About Our Company

WayinTop, Your Top Way to Inspiration, is a professional manufacturer over 2,000 open source motherboards, modules, and components. From designing PCBs, printing, soldering, testing, debugging, and offering online tutorials, WayinTop has been committed to explore and demystify the wonderful world of embedded electronics, including but not limited to Arduino and Raspberry Pi. We aim to make the best designed products for makers of all ages and skill levels. No matter your vision or skill level, our products and resources are designed to make electronics more accessible. Founded in 2013, WayinTop has grown to over 100+ employees and a 50,000+ sq ft. factory in China by now. With our unremitting efforts, we also have expanded offerings to include tools, equipments, connector kits and various DIY products that we have carefully selected and tested.

US Amazon Store Homepage:
https://www.amazon.com/shops/A22PZZC3JNHS9L
CA Amazon Store Homepage:
https://www.amazon.ca/shops/A22PZZC3JNHS9L
UK Amazon Store Homepage:
https://www.amazon.co.uk/shops/A3F8F97TMOROPI
DE Amazon Store Homepage:
https://www.amazon.de/shops/A3F8F97TMOROPI
FR Amazon Store Homepage:
https://www.amazon.fr/shops/A3F8F97TMOROPI
IT Amazon Store Homepage:
https://www.amazon.it/shops/A3F8F97TMOROPI
ES Amazon Store Homepage:
https://www.amazon.es/shops/A3F8F97TMOROPI
JP Amazon Store Homepage:
https://www.amazon.co.jp/shops/A1F5OUAXY2TP0K

Content

Lesson 0 Install and Operate the Raspberry Pi System

Overview

In this lesson, you will learn how to install and operate the Raspberry Pi system.

Parts Required:

1 x Raspberry Pi

1 x LCD Display

1 x HDMI Cable

1 x SD Card

1 x SD Card Reader

1 x Mouse and keyboard

You can obtain the specific installation steps through visiting the following web sites:

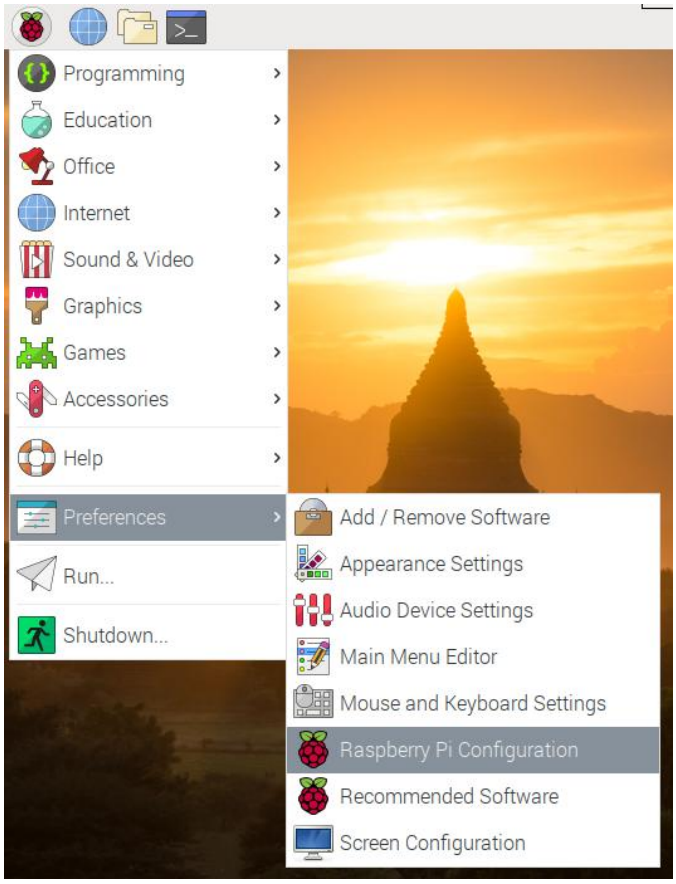
https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up

After the system is installed, you can obtain the basic instructions through visiting the following web sites:

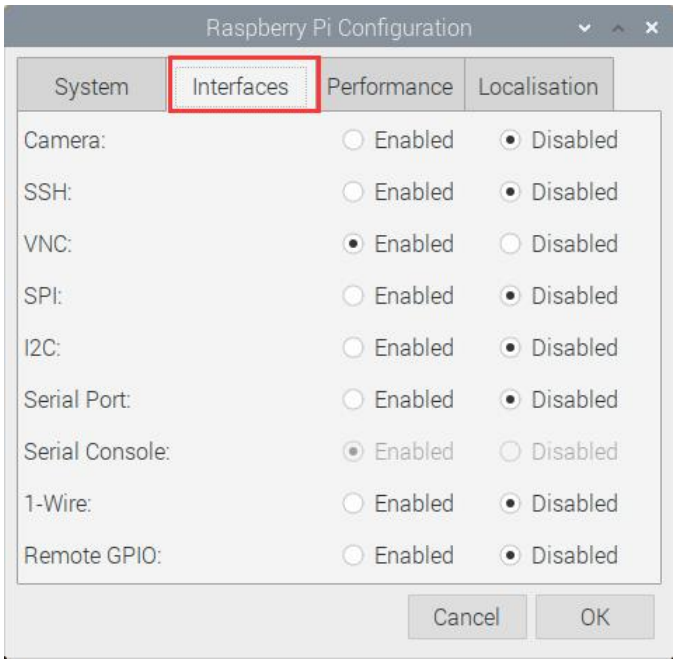
https://projects.raspberrypi.org/en/projects/raspberry-pi-using

After mastering the above basic operations, we will learn to control the Raspberry Pi by using a computer to connect to Wi-Fi:

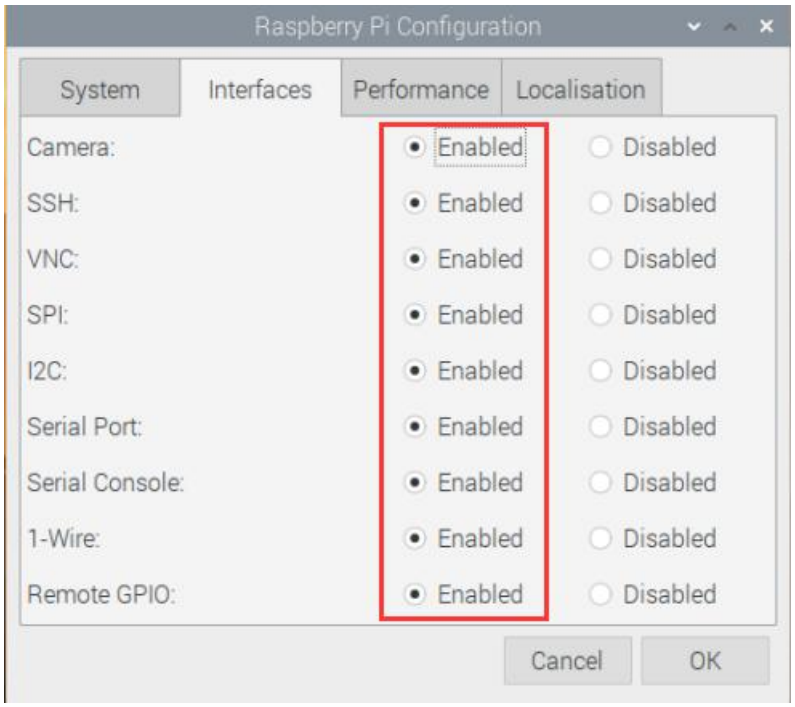
Open the Raspberry Pi interface and click 'Preferences'->'Raspberry Pi Configuration' as shown below.



Select the 'Interfaces'option in the pop-up window as shown below:



Enable all the interfaces as shown below:



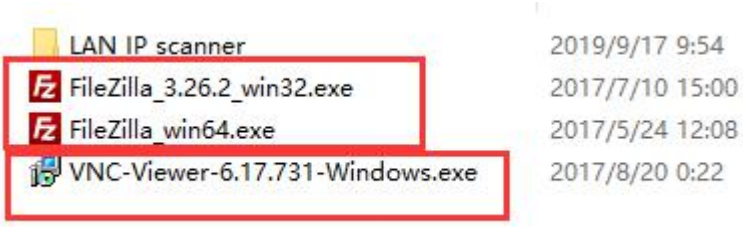
Click 'OK'and restart the Raspberry Pi.

The Raspberry Pi configuration is completed, and then uses the computer to connect to the Raspberry Pi viaWi-Fi:

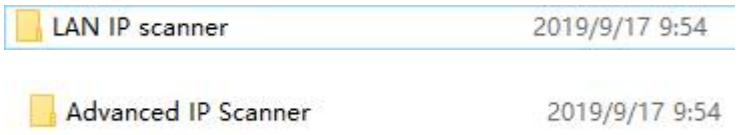
Steps

1) Open the 'Supporting software'file in the companion file as shown below:

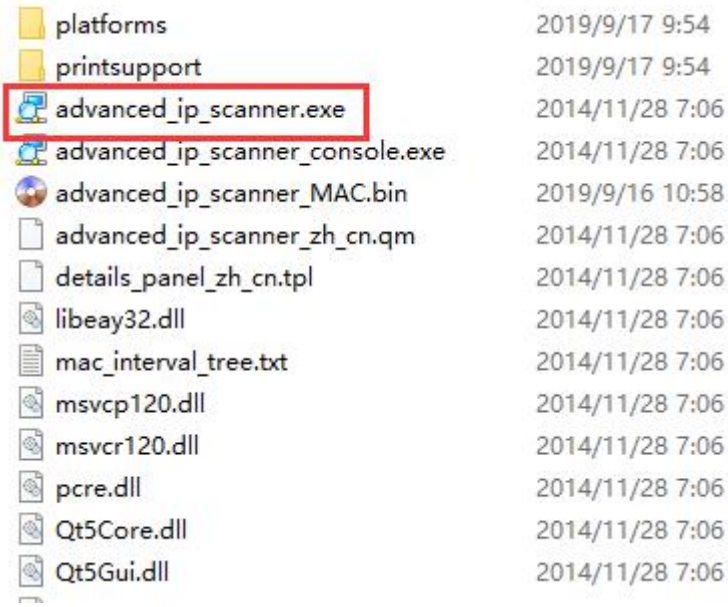
2) Install 'FileZilla'and 'VNC'software in the file:



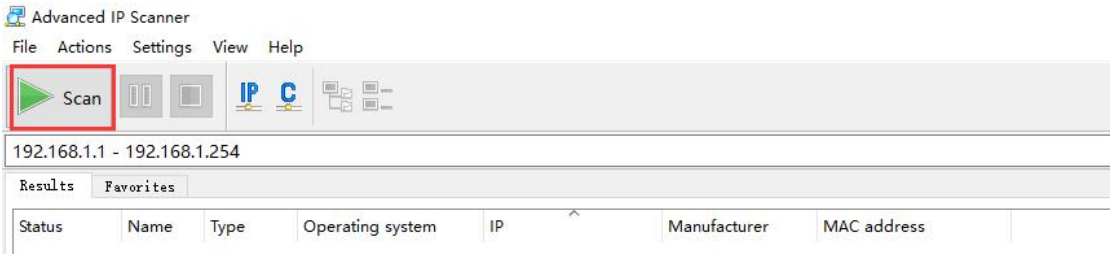
3）After installing the two software, open the Advanced IP Scanner file in the LAN IP scanner file as shown below:



4）Select ‘advanced_ip_scanner.exe’software as shown below：

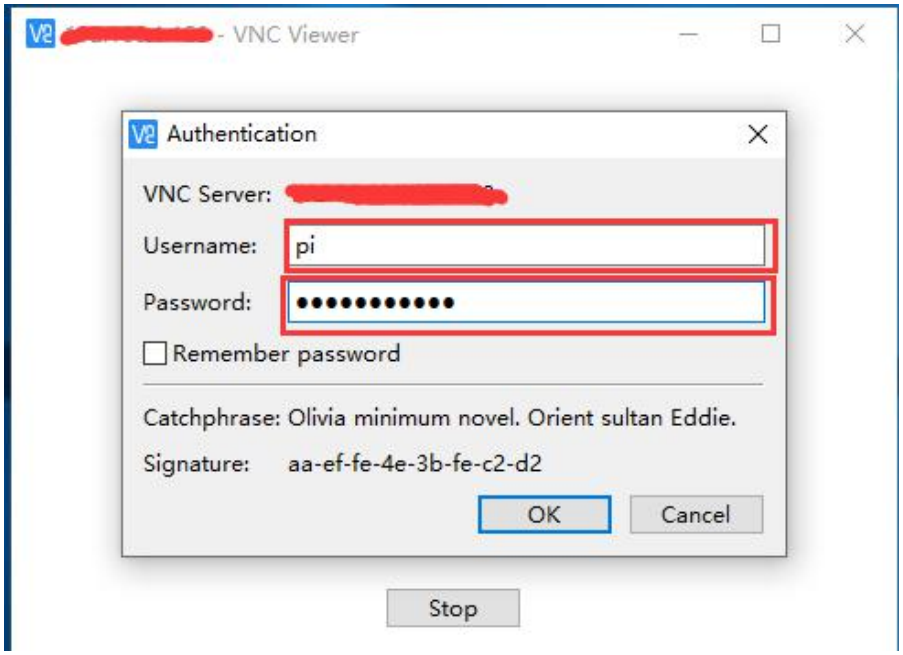


5）Click‘Scan’ to scan the Raspberry Pi IP (In the upper right corner of the desktop in the Raspberry Pi system, you can also view the IP address by placing the mouse on the wireless icon), as shown below:



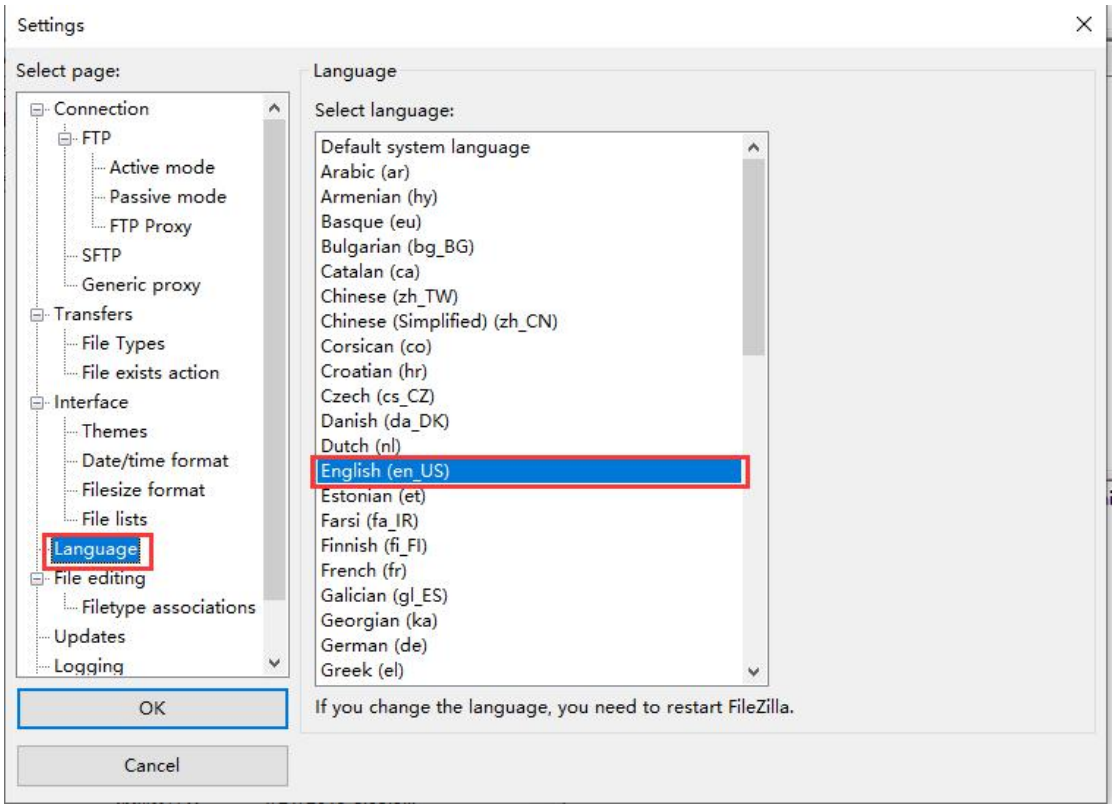
6）Record the IP address after scanning, as shown below:

Open the VNC software after recording the IP address; enter the IP address in the input box, then press the Enter key, enter the user name and password in the pop-up window, click ‘ok’ to connect to the Raspberry Pi. As shown below:

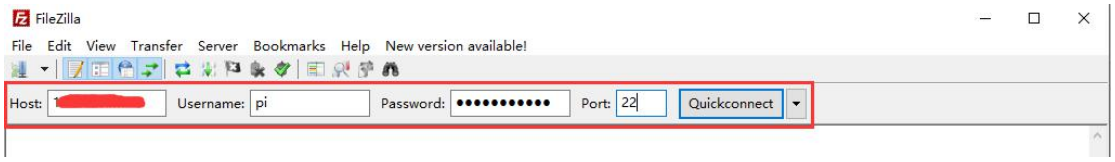


8 ） Use ‘FileZilla’software to achieve wireless transmission of files. Open the ‘FileZilla’software and set the language. As shown below, click the second button in the menu‘Edit’, then select the ‘Settings’button in the drop-down box.

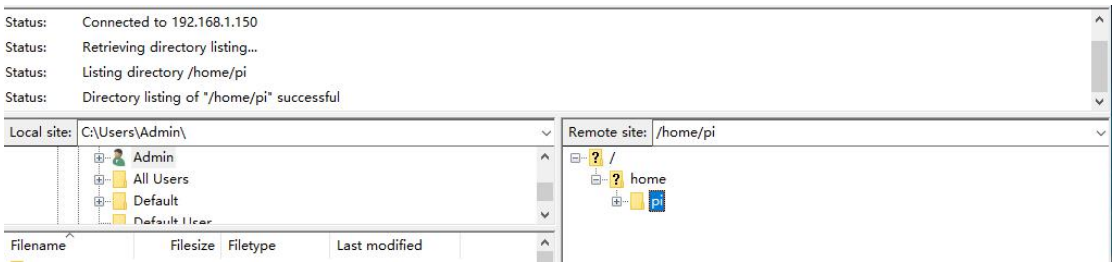
9)This tutorial takes English as an example. Click 'Language', select 'English', and then click'ok'. After setting, close the software and re-open it to set it successfully.



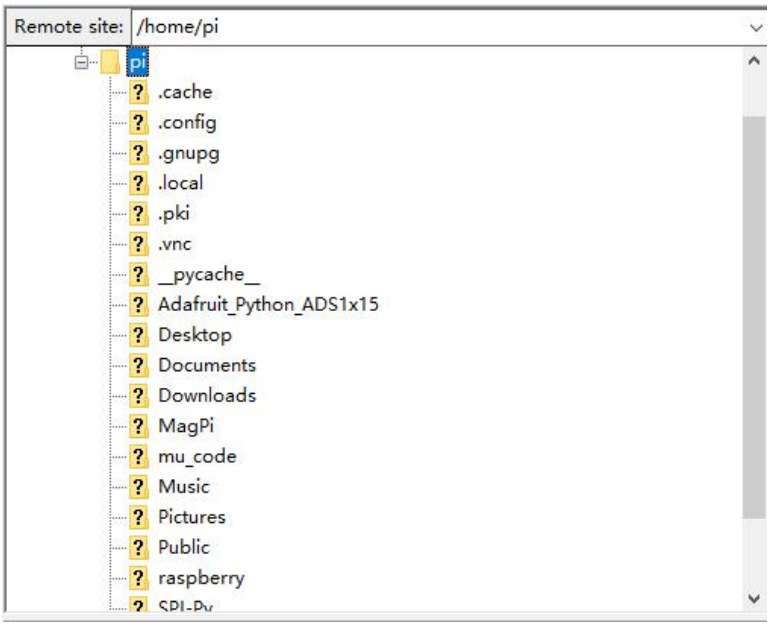
10)After the display language setting is completed, start connecting to the Raspberry Pi, enter the previously recorded IP address in the first input box, enter the user name in the second box, enter the password in the third box, and enter the port number default in the fourth box. 22, then click the 'Quickconnect' button.



The connection is as shown in the following figure:



In the future tutorial, all the files are in the pi folder, so after the connection is successful, you can copy the files on the computer directly into the pi file (This file is the pi file in the Raspberry Pi, you need to copy the code file we provide to this folder before following the tutorial.)



WiringPi GPIO Pins

There are three GPIO wiring pins of Raspberry Pi: based on the BCM chip number, based on the physical serial number and based on connectionPi. The correspondence between these three GPIO numbers is as follows:

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
–	–	3.3v	1 | 2	5v	–	–
8	R1:0/R2:2	SDA0	3 | 4	5v	–	–
9	R1:1/R2:3	SCL0	5 | 6	0V	–	–
7	4	GPIO7	7 | 8	TXD	14	15
–	–	0V	9 | 10	RXD	15	16
0	17	GPIO0	11 | 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 | 14	0V	–	–
3	22	GPIO3	15 | 16	GPIO4	23	4
–	–	3.3v	17 | 18	GPIO5	24	5
12	10	MOSI	19 | 20	0V	–	–
13	9	MISO	21 | 22	GPIO6	25	6
14	11	SCLK	23 | 24	CE0	8	10
–	–	0V	25 | 26	CE1	7	11
30	0	SDA. 0	27 | 28	SCL. 0	1	31
21	5	GP10. 21	29 | 30	0V	–	–
22	6	GP10. 22	31 | 32	GP10. 26	12	26
23	13	GP10. 23	33 | 34	0V	–	–
24	19	GP10. 24	35 | 36	GP10. 27	16	27
25	26	GP10. 25	37 | 38	GP10. 28	20	28
		0V	39 | 40	GP10. 29	21	29
wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin

For RPi B

For RPi B+/ 2 model B

You can also use the ‘gpio readall’command to view their correspondence.

As shown below：

```
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 | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | 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 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 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 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 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---+---+------+---------+-----+-----+
```

If you are using Raspberry Pi 4B, there be errors when you execute the command "gpio readall". As follows:

```
pi@raspberrypi:~ $ gpio readall
Oops - unable to determine board type... model: 17
```

This is because the official version of the library supporting 4B has not yet been released, resulting in some commands can not be used properly. But it won't affect the next project. For this problem, you can solve it by installing a patch. Just execute the commands below in the terminal:

"wget https://project-downloads.drogon.net/wiringpi-latest.deb"

"sudo dpkg -i wiringpi-latest.deb"

(Note：If you are using a Raspberry Pi 4B, this step must be performed, otherwise it will affect the signal output of the GPIO port).

After the installation is completed, execute the"gpio-v" and"gpio readall" commands again.

```
pi@raspberrypi:~ $ gpio -v
gpio version: 2.52
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Pi 4B, Revision: 01, Memory: 1024MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 4 Model B Rev 1.1
  * This Raspberry Pi supports user-level GPIO access.
pi@raspberrypi:~ $ gpio readall
 +-----+-----+---------+------+---+---Pi 4B--+---+------+---------+-----+-----+
 | 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 | 1 |  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 |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 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 4B--+---+------+---------+-----+-----+
```

For more details on connectionPi, please refer to:http://wiringpi.com/

Lesson 1 LED

Overview

In this lesson, you will learn how to use the Raspberry Pi power and resistance to blink the LED light.

Parts Required:

1 x Raspberry Pi

1 x LED

1 x 220 ohm resistor

2 x Jumper Wires

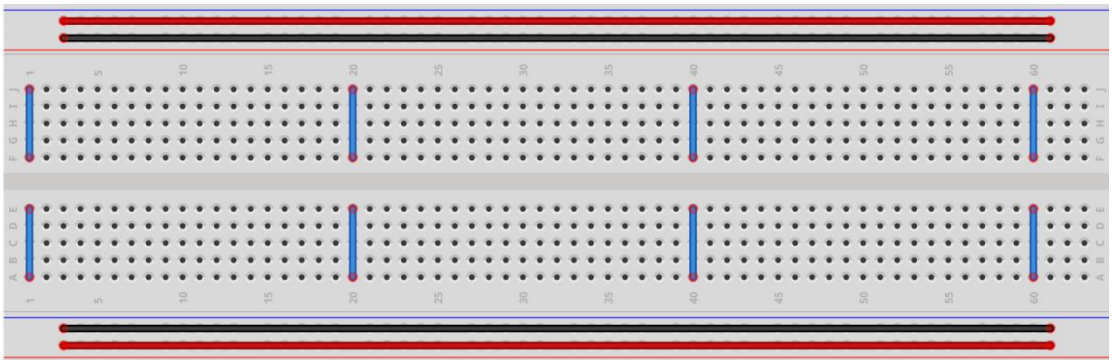
1 x breadboard

Product Introduction

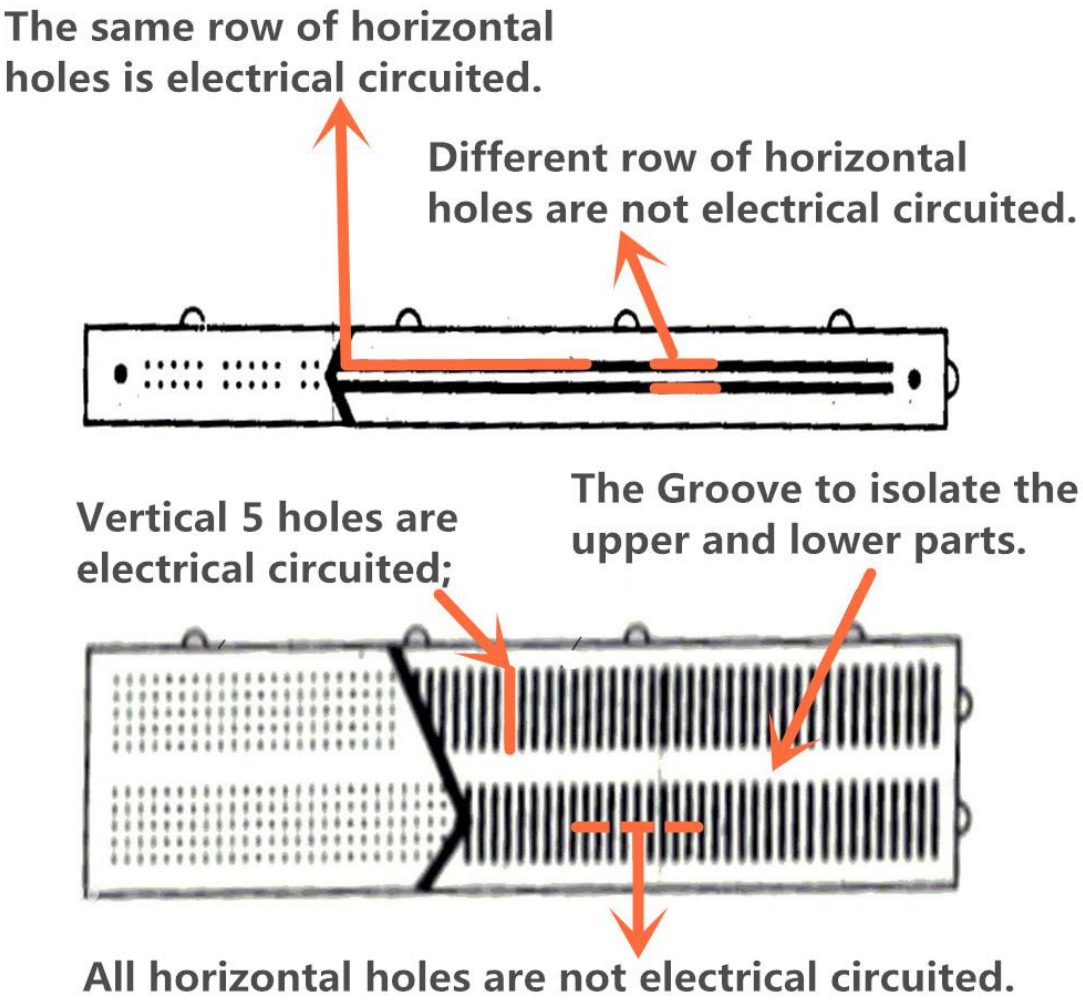
BREADBOARD

A breadboard enables you to prototype circuits quickly without soldering.

The specific principle is as follows:



The inside of the breadboard is made up of metal strips. When inserted into the holes of the board, it can be in contact with the metal strip to achieve the purpose of conduction. Usually 5 holes are linked by a metal strip. There are two rows of vertical holes on both sides of the breadboard board, which are also 5 holes as a group. These two rows are used to power the components on the breadboard.



LED

The LEDs can make great indicator lights. In this lesson, we will use the most common LED, a 5mm blue LED (5mm refers to the diameter of the LED, other common sizes are 3mm and 10mm.)

You cannot directly connect an LED to a battery or voltage, because

1) The LED has a positive and a negative lead and will not light if placed in the wrong way;

2) The LED must be used with a resistor to limit or 'choke' the amount of current flowing through it; otherwise, it will burn out!

The way to distinguish the positive and negative poles of the LED： Long Pin-positive；Short Pin-negative.

RESISTORS

As the name suggests, resistors resist the flow of electricity. The higher the value of the resistor, the more it resists and the less electrical current will flow through it.

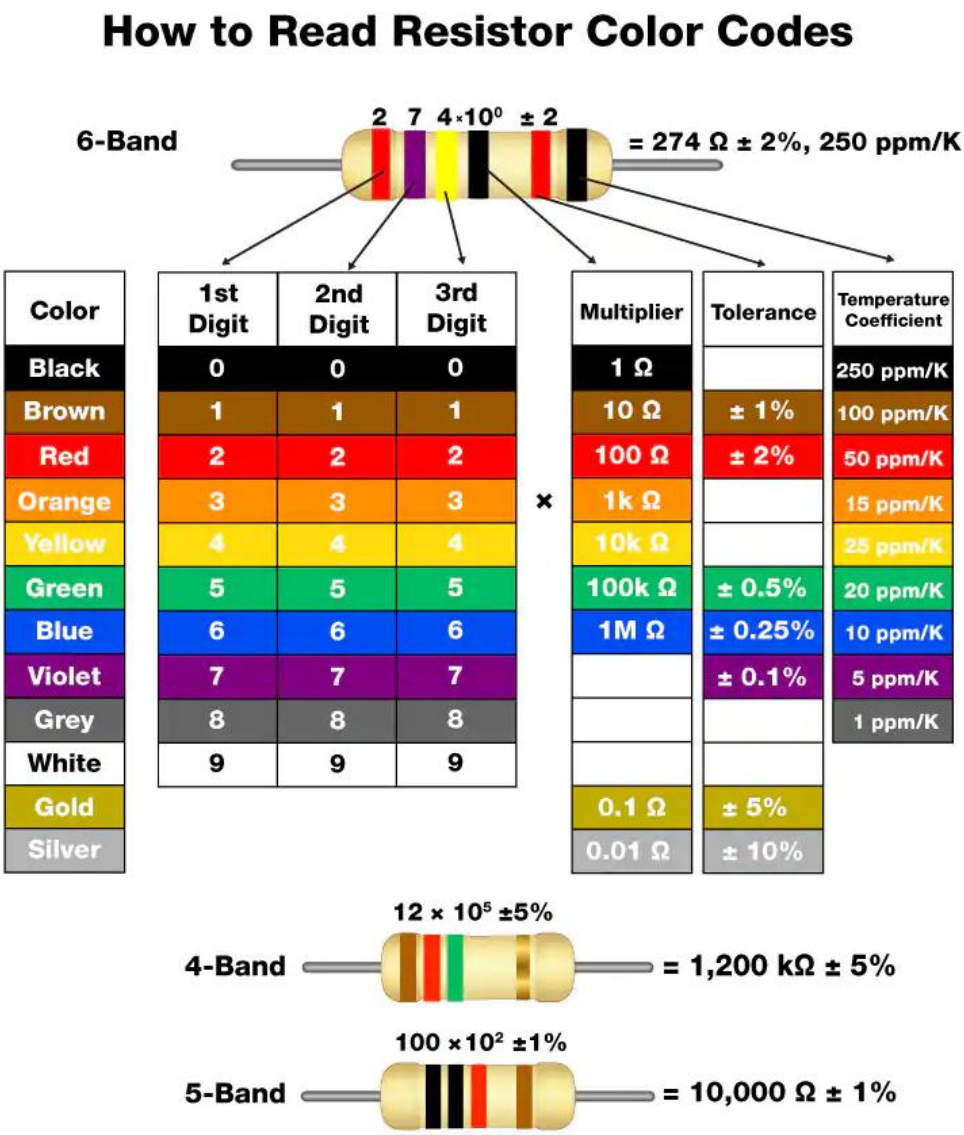
We are going to use this to control how much electricity flows through the LED and therefore, how brightly it shines.

