 0, 'Temp: ')
    LCD1602.write(0, 1, 'Upper: ')
    LCD1602.write(6, 0, str(Cel))
    LCD1602.write(7, 1, str(upperTem))
    time.sleep(0.1)
    if Cel >= upperTem:
        GPIO.output(buzzPin, GPIO.HIGH)
        GPIO.output(ledPin, GPIO.HIGH)
    else:
        GPIO.output(buzzPin, GPIO.LOW)
        GPIO.output(ledPin, GPIO.LOW)
```

As the code runs, the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and LED are started to alarm you.

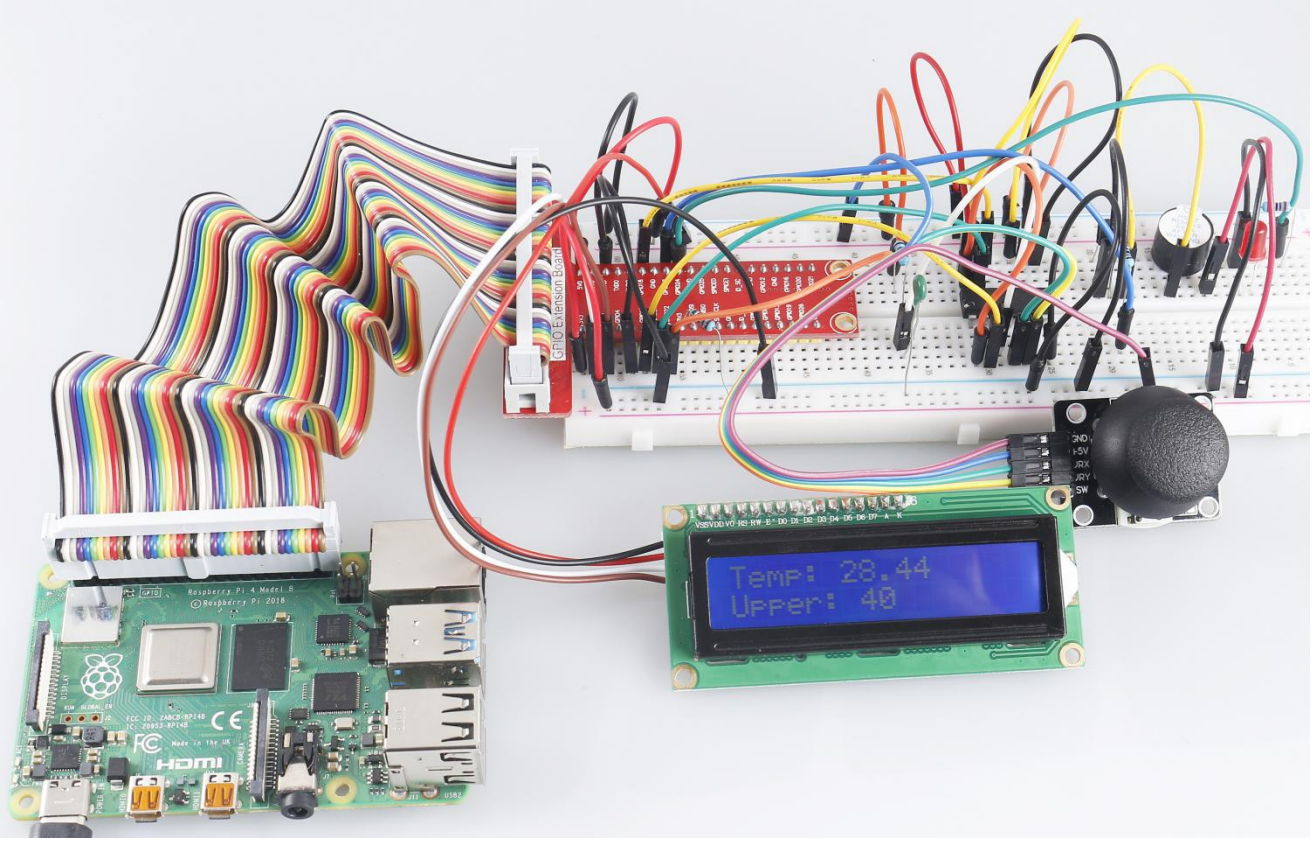
```
def loop():
    lastState=1
    stage=0
    while True:
        currentState=GPIO.input(Joy_BtnPin)
        if currentState==1 and lastState ==0:
            stage=(stage+1)%2
            time.sleep(0.1)
            LCD1602.clear()
        lastState=currentState
        if stage == 1:
            upper_tem_setting()
        else:
            monitoring_temp()
```

The function main() contains the whole program process as shown:

1) When the program starts, the initial value of **stage** is **0**, and the current temperature and the high-temperature threshold **40** are displayed on **I2C LCD1602**. If the current temperature is larger than the threshold, the buzzer and the LED are started to alarm you.

2) Press the Joystick, and **stage** will be **1** and you can adjust the high-temperature threshold. Toggling the Joystick in the direction of X-axis and Y-axis can adjust (turn up or down) the current high-temperature threshold. Press the Joystick once again to reset the threshold to initial value.

Phenomenon Picture



3.1.9 Password Lock

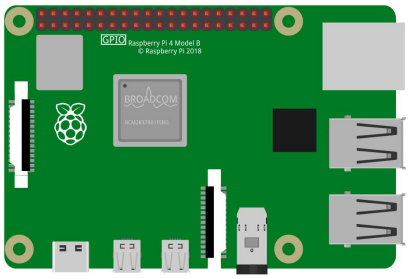
Introduction

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.

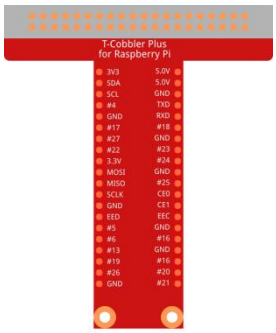
On the basis of this project, we can add additional electronic components, such as buzzer, LED and so on, to add different experimental phenomena for password input.

Components

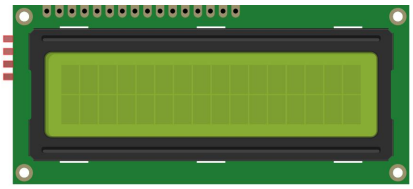
1 * Raspberry Pi



1 * T-Extension Board



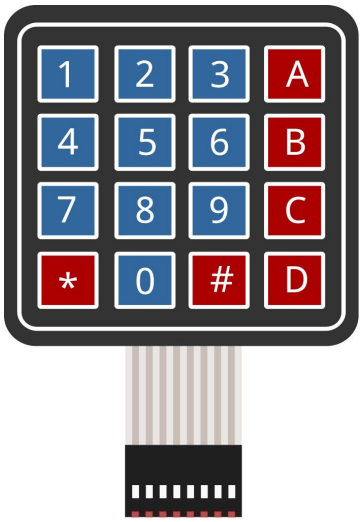
1 * I2C LCD1602



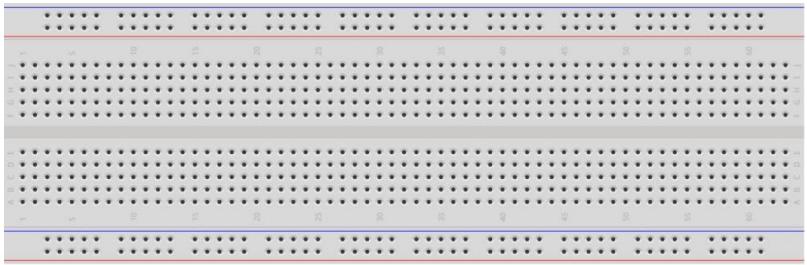
Several Jumper Wires

1 * 40-pin Cable

1 * Keypad



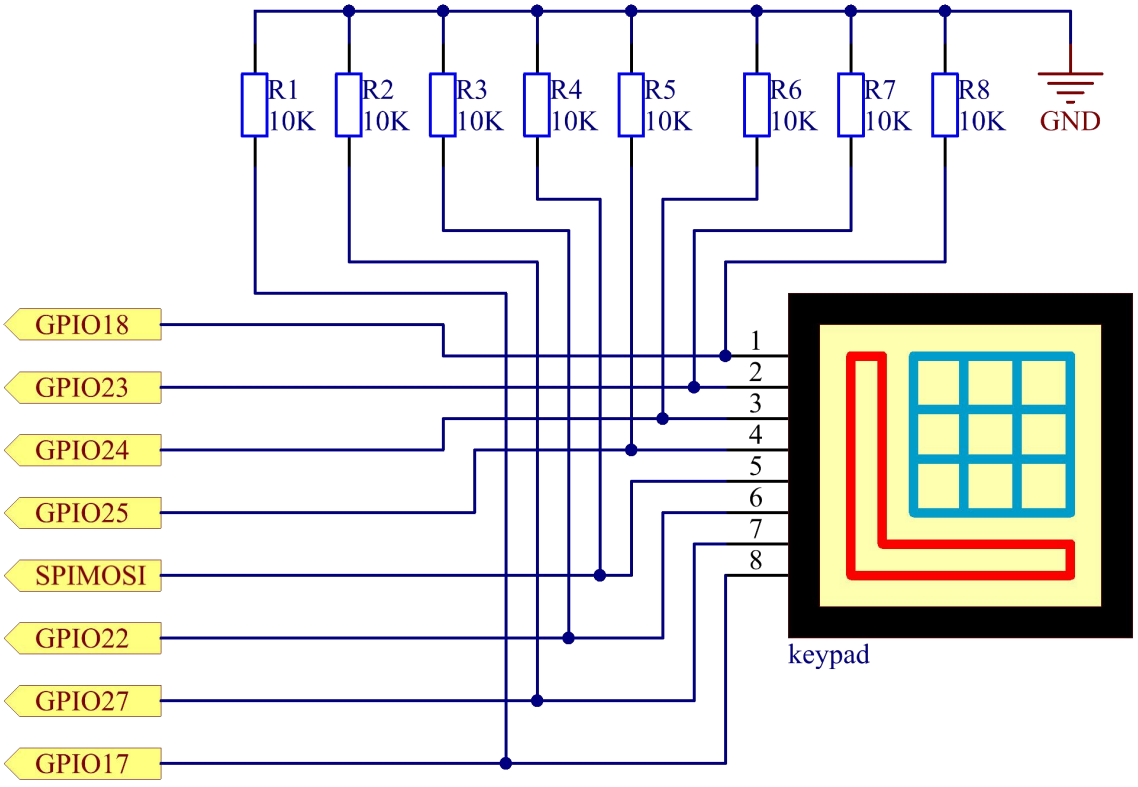
1 * Breadboard

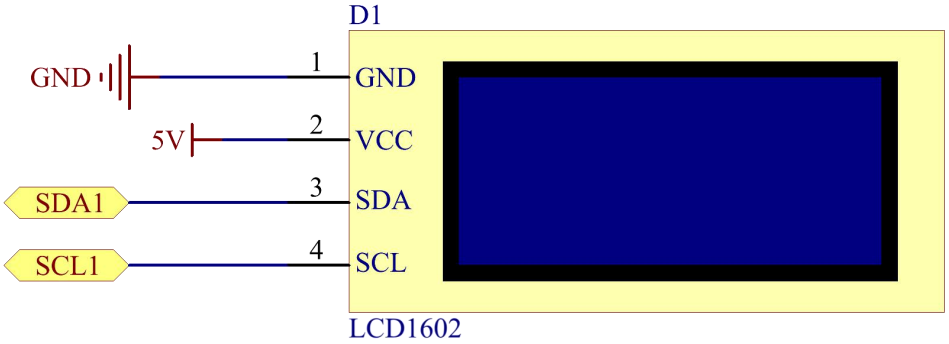


8 * Resistor 10KΩ

Schematic Diagram

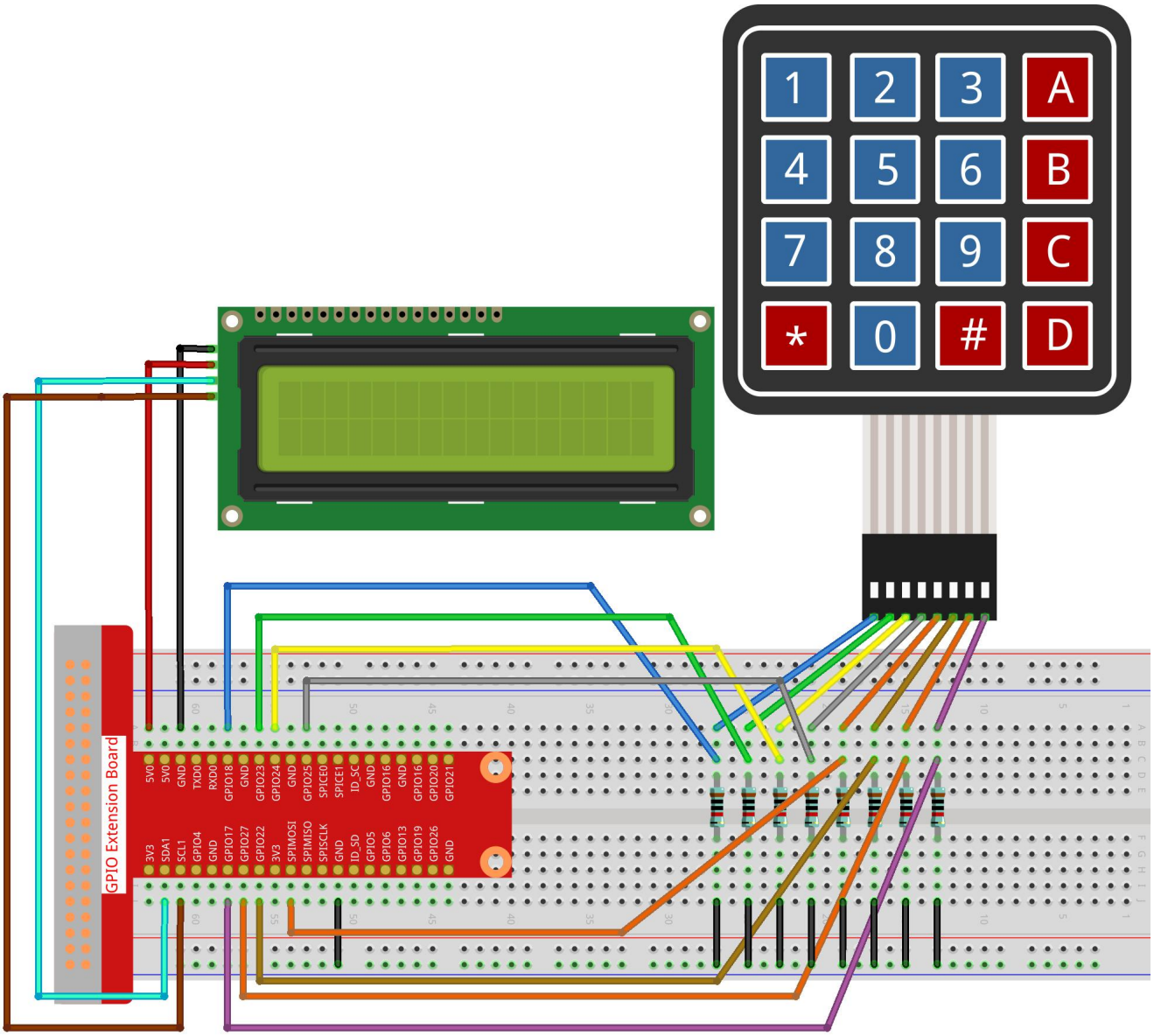
T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
GPIO17	Pin 11	0	17
GPIO27	Pin 13	2	27
GPIO22	Pin 15	3	22
SPIMOSI	Pin 19	12	10
SDA1	Pin 3		
SCL1	Pin 5		





Experimental Procedures

Step 1: Build the circuit.



➢ For C Language Users

Step 2: Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.9/
```

Step 3: Compile.

```
gcc 3.1.9_PasswordLock.cpp -lwiringPi
```

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 Explanation

```
#define ROWS  4
#define COLS  4
#define BUTTON_NUM (ROWS * COLS)
#define LENS  4

unsigned char KEYS[BUTTON_NUM] {
  '1','2','3','A',
  '4','5','6','B',
  '7','8','9','C',
  '*','0','#','D'};

char password[LENS]={'1','9','8','4'};
```

Here, we define the length of the password LENS, storage matrix keyboard key value array KEYS and the array that stores the correct password.

```
void keyRead(unsigned char* result);
bool keyCompare(unsigned char* a, unsigned char* b);
void keyCopy(unsigned char* a, unsigned char* b);
void keyPrint(unsigned char* a);
void keyClear(unsigned char* a);
int keyIndexOf(const char value);
```

There is a declaration of the subfunctions of the matrix keyboard code, refer to **chapter 2.1.5** of this document for more details.

```
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[]);
```

There is a declaration of the subfunctions of LCD1062 code, refer to **chapter 1.1.7** of this document for more details.

```
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        ......
                testword[keyIndex]=pressed_keys[0];
                keyIndex++;
                if(keyIndex==LENS){
                    if(check()==0){
                        clear();
                        write(3, 0, "WRONG KEY!");
                        write(0, 1, "please try again");
                        }
                ......
```

Read the key value and store it in the test array testword. If the number of stored key values is more than 4, the correctness of the password is automatically verified, and the verification results are displayed on the LCD interface.

```
int check(){
    for(int i=0;i<LENS;i++){
        if(password[i]!=testword[i])
        {return 0;}
    }
    return 1;
}
```

Verify the correctness of the password. Return 1 if the password is entered correctly, and 0 if not.

➢ For Python Language Users

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 3.1.9_PasswordLock.py
```

After the code runs, keypad is used to input password:1984. If the "CORRECT" appears on LCD1602, there is no wrong with the password; otherwise, "WRONG KEY" will appear.

Code Explanation

```
LENS = 4
password=['1','9','8','4']
.....
rowsPins = [18,23,24,25]
colsPins = [10,22,27,17]
keys = ["1","2","3","A",
        "4","5","6","B",
        "7","8","9","C",
        "*","0","#","D"]
```

Here, we define the length of the password LENS, the array keys that store the matrix keyboard keys, and the array password that stores the correct password.

```
class Keypad():
    def __init__(self, rowsPins, colsPins, keys):
        self.rowsPins = rowsPins
        self.colsPins = colsPins
        self.keys = keys
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.rowsPins, GPIO.OUT, initial=GPIO.LOW)
        GPIO.setup(self.colsPins, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  ...
```

This class is the code that reads the values of the pressed keys. Refer to **chapter 2.1.5** of this document for more details.

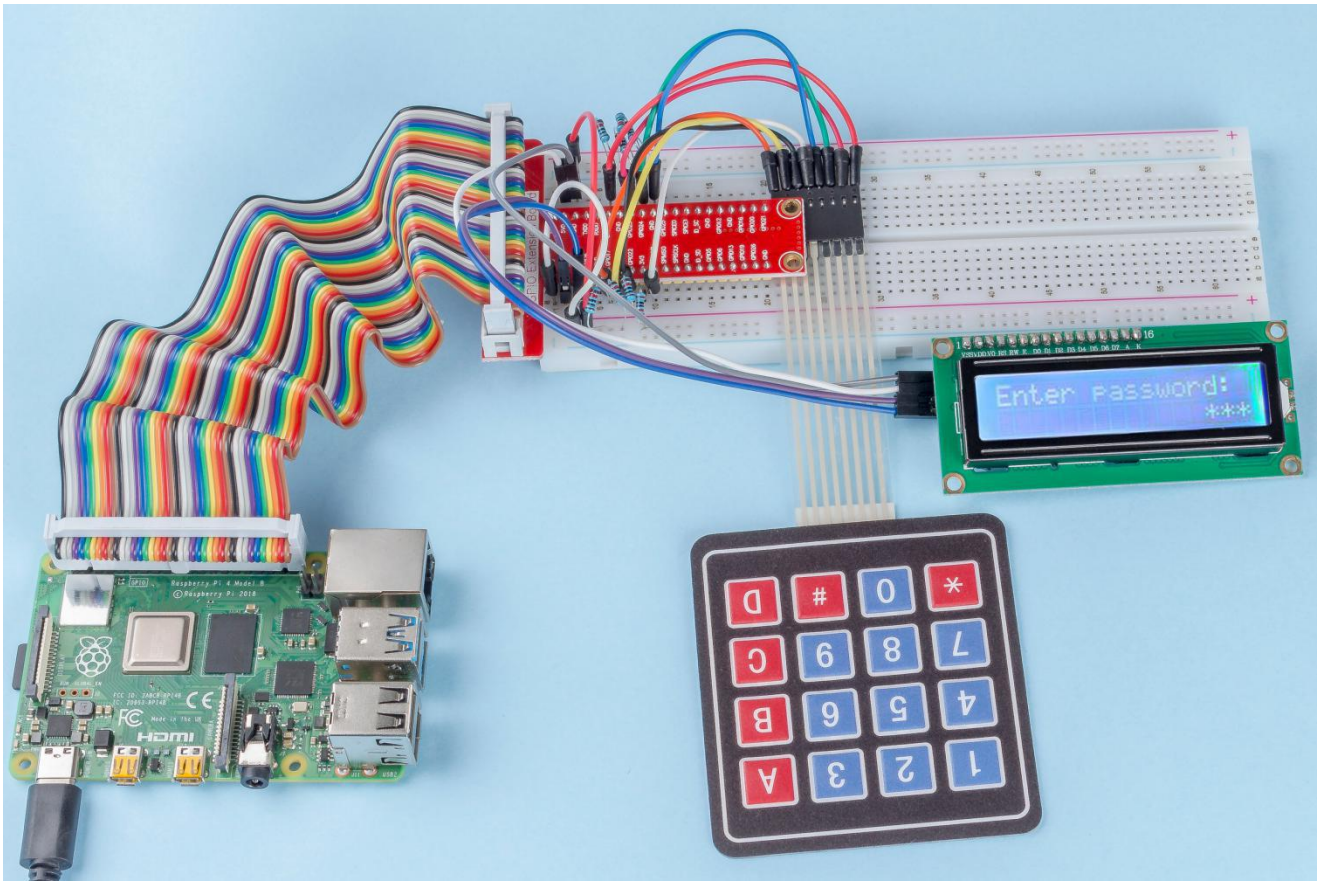
```
    while(True):
        pressed_keys = keypad.read()
        if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
            LCD1602.clear()
            LCD1602.write(0, 0, "Enter password:")
            LCD1602.write(15-keyIndex,1, pressed_keys)
            testword[keyIndex]=pressed_keys
            keyIndex+=1
  ...
```

Read the key value and store it in the test array testword. If the number of stored key values is more than 4, the correctness of the password is automatically verified, and the verification results are displayed on the LCD interface.

```
def check():
    for i in range(0,LENS):
        if(password[i]!=testword[i]):
            return 0
    return 1
```

Verify the correctness of the password. Return 1 if the password is entered correctly, and 0 if not.

Phenomenon Picture

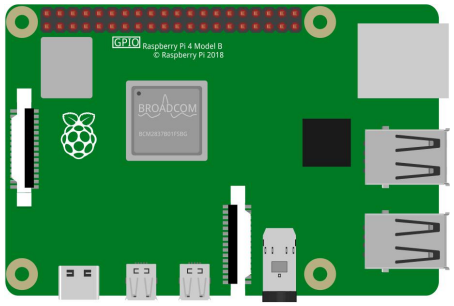
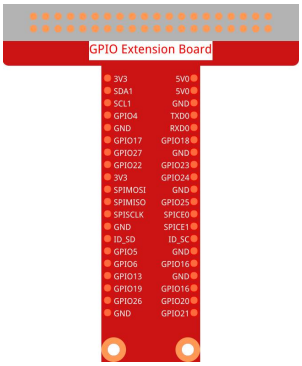
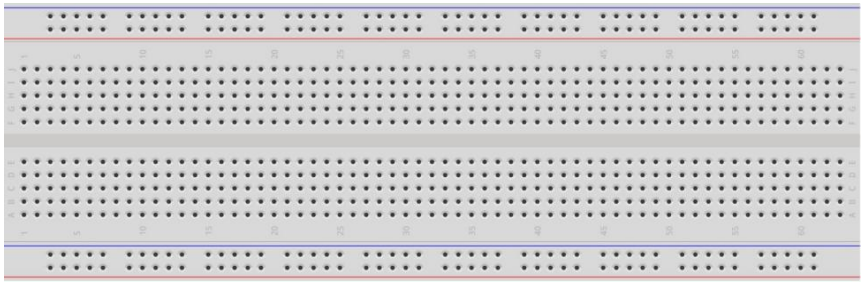
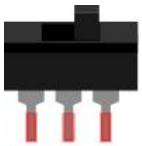
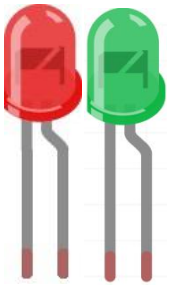


3.1.10 Alarm Bell

Introduction

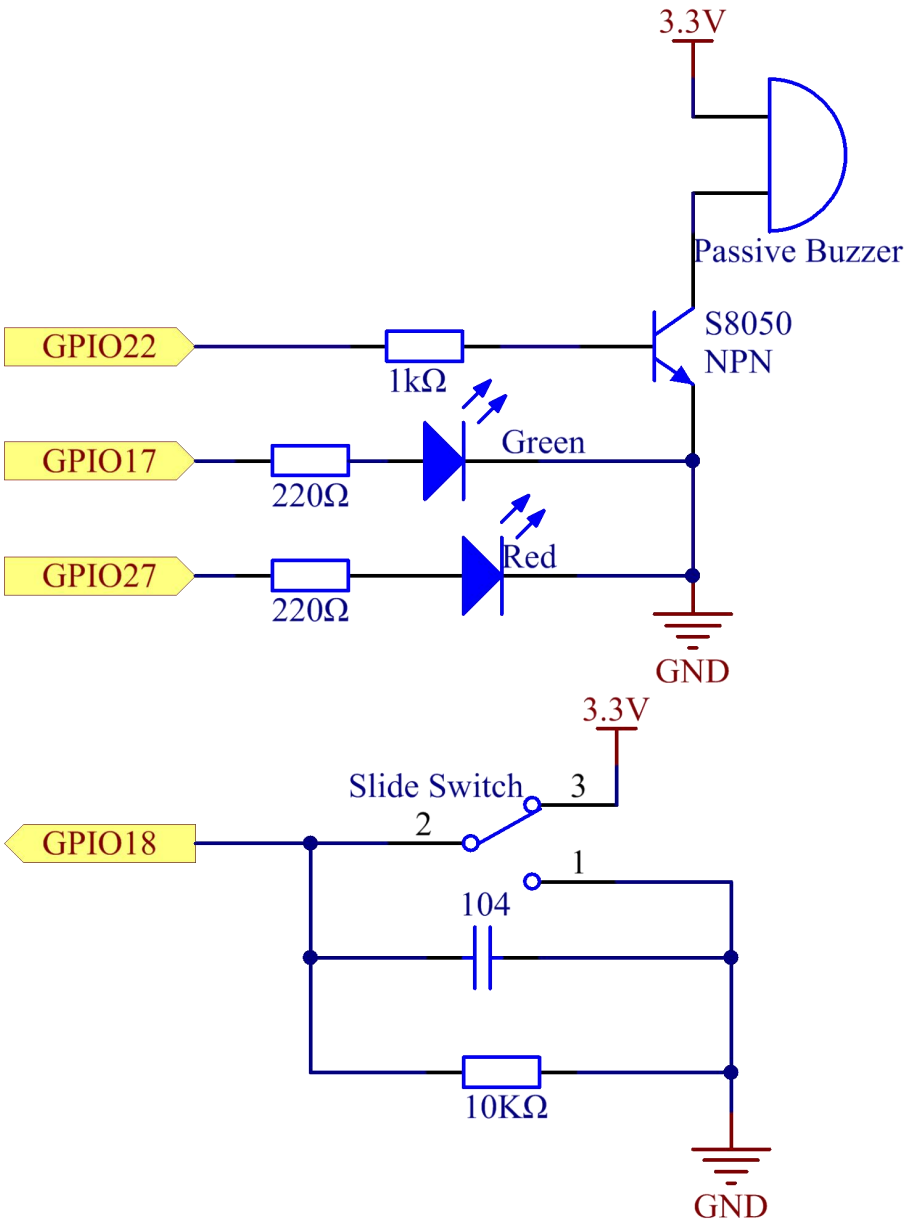
In this course, we will make a manual alarm device. You can replace the toggle switch with a thermistor or a photosensitive sensor to make a temperature alarm or a light alarm.

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
  </td><td>1 * Passive Buzzer
  </td></tr>
<tr><td colspan="2" rowspan="2">1 * 40-pin Cable
  </td><td>1 * Resistor(1kΩ)
  </td></tr>
<tr><td>2 * Resistor(220Ω)
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * Resistor 10KΩ
  </td></tr>
<tr><td colspan="2"></td><td>1 * Slide Switch
  </td></tr>
<tr><td>Several Jumper Wires
  </td><td>2 * LED
  </td><td>1 * S85050 NPN Transistor
  </td><td>1 * 104 Capacitor
  </td></tr>
</table>

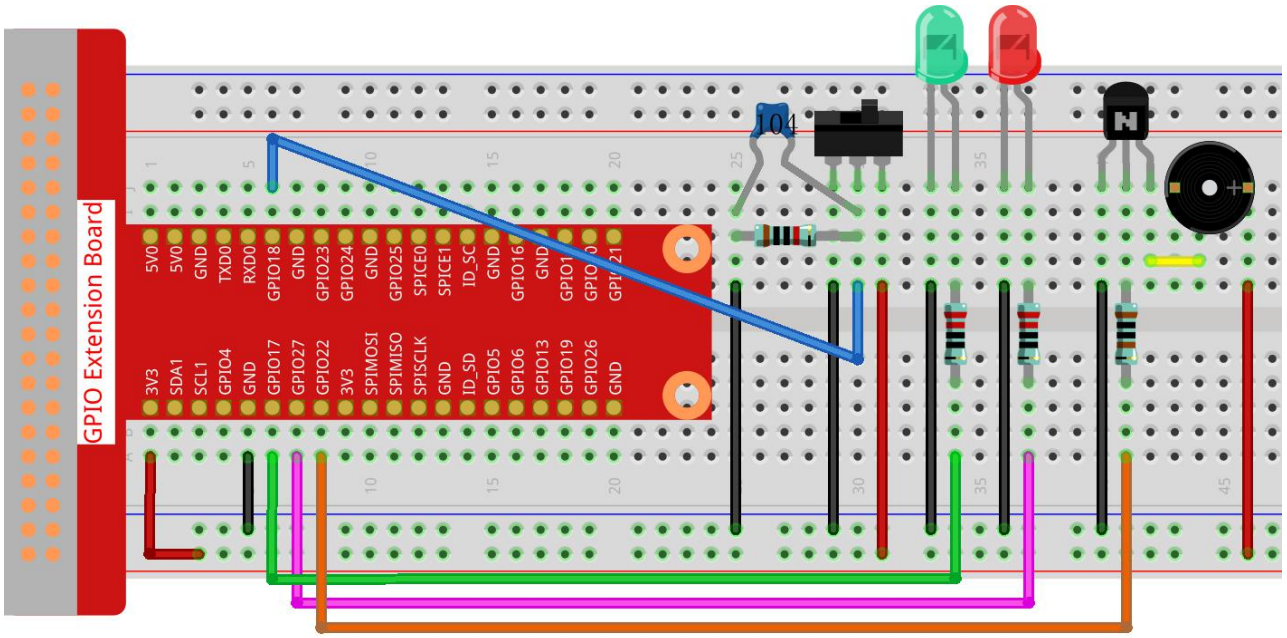
Schematic Diagram

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



Experimental Procedures

Step 1: Build the circuit.



➢ For C Language Users

Step 2: Change directory.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.10/
```

Step 3: Compile.

```
gcc 3.1.10_AlarmBell.c -lwiringPi -lpthread
```

Step 4: Run.

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

After the program starts, the toggle switch will be toggled to the right, and the buzzer will give out alarm sounds. At the same time, the red and green LEDs will flash at a certain frequency.

Code Explanation

```
#include <pthread.h>
```

In this code, you'll use a new library, pthread.h, which is a set of common thread libraries and can realize multithreading. We add the **-lpthread** parameter at compile time for the independent working of the LED and the buzzer.

```
void *ledWork(void *arg){
  while(1)
```

```
    {
        if(flag==0){
            pthread_exit(NULL);
        }
        digitalWrite(ALedPin,HIGH);
        delay(500);
        digitalWrite(ALedPin,LOW);
        digitalWrite(BLedPin,HIGH);
        delay(500);
        digitalWrite(BLedPin,LOW);
    }
}
```

The function ledWork() helps to set the working state of these 2 LEDs: it keeps the green LED lighting up for 0.5s and then turns off; similarly, keeps the red LED lighting up for 0.5s and then turns off.

```
  void *buzzWork(void *arg){
   while(1)
   {
        if(flag==0){
            pthread_exit(NULL);
        }
        if((note>=800)||(note<=130)){
            pitch = -pitch;
        }
        note=note+pitch;
        softToneWrite(BeepPin,note);
        delay(10);
    }
}
```

The function buzzWork() is used to set the working state of the buzzer. Here we set the frequency as between 130 and 800, to accumulate or decay at an interval of 20.

```
 void on(){
    flag = 1;
    if(softToneCreate(BeepPin) == -1){
        printf("setup softTone failed !");
        return;
    }
    pthread_t tLed;
```

```
    pthread_create(&tLed,NULL,ledWork,NULL);
    pthread_t tBuzz;
    pthread_create(&tBuzz,NULL,buzzWork,NULL);
}
```

In the function on():

1) Define the mark "flag=1", indicating the ending of the control thread.

2) Create a software-controlled tone pin **BeepPin**.

3) Create two separate threads so that the LED and the buzzer can work at the same time.

pthread_t tLed: Declare a thread **tLed**.

pthread_create(&tLed,NULL,ledWork,NULL): Create the thread and its prototype is as follows:

```
int pthread_create(pthread_t *restrict tidp,const pthread_attr_t *restrict_attr,void*
  (*start_rtn)(void*),void *restrict arg);
```

Return the Value

If successful, return "**0**"; otherwise, return the **fall number** "**-1**".

Parameter

The first parameter is a pointer to the thread identifier.
The second one is used to set the thread attribute.
The third one is the starting address of the thread running function.
The last one is the one that runs the function.

```
void off(){
    flag = 0;
    softToneStop(BeepPin);
    digitalWrite(ALedPin,LOW);
    digitalWrite(BLedPin,LOW);
}
```

The function Off() defines "flag=0" so as to exit the threads **ledWork** and **BuzzWork** and then turn off the buzzer and the LED.

```
int main(){
    setup();
    int lastState = 0;
    while(1){
        int currentState = digitalRead(switchPin);
        if ((currentState == 1)&&(lastState==0)){
```

```
            on();
        }
        else if((currentState == 0)&&(lastState==1)){
            off();
        }
        lastState=currentState;
    }
    return 0;
}
```

Main() contains the whole process of the program: firstly read the value of the slide switch; if the toggle switch is toggled to the right (the reading is 1), the function on() is called, the buzzer is driven to emit sounds and the the red and the green LEDs blink. Otherwise, the buzzer and the LED don't work.

➢ For Python Language Users

Step 2: Change directory.

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

Step 3: Run.

```
sudo python3 3.1.10_AlarmBell.py
```

After the program starts, the toggle switch will be toggled to the right, and the buzzer will give out alarm sounds. At the same time, the red and green LEDs will flash at a certain frequency.

Code Explanation

```
import threading
```

Here, we import the **Threading** module and it allows you to do multiple things at once, while normal programs can only execute code from top to bottom. With **Threading** modules, the LED and the buzzer can work separately.

```
def ledWork():
    while flag:
        GPIO.output(ALedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(ALedPin,GPIO.LOW)
        GPIO.output(BLedPin,GPIO.HIGH)
        time.sleep(0.5)
        GPIO.output(BLedPin,GPIO.LOW)
```

The function ledWork() helps to set the working state of these 2 LEDs: it keeps the

green LED lighting up for 0.5s and then turns off; similarly, keeps the red LED lighting up for 0.5s and then turns off.

```
def buzzerWork():
    global pitch
    global note
    while flag:
        if note >= 800 or note <=130:
            pitch = -pitch
        note = note + pitch
        Buzz.ChangeFrequency(note)
        time.sleep(0.01)
```

The function buzzWork() is used to set the working state of the buzzer. Here we set the frequency as between 130 and 800, to accumulate or decay at an interval of 20.

```
def on():
    global flag
    flag = 1
    Buzz.start(50)
    tBuzz = threading.Thread(target=buzzerWork)
    tBuzz.start()
    tLed = threading.Thread(target=ledWork)
    tLed.start()
```

In the function on():

1) Define the mark "flag=1", indicating the ending of the control thread.
2) Start the Buzz, and set the duty cycle to 50%.
3) Create **2** separate threads so that the LED and the buzzer can work at the same time.

tBuzz = threading.Thread(target=buzzerWork)：Create the thread and its prototype is as follows:

```
class threading.Thread(group=None, target=None, name=None, args=(), kwargs={}, *,
 daemon=None)
```

Among the construction methods, the principal parameter is **target**, we need to assign a callable object (here are the functions **ledWork** and **BuzzWork**) to **target**.