The unit of resistance is called the Ohm, which is usually shortened to Ω. Because an Ohm is a low value of resistance (it doesn't resist much current), we also denote the values of resistors in kΩ (1,000 Ω) and MΩ (1,000,000 Ω). These are called kilo-ohms and mega-ohms.

In this lesson, we mainly use a 220Ω resistor to protect the LED. You can also use a larger resistance value, but a larger resistance will make the LED's luminous intensity weaker.

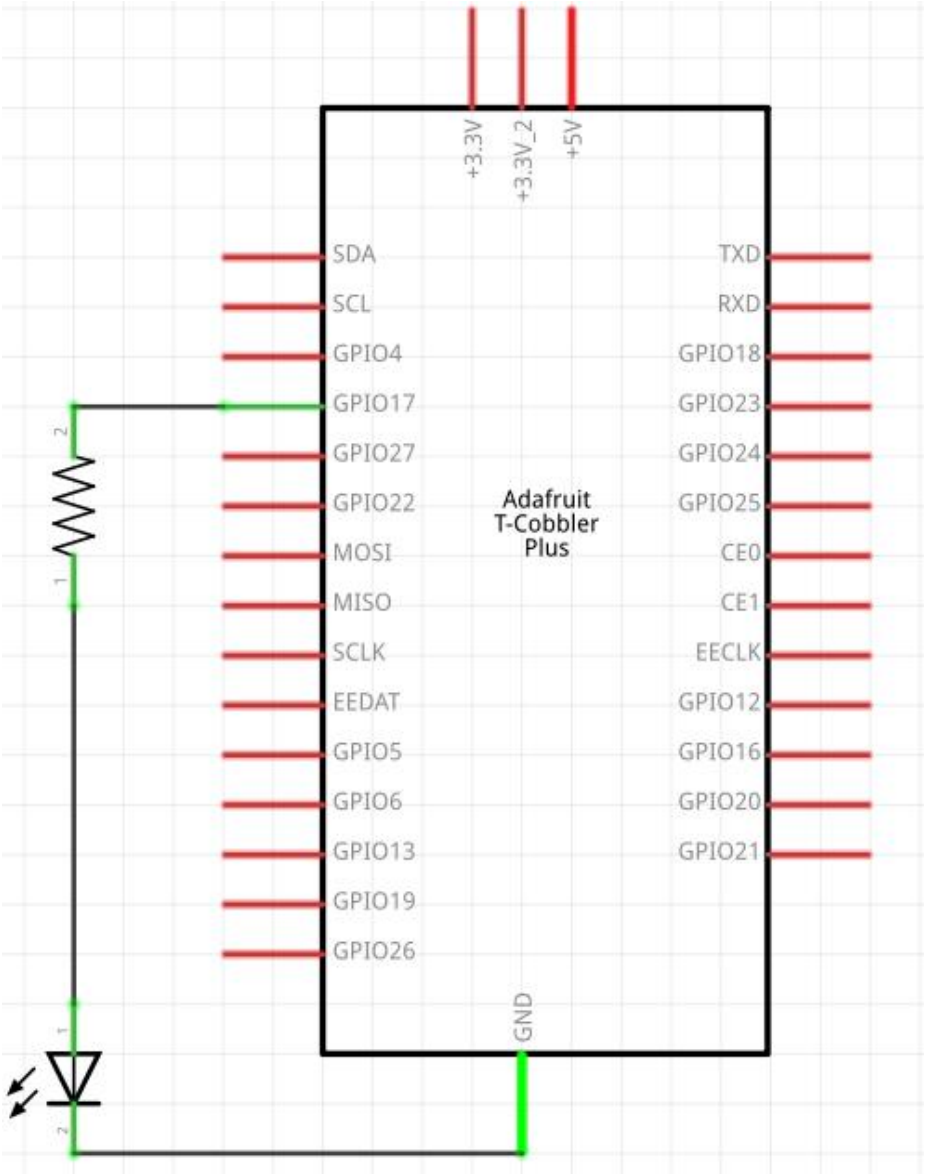
220Ω, 1kΩ and 10kΩ.resistors all look the same, except that they have different colored stripes on them. These stripes tell you the value of the resistor.

The following picture will show you how to distinguish resistance value.

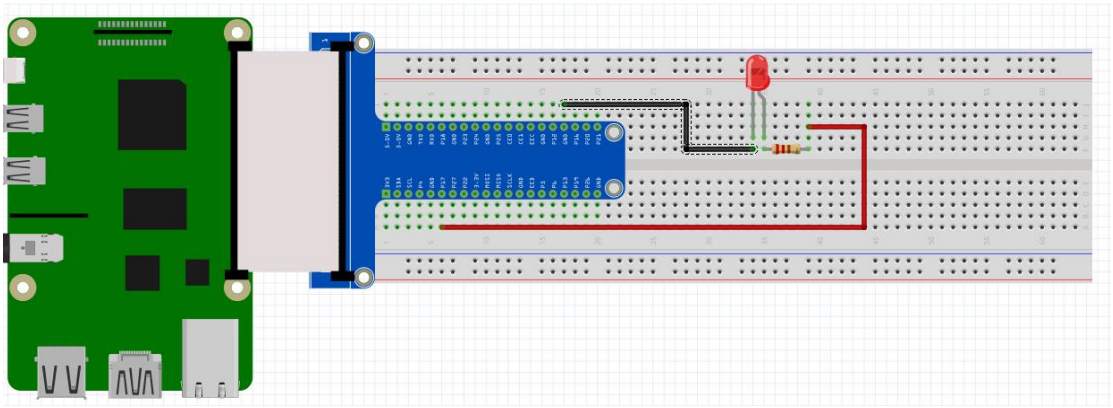


If you find that this discrimination method is too complicated, you can also use a multi-meter to measure the value of the resistance. The resistance is different from LED, it has no positive and negative poles, so it can work normally by connecting it to the negative or positive pole of LED.

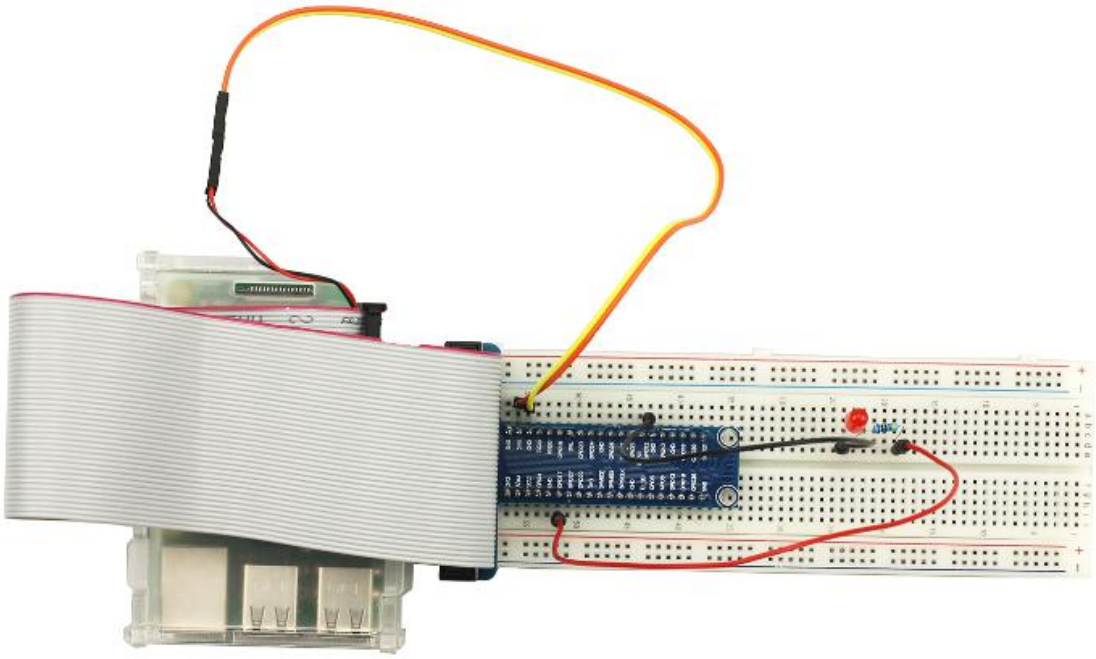
Connection Schematic



Wiring Diagram



Example Figure

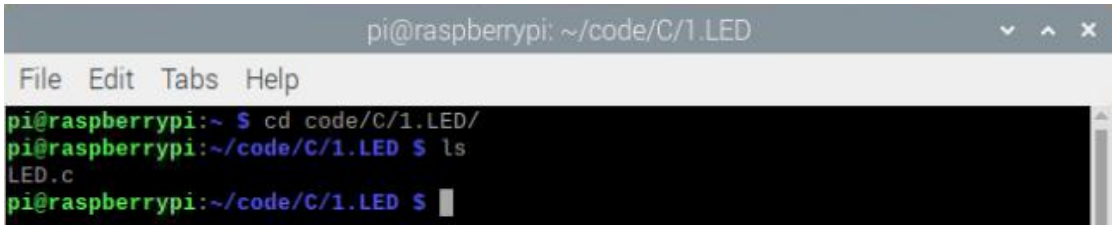


C Code

Tips: Before running the code, you need to move the code file we provide to the pi folder, or create an executable file in the directory where the command is executed (for example: code/C/1.LED/), and then write the code into the file (This reminder won't be given in next lessons).

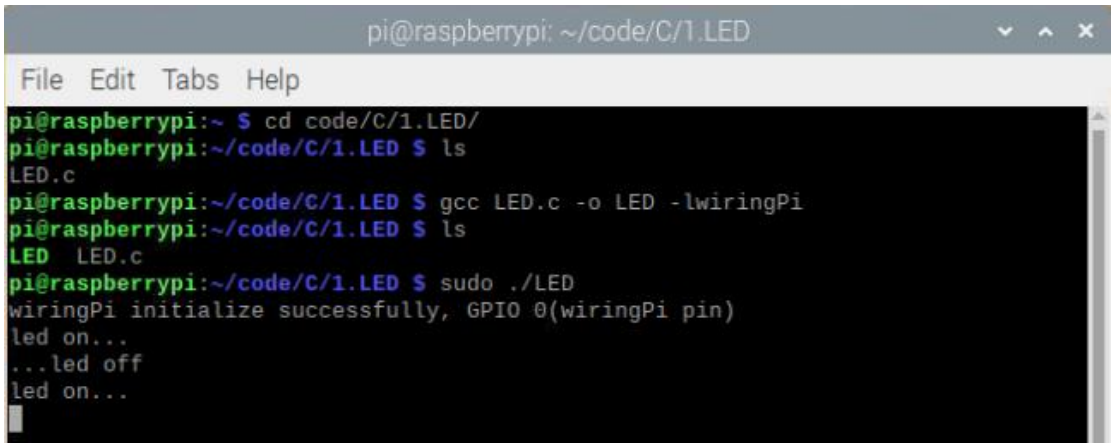
Open the terminal and enter cd code/C/1.LED/command to enter the LED.c code directory;

Enter ls command to view the file LED.c in the directory.



Enter the gcc LED.c -o LED -lwiringPi command to generate the executable file LED for LED.c and enter the ls command to view it.

Enter the sudo ./LED command to run the code, and the result is as follows:



Press "Ctrl + c" to end the program. The following is the program code:

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

#define    ledPin        0

int main(void)
{
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    printf("wiringPi initialize successfully, GPIO %d(wiringPi pin)\n",ledPin);

    pinMode(ledPin, OUTPUT);

    while(1){
            digitalWrite(ledPin, HIGH);
            printf("led on...\n");
            delay(1000);
            digitalWrite(ledPin, LOW);
            printf("...led off\n");
            delay(1000);

}

    return 0;
}
```

Code interpretation

```
#define ledPin 0
```

This code is a GPIO macro definition (pin naming). The GPIO connected to the ledPin in the circuit is GPIO17. GPIO17 is defined as 0 in the connectionPi number. So 'ledPin' should be defined as a 0 pin.

```
If(wiringPiSetup() == -1){
Printf("setup wiringPi failed !");
Return 1;
}

Printf("wiringPi initialize successfully, GPIO %d(wiringPi pin)\n", ledPin);
```

In the main function main(), initialize wiringPi first, and then print out the initial results. This process is mainly judged by the "if" sentence.Once the initialization fails, exit the program.

```
pinMode(ledPin, OUTPUT);

While(1){
digitalWrite(ledPin, HIGH);
Printf("led on...\n");
Delay(1000);
digitalWrite(ledPin, LOW);
Printf("...led off\n");
Delay(1000);
}
```

After the connectionPi is successfully initialized, 'ledPin' is set to the output mode. Then enter the while loop, which is an endless loop. That is, the program will always execute in this loop unless it ends externally. In this loop, use 'digitalWrite(ledPin,HIGH)' to make 'ledPin' output high, then 'LED' illuminates. After 1 second delay, use 'digitalWrite(ledPin, LOW)' to make 'ledPin' output low, then turn off the LED and then delay. Repeat the cycle and the LED will begin to flash all the time.

The configuration function of GPIO is as follows:

```
Void pinMode(int pin, int mode);
```

This sets the mode of the pin to INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only the wiring Pi pin 1 (BCM_GPIO 18) supports the PWM output, and only the wiring Pi pin 7 (BCM_GPIO 4) supports the CLOCK output mode. This function is not available in Sys mode. If you need to change the pin

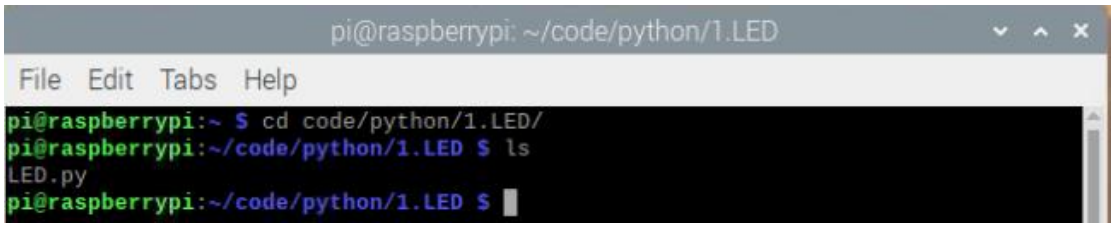
mode, you can use the gpio program in the script before starting the program.
Void digitalWrite (int pin, int value);
Write the value HIGH or LOW (1 or 0) to the pin preset to the output.

Python code

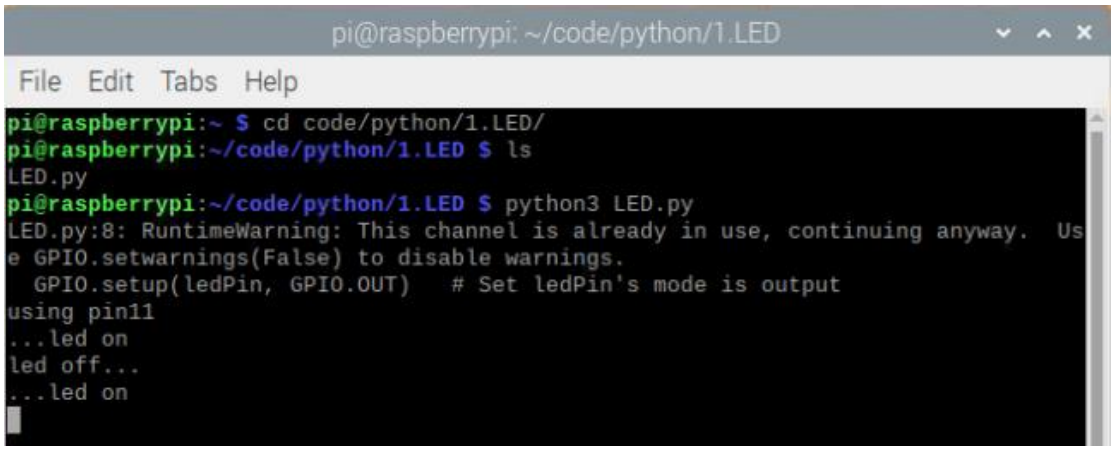
Note:(Because the Raspberry Pi has many models and systems, some running commands are different. Some of the following run commands use python3, and some are python. You can test it according to your own Raspberry Pi. Sometimes both commands are ok.)

Open the terminal and enter cd code/python/1.LED/ command to enter the LED.py code directory.

Enter the ls command to view the files in the directory LED.py



Run the code by typing the python3 LED.py command, and the result is as follows:



Press "Ctrl + c"to end the program. The following is the program code:

```
Import RPi.GPIO as GPIO

Import time

ledPin = 11

Def setup():

GPIO.setmode(GPIO.BOARD)

GPIO.setup(ledPin, GPIO.OUT)

GPIO.output(ledPin, GPIO.LOW)

Print ('using pin%d'%ledPin)

Def loop():

While True:

GPIO.output(ledPin, GPIO.HIGH)

Print ('...led on')

Time.sleep(1)

GPIO.output(ledPin, GPIO.LOW)

Print ('led off...')
```

```
Time.sleep(1)

Def destroy():

GPIO.output(ledPin, GPIO.LOW)

GPIO.cleanup()


If __name__ == '__main__':


Setup()

Try:

Loop()

Except KeyboardInterrupt:

  Destroy() will be executed.

Destroy()
```

Code interpretation

```
Import RPi.GPIO as GPIO

Import time

ledPin = 11
```

Import the required RPi.GPIO module and time module

GPIO17 is to use the pin 11 of the mainboard, so define ledPin as 11

```
Def setup():

GPIO.setmode(GPIO.BOARD)

GPIO.setup(ledPin, GPIO.OUT)
```

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

Print ('using pin%d'%ledPin)
```

In 'Setup()', 'GPIO.setmode(GPIO.BOARD)' is used to set the mode of the GPIO according to the physical position of the pin. Set ‘ledPin’ to output mode (output low) and then print the ‘ledPin’ serial output.

```
Def loop():

While True:



GPIO.output(ledPin, GPIO.HIGH)

Print ('...led on')

Time.sleep(1)

GPIO.output(ledPin, GPIO.LOW)

Print ('led off...')

Time.sleep(1)
```

In deloop(), there s a loop, which is an endless loop. That is to say, the program will always be executed in this loop, unless it is ended by external factor. In this loop, set ledPin output high level, then LED is turned on. After a second (1s) delay, set ledPin output low level, and then LED is turned off, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
Def destroy():

GPIO.output(ledPin, GPIO.LOW)

GPIO.cleanup()
```

Finally, when the program is terminated, sub-function will be executed, the LED will be turned off and then the IO port will be released. If close the program terminal directly, the program will be terminated too, but destroy() function will not be executed. So, GPIO resources won’t be released, in the warning message may appear next time you use GPIO. So, it is not a good habit to close the program terminal directly.

```
if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
  destroy() will be    executed.
        destroy()
```

The code starts with "if __name__ == '__main__':" and then executes in sequence. If you press the Ctal+C program, the "except KeyboardInterrupt:" program terminates.

Lesson 2 Button &LED

Overview

In this lesson, you will learn how to use the Raspberry Pi to control the LED state through a button.

Parts Required

1 x Raspberry Pi

1 x LED

1 x 220 Ohm Resistor

4 x DuPont Wires

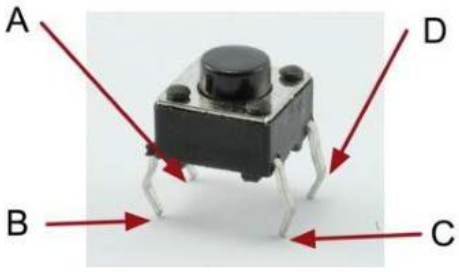
1 x Breadboard

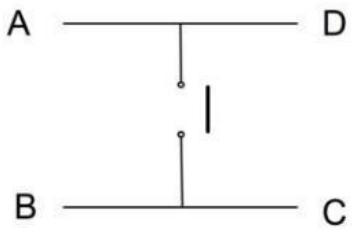
1 x Button

Product Introduction

Push Button

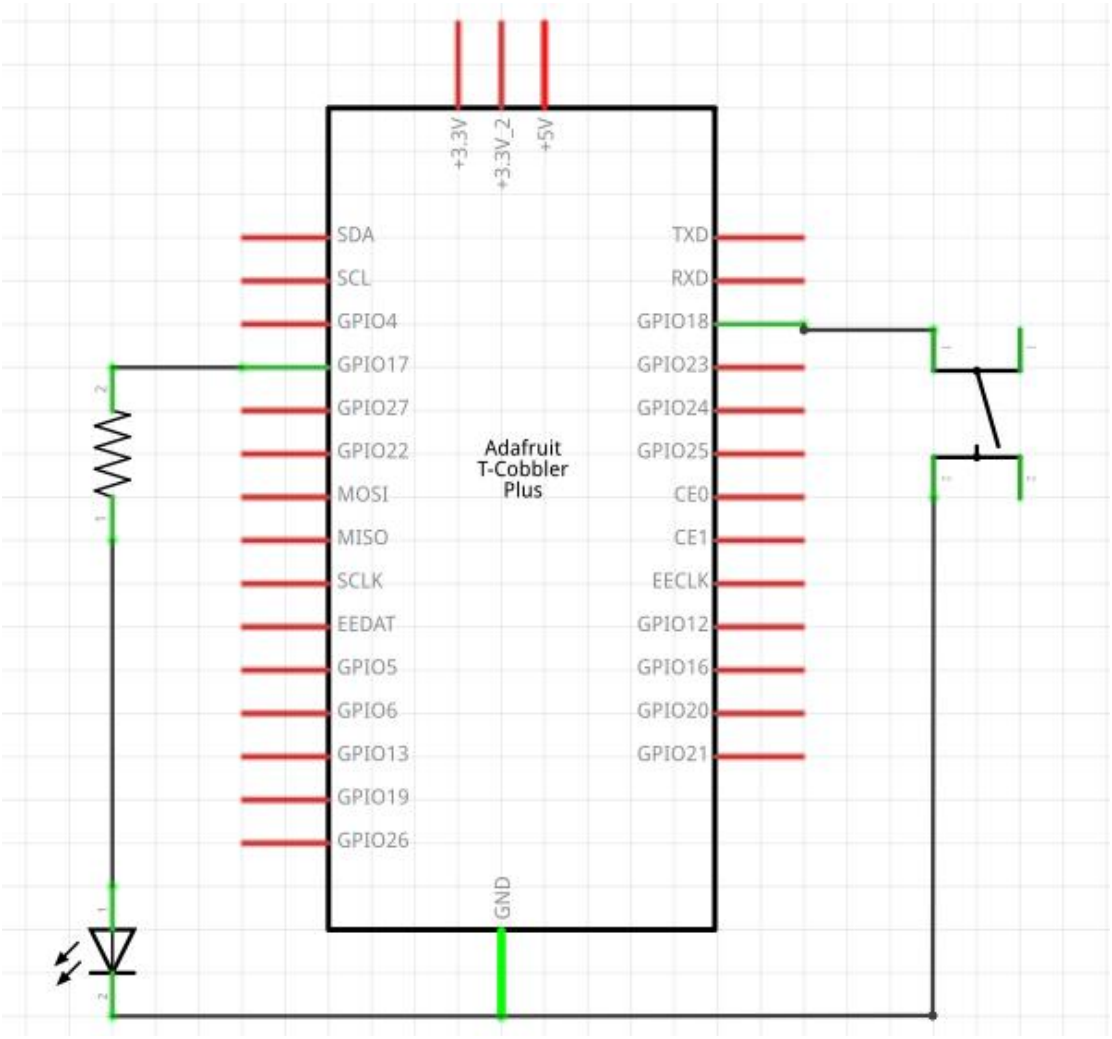
The small push switch used in this lesson has a connection pin.When the push button is pressed, the circuit is turned on.



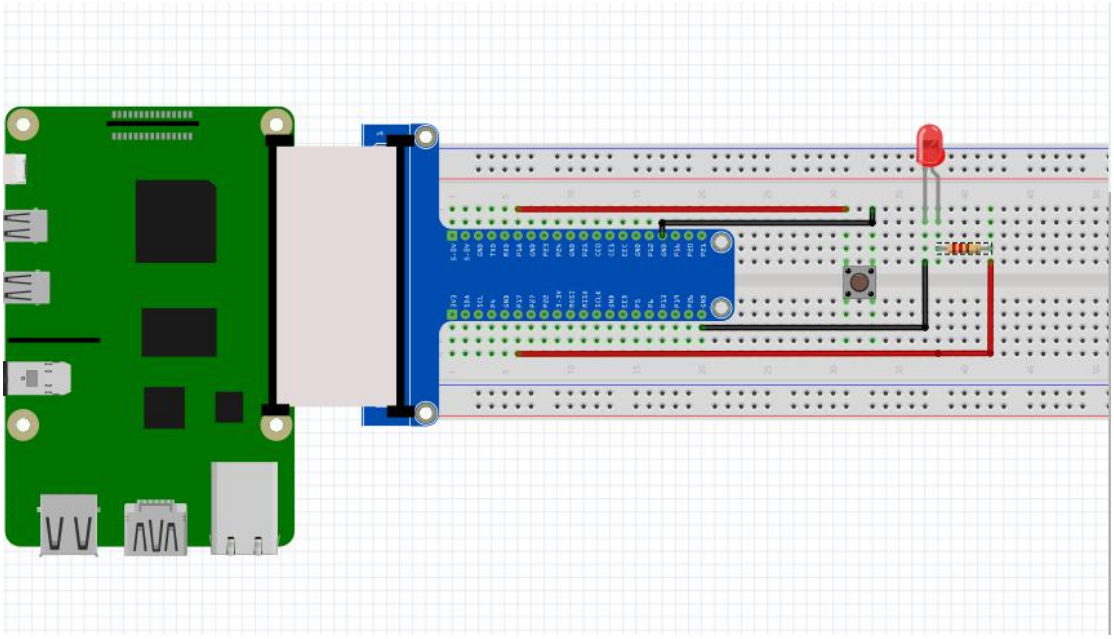


In fact only two pin connections are used. Inside the button, pins B and C are connected together, as are A and D.

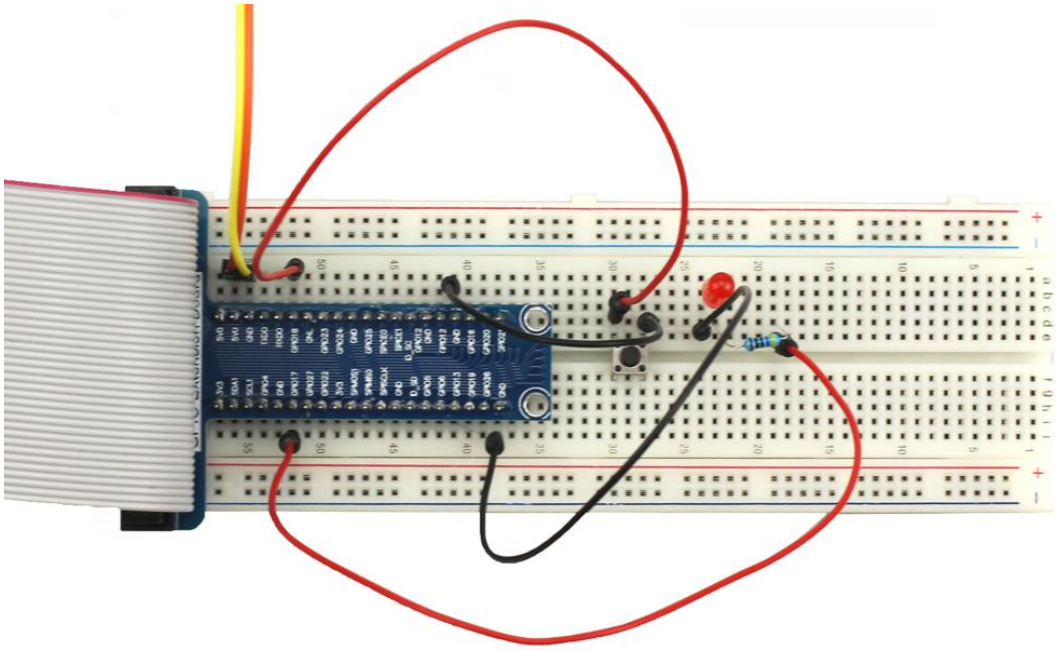
Connection Diagram



Wiring Diagram



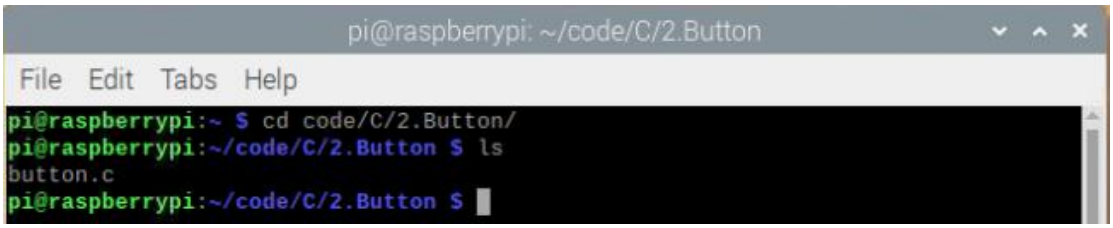
Example Figure



C code

Open the terminal and enter the cd code/C/2.Button/ command to enter the button.c code directory.

Enter the ls command to view the file button.c in the directory.



Enter gcc button.c -o button -lwiringPi command to generate button for executable file in button.c, enter ls command to view;

Run the code by typing sudo ./button command. The result is as follows:

You can press "Ctrl+C" to terminate the program.

The following is the program code:

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

#define ledPin      0
#define buttonPin 1

int main(void)
{
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
```

```
    pullUpDnControl(buttonPin, PUD_UP);
    while(1){

        if(digitalRead(buttonPin) == LOW){
            digitalWrite(ledPin, HIGH);
            printf("led on...\n");
        }
        else {
            digitalWrite(ledPin, LOW);
            printf("...led off\n");
        }
    }

    return 0;
}
```

Code interpretation

```
#define ledPin 0
#define buttonPin 1
```

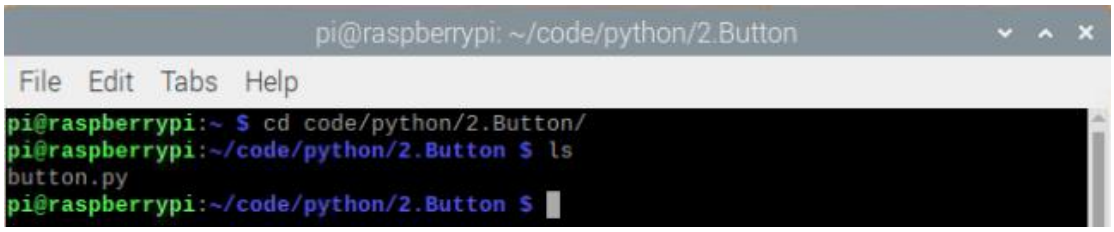
In the circuit connection, the 'LED' and the button are connected to GPIO 17 and GPIO 18, which correspond to 0 and 1 of 'PI' respectively. Therefore, 'ledPin' and 'buttonPin' are defined as 0 and 1 respectively.

```
If(digitalRead(buttonPin) == LOW){
digitalWrite(ledPin, HIGH);
Printf("led on...\n");
}
Else {
digitalWrite(ledPin, LOW);
Printf("...led off\n");
}
```

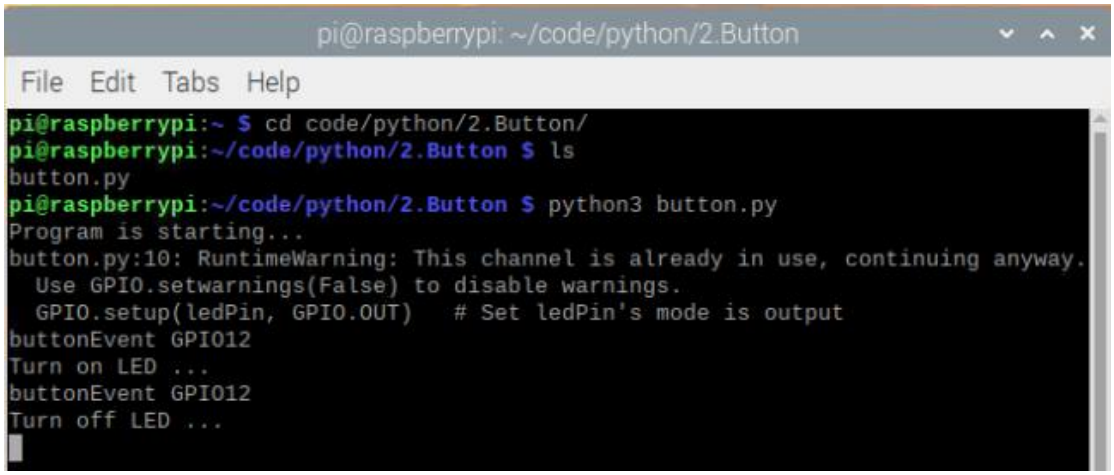
About ‘digitalRead()’:
This function returns the read value on the specified pin. Depending on the logic level of the pin, it will be assigned a value of "High" or "Low" (1 or 0).