Next **start()** is called to start the thread object, ex., tBuzz.start() is used to start the newly installed tBuzz thread.

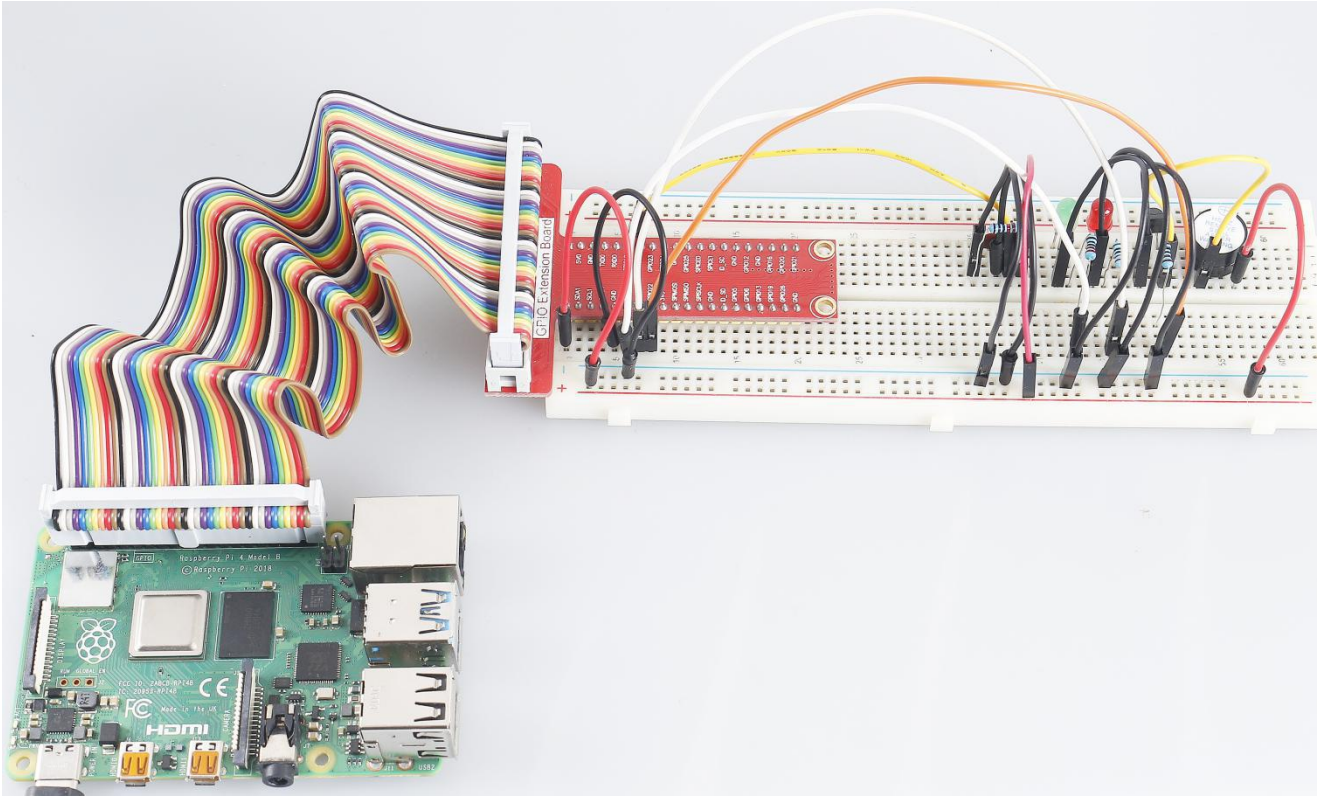
```
def off():
    global flag
    flag = 0
    Buzz.stop()
    GPIO.output(ALedPin,GPIO.LOW)
    GPIO.output(BLedPin,GPIO.LOW)
```

The function Off() defines "flag=0" so as to exit the threads **ledWork** and **BuzzWork** and then turn off the buzzer and the LED.

```
def main():
    lastState=0
    while True:
        currentState =GPIO.input(switchPin)
        if currentState == 1 and lastState == 0:
            on()
        elif currentState == 0 and lastState == 1:
            off()
        lastState=currentState
```

Main() contains the whole process of the program: firstly read the value of the slide switch; if the toggle switch is toggled to the right (the reading is 1), the function on() is called, the buzzer is driven to emit sounds and the the red and the green LEDs blink. Otherwise, the buzzer and the LED don't work.

Phenomenon Picture



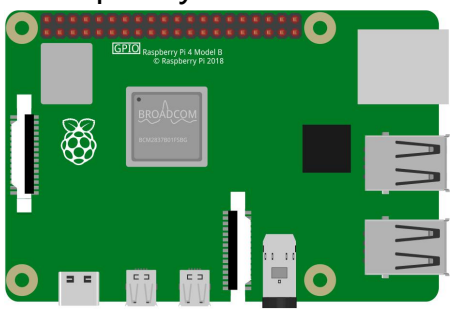
3.1.11 Morse Code Generator

Introduction

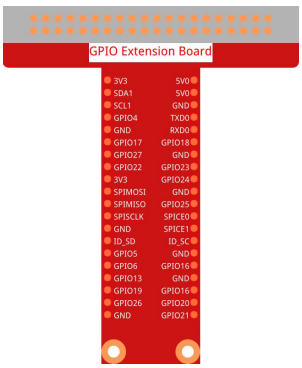
In this lesson, we'll make a Morse code generator, where you type in a series of English letters in the Raspberry Pi to make it appear as Morse code.

Components

1 * Raspberry Pi



1 * T-Extension Board



1 * Active Buzzer

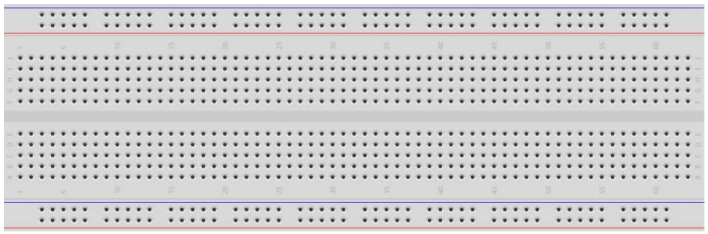
1 * Resistor(1kΩ)

1 * 40-pin Cable

2 * Resistor(220Ω)

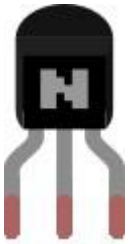
Several Jumper Wires

1 * Breadboard



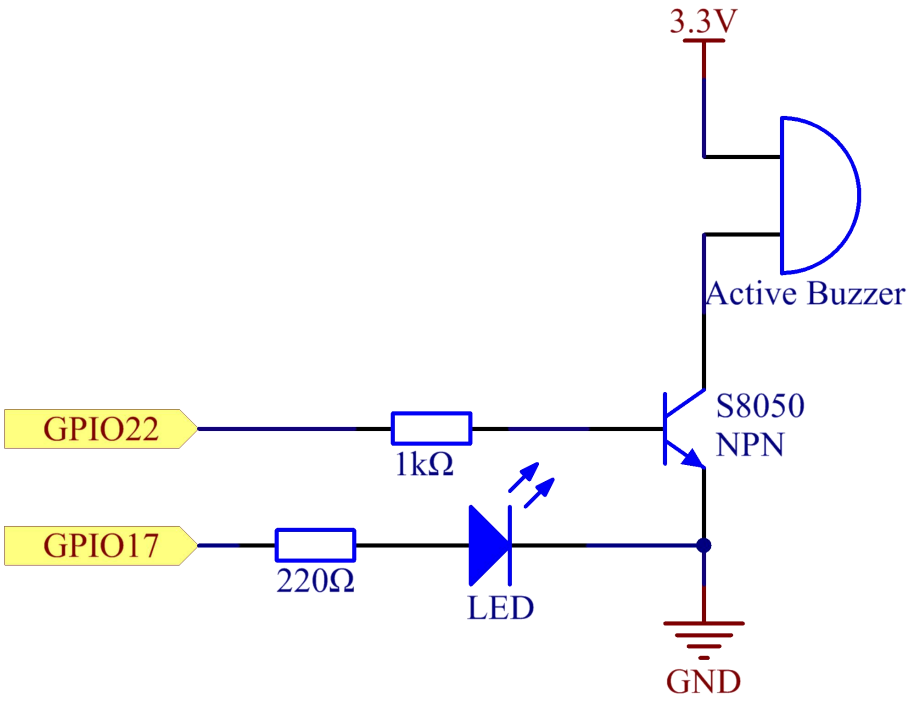
1 * LED

1 * S85050 NPN Transistor



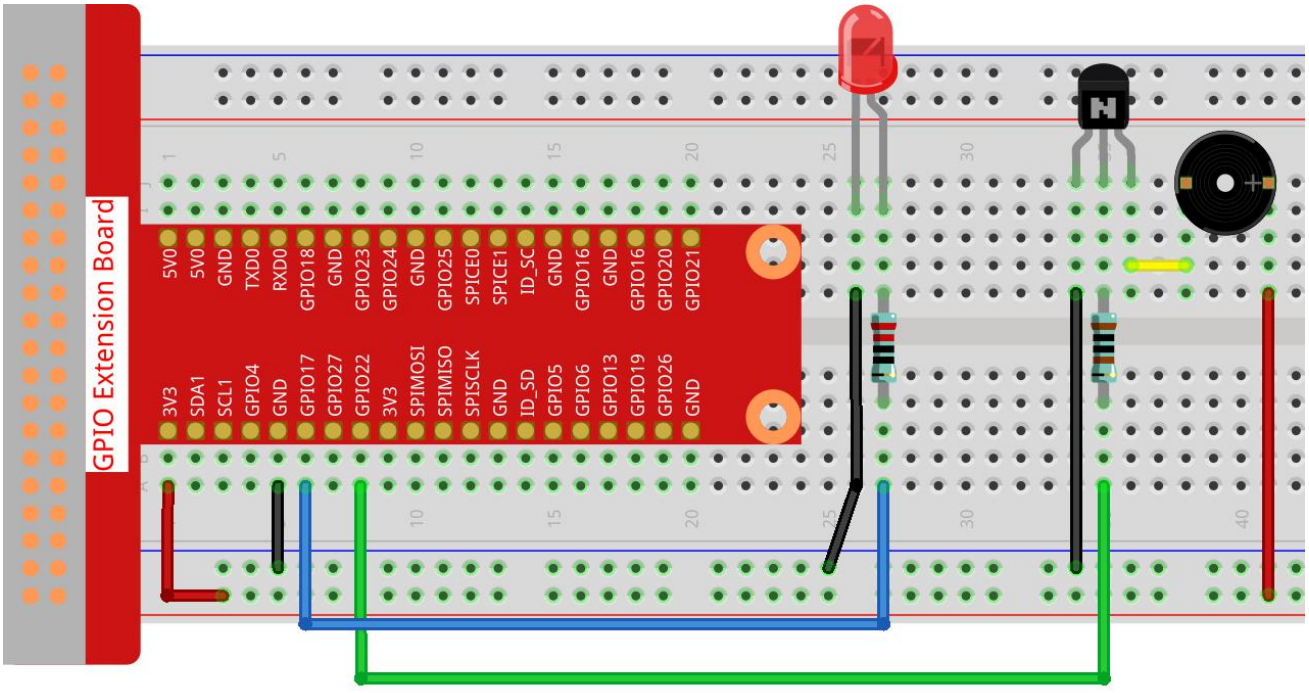
Schematic Diagram

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



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/3.1.11/
```

Step 3: Compile the code.

```
gcc 3.1.11_MorseCodeGenerator.c -lwiringPi
```

Step 4: Run the executable file above.

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

After the program runs, type a series of characters, and the buzzer and the LED will send the corresponding Morse code signals.

Code Explanation

```
struct MORSE{
    char word;
    unsigned char *code;
};

struct MORSE morseDict[]=
{
    {'A',"01"}, {'B',"1000"}, {'C',"1010"}, {'D',"100"}, {'E',"0"},
    {'F',"0010"}, {'G',"110"}, {'H',"0000"}, {'I',"00"}, {'J',"0111"},
    {'K',"101"}, {'L',"0100"}, {'M',"11"}, {'N',"10"}, {'O',"111"},
    {'P',"0110"}, {'Q',"1101"}, {'R',"010"}, {'S',"000"}, {'T',"1"},
    {'U',"001"}, {'V',"0001"}, {'W',"011"}, {'X',"1001"}, {'Y',"1011"},
    {'Z',"1100"},{'1',"01111"}, {'2',"00111"}, {'3',"00011"}, {'4',"00001"},
    {'5',"00000"},{'6',"10000"}, {'7',"11000"}, {'8',"11100"}, {'9',"11110"},
    {'0',"11111"},{'?',"001100"}, {'/',"10010"}, {',',"110011"}, {'.',"010101"},
    {';',"101010"},{'!',"101011"}, {'@',"011010"}, {':',"111000"}
};
```

This structure MORSE is the dictionary of the Morse code, containing characters A-Z, numbers 0-9 and marks “?” “/” “:” “,” “.” “;” “!” “@” .

```
char *lookup(char key,struct MORSE *dict,int length)
{
    for (int i=0;i<length;i++)
    {
        if(dict[i].word==key){
            return dict[i].code;
```

```
    }
  }
}
```

The function **lookup()** works by "checking the dictionary". Define a **key,** search the same words as **key** in the structure **morseDict** and return the corresponding information— "**code**" of the certain word.

```
void on(){
  digitalWrite(ALedPin,HIGH);
  digitalWrite(BeepPin,HIGH);
}
```

Create a function on() to start the buzzer and the LED.

```
void off(){
  digitalWrite(ALedPin,LOW);
  digitalWrite(BeepPin,LOW);
}
```

The function off() turns off the buzzer and the LED.

```
void beep(int dt){
  on();
  delay(dt);
  off();
  delay(dt);
}
```

Define a function beep() to make the buzzer and the LED emit sounds and blink in a certain interval of **dt**.

```
void morsecode(char *code){
  int pause = 250;
  char *point = NULL;
  int length = sizeof(morseDict)/sizeof(morseDict[0]);
  for (int i=0;i<strlen(code);i++)
  {
    point=lookup(code[i],morseDict,length);
    for (int j=0;j<strlen(point);j++){
      if (point[j]=='0')
      {
        beep(pause/2);
      }else if(point[j]=='1')
```

```
        {
            beep(pause);
        }
        delay(pause);
    }
  }
}
```

The function morsecode() is used to process the Morse code of input characters by making the "1" of the code keep emitting sounds or lights and the "0"shortly emit sounds or lights, ex., input "SOS", and there will be a signal containing three short three long and then three short segments " · · · - - - · · · ".

```
int toupper(int c)
{
  if ((c >= 'a') && (c <= 'z'))
    return c + ('A' - 'a');
  return c;
}
char *strupr(char *str)
{
  char *orign=str;
  for (; *str!='\0'; str++)
    *str = toupper(*str);
return orign;
}
```

Before coding, you need to unify the letters into capital letters.

```
void main(){
  setup();
  char *code;
  int length=8;
  code = (char*)malloc(sizeof(char)*length);
  while (1){
    printf("Please input the messenger:");
    scanf("%s",code);
    code=strupr(code);
    printf("%s\n",code);
    morsecode(code);
  }
}
```

When you type the relevant characters with the keyboard, code=strupr(code) will convert the input letters to their capital form.

Printf() then prints the clear text on the computer screen, and the morsecod() function causes the buzzer and the LED to emit Morse code.

Note that the length of the input character mustn't exceed the **length** (can be revised).

➢ For Python Language Users

Step 2: Open the code file.

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

Step 3: Run.

```
sudo python3 3.1.11_MorseCodeGenerator.py
```

After the program runs, type a series of characters, and the buzzer and the LED will send the corresponding Morse code signals.

Code Explanation

```
MORSECODE = {
  'A':'01', 'B':'1000', 'C':'1010', 'D':'100', 'E':'0', 'F':'0010', 'G':'110',
  'H':'0000', 'I':'00', 'J':'0111', 'K':'101', 'L':'0100', 'M':'11', 'N':'10',
  'O':'111', 'P':'0110', 'Q':'1101', 'R':'010', 'S':'000', 'T':'1',
  'U':'001', 'V':'0001', 'W':'011', 'X':'1001', 'Y':'1011', 'Z':'1100',
  '1':'01111', '2':'00111', '3':'00011', '4':'00001', '5':'00000',
  '6':'10000', '7':'11000', '8':'11100', '9':'11110', '0':'11111',
  '?':'001100', '/':'10010', ',':'110011', '.':'010101', ';':'101010',
  '!':'101011', '@':'011010', ':':'111000',
  }
```

This structure MORSE is the dictionary of the Morse code, containing characters A-Z, numbers 0-9 and marks "?" "/" ":" "," "." ";" "!" "@" .

```
def on():
  GPIO.output(BeepPin, 1)
  GPIO.output(ALedPin, 1)
```

The function on() starts the buzzer and the LED.

```
def off():
  GPIO.output(BeepPin, 0)
  GPIO.output(ALedPin, 0)
```

The function off() is used to turn off the buzzer and the LED.

```
def beep(dt):   # x for dalay time.
    on()
    time.sleep(dt)
    off()
    time.sleep(dt)
```

Define a function beep() to make the buzzer and the LED emit sounds and blink in a certain interval of **dt**.

```
def morsecode(code):
    pause = 0.25
    for letter in code:
        for tap in MORSECODE[letter]:
            if tap == '0':
                beep(pause/2)
            if tap == '1':
                beep(pause)
        time.sleep(pause)
```

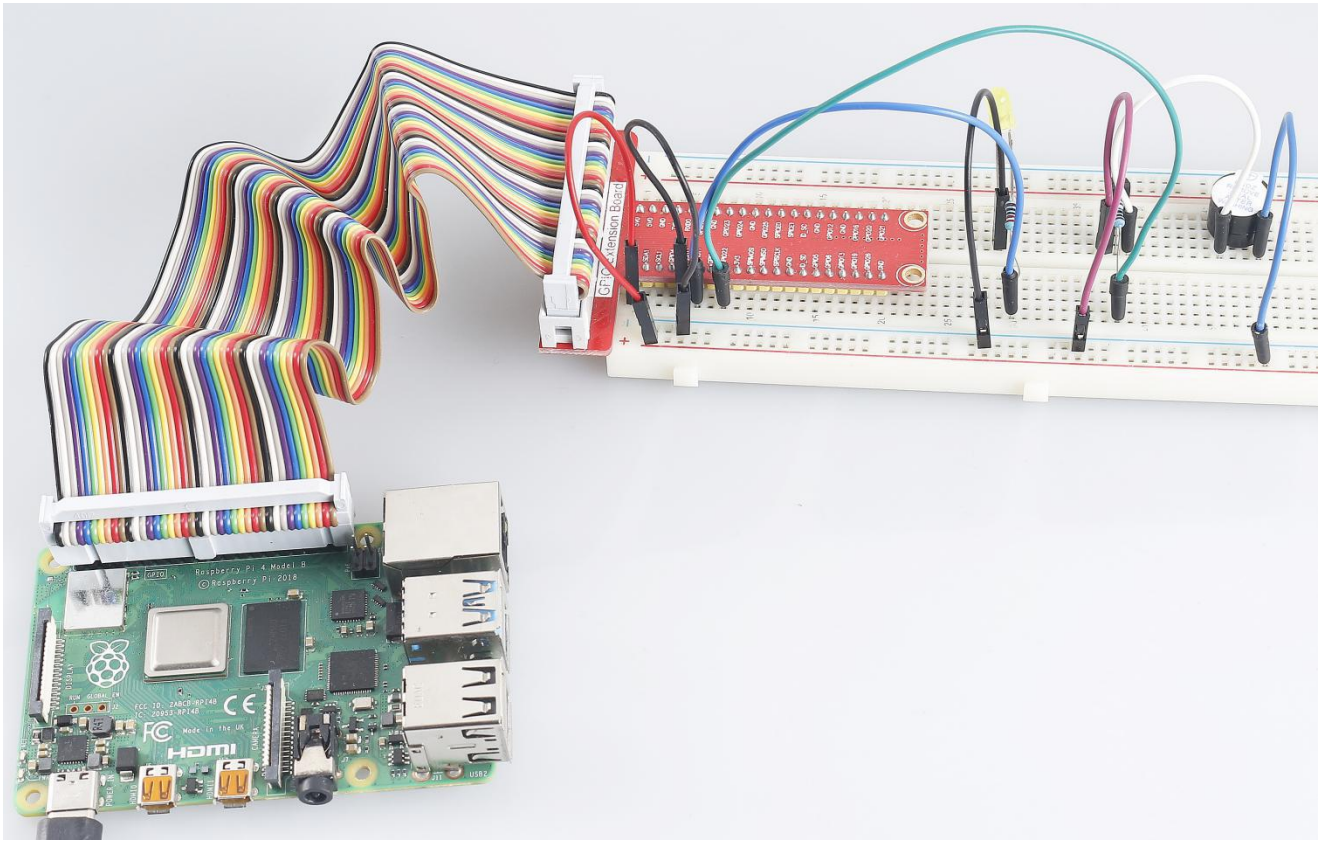
The function morsecode() is used to process the Morse code of input characters by making the "1" of the code keep emitting sounds or lights and the "0"shortly emit sounds or lights, ex., input "SOS", and there will be a signal containing three short three long and then three short segments " · · · - - - · · · ".

```
def main():
    while True:
        code=input("Please input the messenger:")
        code = code.upper()
        print(code)
        morsecode(code)
```

When you type the relevant characters with the keyboard, upper() will convert the input letters to their capital form.

Printf () then prints the clear text on the computer screen, and the morsecod() function causes the buzzer and the LED to emit Morse code.

Phenomenon Picture

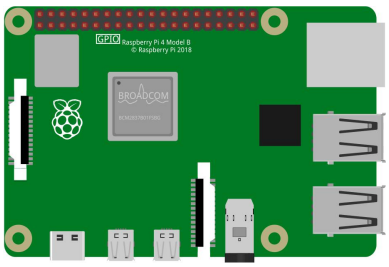
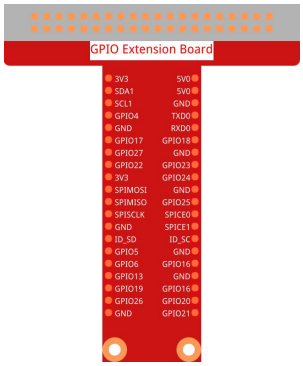
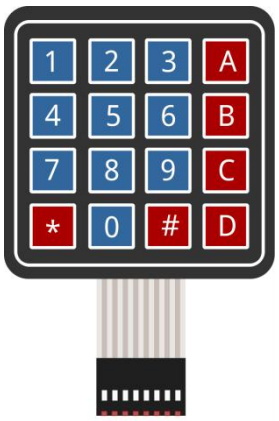
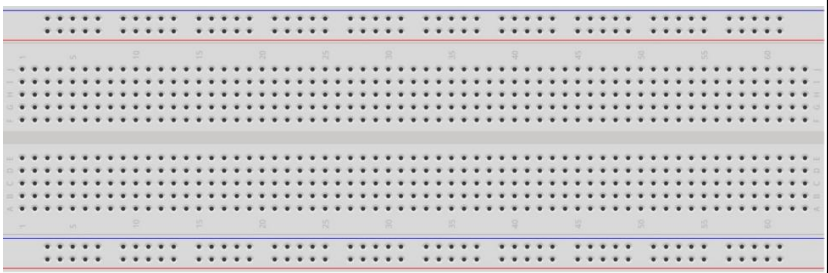
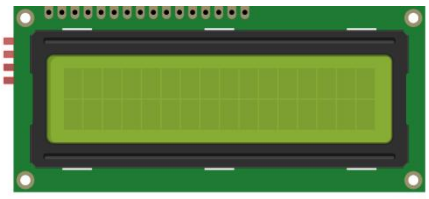


3.1.12 GAME– Guess Number

Introduction

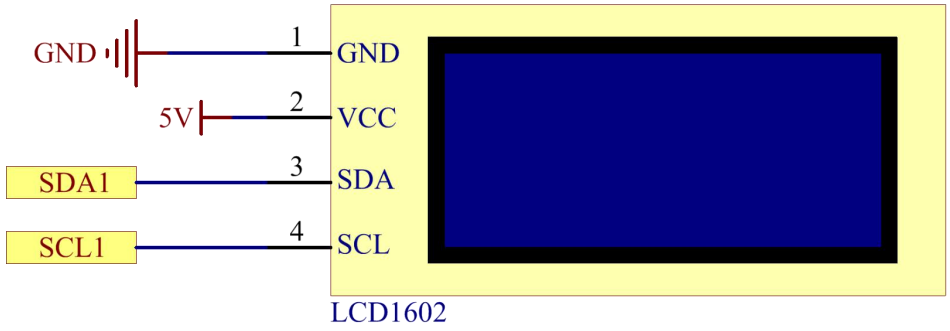
Guessing Numbers is a fun party game where you and your friends take turns inputting a number (0~99). The range will be smaller with the inputting of the number till a player answers the riddle correctly. Then the player is defeated and punished. For example, if the lucky number is 51 which the players cannot see, and the player ① inputs 50, the prompt of number range changes to 50~99; if the player ② inputs 70, the range of number can be 50~70; if the player ③ inputs 51, this player is the unlucky one. Here, we use keypad to input numbers and use LCD to output outcomes.

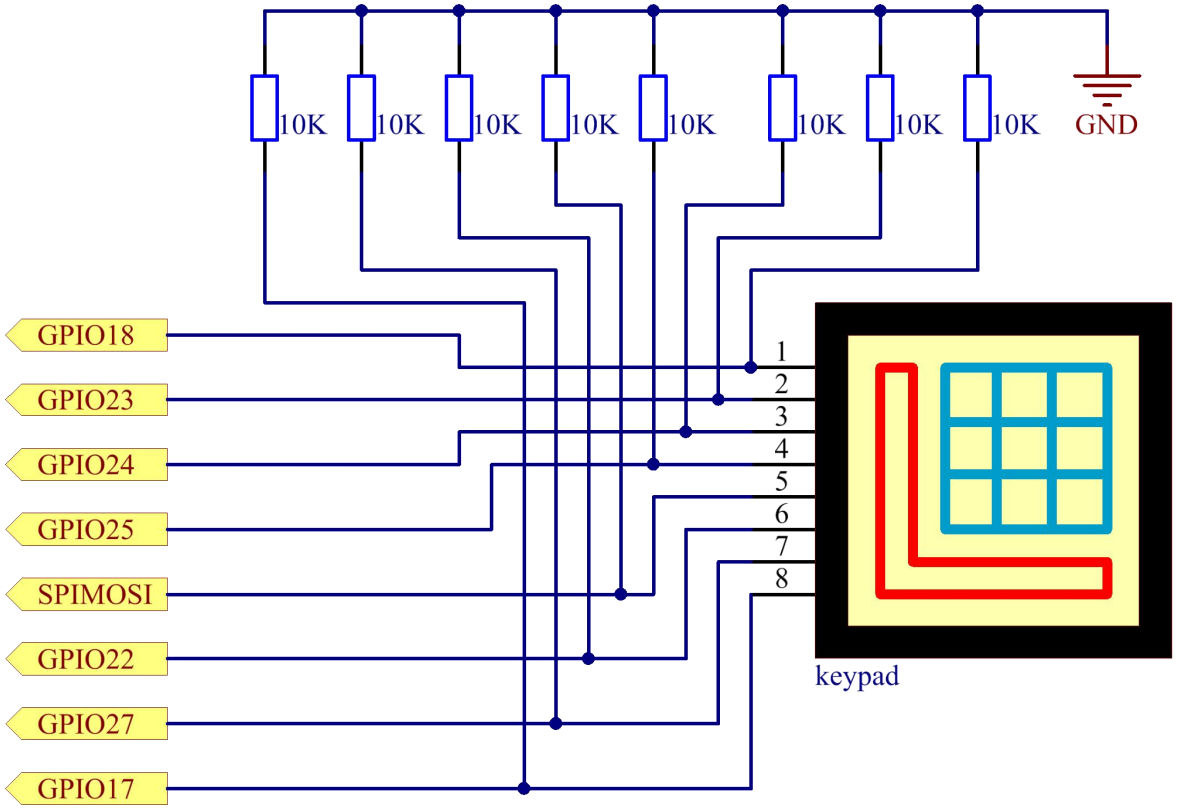
Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * Keypad
 
</td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>Several Jumper Wires
 
8 * Resistor 10KΩ
  </td></tr>
<tr><td colspan="2">1 * Breadboard
  </td><td>1 * I2C LCD1602
  </td></tr>
</table>

Schematic Diagram

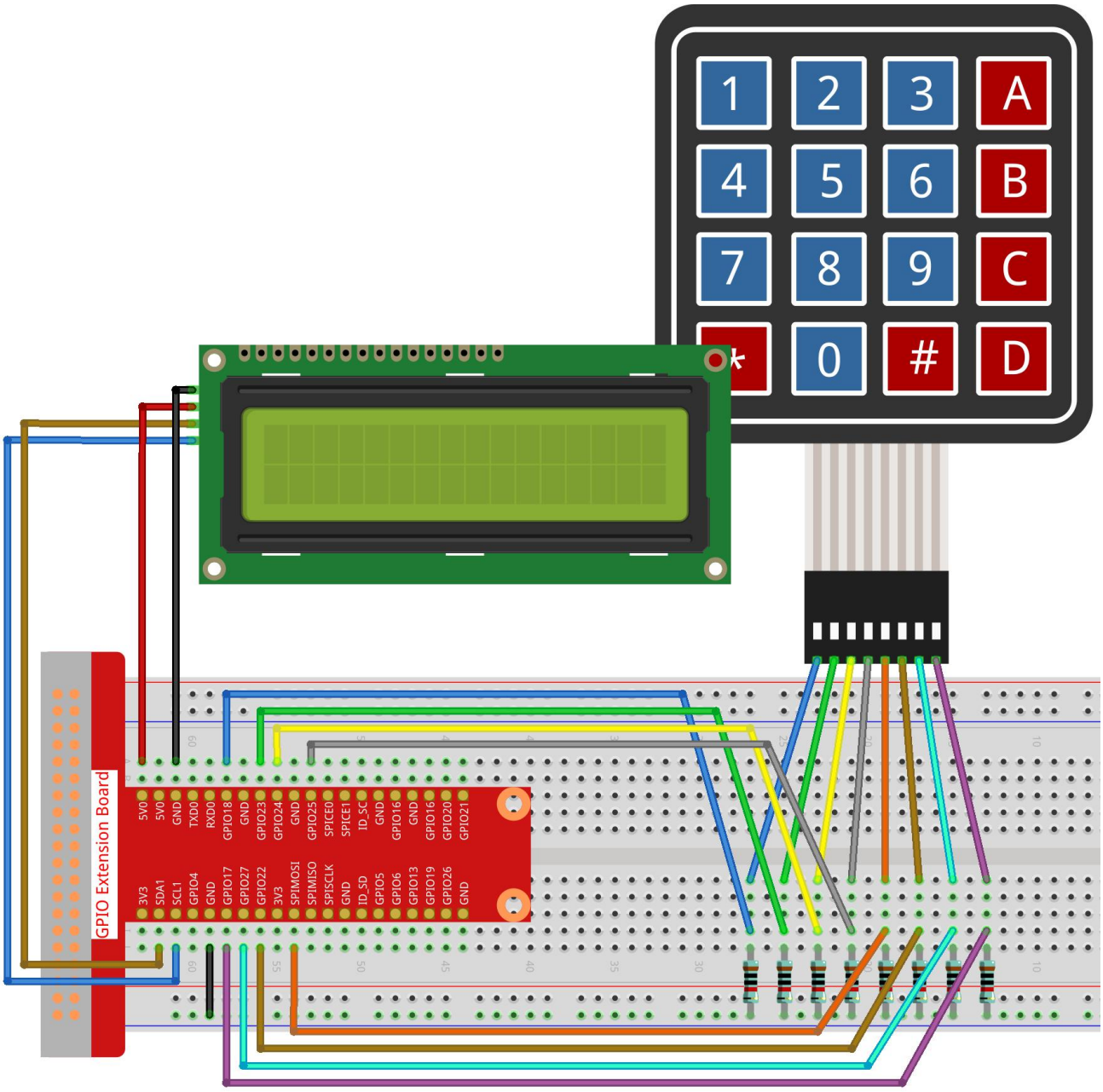
T-Board Name	physical	wiringPi	BCM
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO25	Pin 22	6	25
SPIMOSI	Pin 19	12	10
GPIO22	Pin 15	3	22
GPIO27	Pin 13	2	27
GPIO17	Pin 11	0	17
SDA1	Pin 3	SDA1(8)	SDA1(2)
SCL1	Pin 5	SCL1(9)	SDA1(3)





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/3.1.12/
```

Step 4: Compile.

```
gcc 3.1.12_GAME_GuessNumber.c -lwiringPi
```

Step 5: Run.

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

After the program runs, there displays the initial page on the LCD:

```
Welcome!
Press A to go!
```

Press 'A', and the game will start and the game page will appear on the LCD.

```
Enter number:
0 <point< 99
```

A random number '**point**' is produced but not displayed on the LCD when the game starts, and what you need to do is to guess it. The number you have typed appears at the end of the first line till the final calculation is finished. (Press 'D' to start the comparation, and if the input number is larger than **10**, the automatic comparation will start.)

The number range of 'point' is displayed on the second line. And you must type the number within the range. When you type a number, the range narrows; if you got the lucky number luckily or unluckily, there will appear "You've got it!"

Code Explanation

At the beginning part of the code are the functional functions of **keypad** and **I2C LCD1602**. You can learning more details about them in **1.1.7 I2C LCD1602** and **2.1.5 Keypad.**

Here, what we need to know is as follows:

```
/**************************************/
//Start from here
/**************************************/
void init(void){
    fd = wiringPiI2CSetup(LCDAddr);
    lcd_init();
    lcd_clear();
    for(int i=0 ; i<4 ; i++) {
        pinMode(rowPins[i], OUTPUT);
        pinMode(colPins[i], INPUT);
    }
    lcd_clear();
    write(0, 0, "Welcome!");
    write(0, 1, "Press A to go!");
```

```
}
```

This function is used to initially define **I2C LCD1602** and **Keypad** and to display "Welcome!" and "Press A to go!".

```
void init_new_value(void){
    srand(time(0));
    pointValue = rand()%100;
    upper = 99;
    lower = 0;
    count = 0;
    printf("point is %d\n",pointValue);
}
```

The function produces the random number '**point**' and resets the range hint of the point.

```
bool detect_point(void){
    if(count > pointValue){
        if(count < upper){
            upper = count;
        }
    }
    else if(count < pointValue){
        if(count > lower){
            lower = count;
        }
    }
    else if(count = pointValue){
        count = 0;
        return 1;
    }
    count = 0;
    return 0;
}
```

detect_point() compares the input number with the produced "point". If the comparing outcome is that they are not same, **count** will assign values to **upper** and **lower** and return '**0**'; otherwise, if the outcome indicates they are same, there returns '**1**'.

```
void lcd_show_input(bool result){
    char *str=NULL;
```

```
    str =(char*)malloc(sizeof(char)*3);
    lcd_clear();
    if (result == 1){
        write(0,1,"You've got it!");
        delay(5000);
        init_new_value();
        lcd_show_input(0);
        return;
    }
    write(0,0,"Enter number:");
    Int2Str(str,count);
    write(13,0,str);
    Int2Str(str,lower);
    write(0,1,str);
    write(3,1,"<Point<");
    Int2Str(str,upper);
    write(12,1,str);
}
```

This function works for displaying the game page. Pay attention to the function **Int2Str(str,count)**, it converts these variables **count**, **lower**, and **upper** from **integer** to **character string** for the correct display of **lcd**.

```
int main(){
    unsigned char pressed_keys[BUTTON_NUM];
    unsigned char last_key_pressed[BUTTON_NUM];
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    init_new_value();
    while(1){
        keyRead(pressed_keys);
        bool comp = keyCompare(pressed_keys, last_key_pressed);
        if (!comp){
            if(pressed_keys[0] != 0){
                bool result = 0;
                if(pressed_keys[0] == 'A'){
                    init_new_value();
```

```
                lcd_show_input(0);
            }
            else if(pressed_keys[0] == 'D'){
                result = detect_point();
                lcd_show_input(result);
            }
            else if(pressed_keys[0] >='0' && pressed_keys[0] <= '9'){
                count = count * 10;
                count = count + (pressed_keys[0] - 48);
                if (count>=10){
                    result = detect_point();
                }
                lcd_show_input(result);
            }
        }
        keyCopy(last_key_pressed, pressed_keys);
    }
    delay(100);
  }
  return 0;
}
```

Main() contains the whole process of the program, as show below:

1) Initialize **I2C LCD1602** and **Keypad**.
2) Use **init_new_value()** to create a random number **0-99**.
3) Judge whether the button is pressed and get the button reading.
4) If the button 'A' is pressed, a random number **0-99** will appear then the game starts.
5) If the button 'D' is detected to have been pressed, the program will enter into the outcome judgement and will display the outcome on the LCD. This step helps that you can also judge the outcome when you press only one number and then the button 'D'.
6) If the button **0-9** is pressed, the value of **count** will be changed; if the **count** is larger than **10**, then the judgement starts.
7) The changes of the game and its values are displayed on **LCD1602**.