Python code

Open the terminal and enter the cd code/python/2.Button/ command to enter the code directory.
Enter the ls command to view the file button.py in the directory.



Run the code by typing the python3 button.py command, and the result is as follows:



The following is the program code:

```
import RPi.GPIO as GPIO
ledPin = 11
buttonPin = 12

def setup():
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin, GPIO.OUT)
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            GPIO.output(ledPin,GPIO.HIGH)
            print ('led on ...')
        else :
            GPIO.output(ledPin,GPIO.LOW)
            print ('led off ...')
```

```
def destroy():
    GPIO.output(ledPin, GPIO.LOW)
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
destroy() will be    executed.
    destroy()
```

Code interpretation

```
import RPi.GPIO as GPIO
ledPin = 11
buttonPin = 12
```

Import the required ‘RPi.GPIO’ module
‘GPIO17’ is to use pin 11 of the mainboard, so define ‘ledPin’ as 11
‘GPIO18’ is to use pin 12 of the ,mainboard, so define 'buttonPin’ as 12

```
def setup():
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin, GPIO.OUT)
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

In 'setup()', 'GPIO.setmode(GPIO.BOARD)' is used to set the mode of the GPIO according to the physical location of the pin. Set 'ledPin' to the output mode and 'buttonPin' to the input mode with pull-up resistor.

```
def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            GPIO.output(ledPin,GPIO.HIGH)
            print ('led on ...')
        else :
            GPIO.output(ledPin,GPIO.LOW)
        print ('led off ...')
```

Add the loop statement 'while True' to the ‘def loop()' statement to continue to determine if the button has been pressed. ‘GPIO.input(buttonPin)’ will return low when the button is pressed. Then the result of "if" is ‘true’, ‘ledPin’ outputs high (LED on), otherwise ‘ledPin’ outputs low (LED off).

Lesson 3 Flowing Water Light

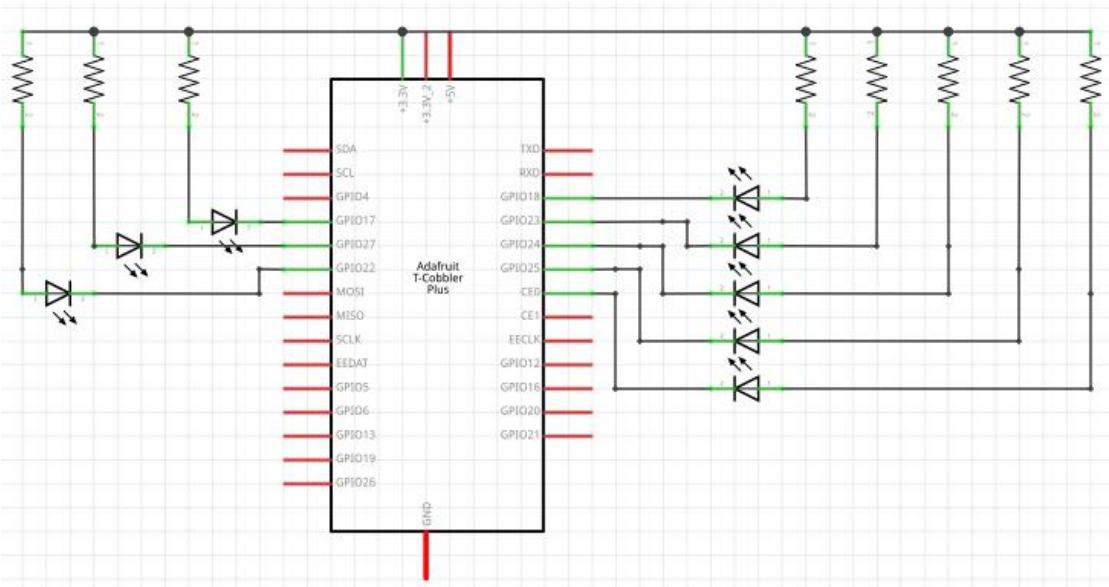
Overview

In this lesson, you will learn how to use a number of LEDs to make flowing water light.

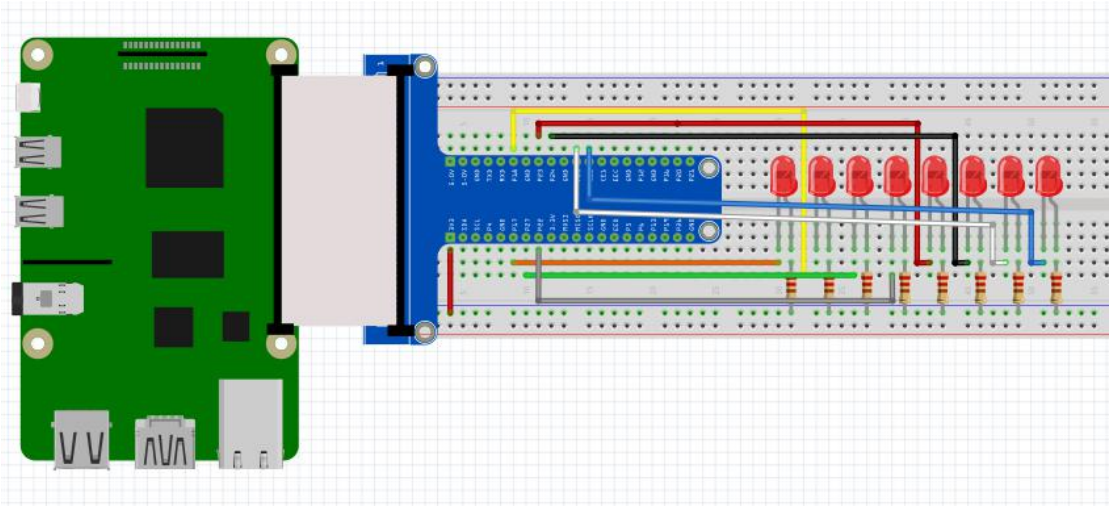
Parts Required

1 x Raspberry Pi
8 x LEDs
8 x 220 ohm resistors
9 x Jumper Wires
1 x breadboard

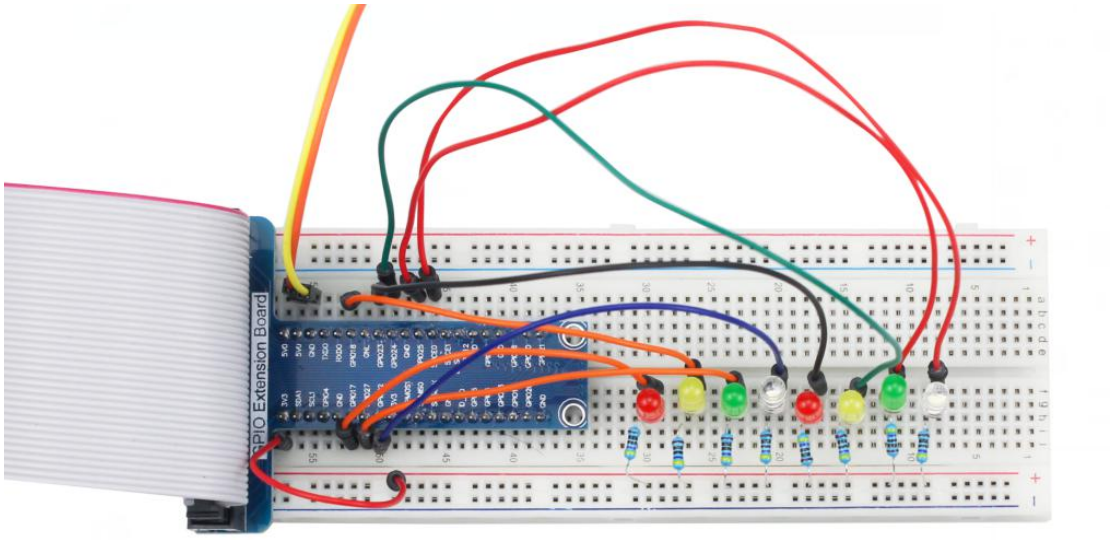
Connection diagram



Wiring diagram



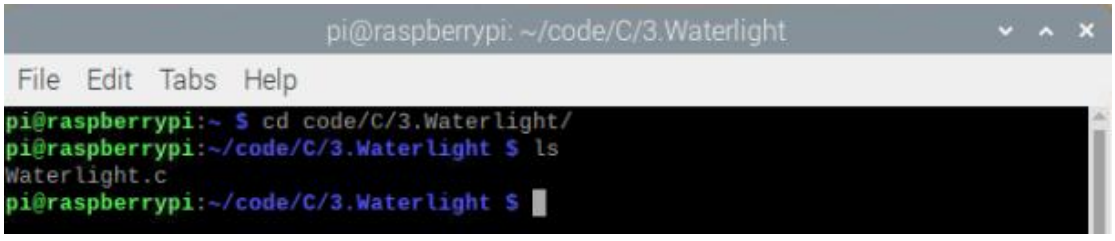
Example Figure



C code

Open the terminal and enter the cd code/C/3.Waterlight/ command to enter the Waterlight.c code directory;

Enter the ls command to view the file in the directory Waterlight.c;



Enter the gcc Waterlight.c -o Waterlight -lwiringPi command to generate Waterlight.c executable file Waterlight, enter the ls command to view;

Run the code by entering sudo ./Waterlight command, and the results are as follows:

Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#define leds 8
int pins[leds] = {0,1,2,3,4,5,6,10};
void led_on(int n)
{
    digitalWrite(n, LOW);
}

void led_off(int n)
{
    digitalWrite(n, HIGH);
}

int main(void)
{
    int i;
    printf("Program is starting ... \n");

if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    for(i=0;i<leds;i++){
        pinMode(pins[i], OUTPUT);
    }
    while(1){
        for(i=0;i<leds;i++){
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
```

```
        for(i=leds-1;i>=0;i--){
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
    }
    return 0;
}
```

Code interpretation

```
while(1){
        for(i=0;i<leds;i++){
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
        for(i=leds-1;i>=0;i--){
            led_on(pins[i]);
            delay(100);
            led_off(pins[i]);
        }
    }
```

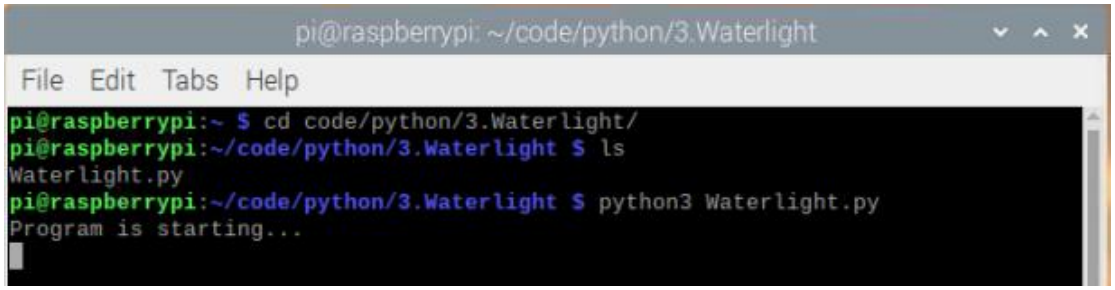
In the program, configure GPIO0-GPIO7 to output mode. Then, in the " while" loop of the main function, two "for" loops are used to effect the flow of water from left to right and from right to left.

Python code

Open the terminal and use the cd code/python/3.Waterlight/ command to enter the code directory;

Enter the command ls to view the file Waterlight.py under the directory;

Run the code by typing Python3 Waterlight.py command, and the results are as follows:



The following is the program code:

```
import RPi.GPIO as GPIO
import time
ledPins = [11, 12, 13, 15, 16, 18, 22, 24]
def setup():
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    for pin in ledPins:
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, GPIO.HIGH)
def loop():
    while True:
        for pin in ledPins:
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)

    for pin in ledPins[::-1]:
            GPIO.output(pin, GPIO.LOW)
            time.sleep(0.1)
            GPIO.output(pin, GPIO.HIGH)

def destroy():
    for pin in ledPins:
        GPIO.output(pin, GPIO.HIGH)
```

```
        GPIO.cleanup()

    if __name__ == '__main__':
        setup()
        try:
            loop()
        except KeyboardInterrupt:
      destroy() will be    executed.
            destroy()
```

Code interpretation

Import RPi.GPIO as GPIO

Import time

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

Import the required RPi.GPIO, time module

We need 8 pins for the streamer, so define a variable array 'ledPins' storage pin.

Def setup():

Print ('Program is starting...')

GPIO.setmode(GPIO.BOARD)

For pin in ledPins:

GPIO.setup(pin, GPIO.OUT)

GPIO.output(pin, GPIO.HIGH)

Def loop():

While True:

For pin in ledPins:

GPIO.output(pin, GPIO.LOW)

Time.sleep(0.1)

GPIO.output(pin, GPIO.HIGH)

For pin in ledPins[::-1]:

GPIO.output(pin, GPIO.LOW)

Time.sleep(0.1)

GPIO.output(pin, GPIO.HIGH)

In the 'loop()' function, two "for" loops are used to implement the right-to-left and left-to-right flow lights, where 'ledPins [:: - 1]' is used to traverse 'ledPins' in reverse order 'Elements.

Lesson 4 Analog & PWM

Overview

In this lesson, we will learn how to control the brightness of a LED.

Parts Required

1 x Raspberry Pi

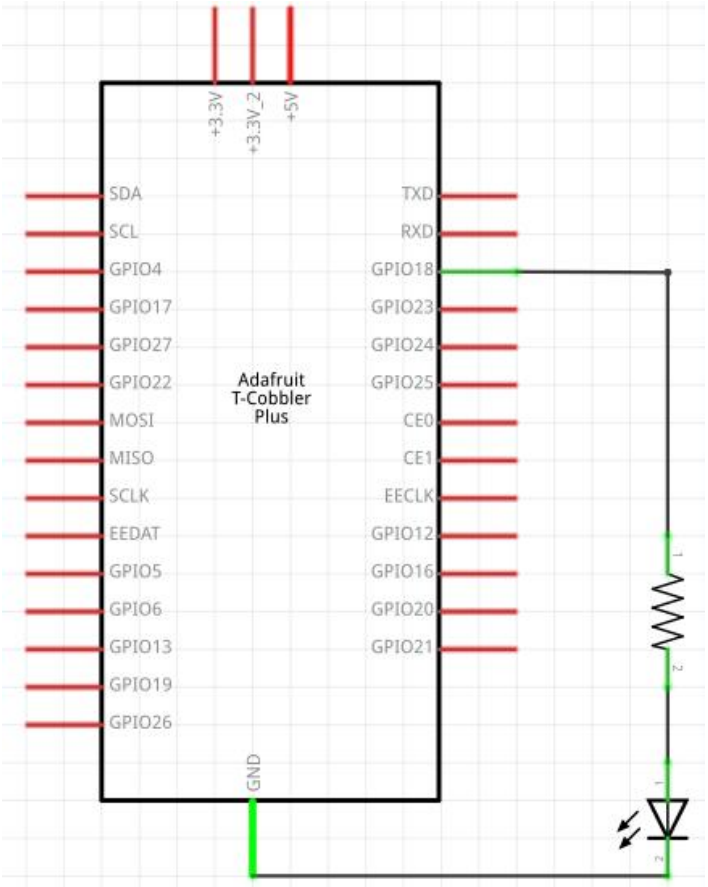
1 x LED

1 x 220 ohm Resistor

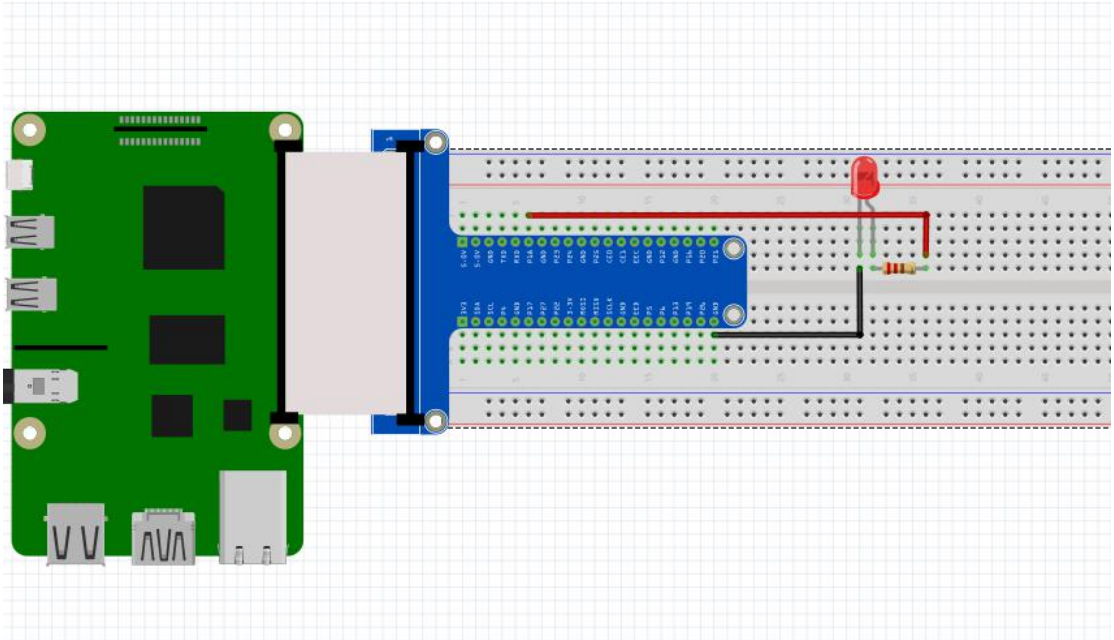
2 x Jumper Wires

1 x Breadboard

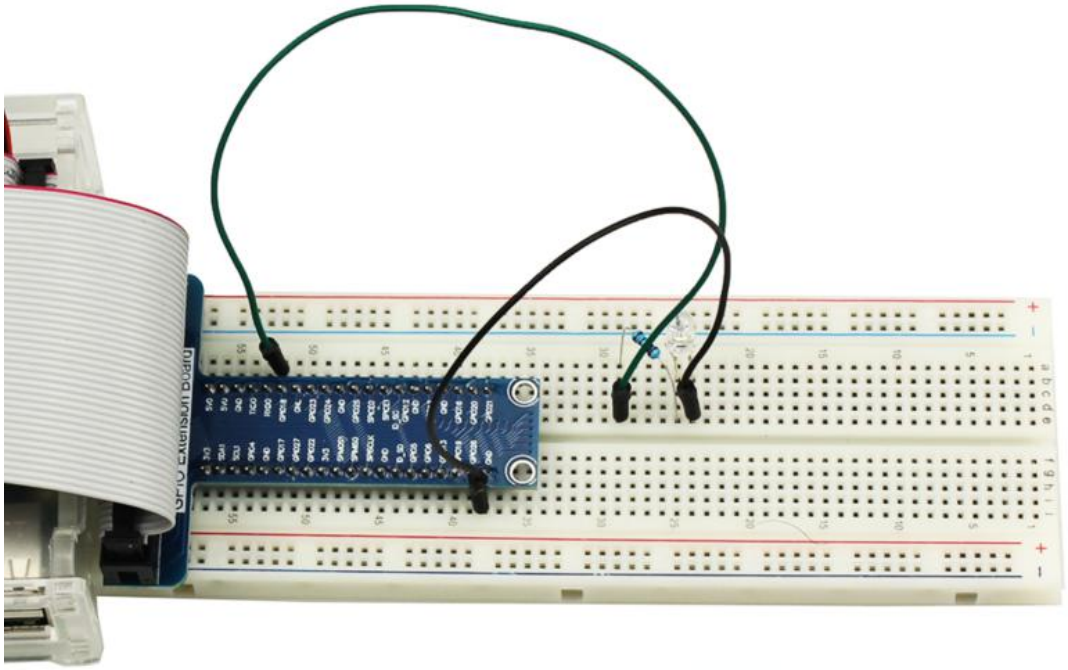
Connection Diagram



Wiring Diagram



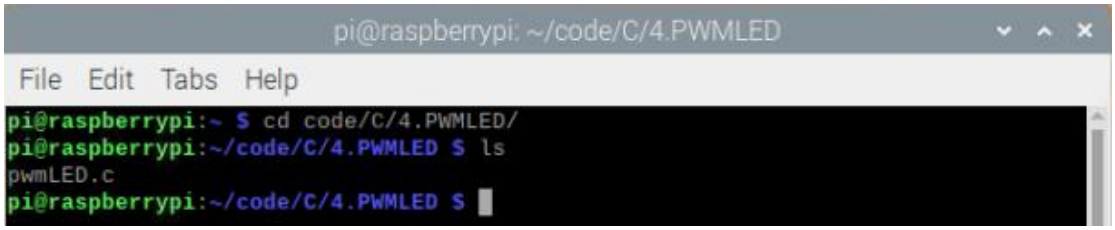
Example Figure



C code

Open the terminal and input cd code/C/4.PWMLED/ command to enter the pwmLED.c code directory;

Enter the ls command to view the file pwmLED.c in the directory.



Enter the gcc Waterlight.c -o Waterlight -lwiringPi command to generate Waterlight.c executable file Waterlight, enter the ls command to view;

Run the code by entering sudo ./Waterlight command, and the results are as follows:

Press "Ctrl + c" to end the program. The following is the program code:

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

#define ledPin      1
int main(void)
{
    int i;

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

}

    softPwmCreate(ledPin,   0, 100);

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

Code interpretation

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

There are two "for" loops in the "while" loop. The first makes the ledPin output PWM from 0% to 100%, and the second makes the ledPin output PWM from 100% to 0%.

You can also adjust the rate at which the LED status changes by changing the parameters of the delay() function in the "for" loop.

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

Create a software controlled PWM pin.

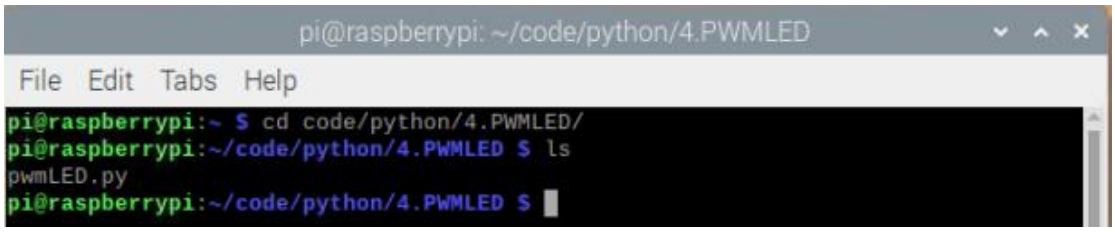
```
void softPwmWrite (int pin, int value) ;
```

Update the PWM value on the pin.

Python code

Open the terminal and use the cd code/python/4.PWMLED/ command to enter the code directory;

Enter the command ls to view the file pwmLED.py in the directory.



Run the code with the command python3 pwmLED.py and the results are as follows:

The following is the program code:

```
import RPi.GPIO as GPIO
import time
LedPin = 12
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(LedPin, GPIO.OUT)
    GPIO.output(LedPin, GPIO.LOW)
p = GPIO.PWM(LedPin, 1000)

p.start(0)

def loop():
    while True:
        for dc in range(0, 101, 1):
            p.ChangeDutyCycle(dc)
            time.sleep(0.01)
        time.sleep(1)
        for dc in range(100, -1, -1):
            p.ChangeDutyCycle(dc)
            time.sleep(0.01)
        time.sleep(1)

def destroy():
    p.stop()
    GPIO.output(LedPin, GPIO.LOW)
```

```
        GPIO.cleanup()

    if __name__ == '__main__':
        setup()
        try:
            loop()
        except KeyboardInterrupt:
    destroy() will be    executed.
            destroy()
```

Code interpretation

```
Def setup():
   Global p
   GPIO.setmode(GPIO.BOARD)
   GPIO.setup(LedPin, GPIO.OUT)
   GPIO.output(LedPin, GPIO.LOW)
p = GPIO.PWM(LedPin, 1000)
   P.start(0)
```

Define a variable 'p' of the type 'global', which is used to set the mode of the GPIO according to the physical location of the pin. The LED is connected to an IO port called GPIO18. And ‘LedPin’ is defined as 12 and is set to the output mode according to the pin physical mode.

Then create a PWM instance and set the PWM frequency to 1000HZ with an initial duty cycle of 0%.

```
Def loop():
   While True:
      For dc in range(0, 101, 1):
         p.ChangeDutyCycle(dc)
```

```
        Time.sleep(0.01)
    Time.sleep(1)
    For dc in range(100, -1, -1):
        p.ChangeDutyCycle(dc)
        Time.sleep(0.01)
    Time.sleep(1)
```

In the "while" loop, there are two "for" loops for LED breathing effects. The first makes the 'ledPin' output PWM from 0% to 100%, and the second makes the 'ledPin' output PWM from 100% to 0%.

Lesson 5 RGBLED

Overview

In this lesson you will learn how to use the Raspberry Pi power supply and resistors to illuminate the RGB LEDs together.

Parts Required

1 x Raspberry Pi

1 x RGB LED

3 x 220 ohm resistor

4 x Jumper Wires

1 x breadboard

Product Introduction

RGB LED

The RGB LED looks like a normal LED from the surface, except that the RGB LED has four pins, and the regular LED only has two pins. In fact, the RGB LED is internally packaged with three LEDs, one red, one green, and one blue.

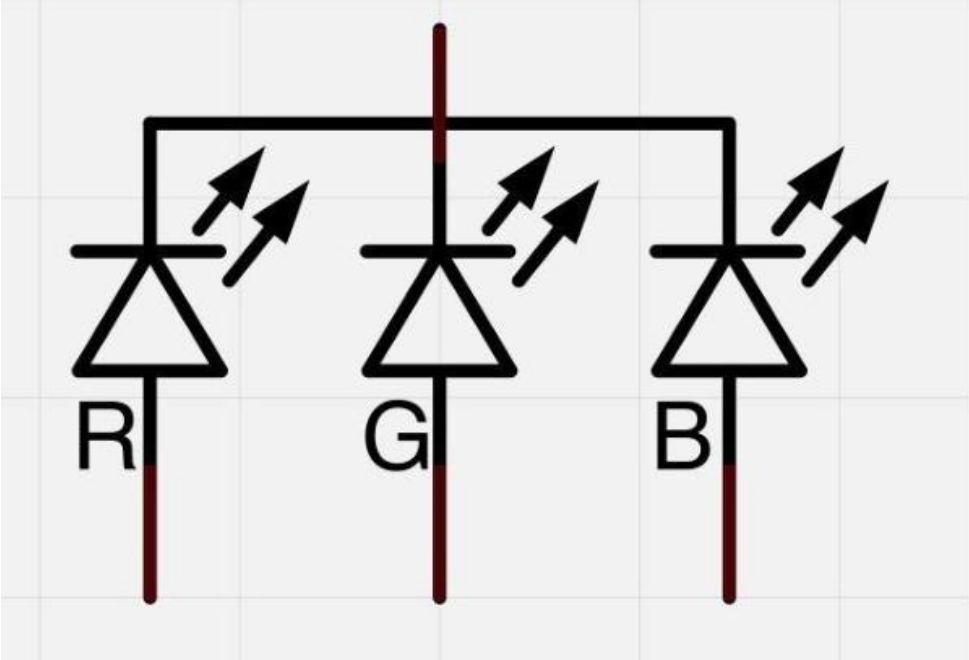
You can show any color you want by adjusting the brightness of each of the three LEDs. The way in which the LED colors are mixed is the same as the way in which the paint is mixed on the palette. It's very difficult to achieve the color adjustment without the main control board. Fortunately, we can use the Raspberry Pi to simplify our workload. This board has the function of analog writing. You can control the LED by using the board marked label "~" pins to output variable power.

In this tutorial, we will use the common cathode LED. One pin is connected to ground, while the other three pins are connected to the Raspberry Pi digital pin with “~”.

This pin does not require a resistor to connect directly to GND. However, the other three pins require resistors to prevent excessive current from flowing. The other three pins also become positive pins. The left side of the CATHODE is the GREEN and BLUE pins in turn, and the right side is the RED pin.

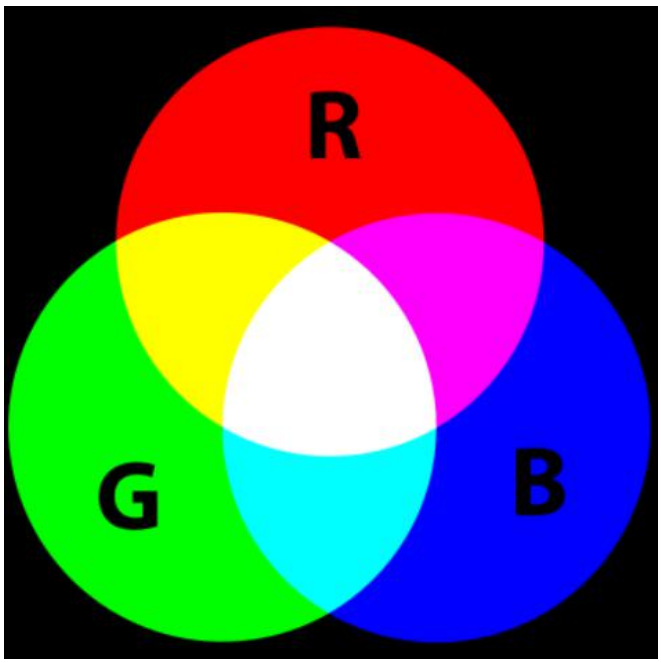
Color

We can mix any color we want by changing the luminance of red, green, and blue LEDs.



Theoretically speaking，our eyes have three types of light receivers (red, green, blue). Our eyes and brain receive and process the red, green, and blue hues and convert them into a spectral color. Nowadays, electronic products such as color TVs, computers and

mobile phones can display different colors, which are mixed with RGB three colors. The principle is the same.

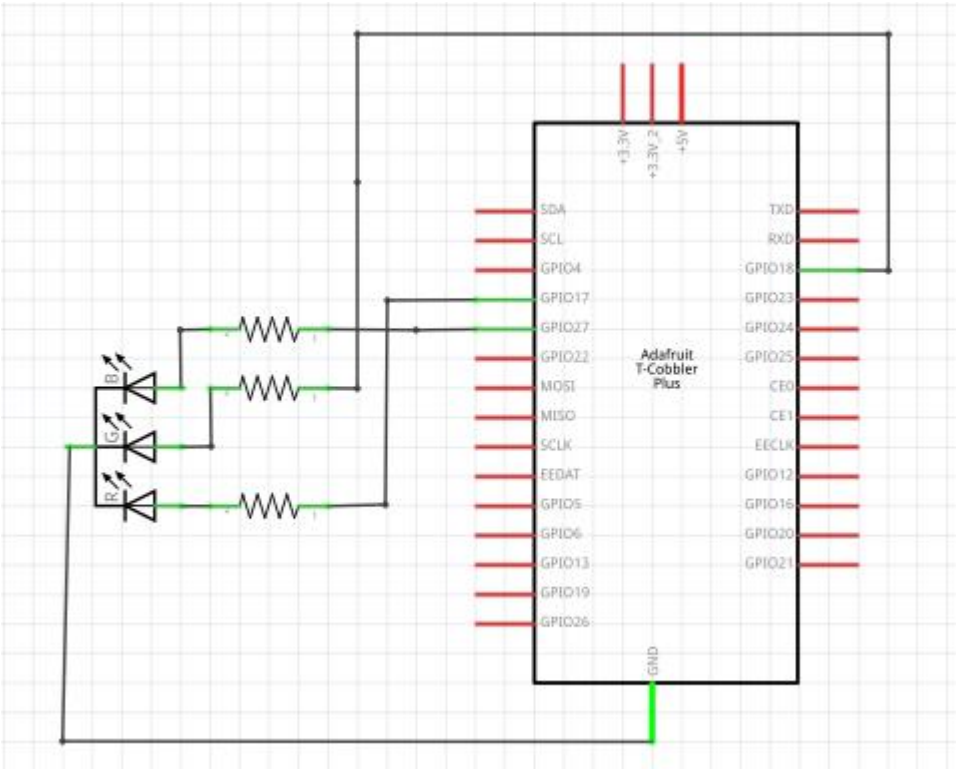


For example, if we set the three LEDs to the same brightness, the overall light color will be displayed in white. If we turn off the blue LED, only the red and green LEDs with the same brightness, then the LED will be color yellow.

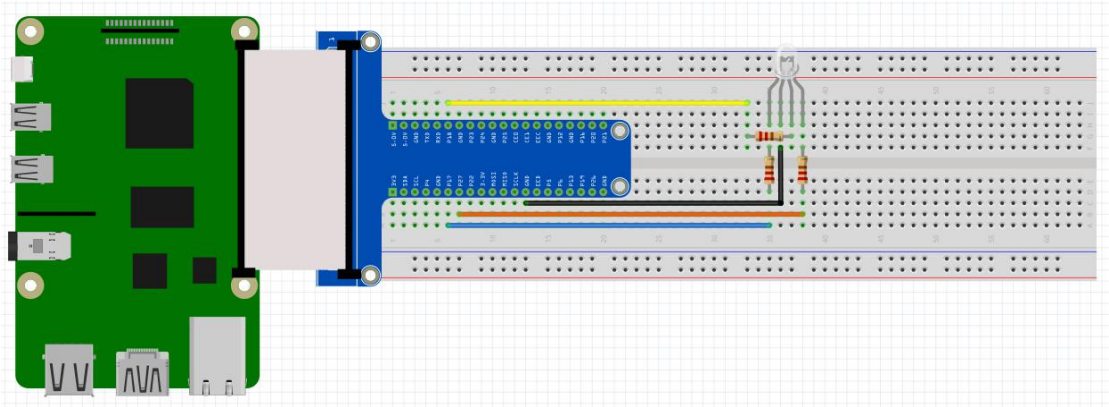
We can easily control the brightness of red, green and blue of each RGB LED, which makes it easy to control different colors.

To adjust to color black, we need to adjust the brightness of the three colors to the lowest. Because black is a nearly matt color.

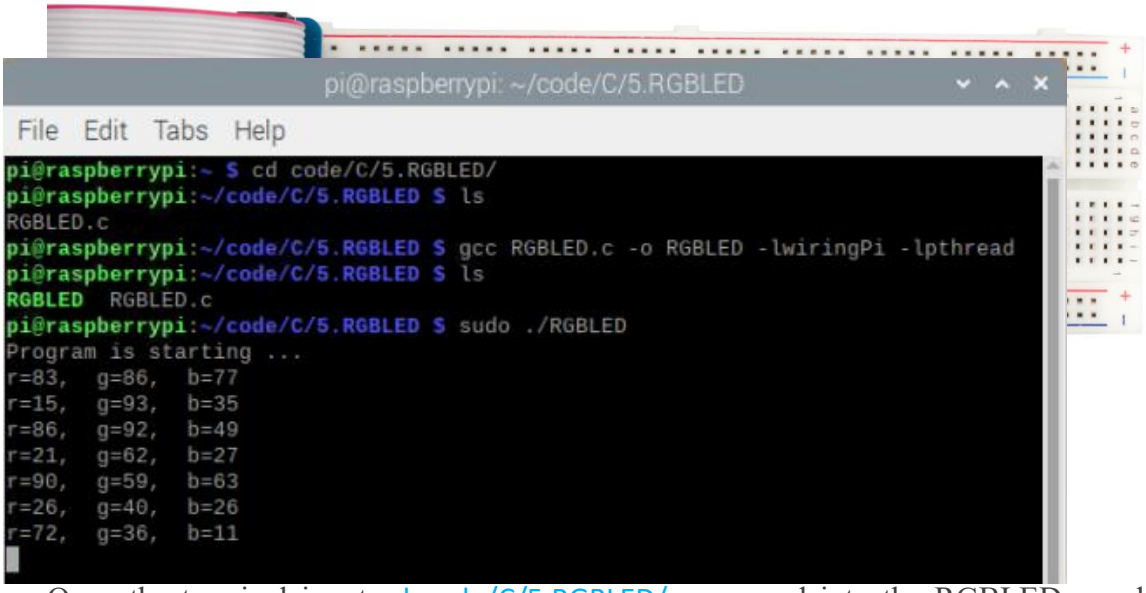
Connection diagram



Wiring Diagram

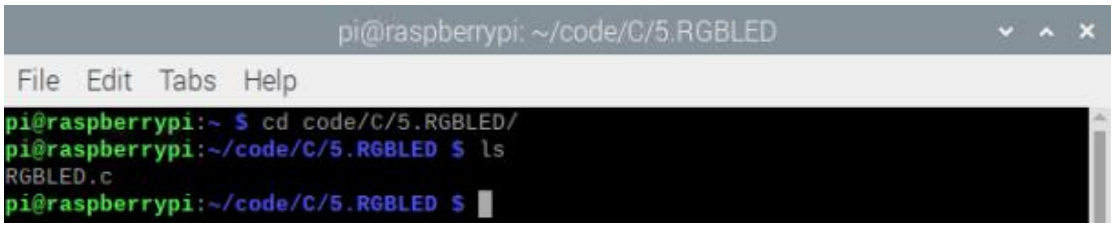


Example Figure



Open the terminal input cd code/C/5.RGBLED/ command into the RGBLED.c code directory;

Enter the ls command to view the file in the directory RGBLED.c;



Enter gcc RGBLED.c -o RGBLED -lwiringPi -lpthread command to generate RGBLED.c executable file RGBLED, enter ls command to view;

Enter the sudo ./RGBLED command to run the code, and the results are as follows:

Press "Ctrl + c" to end the program. The following is the program code:

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

#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(int r_val, int g_val, int b_val)
{
    softPwmWrite(ledPinRed,      r_val);
    softPwmWrite(ledPinGreen, g_val);

softPwmWrite(ledPinBlue,    b_val);
}

int main(void)
{
    int r,g,b;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    printf("Program is starting ...\n");
```

```
    ledInit();

    while(1){
        r=random()%100;
        g=random()%100;
        b=random()%100;
        ledColorSet(r,g,b);
        printf("r=%d,    g=%d,    b=%d \n",r,g,b);
        delay(300);
    }
    return 0;
}
```

Code interpretation

```
Void ledInit(void)

{

softPwmCreate(ledPinRed, 0, 100);

softPwmCreate(ledPinGreen,0, 100);

softPwmCreate(ledPinBlue, 0, 100);

}
```

In the subfunction of 'ledInit()', create a software PWM control pin that controls the R, G, and B pins, respectively.

```
Void ledColorSet(int r_val, int g_val, int b_val)

{

softPwmWrite(ledPinRed, r_val);

softPwmWrite(ledPinGreen, g_val);

softPwmWrite(ledPinBlue, b_val);

}
```

Then create a sub-function and set the PWM for the three pins.

```
While(1){
r=random()%100;
g=random()%100;
b=random()%100;
ledColorSet(r,g,b);
printf("r=%d, g=%d, b=%d \n",r,g,b);
delay(300);
}
```

Finally, in the "while" cycle of the main function, three random numbers are obtained and designated as the PWM duty cycle and assigned to the corresponding pins. Therefore, RGB LED can always switch colors randomly.

The related functions of the software PWM can be described as follows:

```
Long random();
```

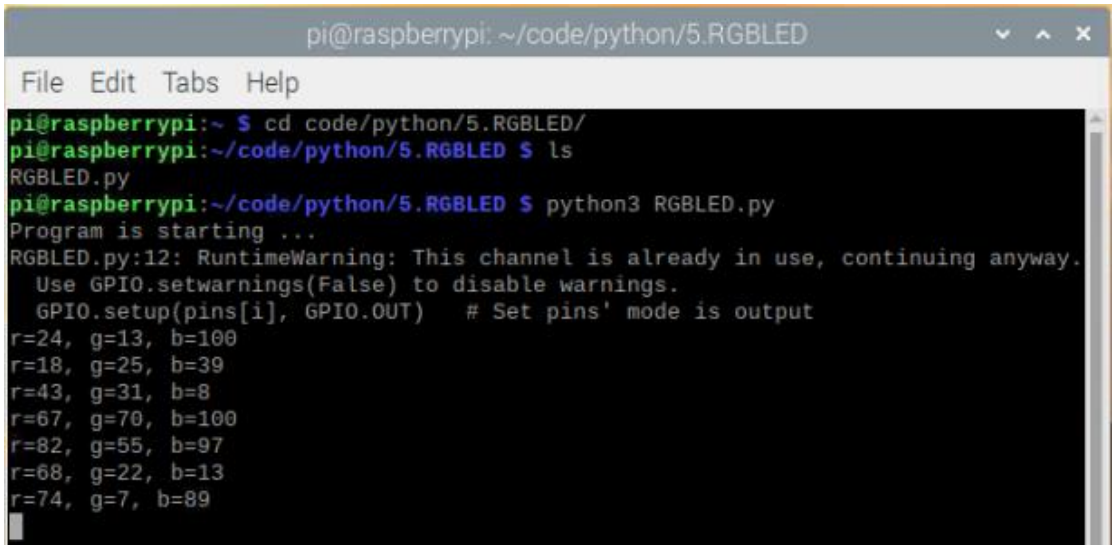
This function will return a random number.

Python code

Open the terminal and use cd code/python/5.RGBLED/ command to enter the code directory;

Enter ls to view the file RGBLED.py in the directory.

Enter python3 RGBLED.py to run the code and the results are as follows:



The following is the code:

```
import RPi.GPIO as GPIO
import time
import random
pins = {'pin_R':11, 'pin_G':12, 'pin_B':13}
def setup():
        global p_R,p_G,p_B
        print ('Program is starting ... ')
        GPIO.setmode(GPIO.BOARD)
        for i in pins:
                GPIO.setup(pins[i], GPIO.OUT)

GPIO.output(pins[i], GPIO.HIGH)
        p_R = GPIO.PWM(pins['pin_R'], 2000)
        p_G = GPIO.PWM(pins['pin_G'], 2000)
        p_B = GPIO.PWM(pins['pin_B'], 2000)
        p_R.start(0)
        p_G.start(0)
        p_B.start(0)
def setColor(r_val,g_val,b_val):
```

```
        p_R.ChangeDutyCycle(r_val)
        p_G.ChangeDutyCycle(g_val)
        p_B.ChangeDutyCycle(b_val)
def loop():
        while True :
                r=random.randint(0,100)#get a random in (0,100)
                g=random.randint(0,100)
                b=random.randint(0,100)
                setColor(r,g,b)#set random as a duty cycle value
                print ('r=%d, g=%d, b=%d ' %(r ,g, b))
                time.sleep(0.3)
def destroy():
        p_R.stop()
        p_G.stop()
        p_B.stop()
        GPIO.cleanup()
if __name__ == '__main__':
        setup()
        try:
                loop()
        except KeyboardInterrupt:
destroy() will be    executed.

        destroy()
```

Code interpretation

```
def loop ():
while    True : :

r= =random. .randint( (0, ,100) )
g= =random. .randint( (0, ,100) )
b= =random. .randint( (0, ,100) )
setColor( (r, ,g, ,b) )
print ( ('r=%d, g=%d, b=%d '    %(r , ,g, , b ))
time. .sleep( (0.3) )
```

In the previous chapter, we learned how to make a pin-out PWM using the Python language. In this project, we have three pins output PWM, the usage is exactly the same as the previous chapter. In the "while" cycle of the "loop" function, we first get three random numbers and then specify the three random numbers as the PWM values of the three pins. Make RGBLEDs randomly switch between different colors.

random.randint(a, b)

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

Lesson 6 Active Buzzer

Overview

In this lesson you will learn how to make a buzzer sound by using the Raspberry Pi.

Parts Required

1 x Raspberry Pi

1 x Active Buzzer

1 x 220 Ohm Resistor

6 x Male to Male Jumper Wires

1 x Breadboard

1 x Button

1 x S8050NPN Triode

Product Introduction

Active Buzzer

Buzzers are DC-powered and equipped with an integrated circuit. They are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic devices, telephones, timers and other electronic products for voice devices.

Buzzers can be categorized as active and passive ones. Looking the side of the pins, the one with a green circuit board is a passive buzzer, while the other enclosed with a black tape is an active one.

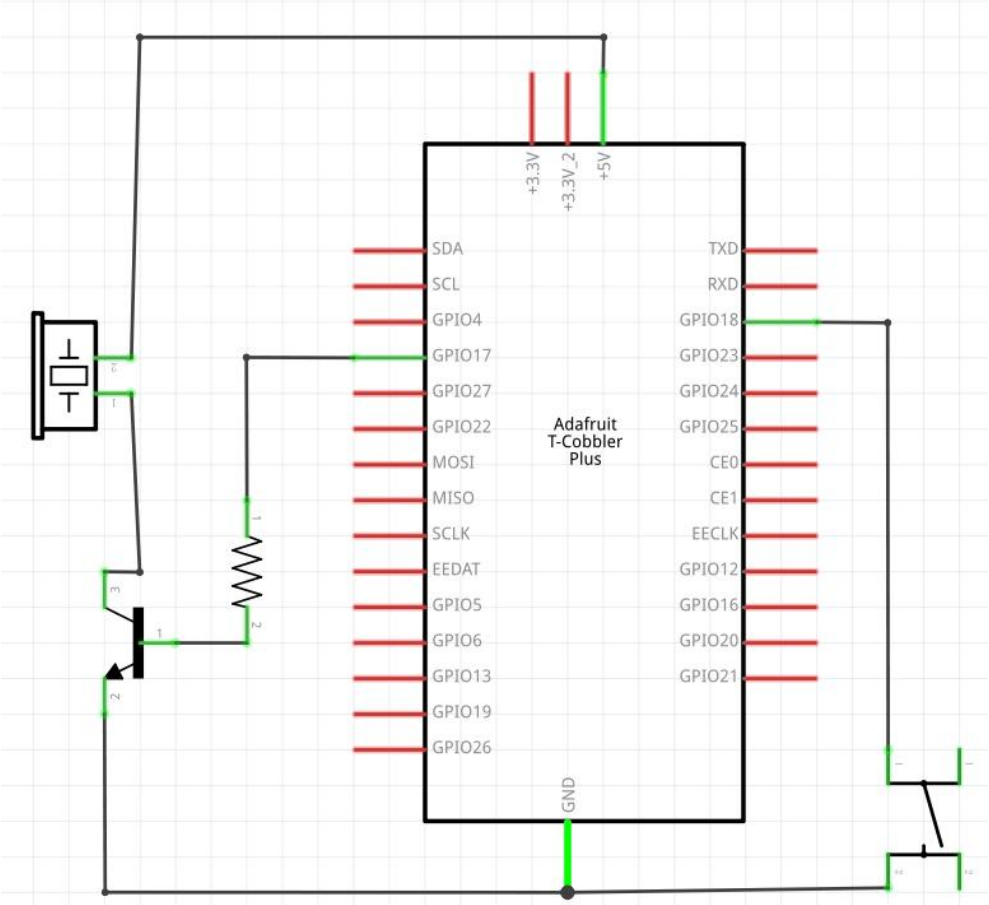
The difference is that the active buzzer has built-in oscillations, so it produces sound when powered. Passive buzzers do not have such a source, so no sound is produced if a DC signal is used; instead, you need to use a square wave with a frequency between 2k and 5k to drive it.

S8050 Triode

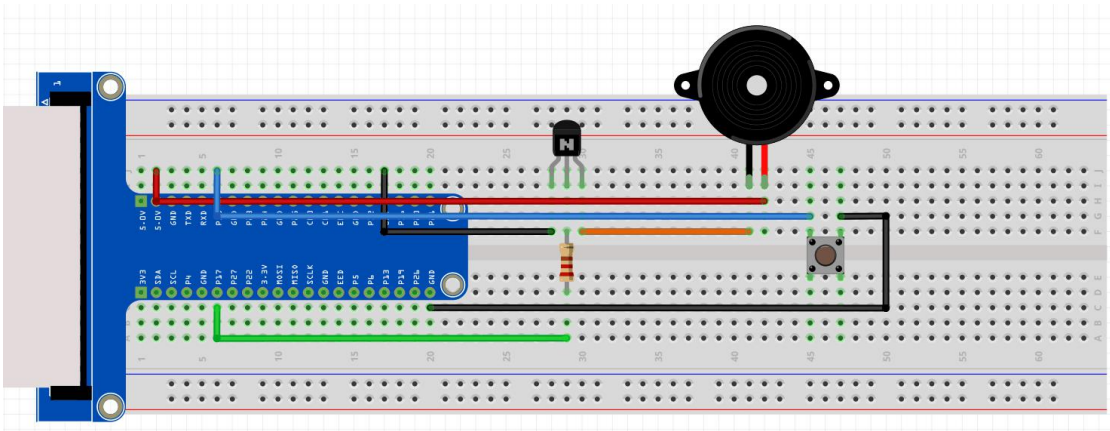
The triode S8050 is a low power NPN silicon tube. The collector-base (Vcbo) voltage can be up to 40V and the collector current is (Ic) 0.5A. S8050 is one of the most commonly used semiconductor triode models for circuit hardware design.

It is often seen in various amplifier circuits, and has a wide range of applications, mainly for high-frequency amplification. Can also be used as a switching circuit. From left to right, the pins are the emitter, base, and collector.

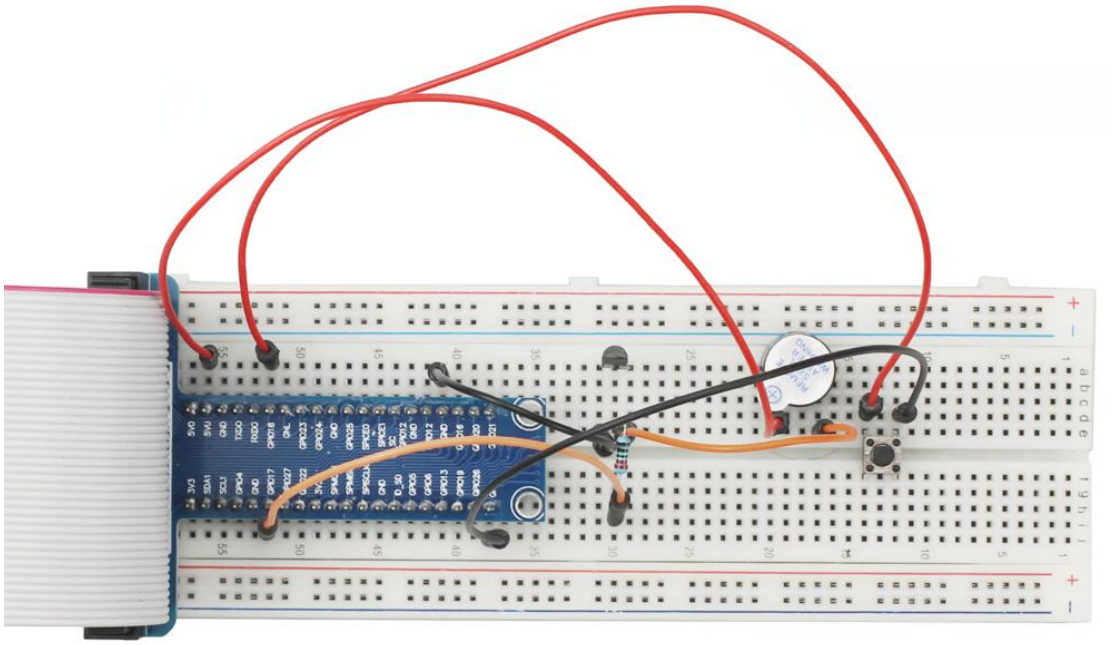
Connection Diagram



Wiring Diagram



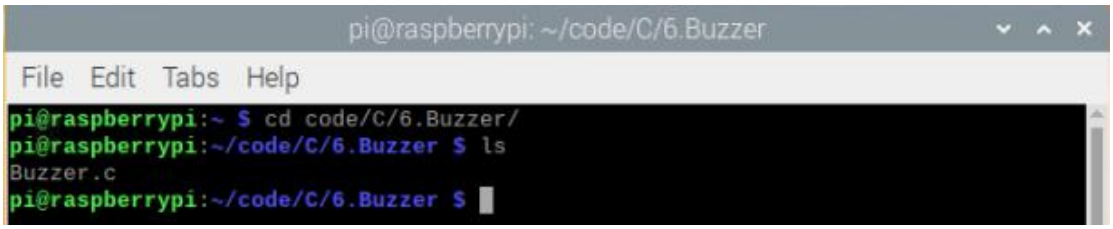
Example picture



C code

Open terminal and enter cd code / C / 6.Buzzer / command to enter Buzzer.c code directory;

Enter the ls command to view the file Buzzer.c in the directory;



Enter gcc Buzzer.c -o buzzer -lwiringPi command to generate Buzzer.c executable buzzer, enter ls command to view;

Enter the sudo ./buzzer command to run the code. The result is as follows:

Press "Ctrl + c" to end the program. The following is the program code:

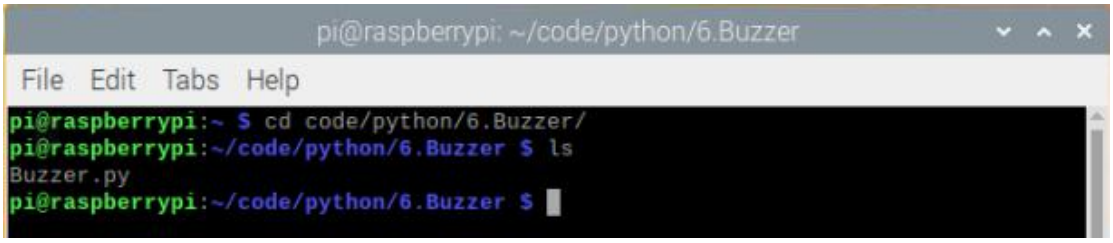
```
#include <wiringPi.h>
#include <stdio.h>

#define buzzerPin       0
#define buttonPin 1

int main(void)
{
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);

    pullUpDnControl(buttonPin, PUD_UP);
    while(1){

        if(digitalRead(buttonPin) == LOW){
            digitalWrite(buzzerPin, HIGH);
            printf("buzzer on...\n");
        }
        else {
            digitalWrite(buzzerPin, LOW);
            printf("...buzzer off\n");
        }
    }
```

```
    return 0;
}
```

Python code

Open terminal and use the cd code / python / 6.Buzzer / command to enter the code directory

Enter the command ls to view the file Buzzer.py in the directory.



Enter the command python3 Buzzer.py to run the code. The result is as follows:

The following is the code:

```
import RPi.GPIO as GPIO
buzzerPin = 11
buttonPin = 12
def setup():
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(buzzerPin, GPIO.OUT)
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            GPIO.output(buzzerPin,GPIO.HIGH)
            print ('buzzer on ...')
        else :
```

```
                GPIO.output(buzzerPin,GPIO.LOW)
                print ('buzzer off ...')
def destroy():
    GPIO.output(buzzerPin, GPIO.LOW)
    GPIO.cleanup()

if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
  destroy() will be    executed.
        destroy()
```

Code Interpretation