➢ For Python Language Users

Step 3: Change directory.

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

Step 4: Run.

```
sudo python3 3.1.12_GAME_GuessNumber.py
```

After the program runs, there displays the initial page on the LCD:

```
Welcome!
Press A to go!
```

Press 'A', and the game will start and the game page will appear on the LCD.

```
Enter number:
0 <point< 99
```

A random number '**point**' is produced but not displayed on the LCD when the game starts, and what you need to do is to guess it. The number you have typed appears at the end of the first line till the final calculation is finished. (Press 'D' to start the comparation, and if the input number is larger than **10**, the automatic comparation will start.)

The number range of 'point' is displayed on the second line. And you must type the number within the range. When you type a number, the range narrows; if you got the lucky number luckily or unluckily, there will appear "You've got it!"

Code Explanation

At the beginning part of the code are the functional functions of **keypad** and **I2C LCD1602**. You can learning more details about them in **1.1.7 I2C LCD1602** and **2.1.5 Keypad.**

Here, what we need to know is as follows:

```
def init_new_value():
    global pointValue,upper,count,lower
    pointValue = random.randint(0,99)
    upper = 99
    lower = 0
    count = 0
    print('point is %d' %(pointValue))
```

The function produces the random number '**point**' and resets the range hint of the point.

```
def detect_point():
    global count,upper,lower
    if count > pointValue:
        if count < upper:
            upper = count
    elif count < pointValue:
        if count > lower:
            lower = count
    elif count == pointValue:
        count = 0
        return 1
    count = 0
    return 0
```

detect_point() compares the input number (**count**) with the produced "**point**". If the comparing outcome is that they are not same, **count** will assign values to **upper** and **lower** and return '**0**'; otherwise, if the outcome indicates they are same, there returns '**1**'.

```
def lcd_show_input(result):
    LCD1602.clear()
    if result == 1:
        LCD1602.write(0,1,'You have got it!')
        time.sleep(5)
        init_new_value()
        lcd_show_input(0)
        return
    LCD1602.write(0,0,'Enter number:')
    LCD1602.write(13,0,str(count))
    LCD1602.write(0,1,str(lower))
    LCD1602.write(3,1,' < Point < ')
    LCD1602.write(13,1,str(upper))
```

This function works for displaying the game page.

str(count): Because **write()** can only support the data type — **character string**, **str()** is needed to convert the **number** into **string**.

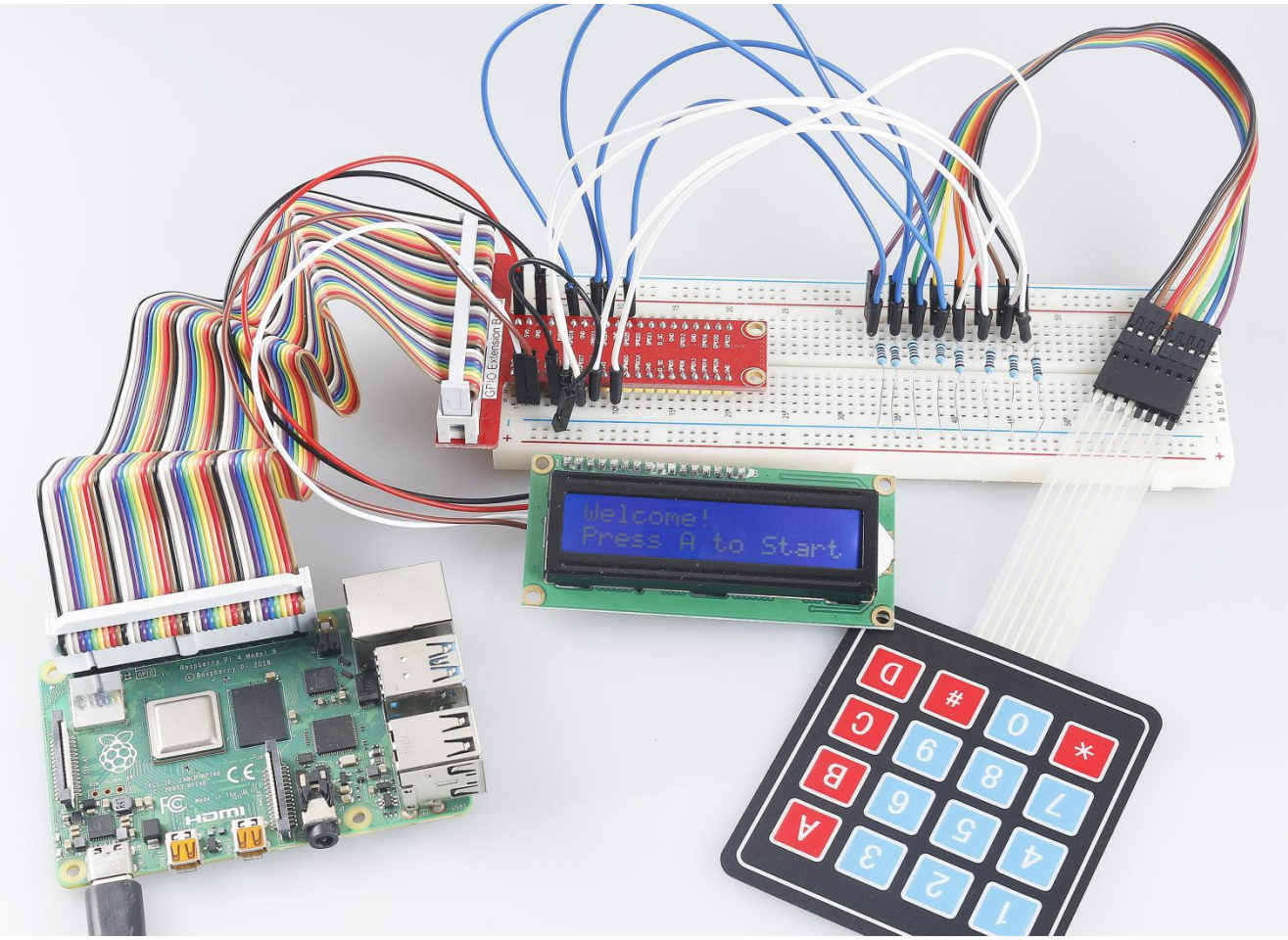
```
def loop():
    global keypad, last_key_pressed,count
    while(True):
        result = 0
        pressed_keys = keypad.read()
```

```
    if len(pressed_keys) != 0 and last_key_pressed != pressed_keys:
        if pressed_keys == ["A"]:
            init_new_value()
            lcd_show_input(0)
        elif pressed_keys == ["D"]:
            result = detect_point()
            lcd_show_input(result)
        elif pressed_keys[0] in keys:
            if pressed_keys[0] in list(["A","B","C","D","#","*"]):
                continue
            count = count * 10
            count += int(pressed_keys[0])
            if count >= 10:
                result = detect_point()
            lcd_show_input(result)
        print(pressed_keys)
    last_key_pressed = pressed_keys
    time.sleep(0.1)
```

Main() contains the whole process of the program, as show below:

1) Initialize **I2C LCD1602** and **Keypad**.
2) Judge whether the button is pressed and get the button reading.
3) If the button '**A**' is pressed, a random number **0-99** will appear then the game starts.
4) If the button '**D**' is detected to have been pressed, the program will enter into the outcome judgement.
5) If the button **0-9** is pressed, the value of **count** will be changed; if the **count** is larger than **10**, then the judgement starts.
6) The changes of the game and its values are displayed on **LCD1602**.

Phenomenon Picture

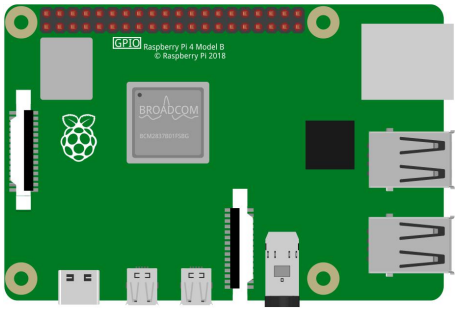
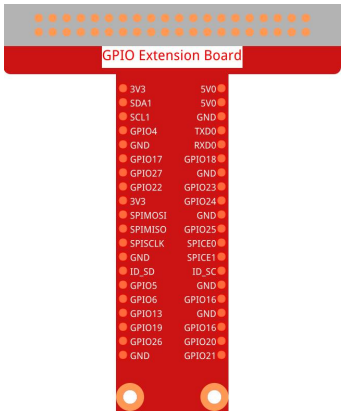
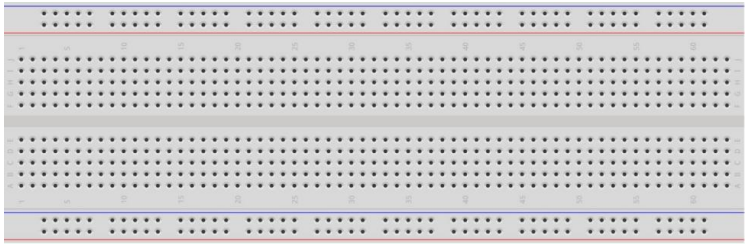


3.1.13 GAME– 10 Second

Introduction

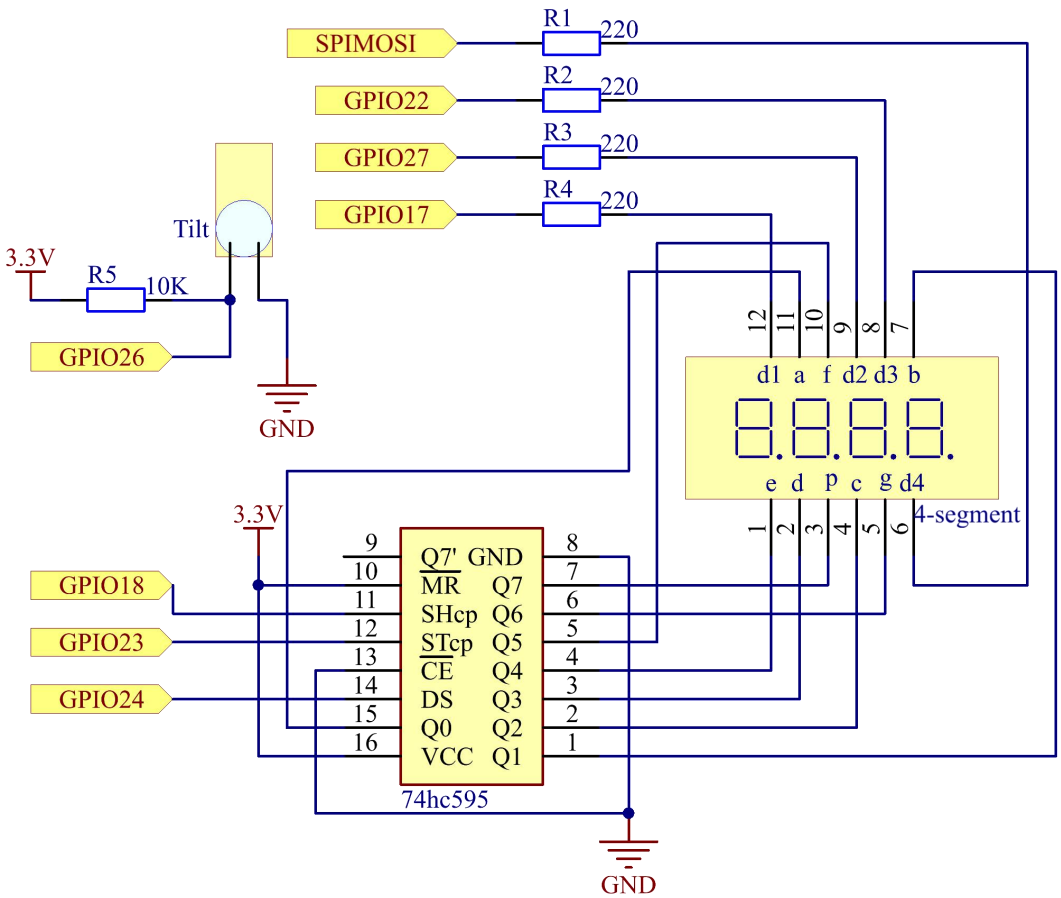
Next, follow me to make a game device to challenge your concentration. Tie the tilt switch to a stick to make a magic wand. Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. You can play the game with your friends to see who is the time wizard.

Components

1 * Raspberry Pi 	1 * T-Extension Board 	1 * 4-Digit 7-Segment Display 
		1 * 74HC595 
1 * 40-pin Cable 		1 * Tilt Switch 
		Several Jumper Wires 
1 * Breadboard 		4 * Resistor(220Ω) 
		1 * Resistor 10KΩ 

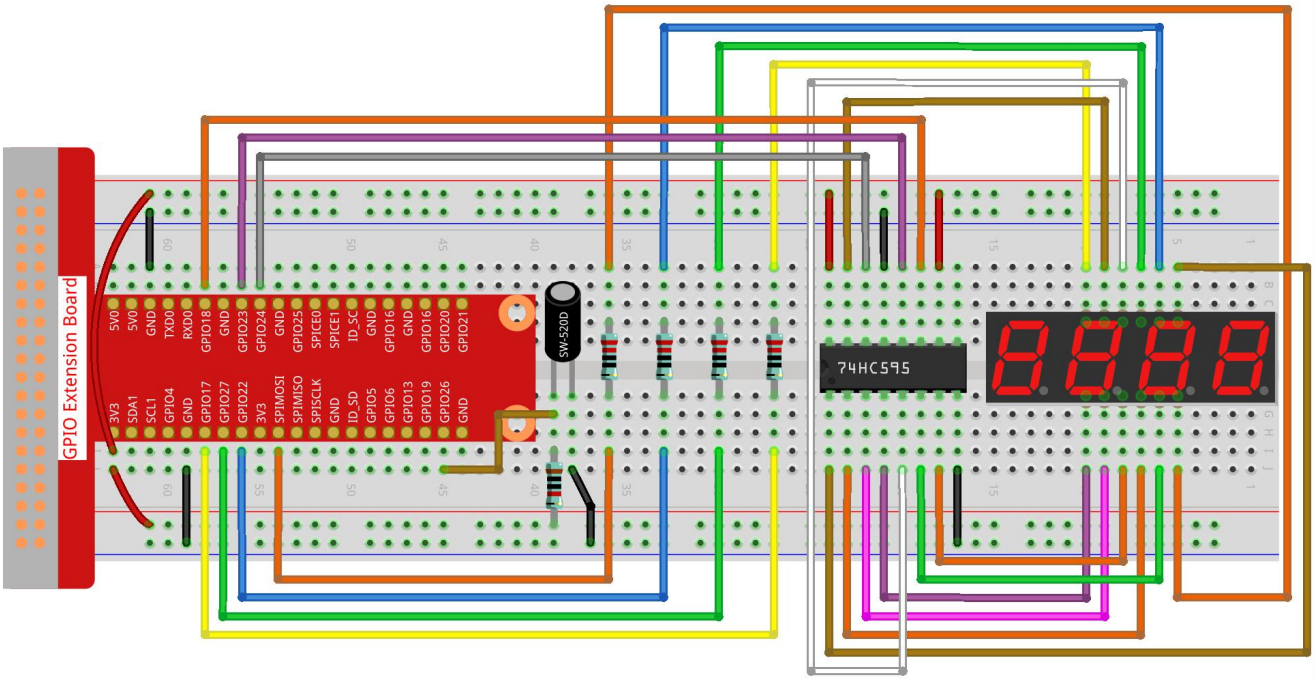
Schematic Diagram

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
GPIO18	Pin 12	1	18
GPIO23	Pin 16	4	23
GPIO24	Pin 18	5	24
GPIO26	Pin 37	25	26



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/3.1.13/
```

Step 3: Compile the code.

```
gcc 3.1.13_GAME_10Second.c -lwiringPi
```

Step 4: Run the executable file.

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

Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. Shake it one more time to start the next round of the game.

Code Explanation

```
void stateChange(){
    if (gameState == 0){
        counter = 0;
        delay(1000);
        ualarm(10000,10000);
    }else{
        alarm(0);
        delay(1000);
```

```
    }
    gameState = (gameState + 1)%2;
}
```

The game is divided into two modes:

gameState=0 is the "start" mode, in which the time is timed and displayed on the segment display, and the tilting switch is shaken to enter the "show" mode.

GameState =1 is the "show" mode, which stops the timing and displays the time on the segment display. Shaking the tilt switch again will reset the timer and restart the game.

```
void loop(){
    int currentState =0;
    int lastState=0;
    while(1){
        display();
        currentState=digitalRead(sensorPin);
        if((currentState==0)&&(lastState==1)){
            stateChange();
        }
        lastState=currentState;
    }
}
```

Loop() is the main function. First, the time is displayed on the 4-bit segment display and the value of the tilt switch is read. If the state of the tilt switch has changed, stateChange() is called.

➢ 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 python3 3.1.13_GAME_10Second.py
```

Shake the wand, the 4-digit segment display will start counting, shake again will let it stop counting. If you succeed in keeping the displayed count at **10.00**, then you win. Shake it one more time to start the next round of the game.

Code Explanation

```
def stateChange():
    global gameState
    global counter
    global timer1
    if gameState == 0:
        counter = 0
        time.sleep(1)
        timer()
    elif gameState ==1:
        timer1.cancel()
        time.sleep(1)
    gameState = (gameState+1)%2
```

The game is divided into two modes:

gameState=0 is the "start" mode, in which the time is timed and displayed on the segment display, and the tilting switch is shaken to enter the "show" mode.