```
def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            GPIO.output(buzzerPin,GPIO.HIGH)
            print ('buzzer on ...')
        else :
            GPIO.output(buzzerPin,GPIO.LOW)
            print ('buzzer off ...')
```

Implement all actions in the 'loop ()' function. Use the 'if' statement to determine when the key is pressed. If pressed, use the 'GPIO.output (buzzerPin, GPIO.HIGH)' statement to enable the buzzer, otherwise use 'GPIO.output (buzzerPin, GPIO.LOW)' statement turns the buzzer off.

Lesson 7 PassiveBuzzer

Overview

In this lesson you will learn how to make a passive buzzer sound with a Raspberry Pi.

Parts Required

1 x Raspberry Pi

1 x Passive Buzzer

1 x 1K Resistor

6 x Double Male Jumper Wires

1 x Breadboard

1 x Button

1 x S8050NPN Triode

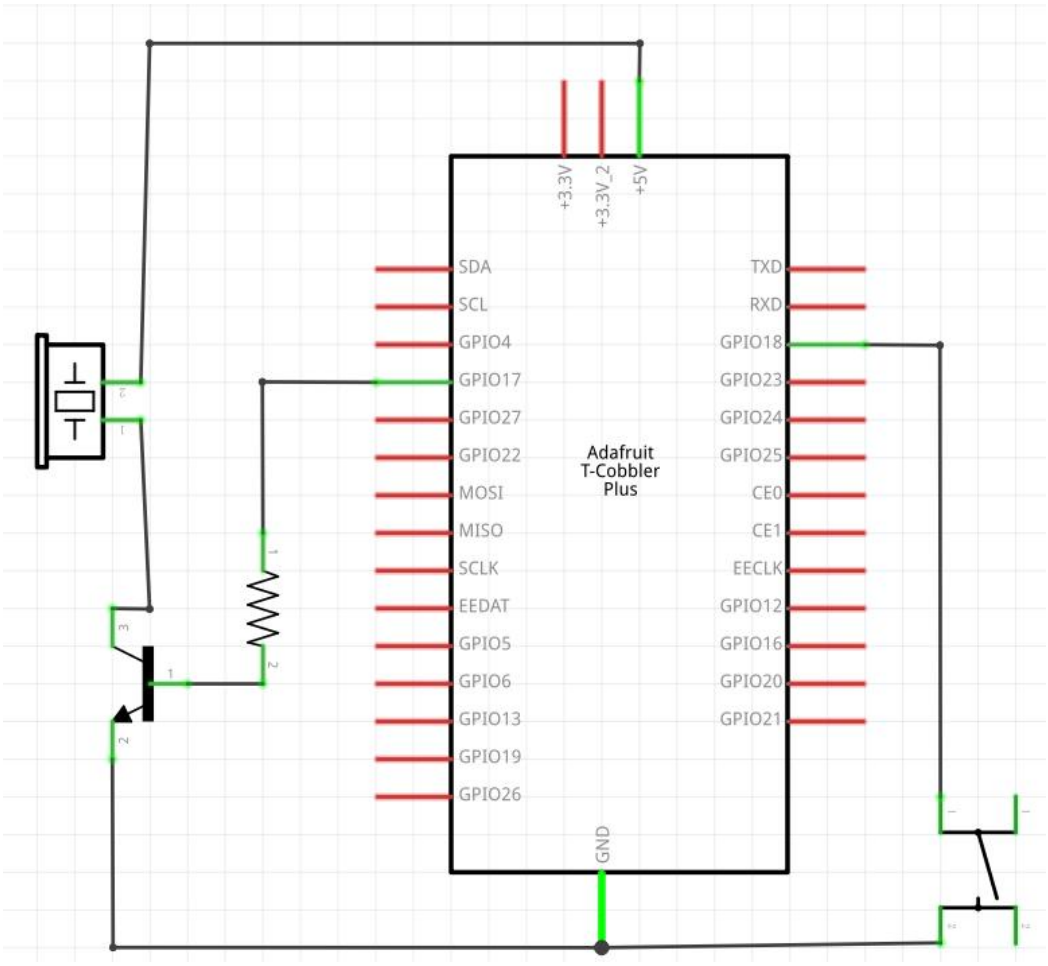
Product Introduction

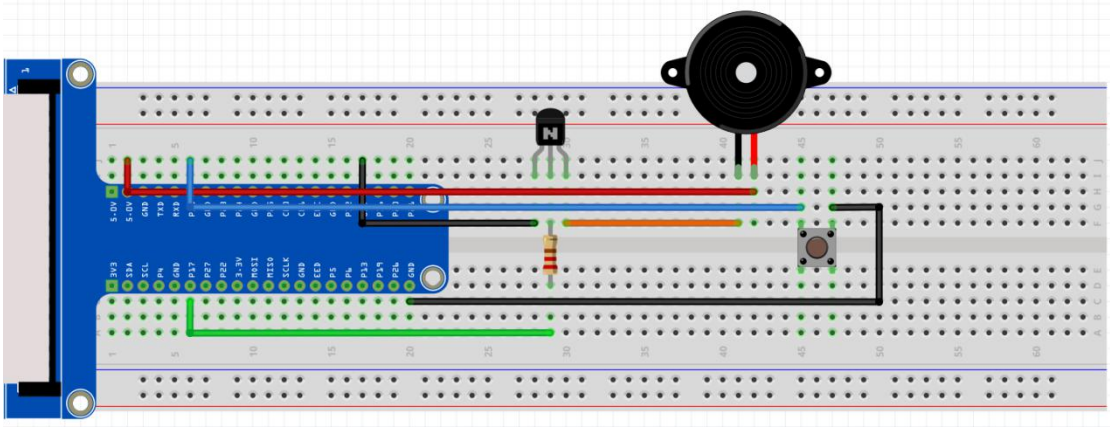
Passive Buzzer

The passive buzzer works by using PWM to generate sound by achieving air vibration. As long as the vibration frequency is changed, different sounds can be produced. For example, send a 523Hz pulse, which can generate Do, pulse 587Hz, it can generate IF Re, 659Hz pulse, it can generate Mi.

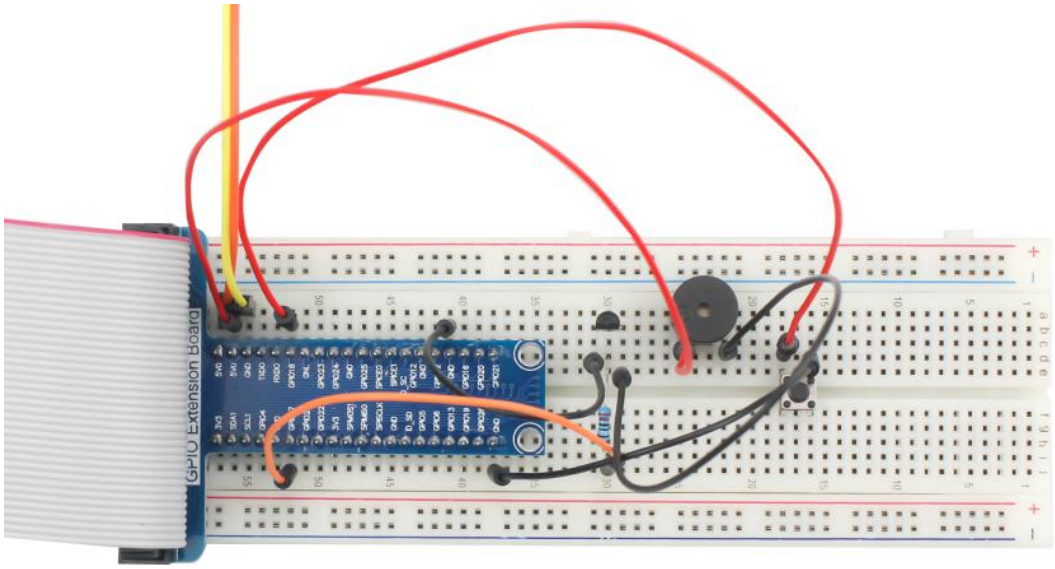
Connection Diagram

Wiring Diagram





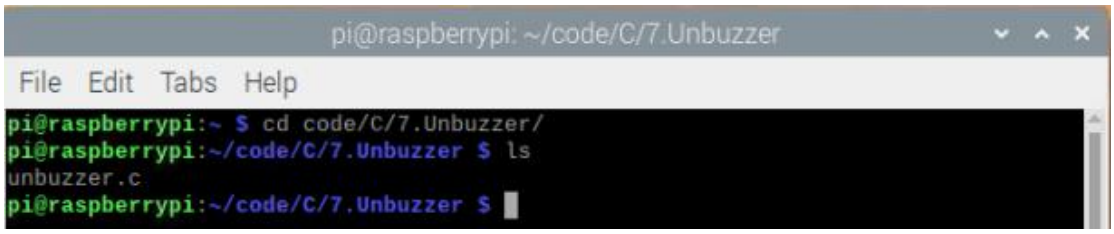
Example picture



C code

Open the terminal and enter the cd code / C / 7.Unbuzzer / command to enter the unbuzzer.c code directory;

Enter the ls command to view the file unbuzzer.c in the directory;



Enter gcc unbuzzer.c -o unbuzzer -lwiringPi -lm -lpthread command to generate unbuzzer.c executable file unbuzzer, enter ls command to view

Enter the sudo ./unbuzzer command to run the code. The result is as follows:

Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softTone.h>
#include <math.h>
#define buzzeRPin 0
#define buttonPin 1
void alertor(int pin){
        int x;
        double sinVal, toneVal;
        for(x=0;x<360;x++){
        sinVal = sin(x * (M_PI / 180));
        toneVal = 2000 + sinVal * 500;
        softToneWrite(pin,toneVal);
        delay(1);
      }
}
void stopAlertor(int pin){
     softToneWrite(pin,0);
}
```

```
int main(void)
{
    if(wiringPiSetup() == -1){
    printf("setup wiringPi failed !");
    return 1;
  }
  pinMode(buzzeRPin, OUTPUT);
  pinMode(buttonPin, INPUT);
  softToneCreate(buzzeRPin);
  pullUpDnControl(buttonPin, PUD_UP);
  while(1){
    if(digitalRead(buttonPin) == LOW){
    alertor(buzzeRPin);
    printf("alertor on...\n");
  }
  else {
    stopAlertor(buzzeRPin);
    printf("...buzzer off\n");
  }
  }
  return 0;
}
```

Code Interpretation

```
softToneCreate(buzzeRPin);
```

Logically, the code is the same as the active buzzer, but the method of controlling the buzzer is different. Passive buzzer requires a certain frequency of PWM to control, so you need to create a software PWM pin through 'softToneCreate' (buzzeRPin). Here 'softTone' is specifically used to generate a square wave with a fixed frequency of 50% and a variable frequency and duty cycle, which is a better choice for controlling the buzzer.

```
void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){
    sinVal = sin(x * (M_PI / 180));
    toneVal = 2000 + sinVal * 500;
    softToneWrite(pin,toneVal);
    delay(1);
```

```
    }
}
```

In the 'while' loop of the main function, when the button is pressed, the sub-function 'alertor ()' will be called and the alarm will sound a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value between 0 and 360 degrees and multiply it by some value (500), plus the resonant frequency of the buzzer. We can set the 'PWM' frequency through 'softToneWrite' (pin, toneVal).

```
void stopAlertor(int pin){
softToneWrite(pin,0);
}
```

To turn the buzzer off, simply set the PWM frequency of the buzzer pin to 0.

The functions of 'softTone' are as follows:

int softToneCreate (int pin) ;

Create a software-controlled tone pin.

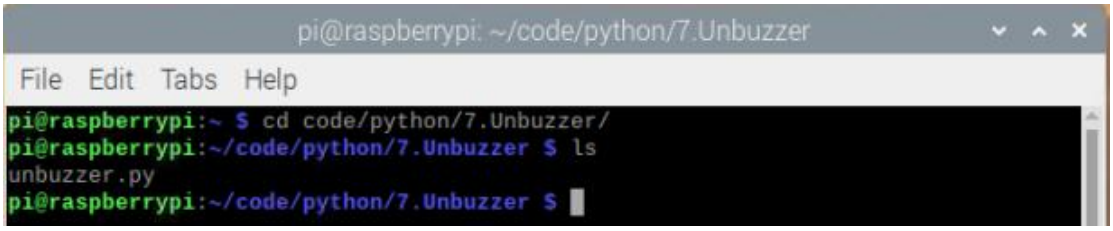
void softToneWrite (int pin, int freq) ;

Updates the audio frequency value on a given pin.

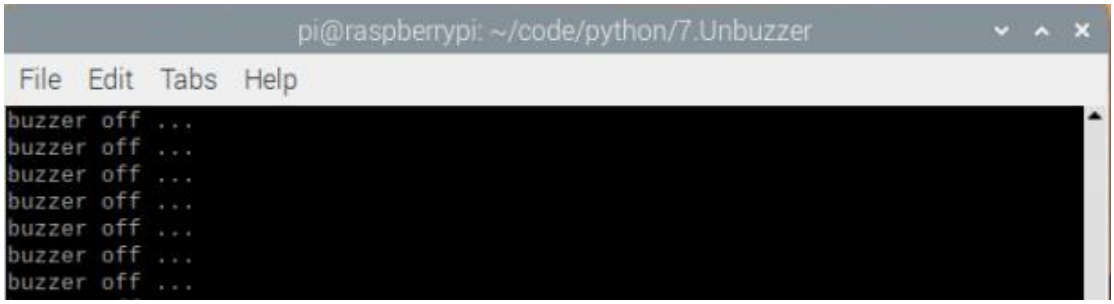
Python code

Open the terminal and use the cd code / python / 7.Unbuzzer / command to enter the code directory

Enter the command ls to view the file Unbuzzer.py in the directory



Enter the command python3 Unbuzzer.py to run the code. The result is as follows:



The following is the code:

```
import RPi.GPIO as GPIO
import time
import math

buzzerPin = 11
buttonPin = 12

def setup():
    global p
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(buzzerPin, GPIO.OUT)
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            alertor()
            print ('buzzer on ...')
        else :
            stopAlertor()
            print ('buzzer off ...')
def alertor():
    p.start(50)
    for x in range(0,361):
        sinVal = math.sin(x * (math.pi / 180.0))
        toneVal = 2000 + sinVal * 500
        p.ChangeFrequency(toneVal)
        time.sleep(0.001)

def stopAlertor():
```

```
        p.stop()

    def destroy():
        GPIO.output(buzzerPin, GPIO.LOW)
        GPIO.cleanup()

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

Code Interpretation

The code is the same to the active buzzer logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so youneed to create a software PWM pin though softToneCreate (buzzeRPin).The method of creating PWM is also introduced in RGB LED lights.

```
    def alertor():
        p.start(50)
        for x in range(0,361):
            sinVal = math.sin(x * (math.pi / 180.0))
            toneVal = 2000 + sinVal * 500
            p.ChangeFrequency(toneVal)
            time.sleep(0.001)
```

In the while cycle of main function, when the button is pressed, subfunction alertor() will be called and the alertor will issue a warning sound. The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (the value: 500) and plus the resonant frequency of buzzer. We can set the PWM frequency through ‘p.ChangeFrequency（toneVal）’

```
    def stopAlertor():
        p.stop()
```

When the button is released, the buzzer will turn off.

Lesson 8 AD/DA Converter

Overview

In this lesson you will learn how to read analog quantities through AD/DA Module, convert it into digital quantity and convert the digital quantity into analog output. That is, ADC and DAC.

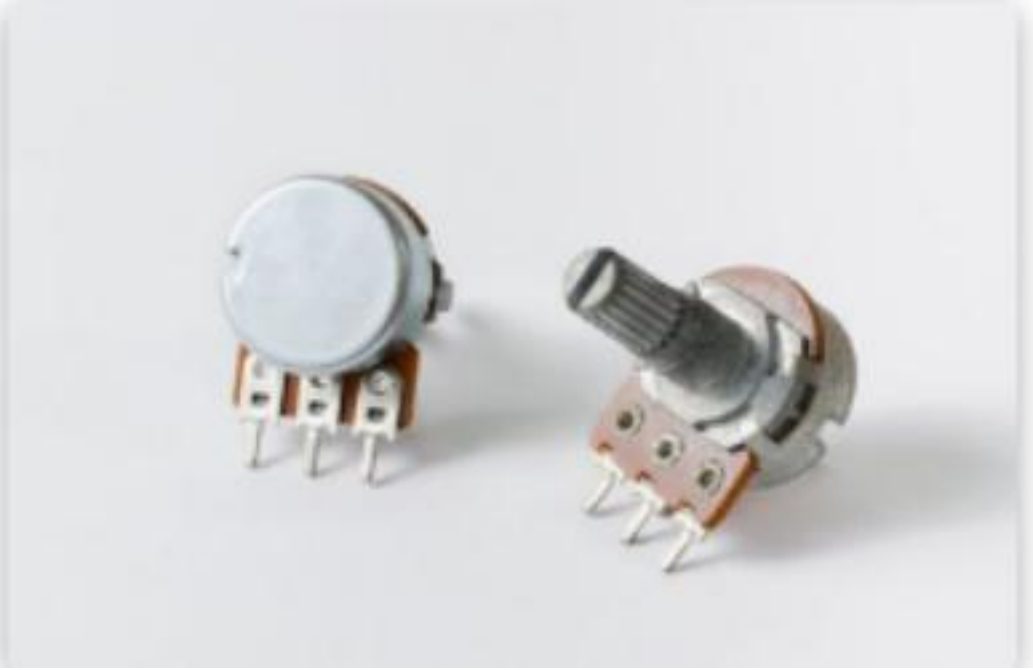
Parts Required

1 x Raspberry Pi
1 x PCF8591 Module
1 x 220 ohm Resistor
11 x Jumper Wires
1 x Breadboard
1 x Potentiometer
1 x LED

Product Introduction

Potentiometer

The potentiometer is a resistance having three pins and the resistance value can be adjusted according to a certain variation rule. Potentiometers usually consist of a resistor and a movable brush. When the brush moves along the resistor, the output end will

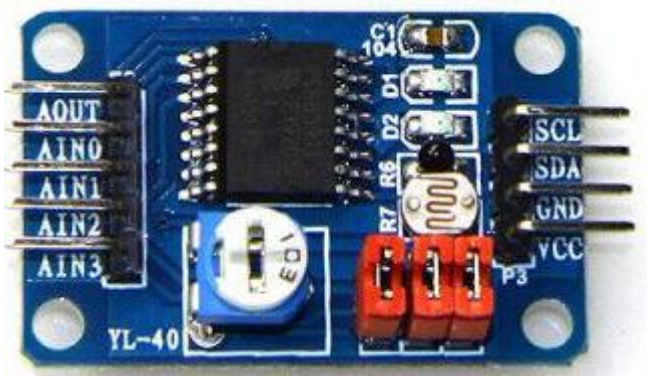


obtain some amount of resistance value or voltage which is in a certain relationship with the displacement. The potentiometer can be used as a three-pins or a two-pin

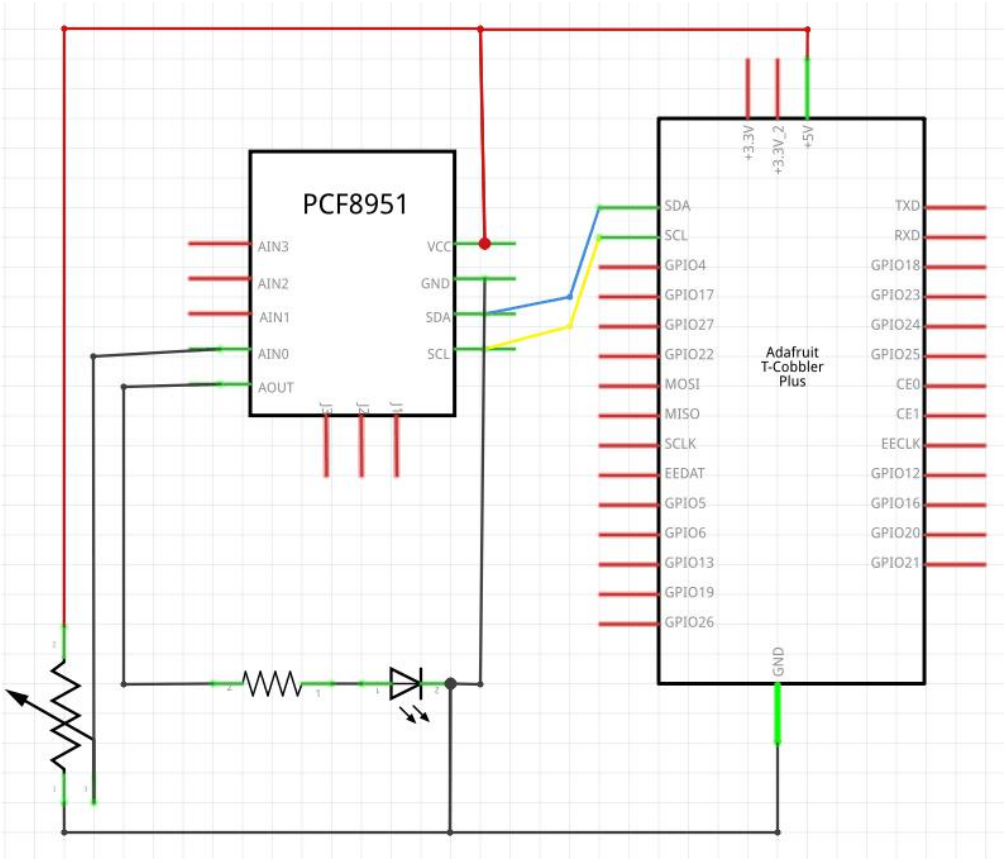
component. The latter can be regarded as a variable resistor. This lesson focuses on its use as a variable resistor.

PCF8591 module

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface.PCF8591's three address pins A0, A1, and A2 can be used for hardware address programming, allowing access to eight PCF8591 devices on the same I2C bus without additional hardware. The address, control, and data signals input and output on the PCF8591 device are transmitted in a serial manner through a two-line bidirectional I2C bus.

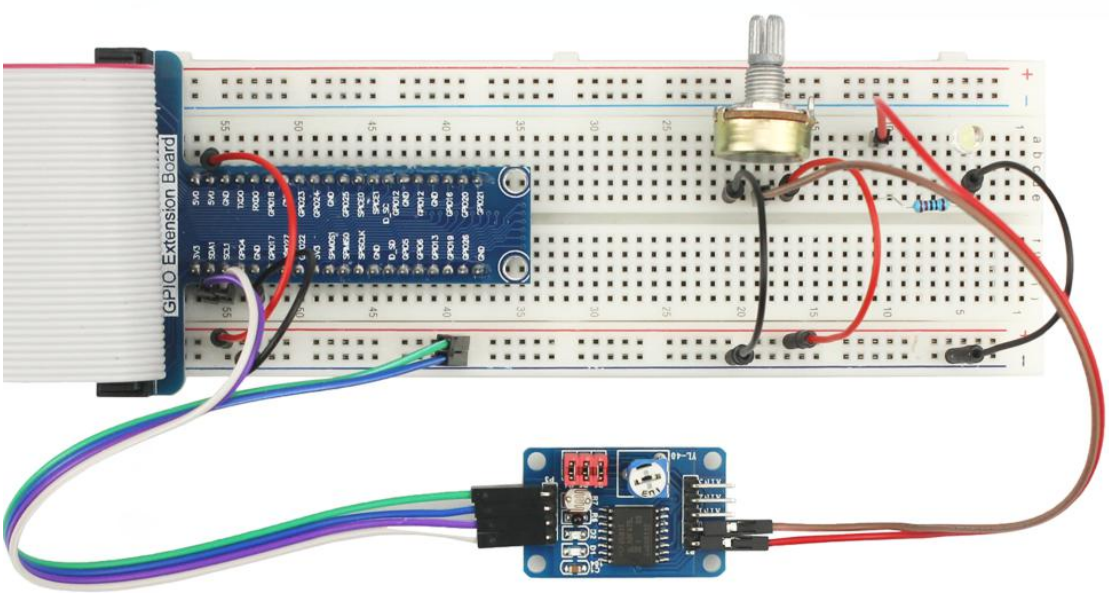


Connection Diagram



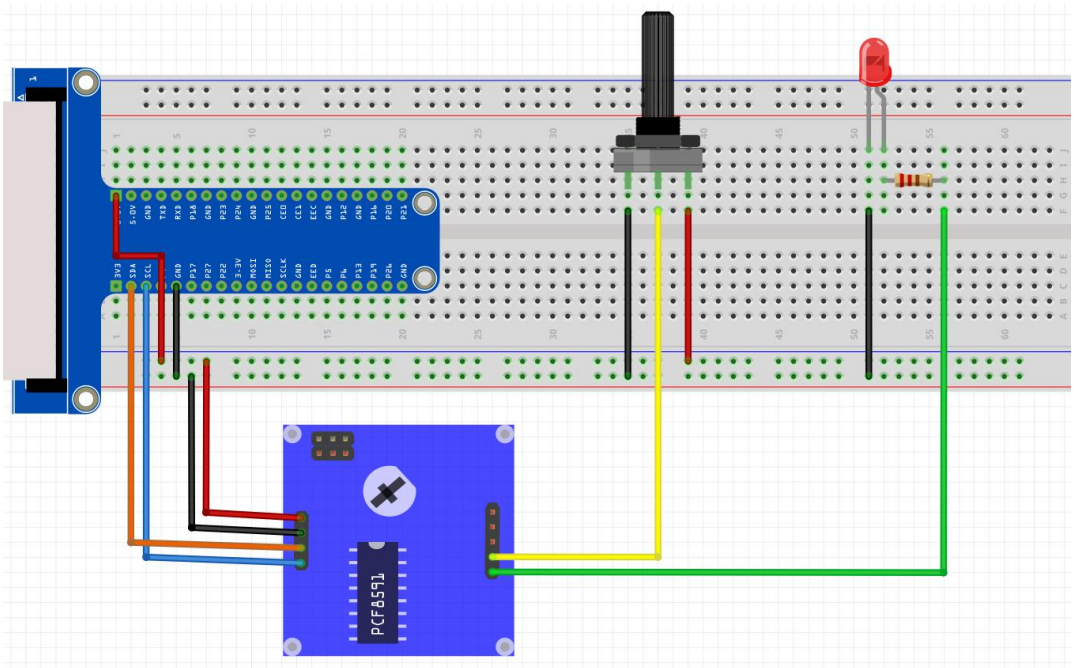
Wiring Diagram

Example Figure



Add the I2C library file:

Enter the sudo apt-get install i2c-tools command and press Enter, as shown in the figure below;

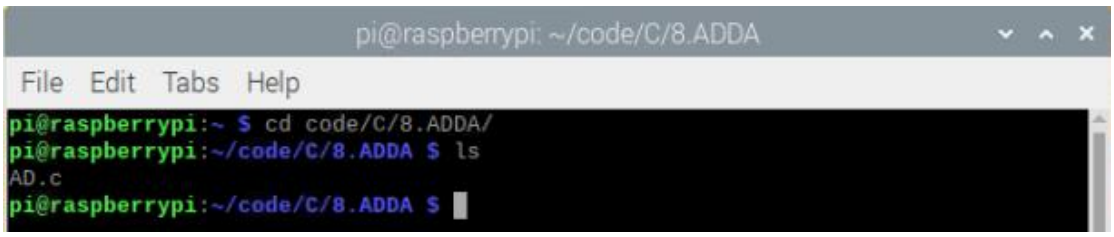


```
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
i2c-tools is already the newest version (3.1.2-3).
i2c-tools set to manually installed.
The following packages were automatically installed and are no longer required:
  libgooglepinyin0 libscim8v5 libsunpinyin3v5 scim scim-gtk-immodule
  scim-im-agent scim-modules-socket sunpinyin-data
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 94 not upgraded.
pi@raspberrypi:~ $
```

C code

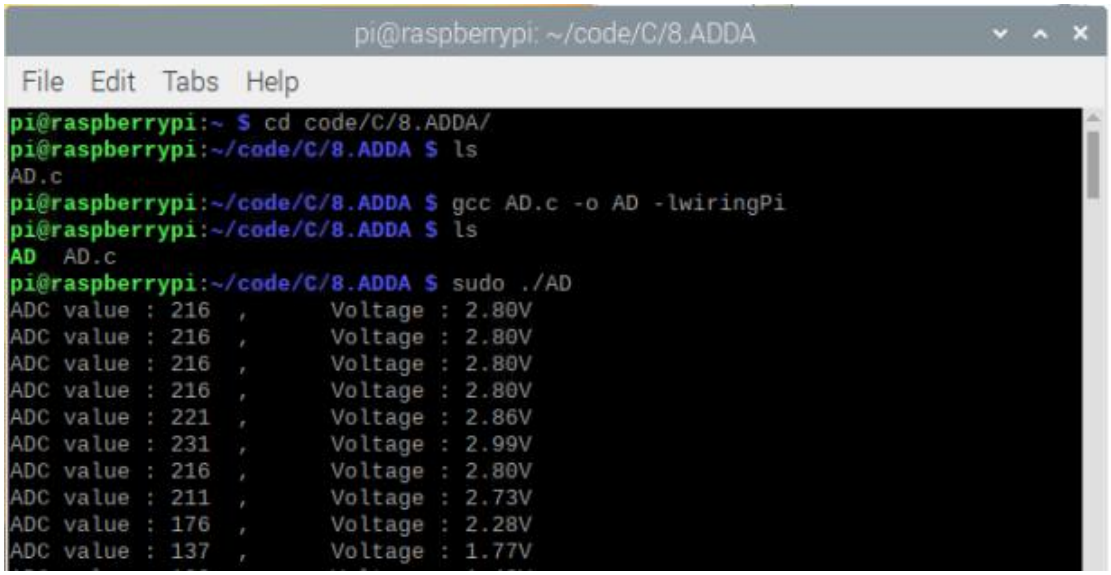
Open terminal and enter cd code / C / 8.ADDA / command to enter AD.c code directory;

Enter the ls command to view the file AD.c in the directory;



Enter gcc AD.c -o AD -lwiringPi command to generate AD.c executable file AD, enter ls command to view;

Enter the sudo ./AD command to run the code. The result is as follows:



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>

#define address 0x48            //pcf8591 default address
#define pinbase 64              //any number above 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

int main(void){
    int value;
    float voltage;
    wiringPiSetup();
    pcf8591Setup(pinbase,address);

    while(1){
        value = analogRead(A0);    //read A0 pin
        analogWrite(pinbase+0,value);
        voltage = (float)value / 255.0 * 3.3;    // calculate voltage
        printf("ADC value : %d    ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
    }
}
```

Code Interpretation

```
#define address 0x48          //pcf8591 default address
#define pinbase 64            //any number above 64
nbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3
```

The default I2C address of PCF8591 is 0x48. The pin base can be any value greater than or equal to 64. We define the ADC input channels A1, A2, A0, A3 of the PCF8591.

```
pcf8591Setup(pinbase,address);
```

In the main function, after PCF8591 is initialized with 'pcf8591Setup' (pinbase, address), the ADC and DAC can be operated using the functions 'analogRead ()' and 'analogWrite ()'.

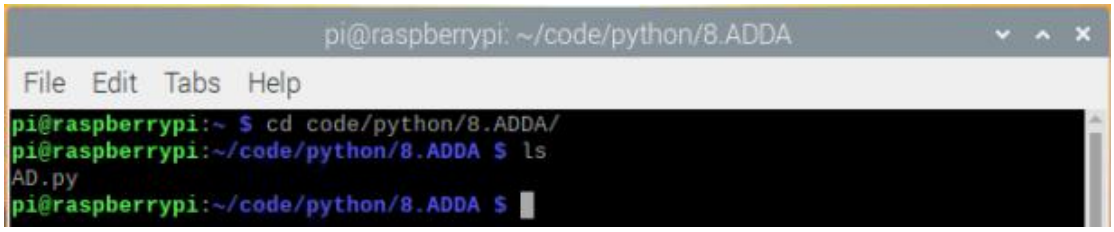
```
while(1){
        value = analogRead(A0);    //read A0 pin
        analogWrite(pinbase+0,value);
        voltage = (float)value / 255.0 * 3.3;    // calculate voltage
        printf("ADC value : %d    ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
    }
```

In the "while" cycle, 'analogRead (A0)' is used to read the ADC value of the A0 port (connected potentiometer), and then the read ADC value is output via 'AnalogWrite ()'. The corresponding actual voltage value will then be calculated and displayed.

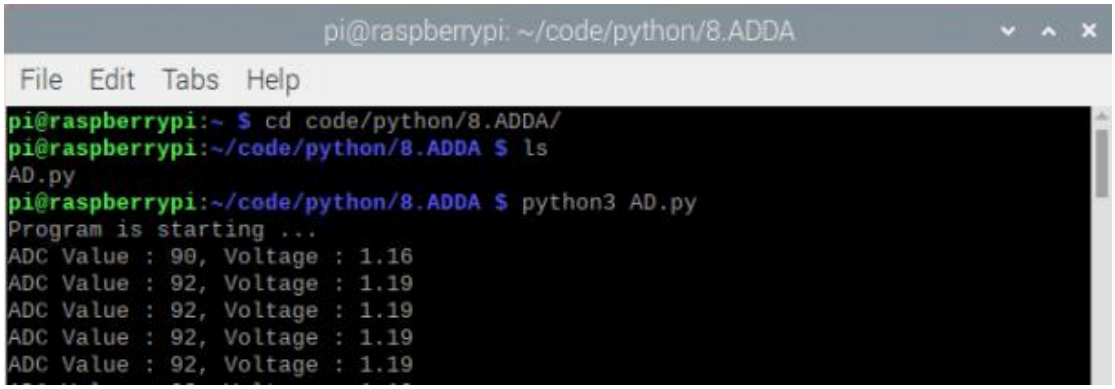
Python code

Open the terminal and use the cd code / python / 8.ADDA / command to enter the code directory

Enter the command ls to view the file ADDA.py in the directory.



Enter the command python3 ADDA.py to run the code. The result is as follows:



The following is the code:

```
import smbus
import time

address = 0x48
bus=smbus.SMBus(1)
cmd=0x40

def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value

def analogWrite(value):
    bus.write_byte_data(address,cmd,value)

def loop():
    while True:
        value = analogRead(0)
        analogWrite(value)
        voltage = value / 255.0 * 3.3
        print ('ADC Value : %d, Voltage : %.2f%(value,voltage))
        time.sleep(0.01)

def destroy():
    bus.close()

if __name__ == '__main__':
    print ('Program is starting ... ')
    try:
        loop()
    except KeyboardInterrupt:
    destroy()
```

Code Interpretation

```
import time
address = 0x48
bus=smbus.SMBus(1)
cmd=0x40
```

'address = 0x48' defines the 'I2C' address and control byte of 'PCF8591', and then instantiates the object bus of 'SMBus',' cmd = 0x40 'is a command for the' bus' object, which can be used to operate PCF8591 ' 'ADC' and 'DAC'

```
def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value
```

This sub-function is used to read 'ADC'. The parameter "chn" represents the input channel number: 0,1,2,3. The return value is the ADC value.

```
def analogWrite(value):
    bus.write_byte_data(address,cmd,value)
```

This sub-function is used to write 'DAC'. Its parameter "value" represents the quality of the number to be written, which is between '0-255'.

```
def loop():
    while True:
        value = analogRead(0)
        analogWrite(value)
        voltage = value / 255.0 * 3.3
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.01)
```

In the "while" cycle, first read the ADC value of channel 0, then write the value as the DAC digital quality and output the corresponding voltage at the output pin of PCF8591. Then calculate the corresponding voltage value and print it out.

```
def destroy():
    bus.close()
```

Release the 'bus' object.

```
import smbus
```

The 'System Management Bus. This' module defines a host that allows the 'SMBus' transaction object type to run the Linux kernel. The host kernel must have I2C support,

I2C device interface support, and a bus adapter driver. All of these can be built into the kernel or loaded from the module. In Python, you can use the help ‘smbus’ to view related features and their descriptions. ‘Bus = smbus.SMBus (1)’: Create an ‘SMBus’ class object. 'Bus.read_byte_data (address, cmd + chn)': Read a data byte from the address and return. 'Bus.write_byte_data (address, cmd, value)': Write one byte of data to an address.

Lesson 9 Potentiometer & LED

Overview

In this course, you will learn how to control the brightness of LED through a potentiometer.

Parts Required

1 x Raspberry Pi

1 x PCF8591 Module

1 x 220 ohm Resistor

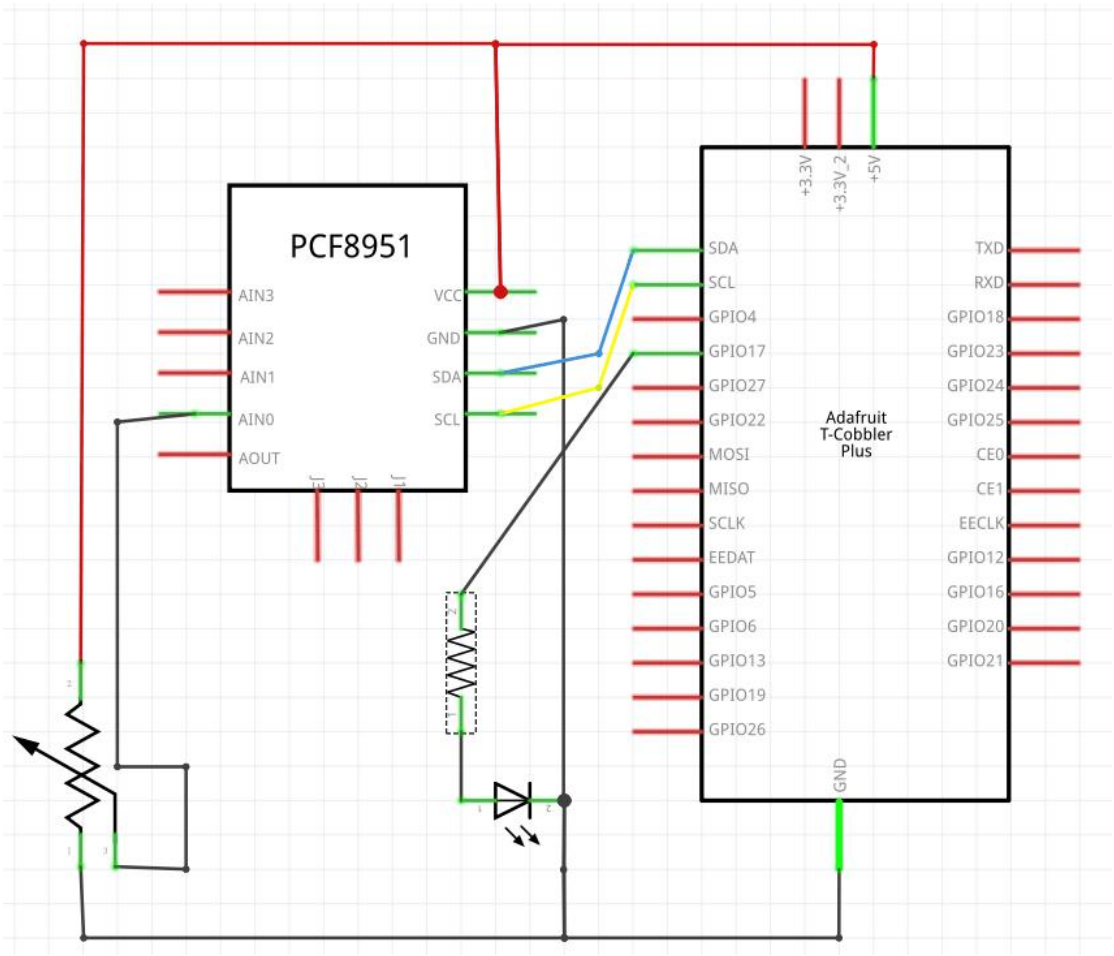
11 x DuPont

1 x Breadboard

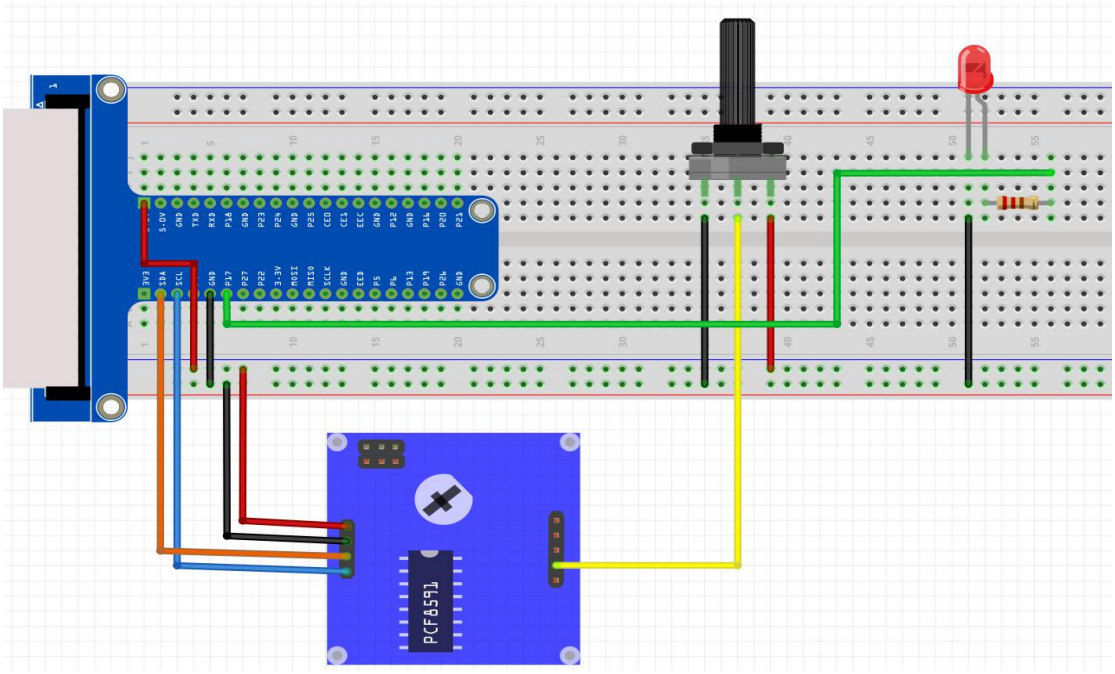
1 x Potentiometer

1 x LED

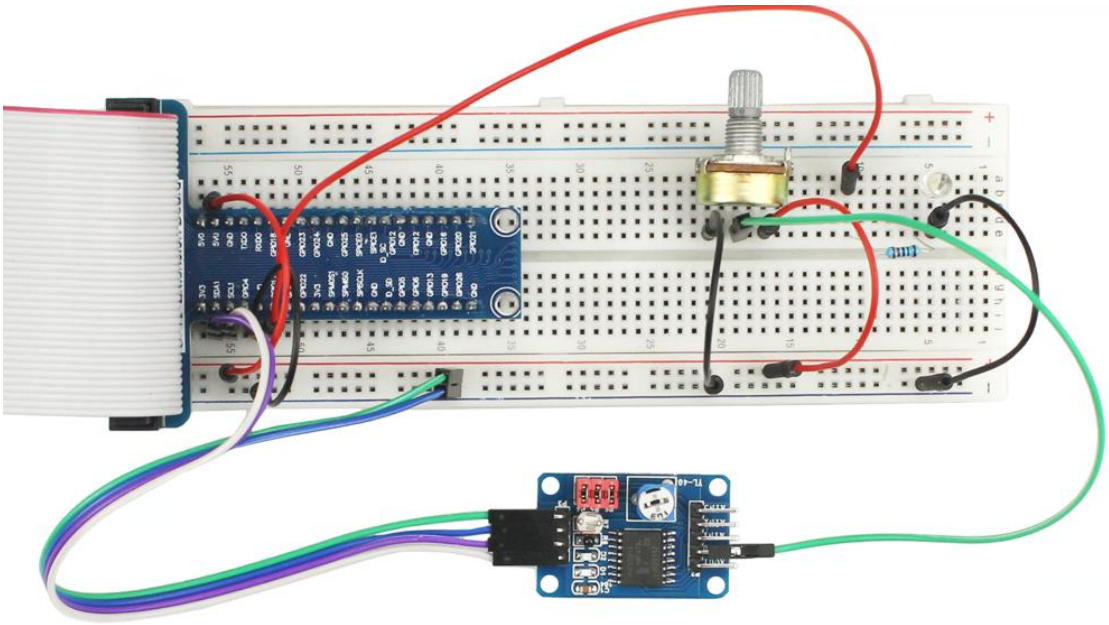
Connection Diagram



Wiring Diagram



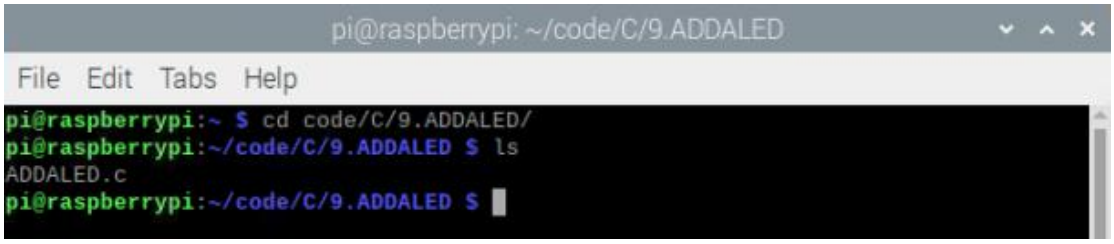
Example picture



C code

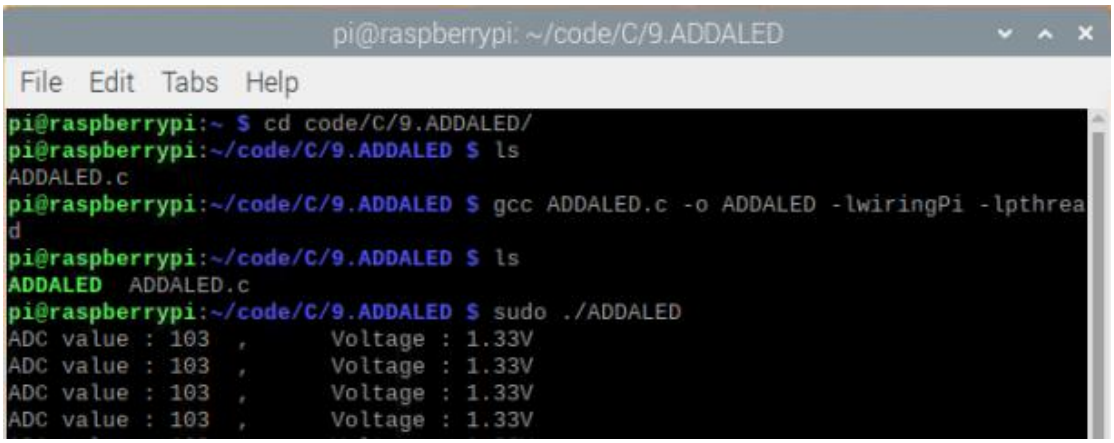
Open the terminal and enter the cd code / C / 9.ADDALED / command to enter the ADDALED.c code directory;

Enter the ls command to view the file ADDALED.c in the directory;



Enter gcc ADDALED.c -o ADDALED -lwiringPi -lpthread command to generate ADDALED.c executable file ADDALED, enter ls command to view;

Enter the sudo ./ADDALED command to run the code. The result is as follows:



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48
#define pinbase 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
```

```
#define A3 pinbase + 3

#define ledPin 0
int main(void){
    int value;
    float voltage;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledPin,0,100);
    pcf8591Setup(pinbase,address);

    while(1){
        value = analogRead(A0);
        softPwmWrite(ledPin,value*100/255);
        voltage = (float)value / 255.0 * 3.3;
        printf("ADC value : %d    ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
    }
    return 0;
}
```

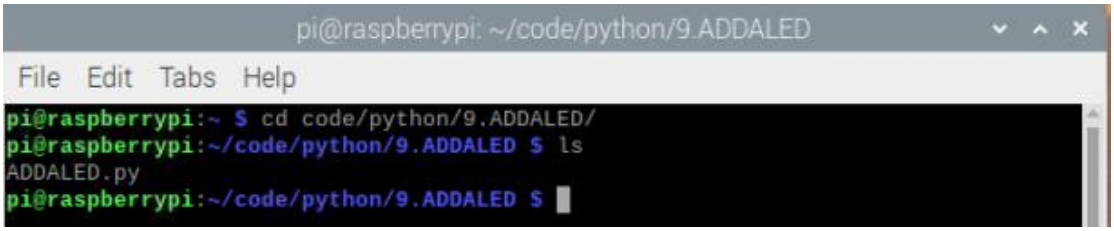
Code Interpretation

In the code, read the ADC value of the potentiometer and map it to the duty cycle of the PWM to control the brightness of the LED. For a detailed explanation, please refer to Lessons 8 and 4.

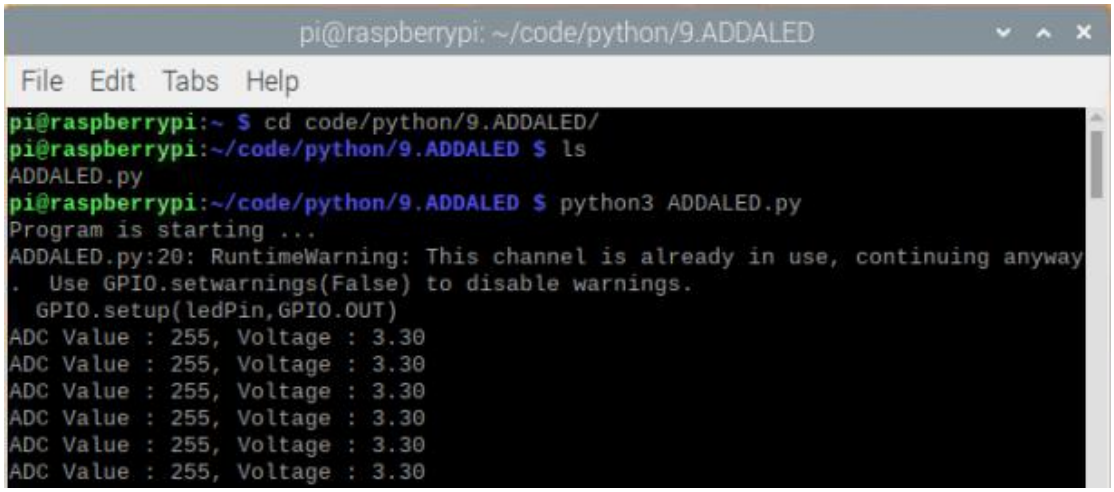
Python code

Open the terminal and use cd code / python / 9.ADDALED / command to enter the code directory

Enter the command ls to view the file ADDALED.py in the directory.



Enter the command python3 ADDALED.py to run the code. The result is as follows:



The following is the code:

```
import RPi.GPIO as GPIO
import smbus
import time
address = 0x48
bus=smbus.SMBus(1)
cmd=0x40
ledPin = 11

def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value
def analogWrite(value):
    bus.write_byte_data(address,cmd,value)
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)
    GPIO.output(ledPin,GPIO.LOW)

    p = GPIO.PWM(ledPin,1000)
    p.start(0)
def loop():
    while True:
        value = analogRead(0)
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 3.3
        print ('ADC Value : %d, Voltage : %.2f%(value,voltage))
        time.sleep(0.01)
def destroy():
```

```
        bus.close()
        GPIO.cleanup()
if __name__ == '__main__':
        print ('Program is starting ... ')
        setup()
        try:
                loop()
        except KeyboardInterrupt:
                destroy()
```

Code Interpretation

```
import RPi.GPIO as GPIO
import smbus
import time
address = 0x48
bus=smbus.SMBus(1)
cmd=0x40
ledPin = 11
```

'address = 0x48' defines the 'I2C' address and control byte of 'PCF8591', and then instantiates the object bus of 'SMBus', 'cmd = 0x40' is a command for the bus object, which can be used to operate the ADC 'and' DAC '.

```
def analogRead(chn):
        value = bus.read_byte_data(address,cmd+chn)
        return value
```

This sub-function is used to read 'ADC'. The parameter "chn" represents the input channel number: 0,1,2,3.

The return value is the ADC value.

```
def analogWrite(value):

bus.write_byte_data(address,cmd,value)
```

This sub-function is used to write ‘DAC’. The parameter "value" represents the digital quality to be written, between‘0-255’.

```
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)
    GPIO.output(ledPin,GPIO.LOW)
```

```
    p = GPIO.PWM(ledPin,1000)
    p.start(0)
```

Define the mode of the 'ledPin' pin, and the period and duty cycle of the PWM

```
def loop():
    while True:
        value = analogRead(0)
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 3.3
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.01)
```

In the "while" cycle, first read the ADC value of channel 0, then write the value as the DAC digital quality and output the corresponding voltage at the output pin of PCF8591. Then calculate the corresponding PWM duty cycle to control the LED brightness and voltage value, and print out.

Lesson 10 Potentiometer & RGBLED

Overview

As for this lesson, we will learn how to use 3 potentiometers to control the brightness of 3 LEDs of RGBLED to make it show different colors.

Parts Required

1 x Raspberry Pi

1 x PCF8591 Module

3 x 220 ohm Resistor

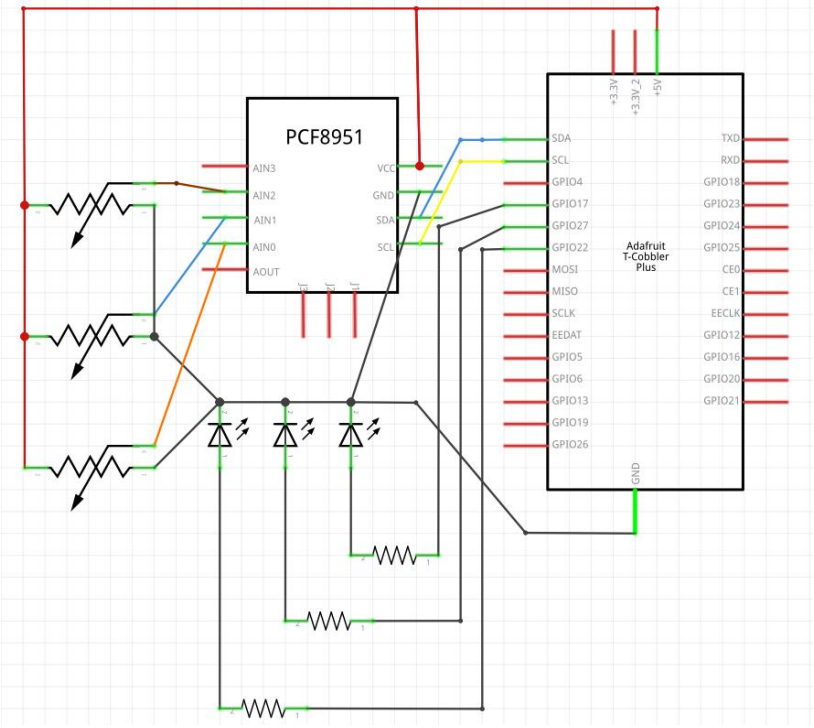
21 x DuPont

1 x Breadboard

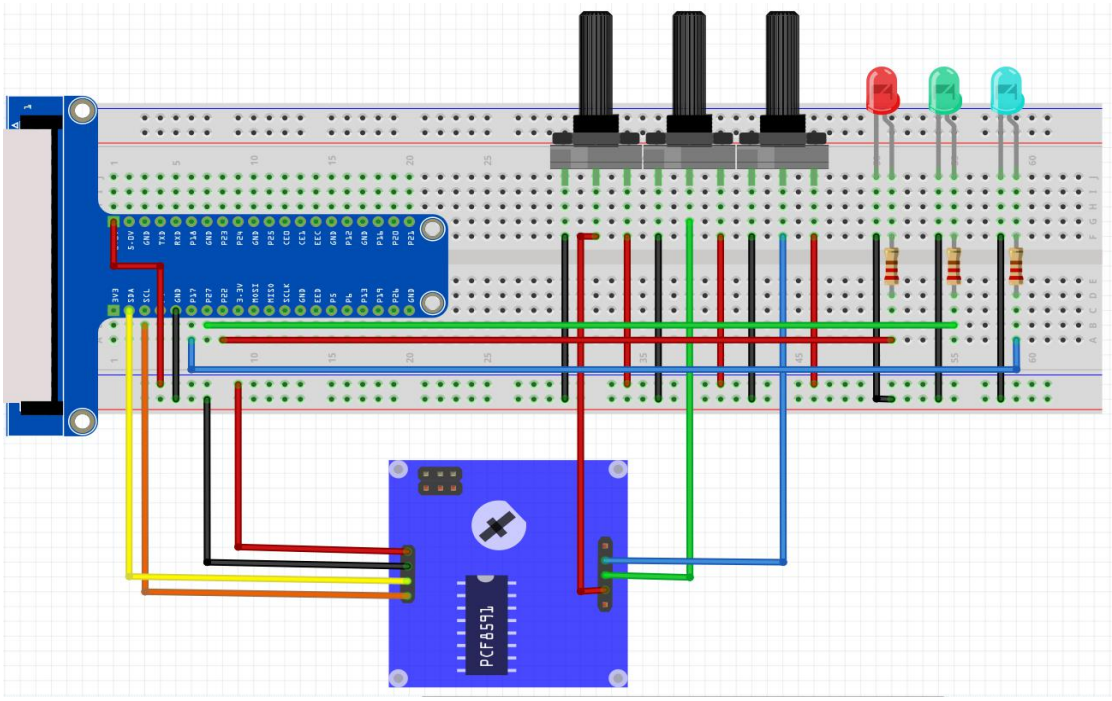
1 x RGBLED

3 x Potentiometer

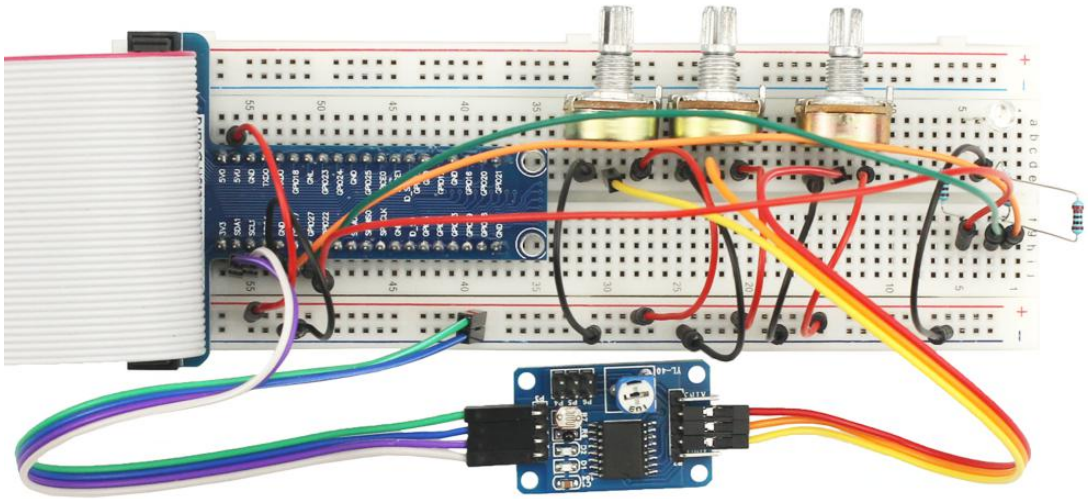
Connection Diagram



Wiring Diagram



Example picture



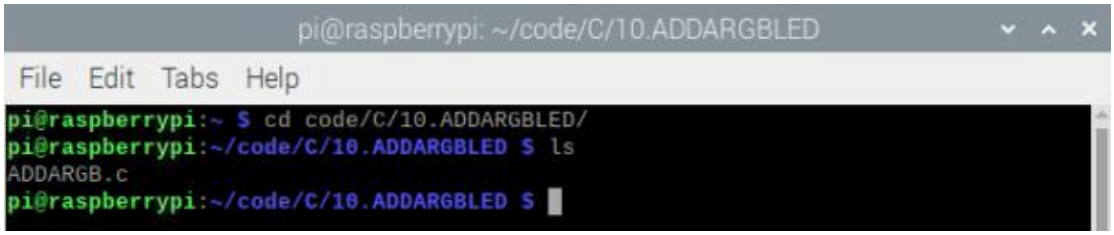
C code

Open the terminal and enter cd code / C / 10.ADDARGBLED / command to enter the ADDARGB.c code directory;

Enter the ls command to view the file ADDARGB.c in the directory;

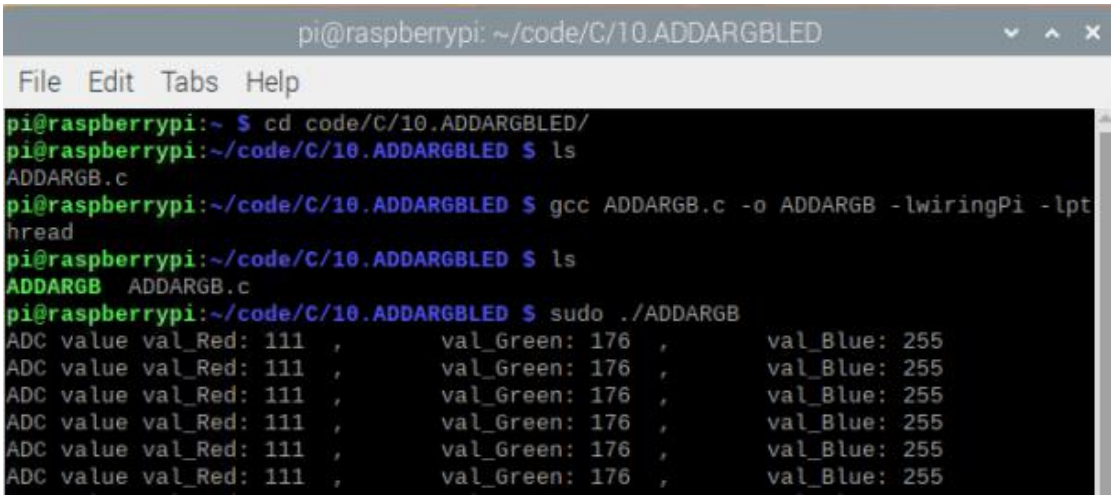
Enter gcc ADDARGB.c -o ADDARGB -lwiringPi -lpthread command to generate ADDARGB.c executable file ADDARGB, enter ls command to view;

Enter sudo ./ADDARGB command to run the code. The result is as follows:



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
```




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

#define address 0x48
#define pinbase 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define ledRedPin 3
#define ledGreenPin 2
#define ledBluePin 0
int main(void){
    int val_Red,val_Green,val_Blue;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledRedPin,0,100);
    softPwmCreate(ledGreenPin,0,100);
    softPwmCreate(ledBluePin,0,100);
    pcf8591Setup(pinbase,address);

    while(1){
        val_Red = analogRead(A0);
        val_Green = analogRead(A1);
        val_Blue = analogRead(A2);
```

```
                softPwmWrite(ledRedPin,val_Red*100/255);
                softPwmWrite(ledGreenPin,val_Green*100/255);
                softPwmWrite(ledBluePin,val_Blue*100/255);
                printf("ADC value val_Red: %d    ,\tval_Green: %d    ,\tval_Blue: %d
\n",val_Red,val_Green,val_Blue);
                delay(100);
            }
            return 0;
        }
```

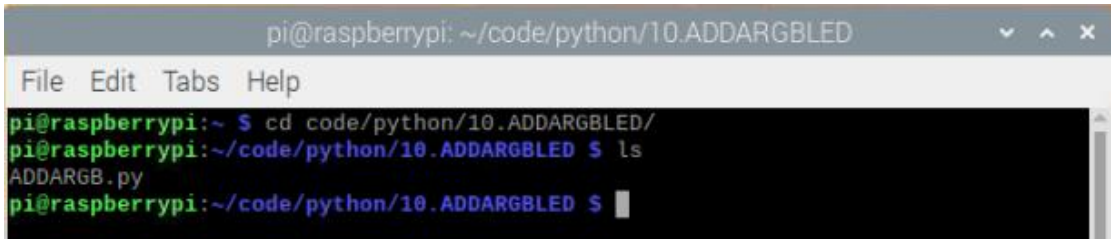
Code Interpretation

In the code, the ADC values of the 3 potentiometers are read and mapped to the PWM duty cycle to control the 3 LEDs with RGBLEDs of different colors, respectively. For detailed interpretation, please refer to Lessons 8 and 5.

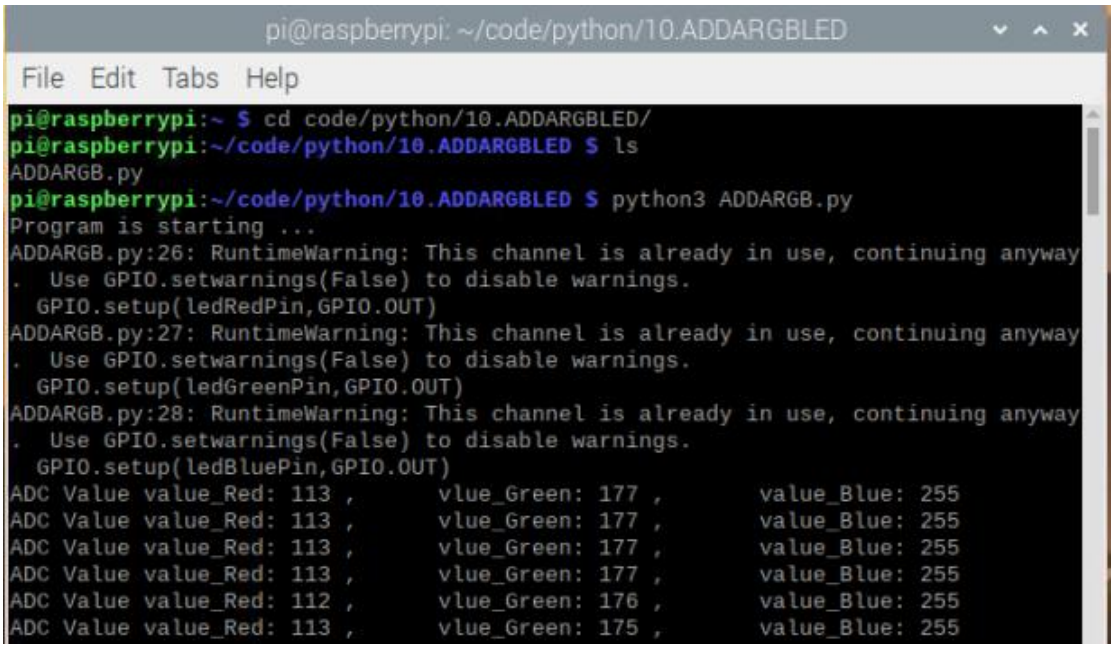
Python code

Open the terminal and use the cd code / python / 10.ADDARGBLED / command to enter the code directory;

Enter the command ls to view the file ADDARGB.py in the directory.



Enter the command python3 ADDARGB.py to run the code. The result is as follows:



The following is the code:

```
import RPi.GPIO as GPIO
import smbus
import time

address = 0x48
bus=smbus.SMBus(1)
cmd=0x40

ledRedPin = 15
ledGreenPin = 13
ledBluePin = 11

def analogRead(chn):
    bus.write_byte(address,cmd+chn)
    value = bus.read_byte(address)
    value = bus.read_byte(address)
    return value

def analogWrite(value):
    bus.write_byte_data(address,cmd,value)

def setup():
    global p_Red,p_Green,p_Blue
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledRedPin,GPIO.OUT)
```