GameState =1 is the "show" mode, which stops the timing and displays the time on the segment display. Shaking the tilt switch again will reset the timer and restart the game.

```
def loop():
    global counter
    currentState = 0
    lastState = 0
    while True:
        display()
        currentState=GPIO.input(sensorPin)
        if (currentState == 0) and (lastState == 1):
            stateChange()
        lastState=currentState
```

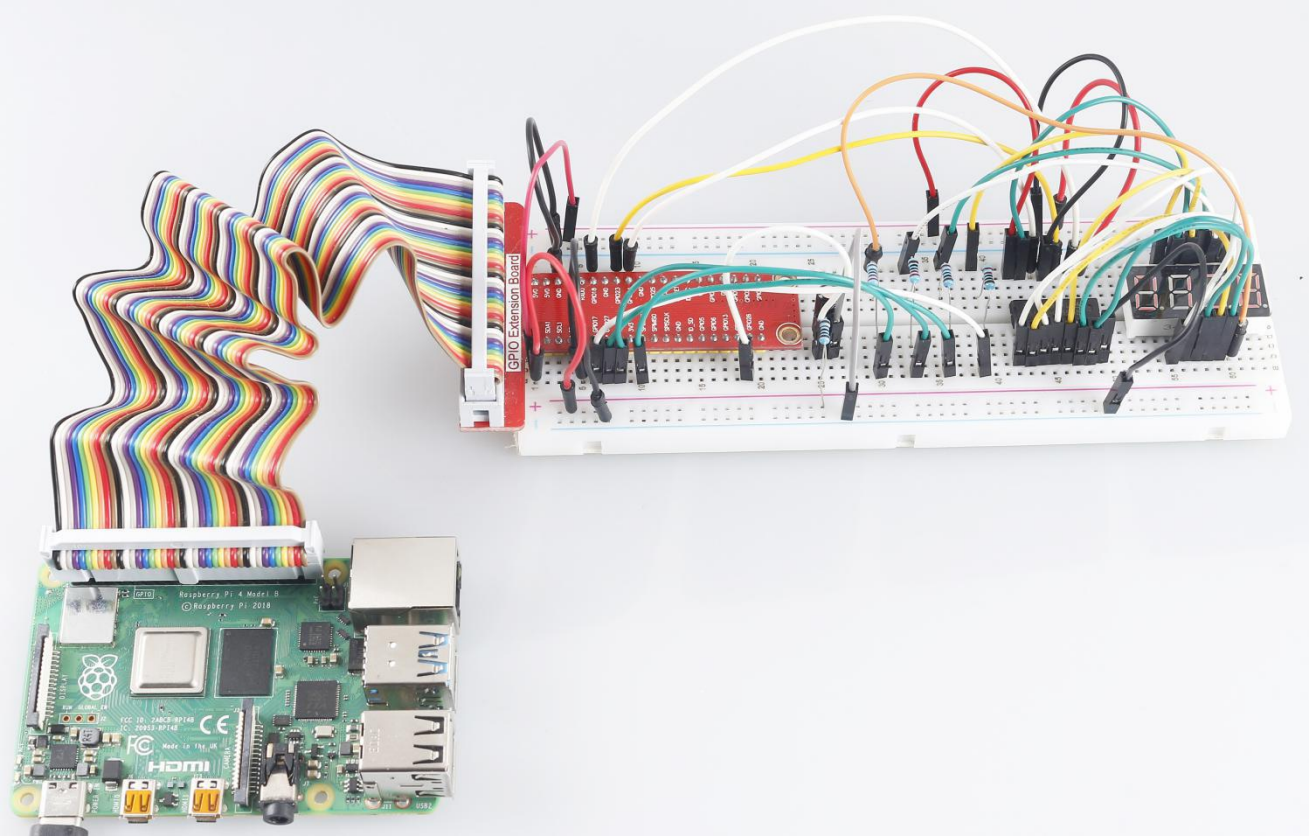
Loop() is the main function. First, the time is displayed on the 4-bit segment display and the value of the tilt switch is read. If the state of the tilt switch has changed, stateChange() is called.

```
def timer():
    global counter
    global timer1
    timer1 = threading.Timer(0.01, timer)
    timer1.start()
```

```
    counter += 1
```

After the interval reaches 0.01s, the timer function is called; add 1 to counter, and the timer is used again to execute itself repeatedly every 0.01s.

Phenomenon Picture



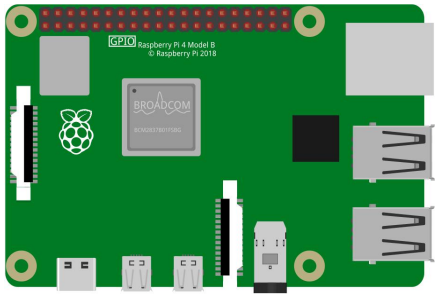
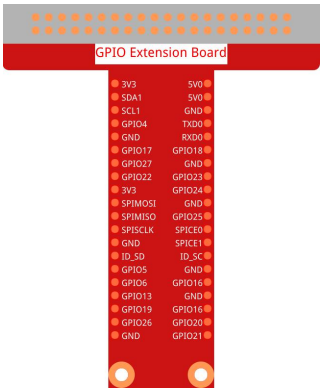
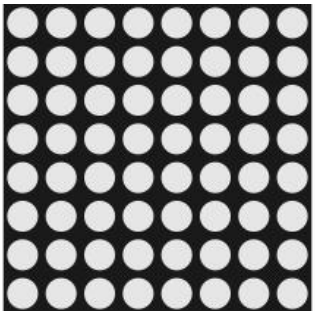
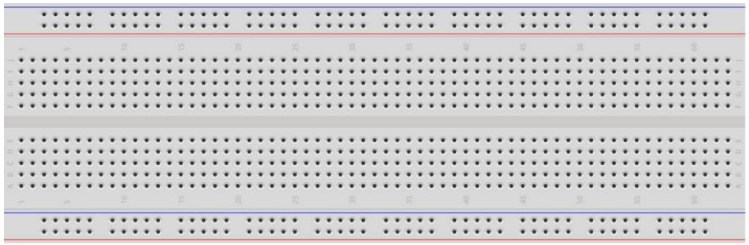
3.1.14 GAME– Not Not

Introduction

In this lesson, we will make an interesting game device, and we call it "Not Not". During the game, the dot matrix will refresh an arrow randomly. What you need to do is to press the button in the opposite direction of the arrow within a limited time. If the time is up, or if the button in the same direction as the arrow is pressed, you are out.

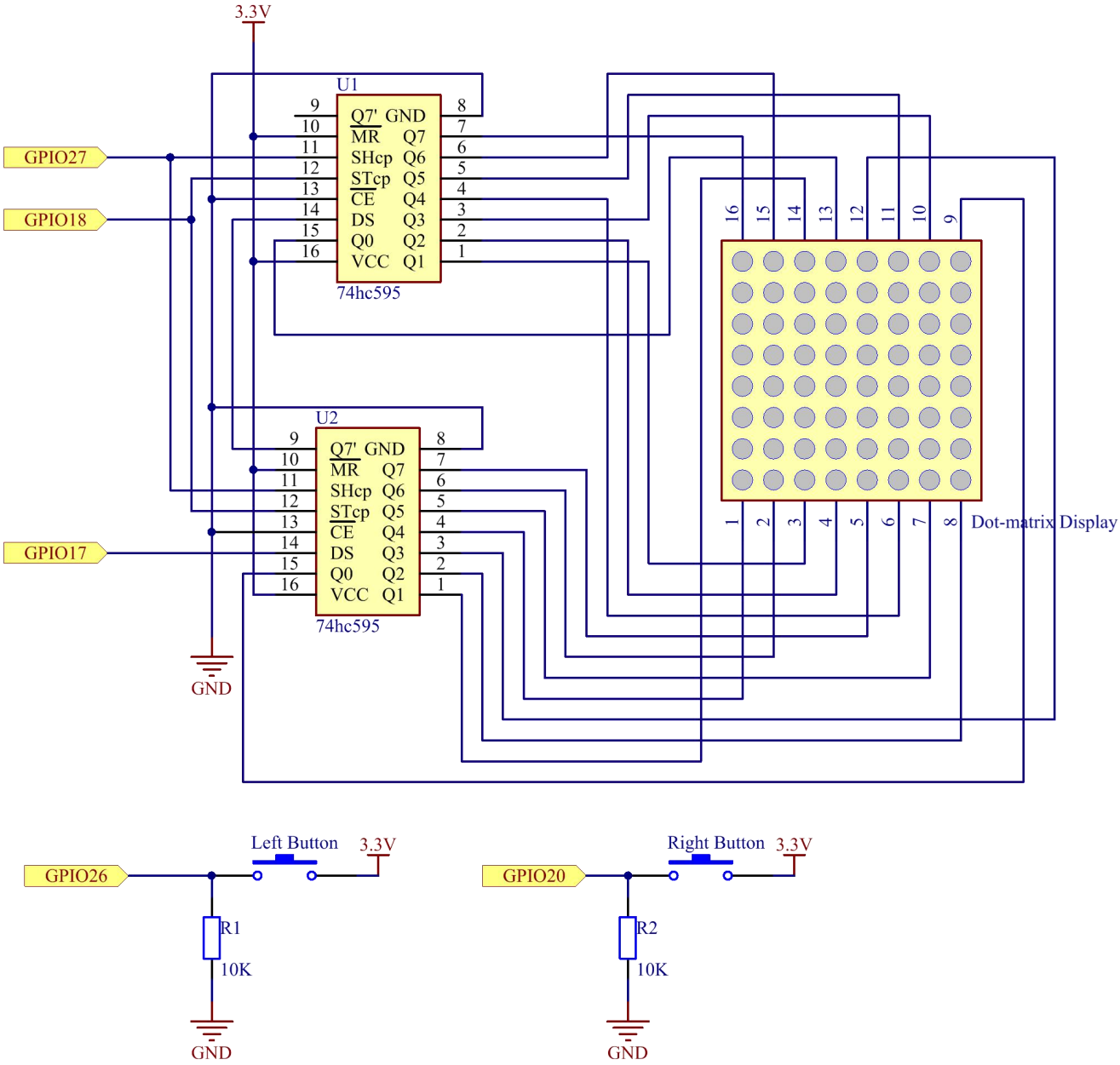
This game can really practice your reverse thinking, and now shall we have a try?

Components

<table>
<tr><td>1 * Raspberry Pi
  </td><td>1 * T-Extension Board
 
</td><td>1 * LED Dot Matrix
  </td></tr>
<tr><td colspan="2">1 * 40-pin Cable
  </td><td>Several Jumper Wires
  </td></tr>
<tr><td colspan="2" rowspan="2">1 * Breadboard
  </td><td>2 * Button
  </td></tr>
<tr><td>2 * 74HC595
 
</td></tr>
<tr><td colspan="2"></td><td>2 * Resistor 10KΩ
  </td></tr>
</table>

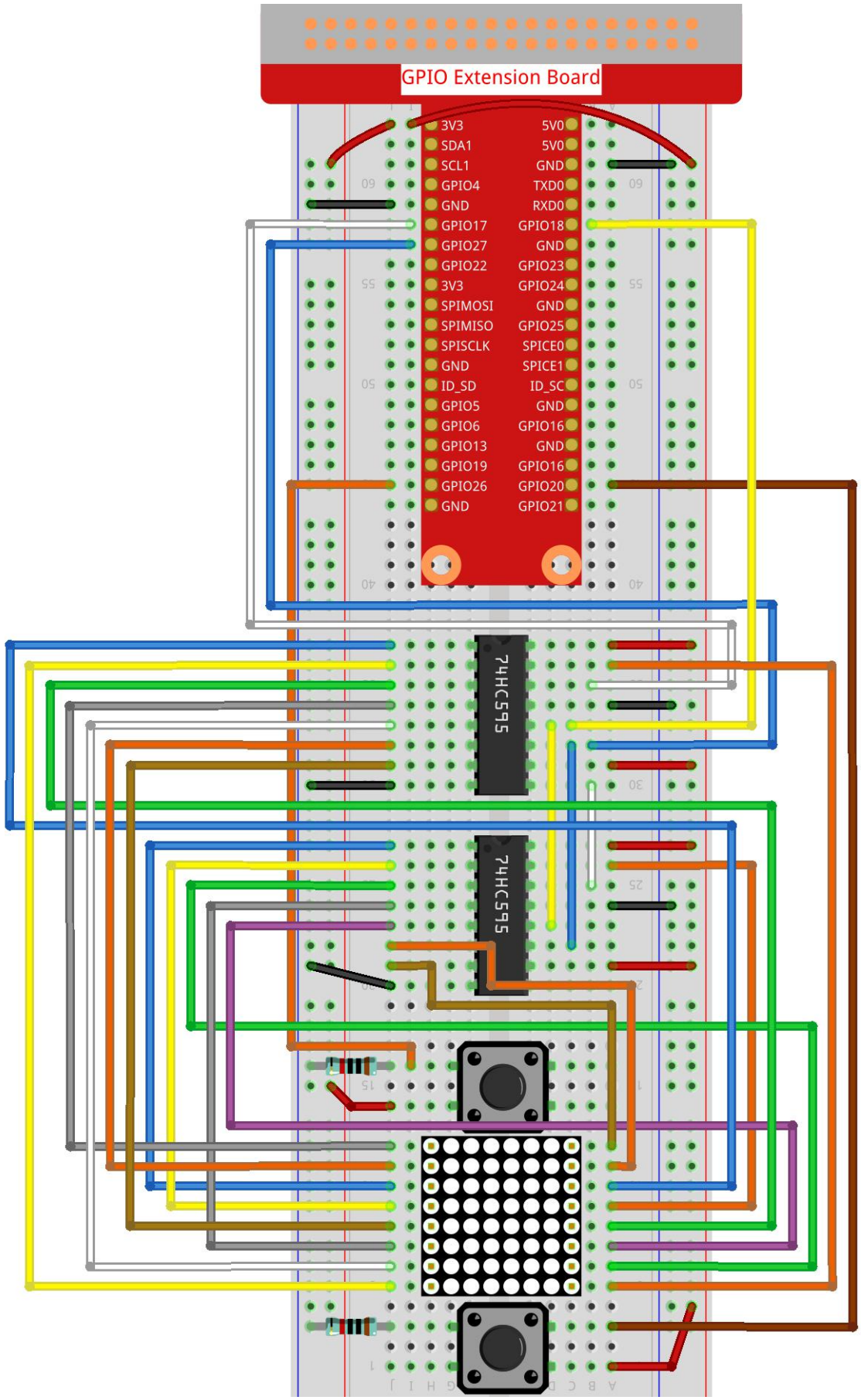
Schematic Diagram

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27
GPIO20	Pin 38	28	20
GPIO26	Pin 37	25	26



Experimental Procedures

Step 1: Build the circuit.



➢ For C Language Users

Step 5: Go to the folder of code.

```
cd /home/pi/davinci-kit-for-raspberry-pi/c/3.1.14/
```

Step 6: Compile.

```
gcc 3.1.14_GAME_NotNot.c -lwiringPi
```

Step 7: Run.

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

After the program starts, a left or right arrow will be refreshed at random on the dot matrix. What you need to do is to press the button in the opposite direction of the arrow within a limited time. Then "√" appears on the dot matrix. If the time is up, or if the button in the same direction as the arrow is pressed, you are out and the dot matrix displays "x". You can also add 2 new buttons or replace them with Joystick keys for up, down, left and right— 4 directions to increase the difficulty of the game.

Code Explanation

Based on **1.1.6 LED Dot Matrix**, this lesson adds **2** buttons to make an amusing game device. So, if you are not very familiar with the dot matrix, please refer to **1.1.6 LED Dot Matrix**.

The whole program process is as below:

1. Randomly select an arrow direction and generate **timer 1**.
2. Display the arrow image on the dot matrix.
3. Judge the button input. If the button is pressed or **timer 1** reminds time's up, judgement starts.
4. Display the image on the basis of a judging result; meanwhile, generate **timer 2**.
5. Rerun **step 1** when **timer 2** reminds time's up.

```
struct GLYPH{
    char *word;
    unsigned char code[8];
};

struct GLYPH arrow[2]=
{
    {"right",{0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF}},
```

```
    // {"down",{0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF}},
    // {"up",{0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF}},
    {"left",{0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF}}
};

struct GLYPH check[2]=
{
    {"wrong",{0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF}},
    {"right",{0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF}}
};
```

GLYPH structure works like a dictionary: the **word** attribute corresponds to the **key** on the dictionary; the **code** attribute corresponds to the **value**.

Here, code is used to store an array for dot matrix to display images (an 8x8 bit array).

Here, the array **arrow** can be used to display the arrow pattern in up, down, left and right directions on the LED dot matrix.

Now **down** and **up** are commented and uncomment them if needed.

The array **check** is used to display these two images: "×" and "√".

```
char *lookup(char *key,struct GLYPH *glyph,int length){
    for (int i=0;i<length;i++)
    {
        if(strcmp(glyph[i].word,key)==0){
            return glyph[i].code;
        }
    }
}
```

The function **lookup()** works by "checking the dictionary". Define a **key,** search the same words as **key** in the structure **GLYPH *glyph** and return the corresponding information— "**code**" of the certain word.

The function **Strcmp()** is used to compare the identity of two character strings **glyph[i].word** and **key**; if the identity is judged, return **glyph[i].code** (as shown).

```
void display(char *glyphCode){
    for(int i;i<8;i++){
        hc595_in(glyphCode[i]);
        hc595_in(0x80>>i);
        hc595_out();
```

```
    }
}
```

Display the specified pattern on the dot matrix.

```
void createGlyph(){
    srand(time(NULL));
    int i=rand()%(sizeof(arrow)/sizeof(arrow[0]));
    waypoint=arrow[i].word;
    stage="PLAY";
    alarm(2);
}
```

The function **createGlyph()** is used to randomly select a direction (the word attribute of an element in the array **arrow[]:** "**left**", "**right**"...). Set the stage as "PLAY" and start a 2-second alarm clock function.

srand(time(NULL)): Initializes random seeds that are from the system clock.

(sizeof(arrow)/sizeof(arrow[0])): Get the length of the array, the outcome is 2.

rand()%2: The remainder is **0** or **1,** gotten from dividing a generated random number by 2**.**

waypoint=arrow[i].word: The outcome should be "right" or "left".

```
void checkPoint(char *inputKey){
    alarm(0)==0;
    if(inputKey==waypoint||inputKey=="empty")
    {
        waypoint="wrong";
    }
    else{
        waypoint="right";
    }
    stage="CHECK";
    alarm(1);
}
```

checkPoint() is used to check the button input; if the button is not pressed or the button in the same direction as the arrow is pressed, the outcome of the waypoint is wrong and "**x**" appears on the dot matrix. Otherwise, the waypoint is right and dot matrix displays "√". Here the **stage** is **CHECK,** and there can be set a 1-second alarm clock function.

alarm() is also called "alarm clock", in which a timer can be set, and it sends **SIGALRM** signals to the progress when the defined time is up.

```
void getKey(){
  if (digitalRead(AButtonPin)==1&&digitalRead(BButtonPin)==0)
  {checkPoint("right");}
  else if (digitalRead(AButtonPin)==0&&digitalRead(BButtonPin)==1)
  {checkPoint("left");}
}
```

getKey() reads the states of the these two buttons; if the right button is pressed, the parameter of the function checkPoint() is **right** and if the left button is pressed, the parameter is **left**.

```
void timer(){
  if (stage=="PLAY"){
    checkPoint("empty");
  }
  else if(stage=="CHECK"){
    createGlyph();
  }
}
```

Previously, timer() was called when set as the alarm() time's up. Then under the "PLAY" mode, checkPoint() is to be called to judge the outcome. If the program is set to "CHECK" mode, the function createGlyph() should be called to select new patterns.

```
void main(){
  setup();
  signal(SIGALRM,timer);
  createGlyph();
  char *code = NULL;
  while(1){
    if (stage == "PLAY")
    {
      code=lookup(waypoint,arrow,sizeof(arrow)/sizeof(arrow[0]));
      display(code);
      getKey();
    }
    else if(stage == "CHECK")
    {
      code = lookup(waypoint,check,sizeof(check)/sizeof(check[0]));
      display(code);
```

```
        }
    }
}
```

The working of the function signal(SIGALRM,timer): calling the timer() function when a SIGALRM signal (generated by the alarm clock function alarm()) is received.

When the program starts, call createGlyph() one time at first and then start the loop.

In the loop: under PLAY mode, the dot matrix displays arrow patterns and check the button state; if under CHECK mode, what is displayed is "**x**" or "√".

➢ 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 python3 3.1.14_GAME_NotNot.py
```

After starting the program, on the dot matrix appears an arrow pointing to the right or the left. What you need to do is to press the button in the opposite direction of the arrow within a limited time. Then "√" appears on the dot matrix. If the time is up, or if the button in the same direction as the arrow is pressed, you are out and the dot matrix displays "x". You can also add 2 new buttons or replace them with Joystick keys for up, down, left and right— 4 directions to increase the difficulty of the game.

Code Explanation

Based on **1.1.6 LED Dot Matrix**, this lesson adds **2** buttons to make an amusing game device. So, if you are not very familiar with the dot matrix, please refer to **1.1.6 LED Dot Matrix**.

The whole program process is as below:

1. Randomly select an arrow direction and generate **timer 1**.
2. Display the corresponding arrow image on the dot matrix.
3. Judge the button input. If the button is pressed or **timer 1** reminds time's up, judgement starts.
4. Display the image on the basis of a judging result; meanwhile, generate **timer 2**.
5. Rerun **step 1** when **timer 2** reminds time's up.

```
def main():
```

```
    creatGlyph()
    while True:
        if stage == "PLAY":
            display(arrow[waypoint])
            getKey()
        elif stage == "CHECK":
            display(check[waypoint])
```

Main() contains the whole running process.

When the program starts, call createGlyph() one time at first and then start the loop.

In the loop: under PLAY mode, the dot matrix displays arrow patterns and check the button state; if under CHECK mode, what is displayed is "**x**" or "**√**".

```
arrow={
    #"down" :[0xFF,0xEF,0xC7,0xAB,0xEF,0xEF,0xEF,0xFF],
    #"up":[0xFF,0xEF,0xEF,0xEF,0xAB,0xC7,0xEF,0xFF],
    "right" : [0xFF,0xEF,0xDF,0x81,0xDF,0xEF,0xFF,0xFF],
    "left":[0xFF,0xF7,0xFB,0x81,0xFB,0xF7,0xFF,0xFF]
}
check={
    "wrong":[0xFF,0xBB,0xD7,0xEF,0xD7,0xBB,0xFF,0xFF],
    "right":[0xFF,0xFF,0xF7,0xEB,0xDF,0xBF,0xFF,0xFF]
}
```

Here, the **dictionary** arrow can be used to display the arrow pattern in up, down, left and right directions on the LED dot matrix.

Now down and up are commented and uncomment them if needed.

The **dictionary** check is used to display these two images: "×" and "√".

```
def display(glyphCode):
    for i in range(0, 8):
        hc595_shift(glyphCode[i])
        hc595_shift(0x80>>i)
        GPIO.output(RCLK, GPIO.HIGH)
        GPIO.output(RCLK, GPIO.LOW)
```

Display the specified pattern on the dot matrix.

```
def creatGlyph():
    global waypoint
    global stage
```

```
    global timerPlay
    waypoint=random.choice(list(arrow.keys()))
    stage = "PLAY"
    timerPlay = threading.Timer(2.0, timeOut)
    timerPlay.start()
```

The function **createGlyph()** is used to randomly select a direction (the word attribute of an element in the array **arrow[]:** "**left**", "**right**"...). Set the stage as "PLAY" and start a 2-second alarm clock function.

arrow.keys(): Select the keys "right"and "left" in the arrow array.

list(arrow.keys()): Combine these keys into an array.

random.choice(list(arrow.keys())): Randomly select an element in the array.

So, The outcome of **waypoint=random.choice(list(arrow.keys()))** should be "right" or "left".

```
def checkPoint(inputKey):
    global waypoint
    global stage
    global timerCheck
    if inputKey == "empty" or inputKey == waypoint:
        waypoint = "wrong"
    else:
        waypoint = "right"
    timerPlay.cancel()
    stage = "CHECK"
    timerCheck = threading.Timer(1.0, creatGlyph)
    timerCheck.start()
```

checkPoint() is to detect the current state of button input:

If no button is pressed or the button in the same direction as the the arrow is pressed, the assigned value of the **waypoint** is **wrong** and there displays "**x**" on the dot matrix.

Otherwise, the waypoint is right and "√" appears.

Now the stage is **CHECK** and start a 1-second timer **timerCheck** to call the function creatGlyph() in a second.

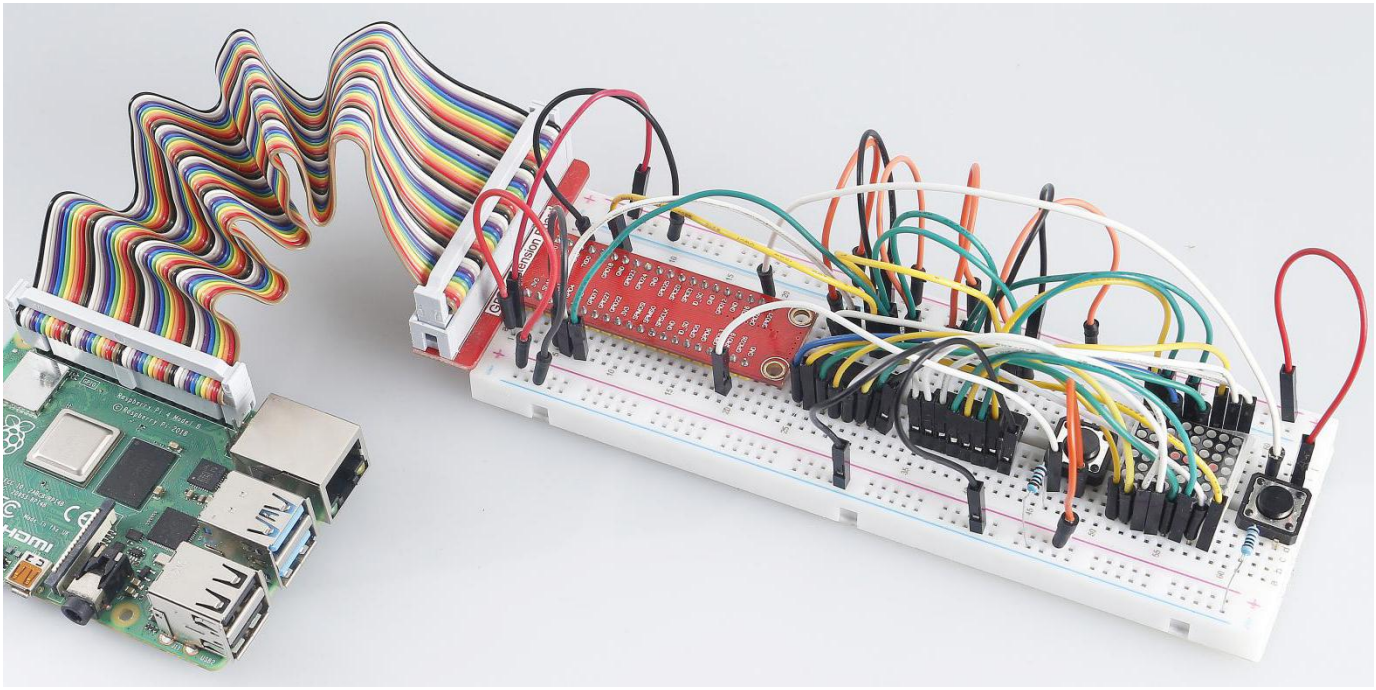
```
def timeOut():
    checkPoint("empty")
```

In the function timeout(), set the parameter of checkPoint() as "**empty**".