```
        GPIO.setup(ledGreenPin,GPIO.OUT)
        GPIO.setup(ledBluePin,GPIO.OUT)

        p_Red = GPIO.PWM(ledRedPin,1000)
        p_Red.start(0)
        p_Green = GPIO.PWM(ledGreenPin,1000)
        p_Green.start(0)
        p_Blue = GPIO.PWM(ledBluePin,1000)
        p_Blue.start(0)

    def loop():
        while True:
            value_Red = analogRead(0)
            value_Green = analogRead(1)
            value_Blue = analogRead(2)
            p_Red.ChangeDutyCycle(value_Red*100/255)
            p_Green.ChangeDutyCycle(value_Green*100/255)
            p_Blue.ChangeDutyCycle(value_Blue*100/255)
            #print read ADC value
            print ('ADC Value
value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value
_Blue))
            time.sleep(0.01)

    def destroy():
        bus.close()
        GPIO.cleanup()

    if __name__ == '__main__':
        print ('Program is starting ... ')
        setup()
        try:
            loop()
        except KeyboardInterrupt:
        destroy()
```

Code Interpretation

```
    def analogRead(chn):
        bus.write_byte(address,cmd+chn)
        value = bus.read_byte(address)
        value = bus.read_byte(address)
```

```
        return value
```

This function always returns the last data read. If you want to return the current data, you need to do it twice

```
    def loop():
        while True:
            value_Red = analogRead(0)
            value_Green = analogRead(1)
            value_Blue = analogRead(2)
            p_Red.ChangeDutyCycle(value_Red*100/255)
            p_Green.ChangeDutyCycle(value_Green*100/255)
            p_Blue.ChangeDutyCycle(value_Blue*100/255
            Print('ADC Value
value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value
_Blue))
            time.sleep(0.01)
```

This function reads the ADC values of the 3 potentiometers and maps them to the PWM duty cycle to control the 3 LED with different color RGBLED respectively.

Lesson 11 Photoresistor & LED

Overview

In this lesson, you will learn how touse photoresistor.Photoresistor is very sensitive to illumination strength. So we can use this feature to make a nightlamp, when ambient light gets darker, LED will become brighter automatically, and when the ambient light gets brighter, LED will become darker automatically.

Parts Required

1 x Raspberry Pi

1 x PCF8591 Module

1 x 220 ohm Resistor

1 x 10k Resistor

9 x Jumper Wires

1 x Breadboard

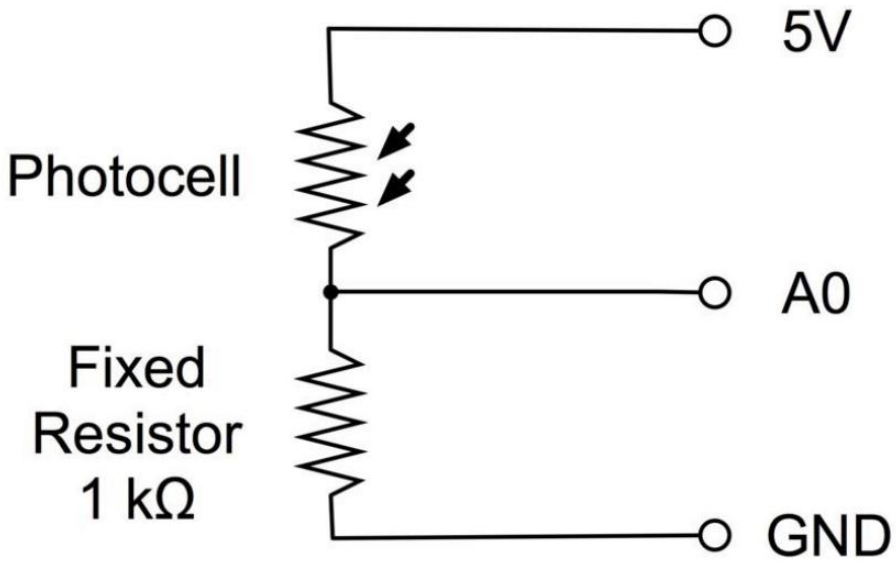
1 x LED

1 x Photoresistor

Product Introduction

Photosensitive Resistor

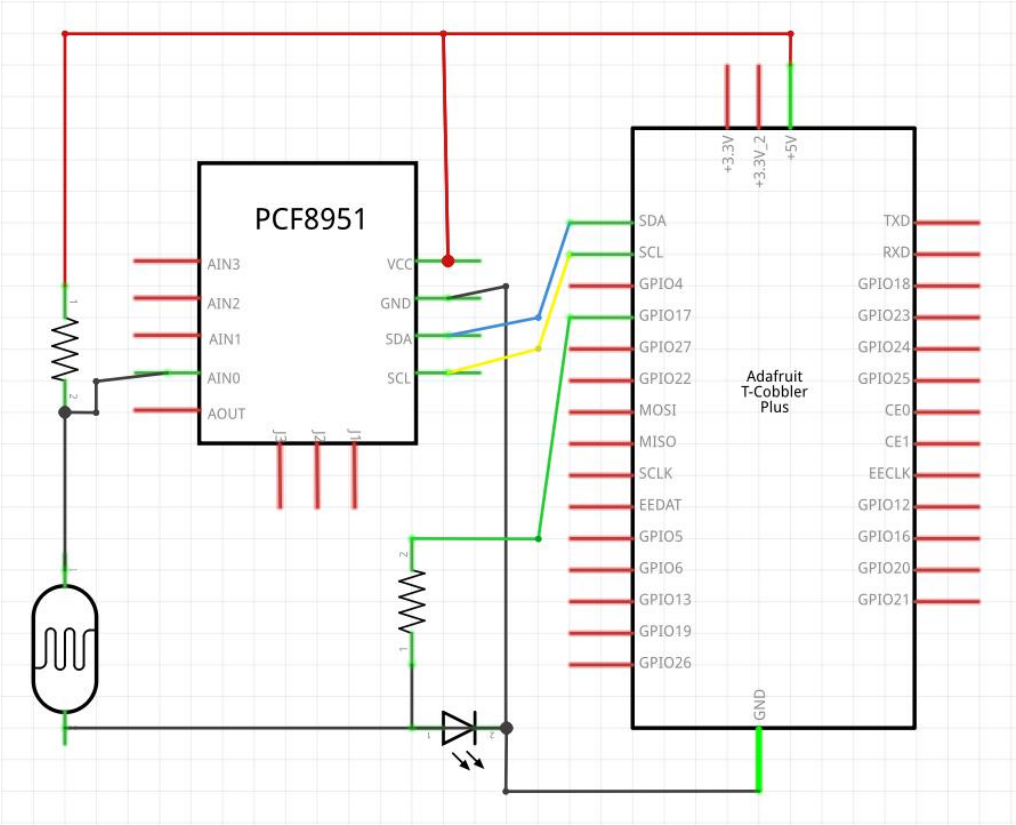
Photosensitive resistors, also known as LDRs. Photoresistors are like ordinary resistors, except that the resistance of the resistor changes as the light falls on them. This has a resistance of about 50 kΩ in the near dark and 500Ω in the strong light. To convert this changed resistance value to a value that we can measure on the analog input of the Raspberry Pi; it needs to be converted to voltage. The easiest way is to combine it with a fixed resistor.



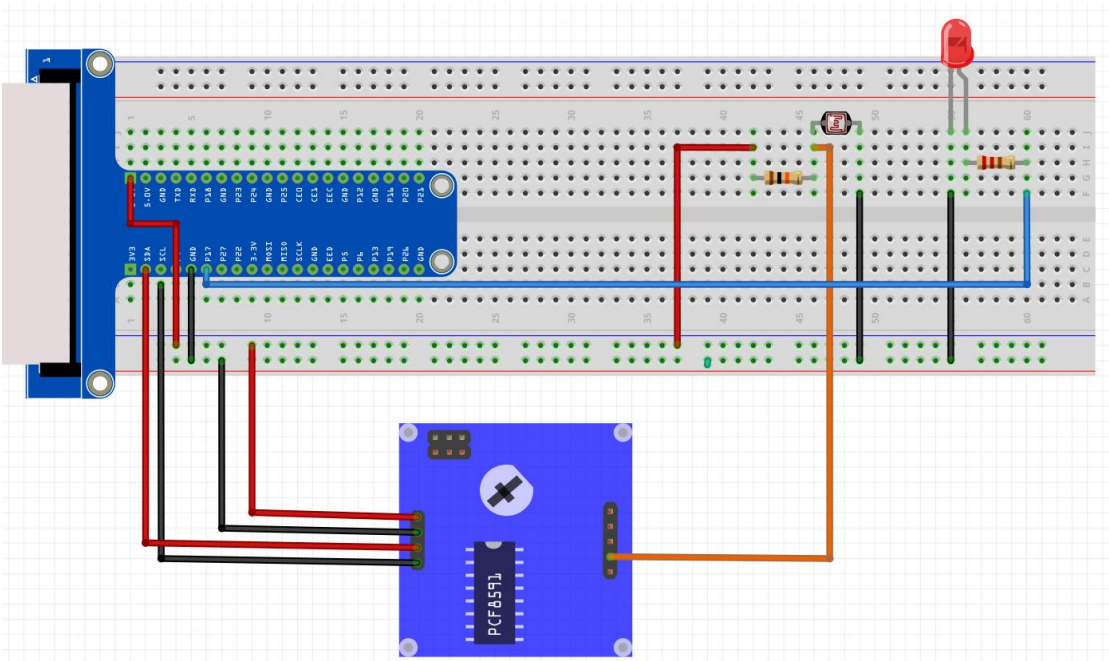
The values of the resistor and the photoresistor are put together to behave like a single data. When the light is very bright, the resistance of the photoresistor is very low compared to a fixed value resistor, so it is as if the potentiometer is set to maximum.

When the photoresistor is in dim light, the resistance becomes greater than a fixed 1kΩ resistor as if the potentiometer is facing GND. Load the code given in this section, cover the photo-sensitive resistor with your finger, and place it near the light source. You can see the change of LED.

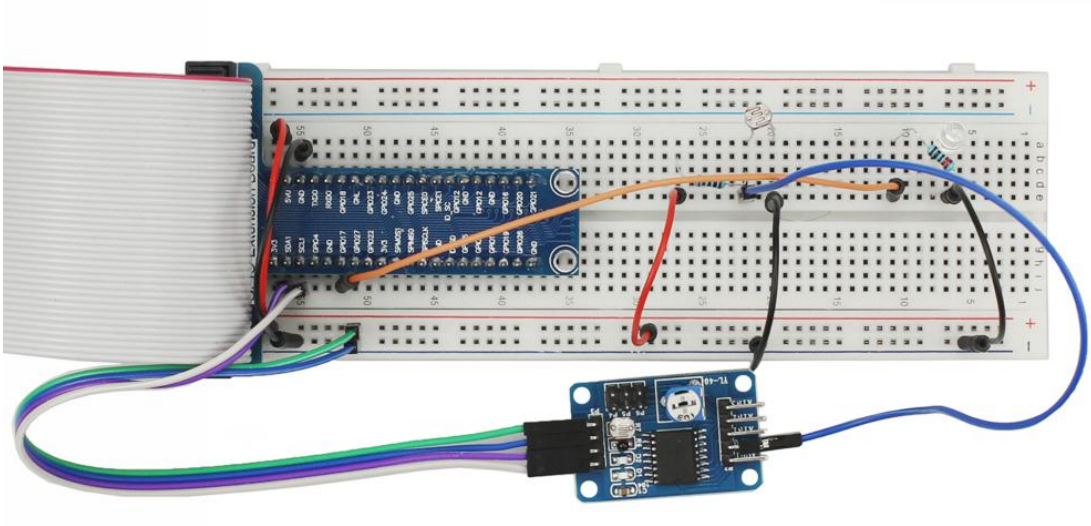
Connection Diagram



Wiring Diagram



Example Figure



C code

Open the terminal and enter the cd code / C / 11.Photoresistor / command to enter the photoresistor.c code directory;

Enter the ls command to view the file photoresistor.c in the directory;

Enter gcc photoresistor.c -o photoresistor -lwiringPi -lpthread to generate the photoresistor.c executable file photoresistor.

Enter the sudo ./photoresistor command to run the code. The result is as follows:

Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <softPwm.h>

#define address 0x48
#define pinbase 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

#define ledPin 0
int main(void){
    int value;
    float voltage;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(ledPin,0,100);
    pcf8591Setup(pinbase,address);

    while(1){
        value = analogRead(A0);
        softPwmWrite(ledPin,value*100/255);
        voltage = (float)value / 255.0 * 3.3;
        printf("ADC value : %d    ,\tVoltage : %.2fV\n",value,voltage);
        delay(100);
```

```
        }
        return 0;
    }
```

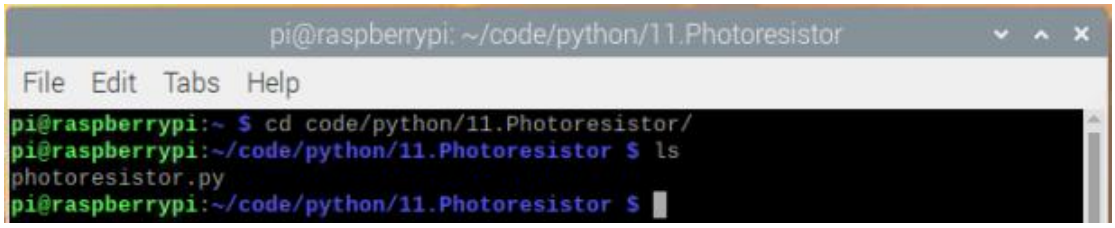
Code Interpretation

In the code, read the ADC value of the photoresistor and map it to the PWM duty cycle to control the brightness of the LED. When the photoresistor is covered or the flashlight is illuminated toward the photoresistor, the brightness of the LED will increase or decrease. For a detailed explanation, please refer to Lessons 8 and 4.

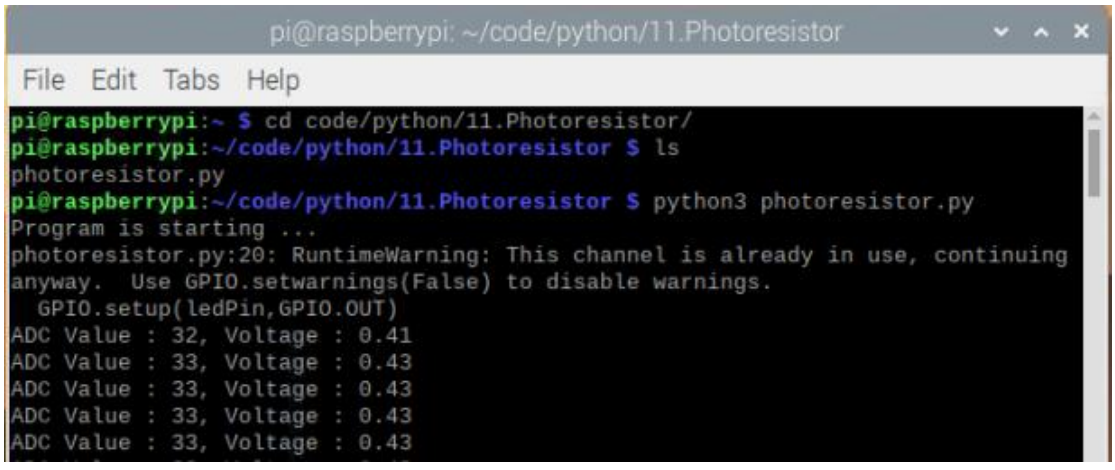
Python code

Open the terminal and use the cd code / python / 11.Photoresistor / command to enter the code directory

Enter the command ls to view the file photoresistor.py in the directory.



Enter the python3 photoresistor.py command to run the code. The result is as follows:



The following is the code:

```
import RPi.GPIO as GPIO
import smbus
import time
address = 0x48
```

```
bus=smbus.SMBus(1)
cmd=0x40
ledPin = 11
def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value

def analogWrite(value):
    bus.write_byte_data(address,cmd,value)
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(ledPin,GPIO.OUT)
    GPIO.output(ledPin,GPIO.LOW)
    p = GPIO.PWM(ledPin,1000)
    p.start(0)
def loop():
    while True:
        value = analogRead(0)
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 3.3
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.01)

def destroy():
    bus.close()
    GPIO.cleanup()

if __name__ == '__main__':
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

Code Interpretation

This program is the same as the Potentiometer& LED course. For a detailed explanation, please refer to the Potentiometer& LED course.

Lesson 12 Thermistor

Overview

In this lesson you will learn how to read the temperature using a Raspberry Pi controlled thermistor.

Parts Required

1 x Raspberry Pi

1 x PCF8591 Module

1 x 10K Resistor

9 x Jumper Wires

1 x Breadboard

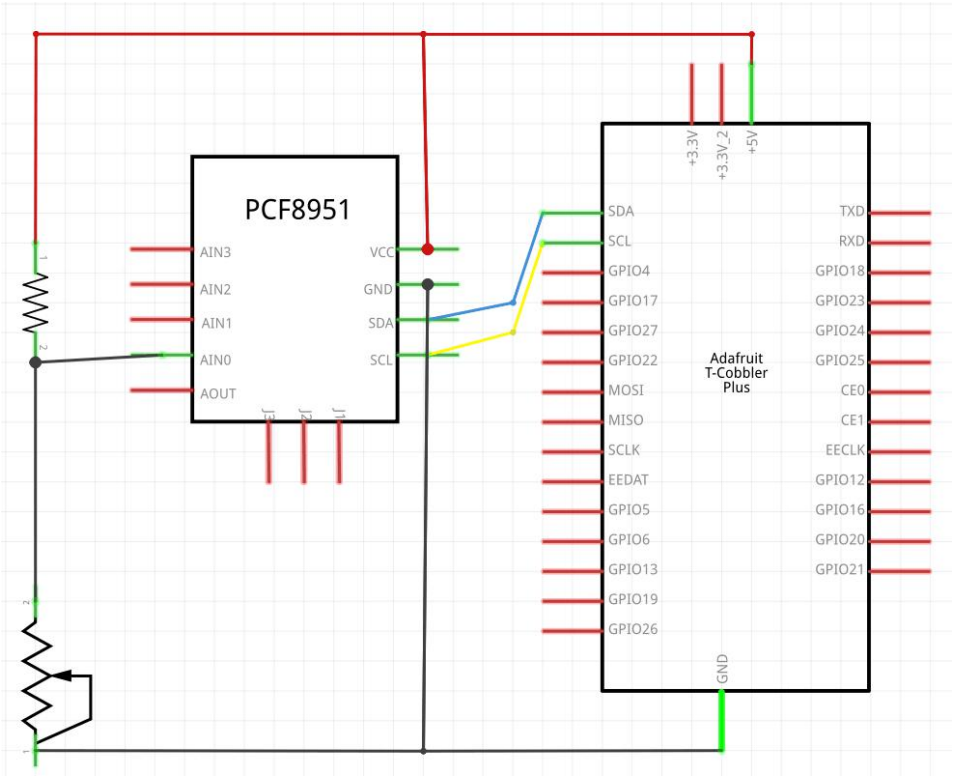
1 x Thermistor

Product Introduction

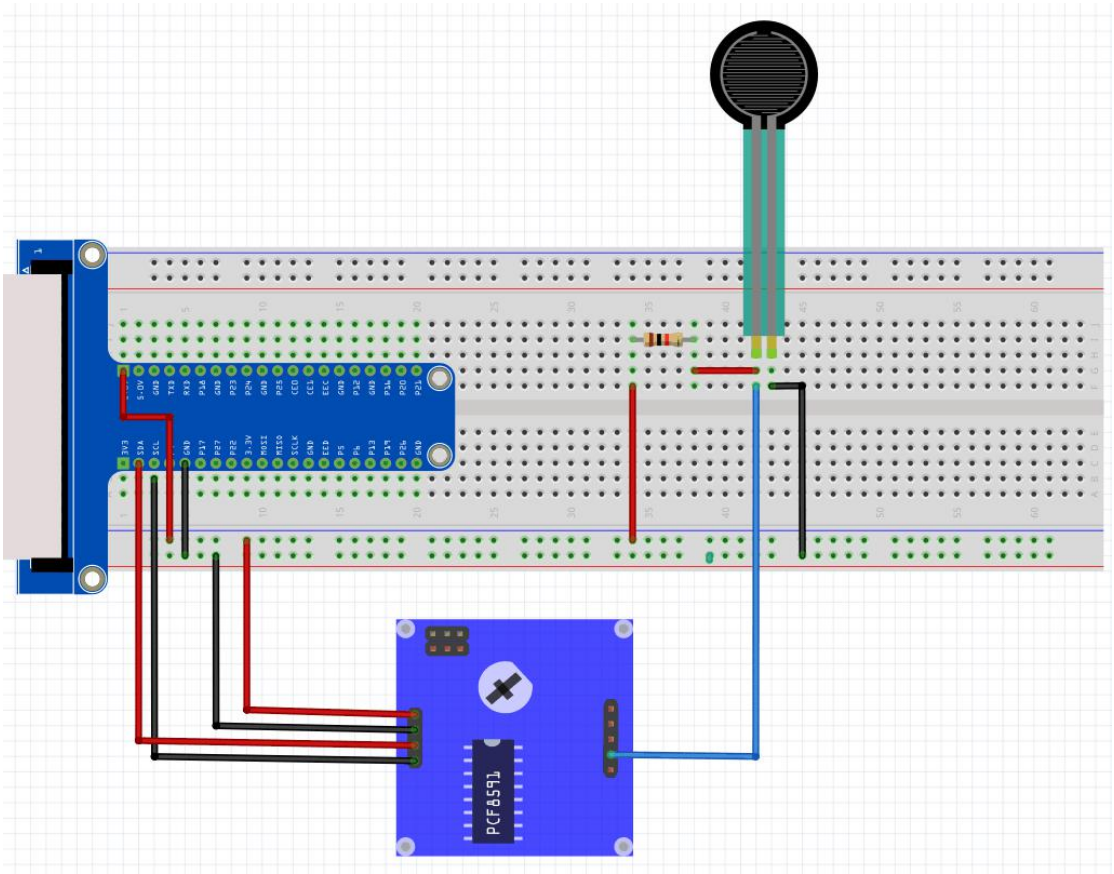
Thermistor

Thermistors are a type of sensitive components. They are divided into positive temperature coefficient thermistors (PTC) and negative temperature coefficient thermistors (NTC) according to different temperature coefficients. The typical characteristic of a thermistor is that it is sensitive to temperature and exhibits different resistance values at different temperatures. Positive temperature coefficient thermistors (PTC) have higher resistance values at higher temperatures, and negative temperature coefficient thermistors (NTC) have lower resistance values at higher temperatures. They both belong to semiconductor devices.

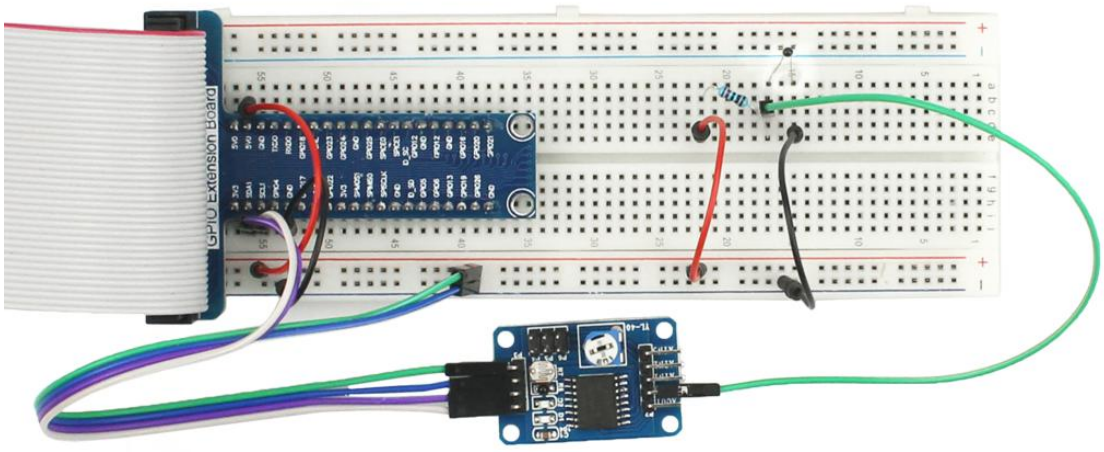
Connection Diagram



Wiring Diagram



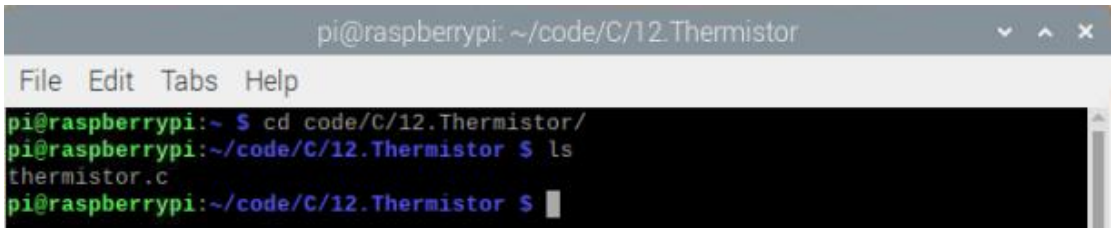
Example Figure



C code

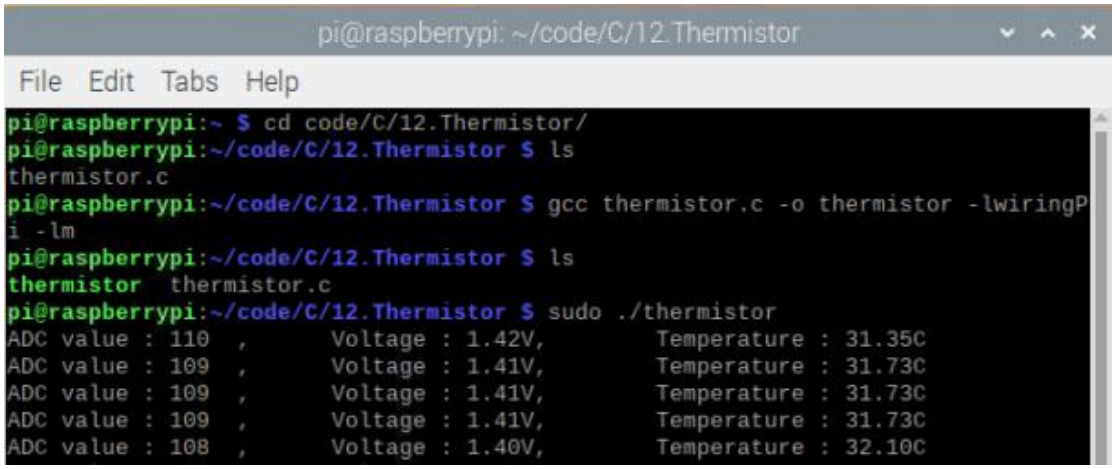
Open the terminal and enter the cd code / C / 12.Thermistor / command to enter the thermistor.c code directory;

Enter the ls command to view the thermistor.c file in the directory;



Enter the gcc thermistor.c -o thermistor -lwiringPi -lm command to generate the thermostor.c executable file thermostor, and enter the ls command to view;

Enter the sudo ./thermistor command to run the code. The result is as follows:



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <pcf8591.h>
#include <stdio.h>
#include <math.h>

#define address 0x48
#define pinbase 64
#define A0 pinbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

int main(void){
    int adcValue;
    float tempK,tempC;
```

```
        float voltage,Rt;
        if(wiringPiSetup() == -1){
            printf("setup wiringPi failed !");
            return 1;
        }
        pcf8591Setup(pinbase,address);
        while(1){
            adcValue = analogRead(A0);
            voltage = (float)adcValue / 255.0 * 3.3;
            Rt = 10 * voltage / (3.3 - voltage);
            tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0);
            tempC = tempK -273.15;
            printf("ADC value : %d    ,\tVoltage : %.2fV,
\tTemperature : %.2fC\n",adcValue,voltage,tempC);
            delay(100);
        }
        return 0;
    }
```

Code Interpretation

```
        while(1){
            adcValue = analogRead(A0);
            voltage = (float)adcValue / 255.0 * 3.3;
            Rt = 10 * voltage / (3.3 - voltage);
            tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0);
            tempC = tempK -273.15;
            printf("ADC value : %d    ,\tVoltage : %.2fV,
\tTemperature : %.2fC\n",adcValue,voltage,tempC);
            delay(100);
        }
```

In the code, read the ADC value of the PCF8591 A0 port, then calculate the voltage and resistance.

Thermistor according to Ohm's law. Finally, calculate the current temperature. According to the previous formula.

Python code

Open the terminal and use the cd code / python / 12.Thermistor / command to enter the code directory;

Enter the command ls to view the file thermistoe.py in the directory.

Enter the command python3 thermistoe.py to run the code. The result is as follows:

The following is the code:

```
import RPi.GPIO as GPIO
import smbus
import time
import math
address = 0x48
bus=smbus.SMBus(1)
cmd=0x40
def analogRead(chn):
    value = bus.read_byte_data(address,cmd+chn)
    return value
def analogWrite(value):
    bus.write_byte_data(address,cmd,value)
def setup():
    GPIO.setmode(GPIO.BOARD)
def loop():
    while True:
        value = analogRead(0)
        voltage = value / 255.0 * 3.3
        Rt = 10 * voltage / (3.3 - voltage)
        tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0)
        tempC = tempK -273.15
        print('ADC Value : %d, Voltage : %.2f,
Temperature : %.2f'%(value,voltage,tempC))
```

```
            time.sleep(0.01)
def destroy():
    GPIO.cleanup()
if __name__ == '__main__':
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()
```

Code Interpretation

This code is similar to the potentiometer controlling LED lights, but the algorithm for calculating temperature is different. The following program is the algorithm for calculating temperature.

```
def loop():
    while True:
        value = analogRead(0)
        voltage = value / 255.0 * 3.3
        Rt = 10 * voltage / (3.3 - voltage)
        tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0)
        tempC = tempK -273.15
        print('ADC Value : %d, Voltage : %.2f,
       Temperature : %.2f'%(value,voltage,tempC))
        time.sleep(0.01)
```

First read the ADC value of the PCF8591 A0 port, then calculate the voltage and resistance of the thermistor according to Ohm's law, and then convert it into a temperature value according to the formula.

Lesson 13 Relay & Motor

Overview

In this course, you will learn a kind of special switch module, Relay Module.we will use a push button to control a relay and drive the motor.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board

1 x Breadboard

1 x 1kΩ Resistor

1 x 220Ω Resistor

1 x LED

1 x Button

1 x Relay

1 x Motor

1 x NPN Transistor

1 x Diode

Product Introduction

Relay

Relay is a safe switch which can use low power circuit to control high power circuit. It consists of electromagnet and contacts. The electromagnet is controlled by low power circuit and contacts is used in high power circuit. When the electromagnet is energized, it will attract contacts.