```
def getKey():
    if GPIO.input(AButtonPin)==1 and GPIO.input(BButtonPin)==0:
        checkPoint("right")
    elif GPIO.input(AButtonPin)==0 and GPIO.input(BButtonPin)==1:
        checkPoint("left")
```

getKey() reads the state of these two buttons, and if the right button is pressed, the parameter of checkPoint() is **right**; if the left button is pressed, the parameter is **left**.

Phenomenon Picture



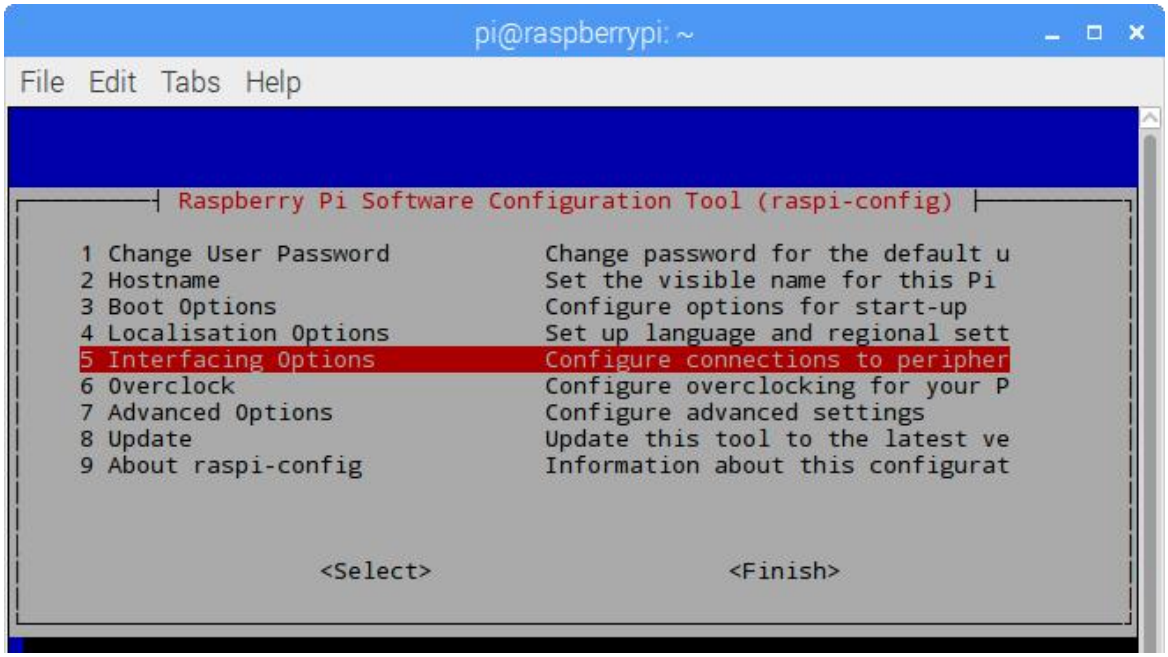
3.2 Appendix

3.2.1 I2C Configuration

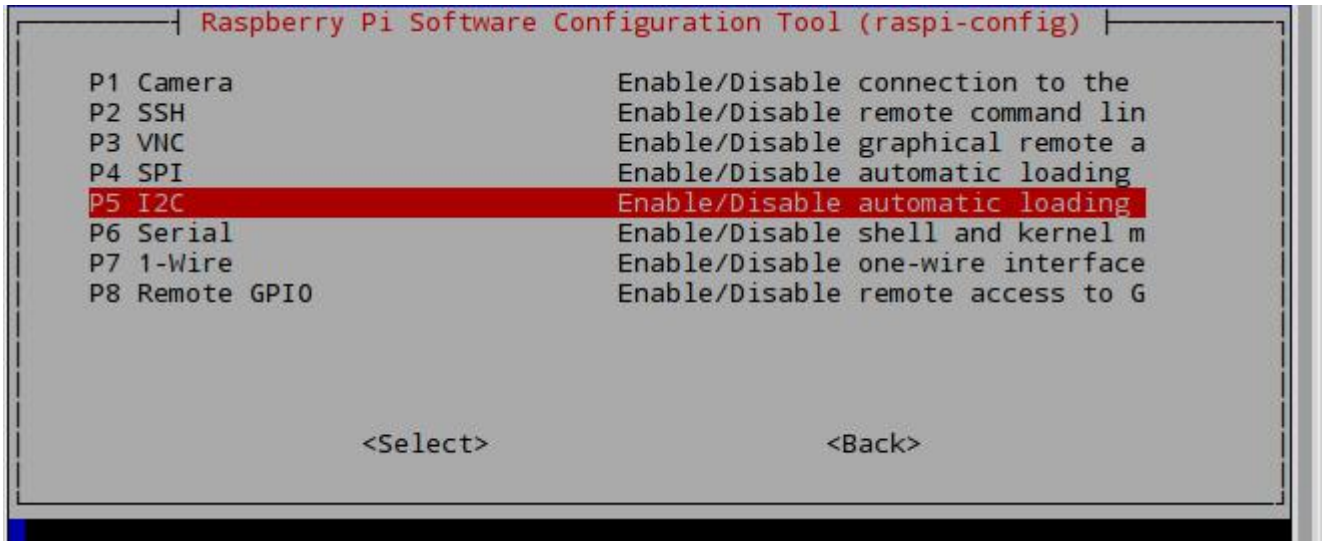
Step 1: Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

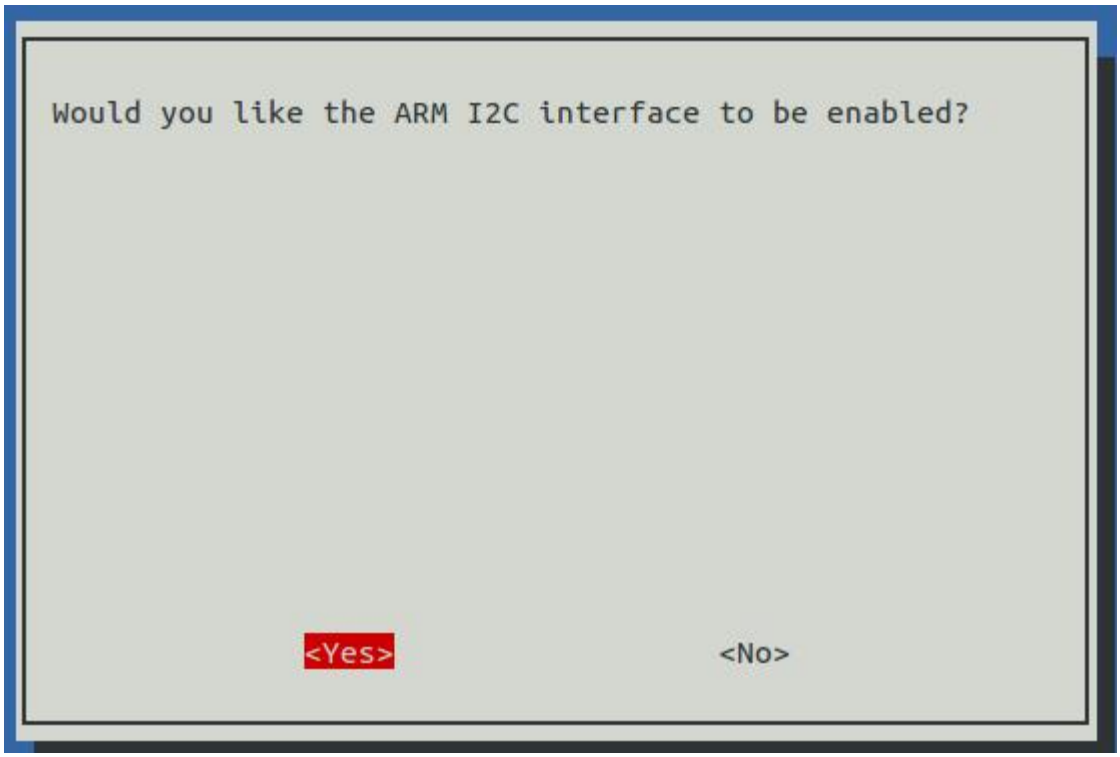
5 Interfacing options



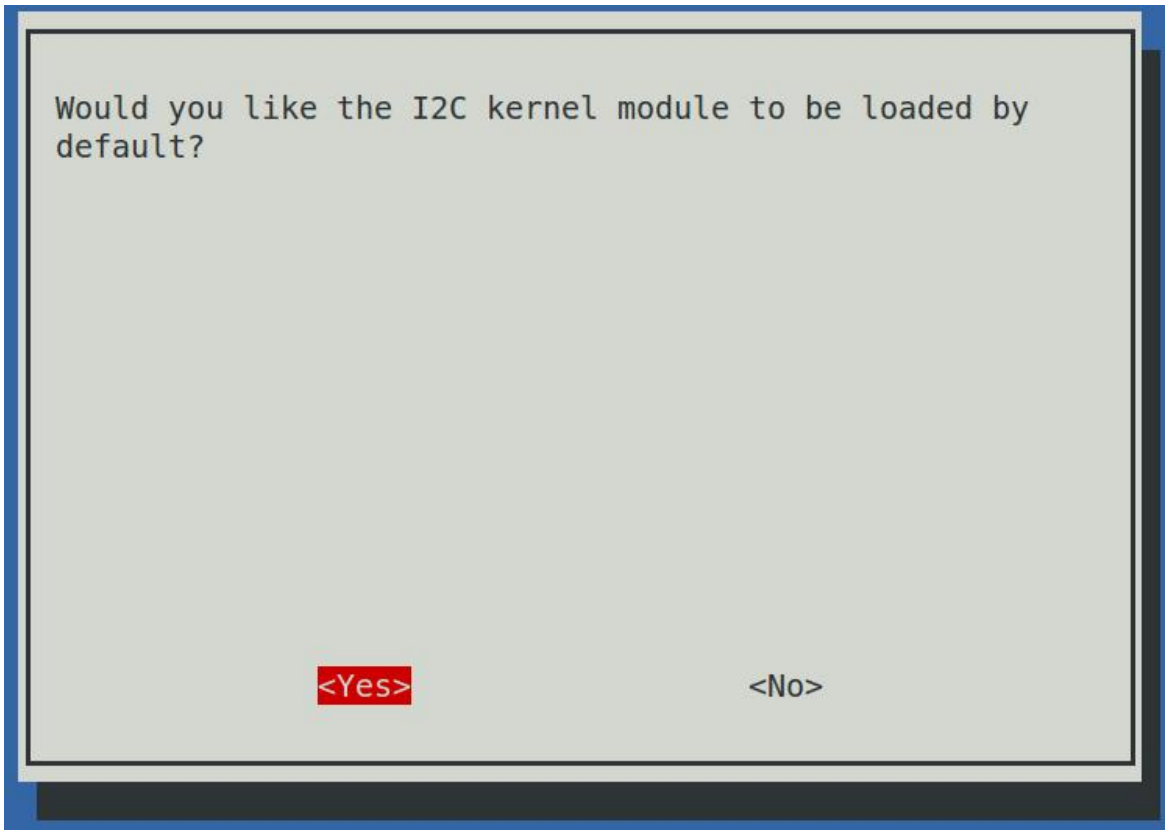
P5 I2C



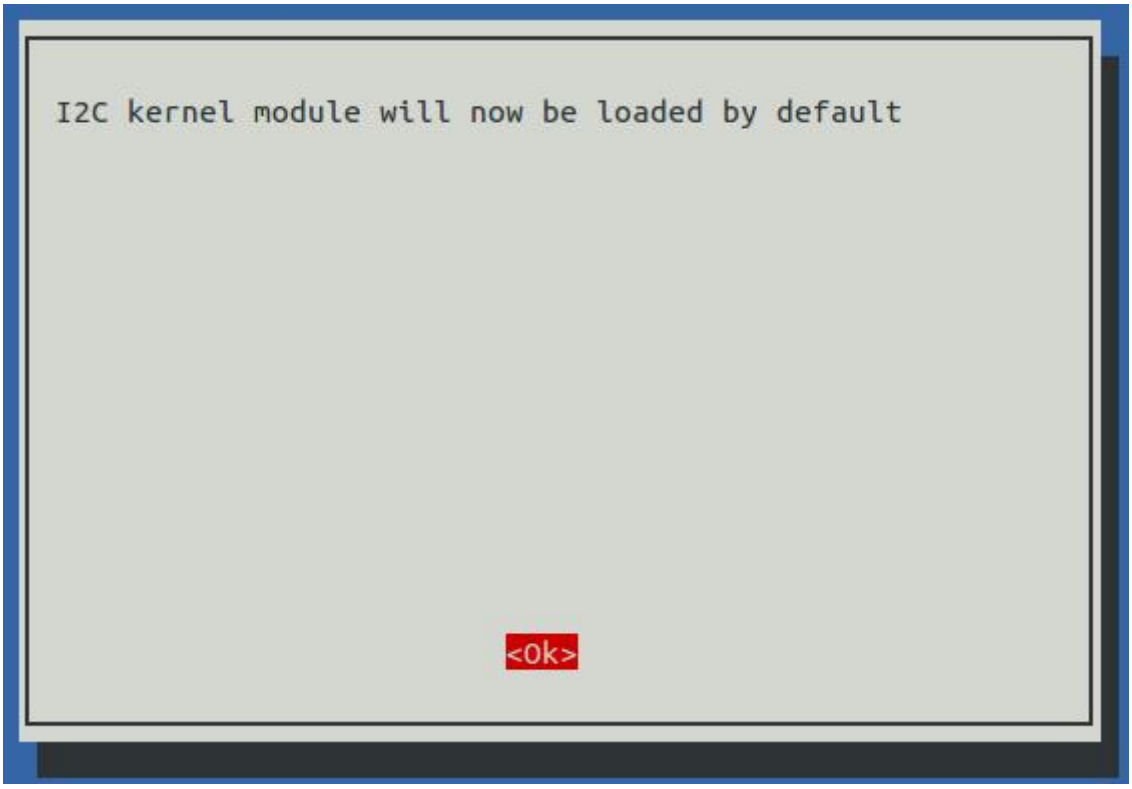
<Yes>


Would you like the ARM I2C interface to be enabled?
<Yes>
<No>

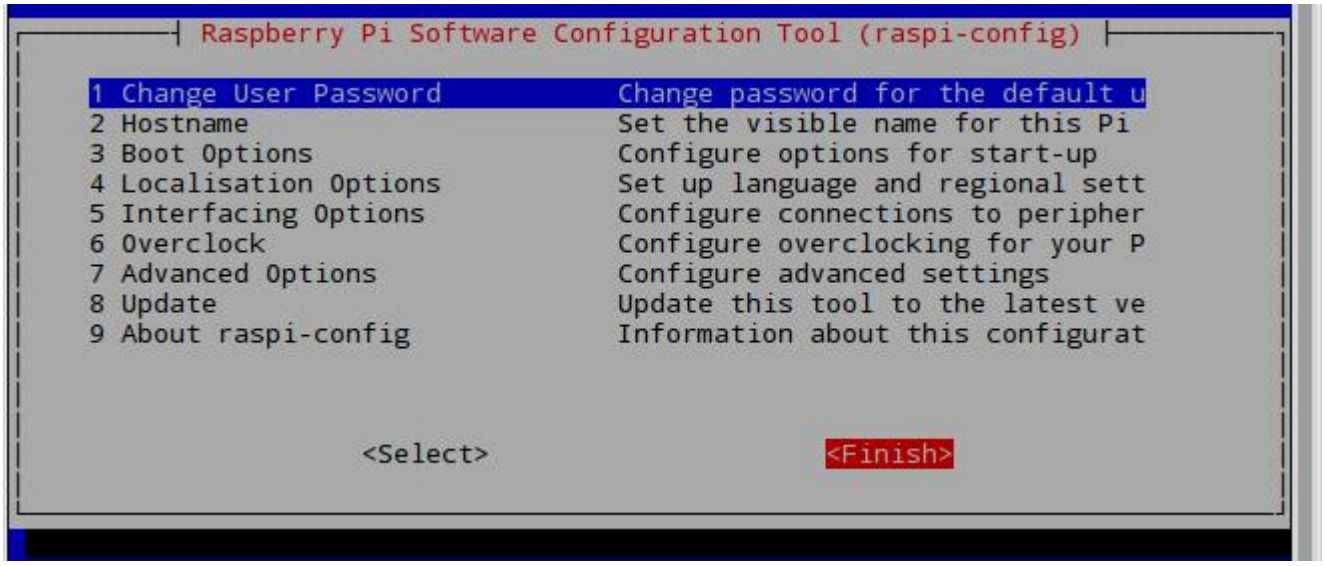
<Yes>


Would you like the I2C kernel module to be loaded by default?
<Yes>
<No>

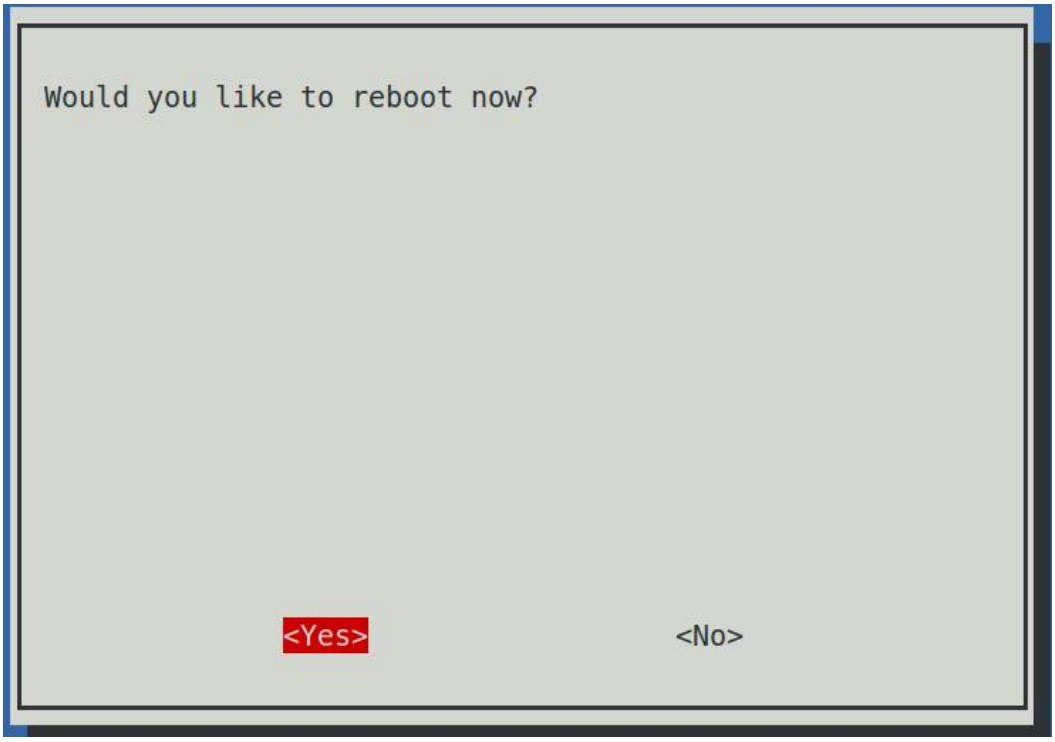
<Ok>



<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 and higher version
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.

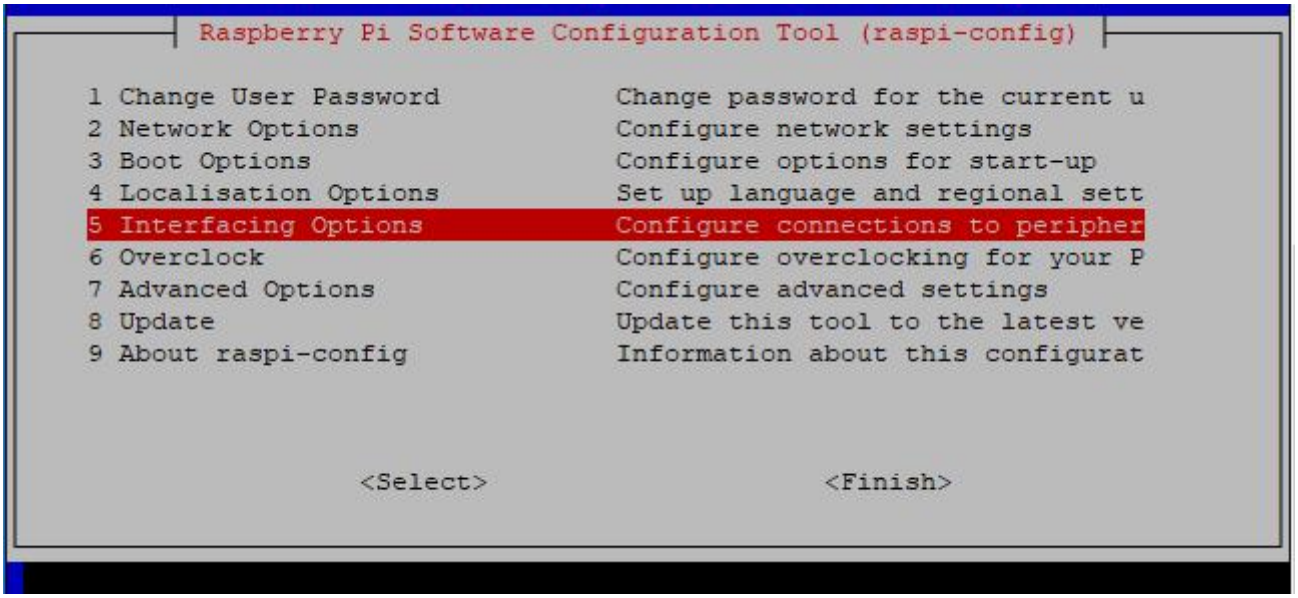
```
sudo apt-get install python-smbus
```

3.2.2 SPI Configuration

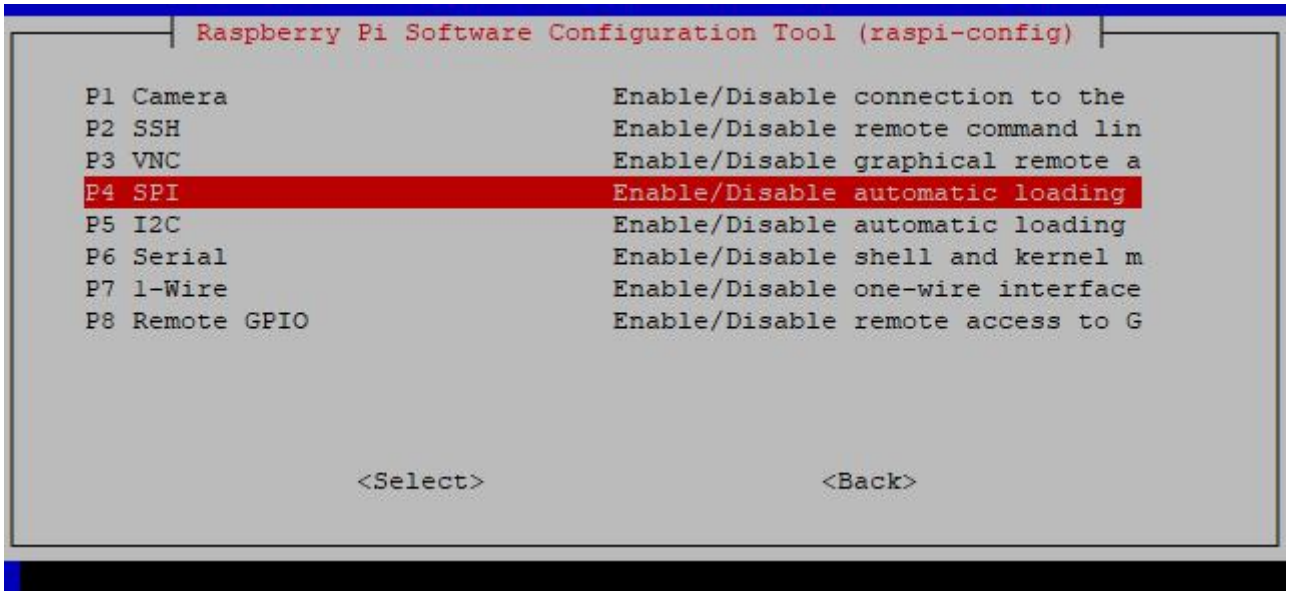
Step 1: Enable the SPI port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

```
sudo raspi-config
```

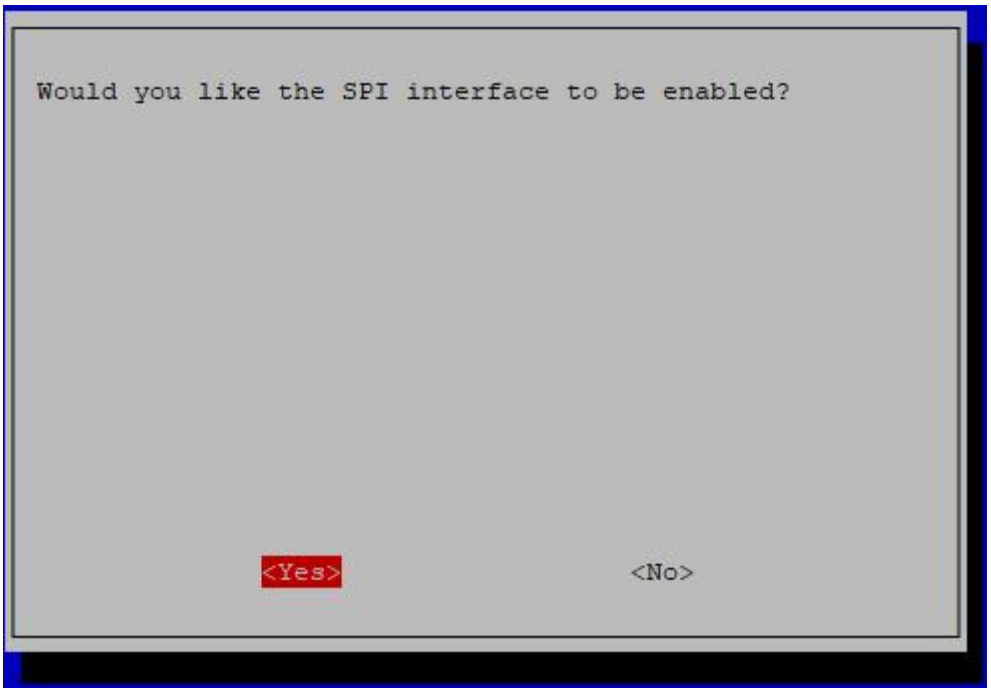
5 Interfacing options



P4 SPI



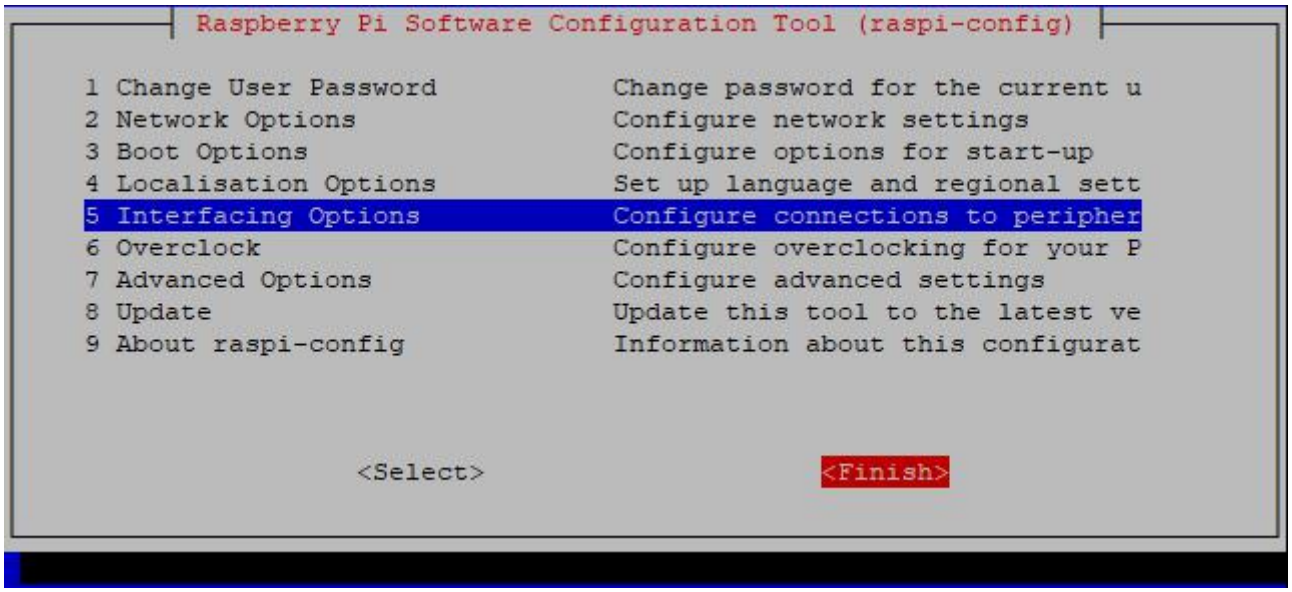
<YES>


Would you like the SPI interface to be enabled?
<Yes>
<No>

<OK>


The SPI interface is enabled
<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 python3 setup.py install
```

Note: This step is for python users, if you use C language, please skip.

Copyright Notice