Motor

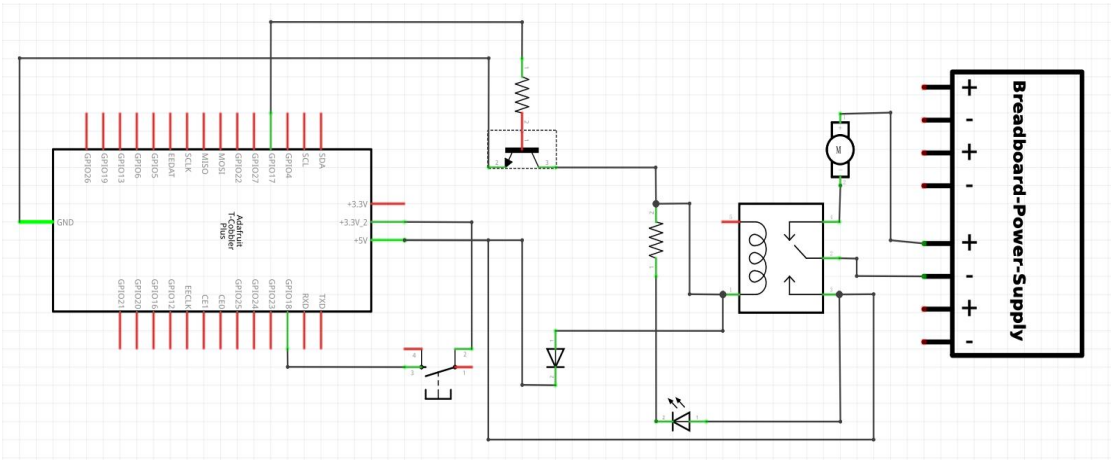
Motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually the shaft of motor, and can drive

other mechanical devices to run. Diagram below is a small DC motor with two pins.When motor get connected to the power supply, it will rotate in one direction. Reverse the polarity of power supply, then motor rotates in opposite direction.

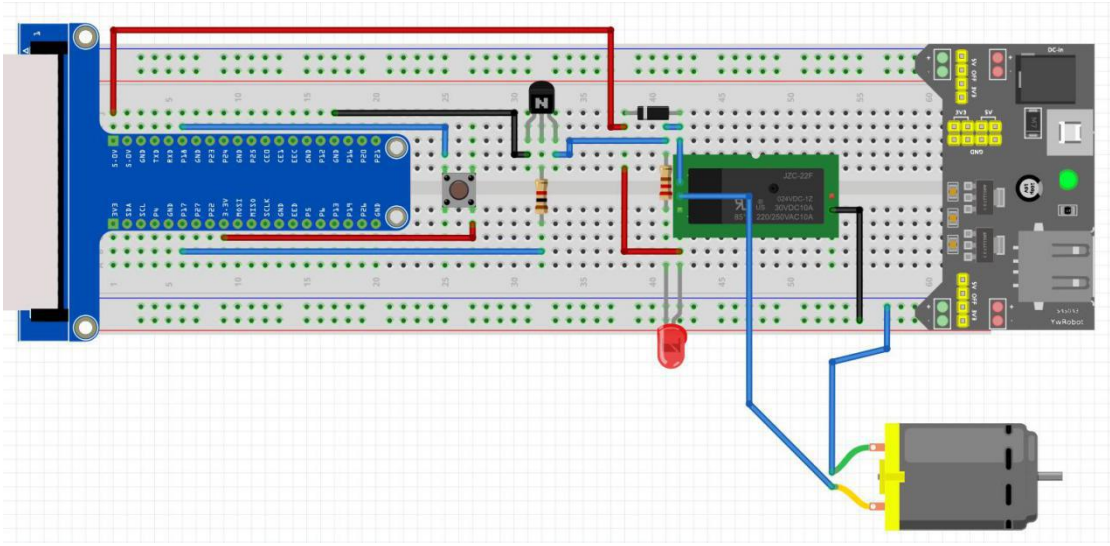
Circuit

When connecting the circuit, pay attention to that because the motor is a high-power component, do not use the power provided by the RPi, which may do damage to your RPi. the logic circuit can be powered by RPi power or external power supply which should have the common ground with RPi.

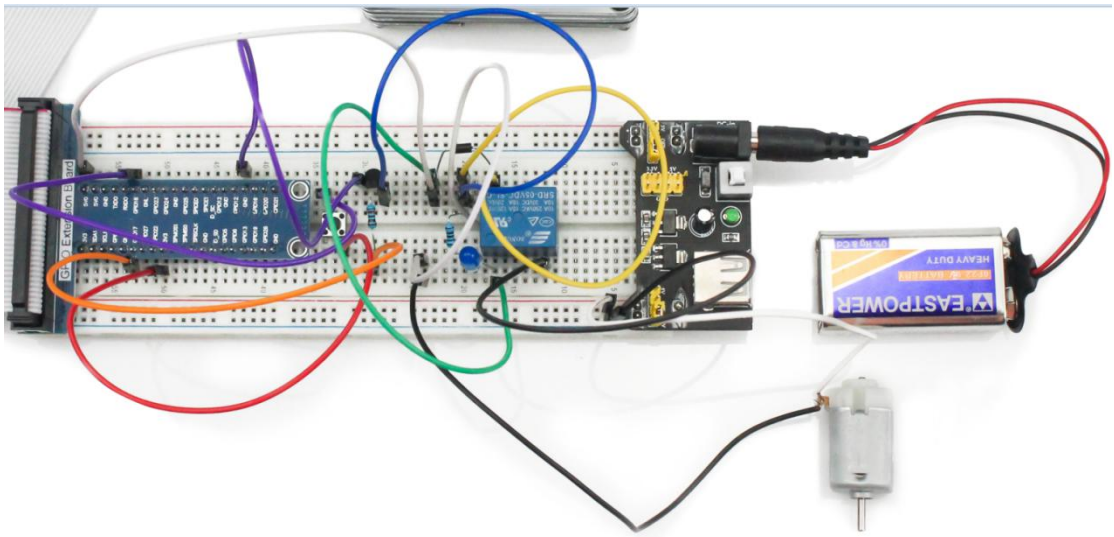
Connection diagram



Wiring diagram

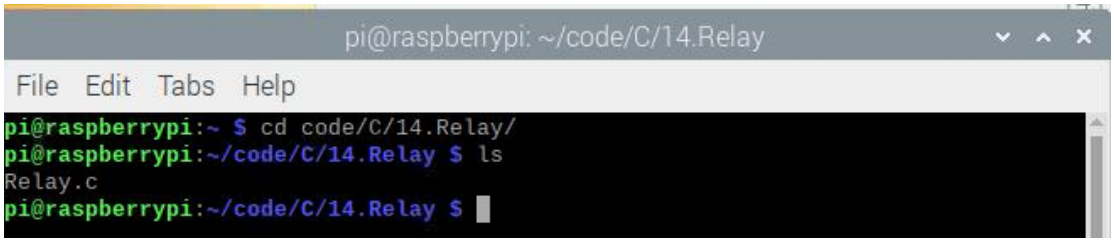


Example Picture



C code

Open the terminal and enter the "cd code / C / 14.Relay /" command to enter the "Relay" code directory;
Enter the "ls" command to view the file "Relay.c" in the directory;



Enter "gcc Relay.c -o Relay -lwiringPi" command to generate "Relay.c" executable file "Relay", enter "ls" command to view;
Enter the "sudo ./Relay" command to run the code. The result is as follows:

Press "Ctrl + c" to end the program. The following is the program code:

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

#define relayPin      0     //define the relayPin
#define buttonPin 1         //define the buttonPin
int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiring fairelay,print messageto screen
        printf("fairelay !");
        return 1;
    }
    printf("starting...\n");
    pinMode(relayPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    pullUpDnControl(buttonPin, PUD_UP);    //pull up to high level
    int i=0;
    int g=0;
    while(1){
        if(digitalRead(buttonPin)==HIGH && g==0)
        {
                delay(20);
                if(digitalRead(buttonPin)==HIGH)
            {
```

```
                        i=i+1;
                        i=i%2;
                        g=1;
                        if(i==0)
                {
                                digitalWrite(relayPin,LOW);
                                printf("relayPin.....off\n");
                }
                        if(i==1)
                {
                                digitalWrite(relayPin,HIGH);
                                printf("relayPin.....on\n");
                }
                  }
        }
                else if(digitalRead(buttonPin)==LOW)
        {
        g=0;
        }
    }

    return 0;
}
```

Code Interpretation

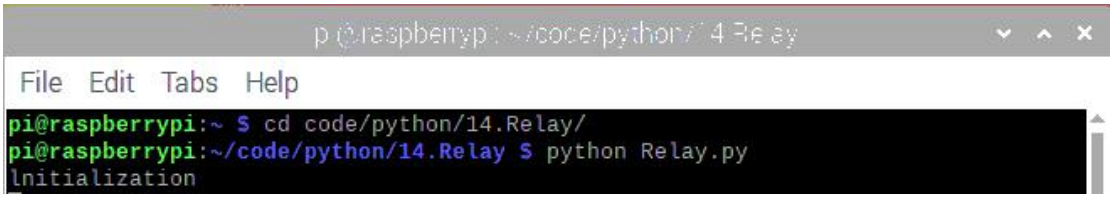
In the main function "main()", first define two flags "i=0" and "g=0", "i" is used to determine whether the previous key was pressed, "g" is a flag bit, which is used to judge whether the button is still pressed, "if(digitalRead(buttonPin)==HIGH && g==0)" is the judgment statement of the state of the button, "delay(20)" is a delay function, this delay The function of the time function is to eliminate the jitter of the button.

```
    int i=0;
        int g=0;
        while(1){
            if(digitalRead(buttonPin)==HIGH && g==0)
            {
                        delay(20);
                        if(digitalRead(buttonPin)==HIGH)
```

Python code

1. Use "cd code / python / 14.Relay /" command to enter the directory of Relay code.

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



Press "Ctrl + c" to end the program. The following is the program code:

```
import RPi.GPIO as GPIO
import time

relayPin = 11       # define the relayPin
buttonPin = 12      # define the buttonPin

def setup():
    print ('Initialization')
    GPIO.setmode(GPIO.BOARD)        # Numbers GPIOs by physical location
    GPIO.setup(relayPin, GPIO.OUT)    # Set relayPin's mode is output
    GPIO.setup(buttonPin, GPIO.IN)    # Set buttonpin mode input

def loop():
    i=0
    g=0
    while True:
        reading = GPIO.input(buttonPin)
        if reading == GPIO.HIGH & g==0:
            time.sleep(0.2)
            reading = GPIO.input(buttonPin)
            if reading == GPIO.HIGH:
                i=i+1
                i=i%2
                g=1
                if i==0:
                    GPIO.output(relayPin,GPIO.LOW)
                    print("relayPin.....off")
                if i==1:
                    GPIO.output(relayPin,GPIO.HIGH)
                    print("relayPin.....on")
        else:
            g=0

def destroy():
    GPIO.output(relayPin, GPIO.LOW)       # relay off
```

```
    GPIO.cleanup()                          # Release resource

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

Code Interpretation

The first thing we have to do is definetwo symbols "i = 0" and "g = 0"in the 'loop ()' function. "I" is used to determinewhether the previous key was pressed."g" is used to determinewhether the key is continuously pressed.

```
i=0
g=0
```

Use the 'reading = GPIO.input (buttonPin)' statement to check the status. The code 'if reading == GPIO.HIGH & g == 0' is used to determine whether the key is pressed.When the button is pressed, give it a delay function 'time.sleep (0.2)' to debounce to prevent the button from not being pressed.After giving a delay, if the key is still pressed, it is judged that the current state of the key is pressed.

```
reading = GPIO.input(buttonPin)
        if reading == GPIO.HIGH & g==0:
            time.sleep(0.2)
            reading = GPIO.input(buttonPin)
            if reading == GPIO.HIGH:
```

Lesson 14 Servo

Overview

In this lesson, we will learn how to control the rotation of SG90 servo by Raspberry pi, and how to control the rotation angle of servo. The servo is a type of geared motor that can only rotate 180 degrees. It is controlled by the electrical pulse sent by Raspberry pi. It has three wires.The brown wire is the ground wire, connected to the GND pin of the Raspberry pi, the red wire is the positive pole of the power supply, connected to the 5V pin of the Raspberry pi, and the orange wire is the signal

line, connected to the GPIO18 pin of the Raspberry pi.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

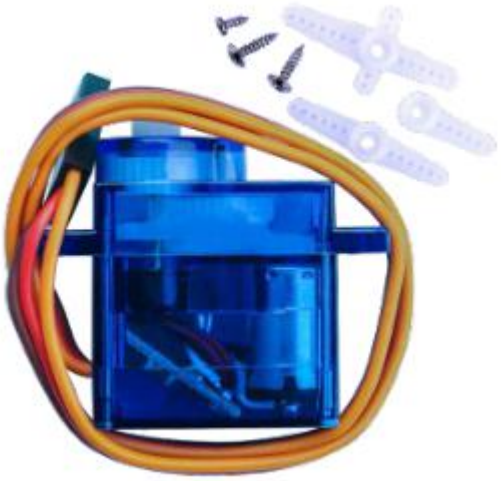
1 x SG90 Servo

3 x Jumper Wires

Product Introduction

SG90

Servo is an auto-control system, consisting of DC motor, reduction gear,sensorand control circuit. Usually, it can rotate in the range of 180 degrees. Servo can output larger torque and is widely used in model airplane, robot and so on. It has three lines, including two for electric power line positive (2-VCC, red), negative (3- GND, brown), and the signal line (1-Signal, orange).



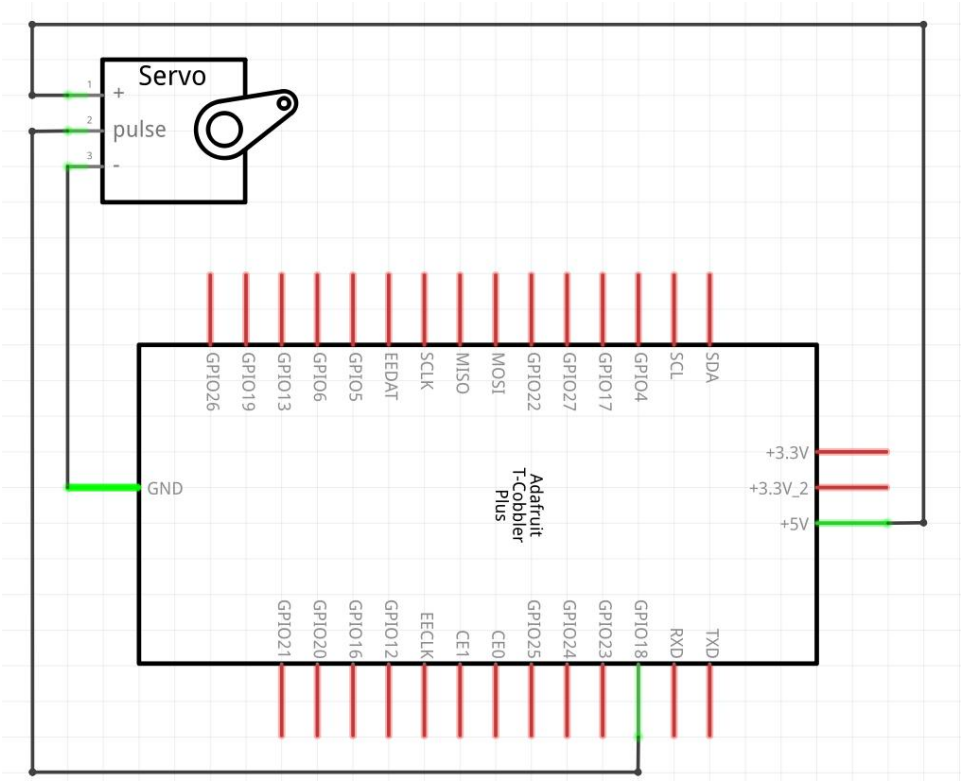
We use 50Hz PWM signal with a duty cycle in a certain range to drive the servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the servo angle 0 degrees - 180 degree linearly.

Part of the corresponding values are as follows:

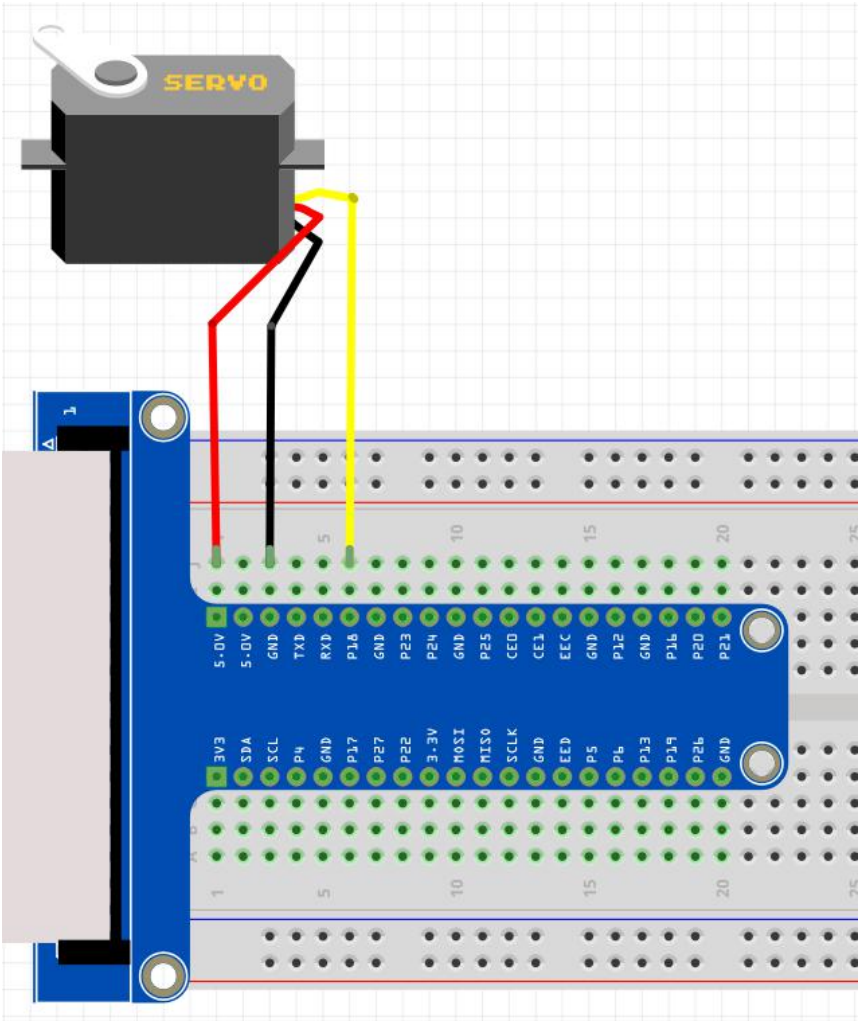
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

When you change the servo signal, servo will rotate to the designated position.

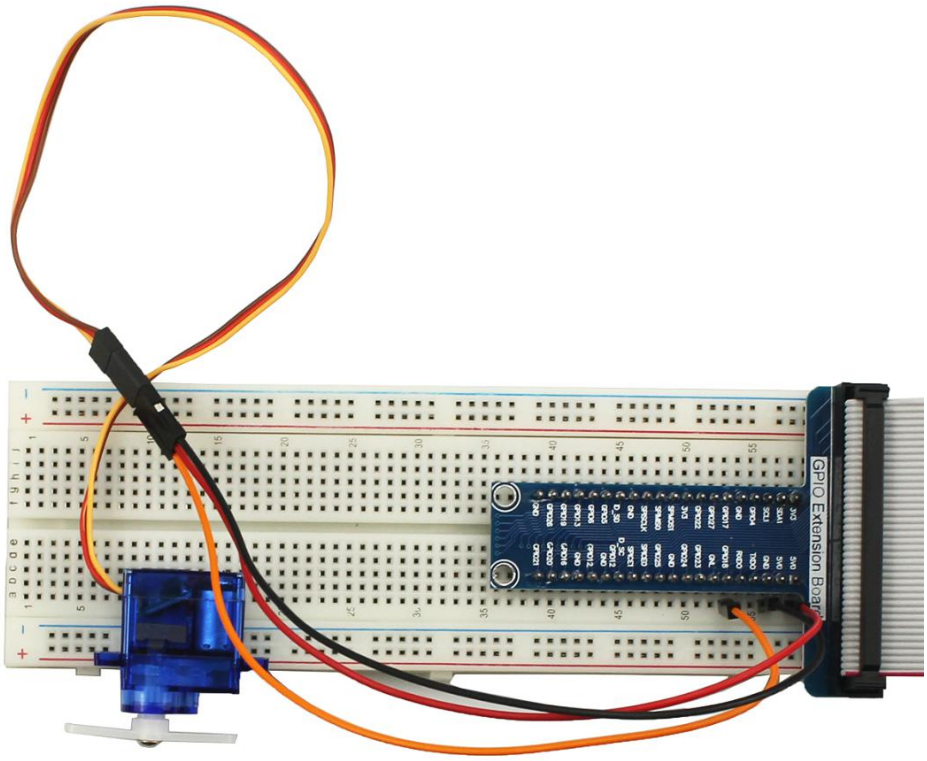
Connection Diagram



Wiring Diagram



Example Figure



C code

Open the terminal and enter the"cd code / C / 15.Servo /" command to enter the "Servo" code directory;

Enter the "ls"command to view the file "Servo.c" in the directory;

Enter "gcc Servo.c -o Servo -lwiringPi" command to generate "Servo.c" executable file "Servo", enter "ls" command to view;

Enter the "sudo ./Servo" command to run the code. The result is as follows:

Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define OFFSET_MS 3          //Define the unit of servo pulse offset: 0.1ms
#define SERVO_MIN_MS 5+OFFSET_MS              //define the pulse duration for
minimum angle of servo
#define SERVO_MAX_MS 25+OFFSET_MS             //define the pulse duration for
maximum angle of servo

#define servoPin        1             //define the GPIO number connected to servo
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
        return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void servoInit(int pin){              //initialization function for servo PMW pin
        softPwmCreate(pin,    0, 200);
}
void servoWrite(int pin, int angle){        //Specif a certain rotation angle (0-180) for
the servo
        if(angle > 180)
                angle = 180;
        if(angle < 0)
                angle = 0;
        softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
}
void servoWriteMS(int pin, int ms){         //specific the unit for pulse(5-25ms) with
specific duration output by servo pin: 0.1ms
        if(ms > SERVO_MAX_MS)
                ms = SERVO_MAX_MS;
        if(ms < SERVO_MIN_MS)
                ms = SERVO_MIN_MS;
        softPwmWrite(pin,ms);
}
```

```
int main(void)
{
	int i;
	if(wiringPiSetup() == -1){ //when initialize wiring faiservo,print messageto
screen
		printf("setup wiringPi faiservo !");
		return 1;
	}
	printf("starting ...\n");
	servoInit(servoPin);		//initialize PMW pin of servo
	while(1){
		for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){	//make servo rotate
from minimum angle to maximum angle
			servoWriteMS(servoPin,i);
			delay(10);
		}
		delay(1000);
		for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){	//make servo rotate
from maximum angle to minimum angle
			servoWriteMS(servoPin,i);
			delay(10);
		}
		delay(1000);
	}
	return 0;
}
```

Code Interpretation

50 Hz pulse, namely cycle for 20ms, is required to control Servo. In function softPwmCreate (int pin, int initialValue, int pwmRange), the unit of third parameter pwmRange is 100US, namely 0.1ms. In orde to get the PWM with cycle of 20ms, the pwmRange shoulde be set to 200. So in subfunction of servoInit (), we create a PWM pin with pwmRange 200.

```
void servoInit(int pin){
	softPwmCreate(pin,	0, 200);
}
```

As 0-180 degrees of servo corresponds to PWM pulse width 0.5-2.5ms, with PwmRange 200 and unit 0.1ms. So, in function softPwmWrite (int pin, int value), the

scope 5-25 of parameter value corresponds to 0-180degrees of servo. What's more, the number writen in subfunction servoWriteMS () should be within the rangeof 5-25. However, in practice, due to the manufacture error of each servo, pulse width will also have deviation. So we define a minimum pulse width and a maximum one and an error offset.

```
void servoWriteMS(int pin, int ms){
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}
```

In subfunction "servoWrite (int pin, int angle)", input directly angle (0-180 degrees), and map the angle to the pulse width and then output it.

```
    void servoWrite(int pin, int angle){            if(angle > 180)
            angle = 180;
        if(angle < 0)
            angle = 0;

softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
    }
```

Finally, in the "while" cycle of main function, use two "for" cycle to make servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
while(1){
        for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(1000);
        for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(1000);
    }
```

Python code

1. Use the "cd code / python / 15.Servo /" command to go to the "Servo" code directory.

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

Press "Ctrl + c" to end the program. The following is the program code:

```
import RPi.GPIO as GPIO
import time

DUTY = 0.5
MIN_DUTY=2.5+DUTY
MAX_DUTY=12.5+DUTY
servo = 12

def setup():
    global P
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(servo,GPIO.OUT)
    GPIO.output(servo,GPIO.LOW)

    P = GPIO.PWM(servo,50)
    P.start(0)

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

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

def loop():
```

```
        while True:
            for i in range(0,181,1):
                servoWrite(i)
                time.sleep(0.01)
            time.sleep(0.5)
            for i in range (180,-1,-1):
                servoWrite(i)
                time.sleep(0.01)
            time.sleep(0.5)

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

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

Code Interpretation

50 Hz pulse, namely cycle for 20ms, is required to control Servo. So we need to use the code 'P = GPIO.PWM(servo,50)' to set the PWM frequency of 'servo' to 50Hz.

```
P = GPIO.PWM(servo,50)
```

Since the steering gear of 0-180 degrees corresponds to a PWM pulse width of 0.5-2.5ms and a duty ratio of 2.5%-12.5% within a period of 20ms. Write the angle in the sub-function 'def servoWrite(a)', and map the angle to the duty cycle to output PWM, and then output the angle at which the servo will rotate. However, in fact, due to the manufacturing error of each steering gear, the pulse width will also vary. Therefore, we define the function 'def map(value, fromLow, fromHigh, toLow, toHigh)' to calculate the minimum pulse width and maximum pulse width and the error offset.

```
def servoWrite(a):    # make the servo rotate to specific angle (0-180 degrees)
    if(a<0):
        a=0
    elif(a>180):
        a=180
```

```
            P.ChangeDutyCycle(map(a,0,180,MIN_DUTY,MAX_DUTY))      #map the
angle to duty cycle and output it
```

Finally, in the "while" cycle of main function, use two "for" cycle to make servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
def loop():
    while True:
        for i in range(0,181,1):
            servoWrite(i)
            time.sleep(0.01)
        time.sleep(0.5)
        for i in range (180,-1,-1):
            servoWrite(i)
            time.sleep(0.01)
        time.sleep(0.5)
```

Lesson 15 74HC595 & LEDBar Graph

Overview

In this lesson, we will learn how to use the 74HC595 chip to expand the Raspberry pi pins to light up eight LED lights and present them in the form of running lights.

Although we can directly connect eight LED to the Raspberry pi development board, this will quickly consume the GPIO port resources of the development board. More GPIO ports means more peripherals can be connected to the Raspberry pi, so GPIO resources are very valuable.

You will use a 74HC595 serial-to-parallel converter chip, which will save a lot of pin resources. The chip has eight outputs and three inputs, and you can use them to input data one at a time.

The chip makes driving LED slightly slower, about 500,000 times per second, but it is still faster than humans can watch, so the effect will not change significantly after use.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

1 x 74HC595

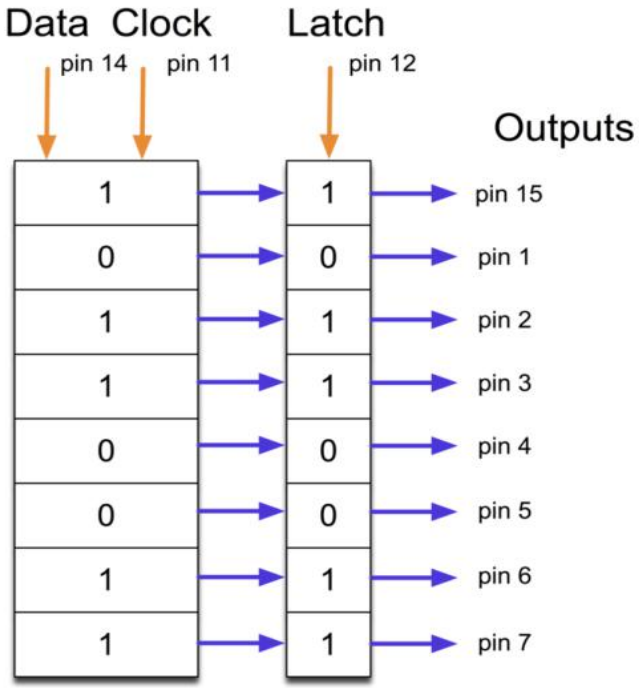
8 x 220Ω Resistor

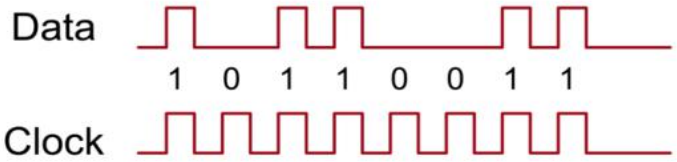
8 x LED

Product Introduction

74HC595

74HC595 chip is used to convert serial data into parallel data. 74HC595 can convert the serialdata ofone byte to 8 bits, and send its corresponding level to the corresponding 8 ports. With this feature, 74HC595 can be used to expand the IO port of Raspberry Pi. At least 3 ports on the RPI board are need to control 8 ports of 74HC595.





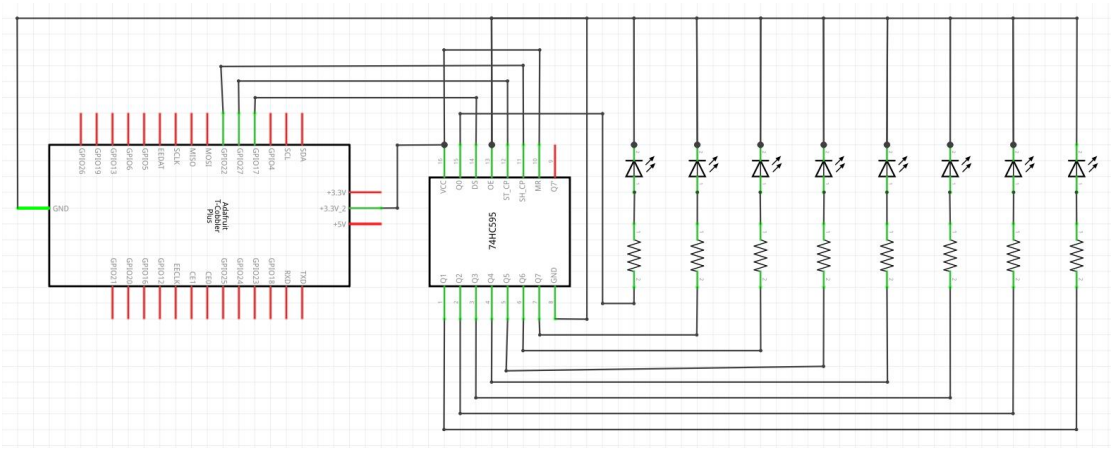
The clock pin needs to receive 8 pulses. In each pulse, if the data pin is high, 1 is pushed into the shift register; when the data pin is low, it is directly 0. When all 8 pulses are received, enabling the "Latch" pin will copy these 8 values to the shift register; this is necessary; otherwise, the LED will be wrong when data is loaded into the shift register flashes.

The chip also has an output enable (OE) pin, which is used to enable or disable all outputs. You can connect it to the GPIO port to output PWM, and use the PWM signal to control the brightness of the LED. This pin is active low, so we connect it to GND.

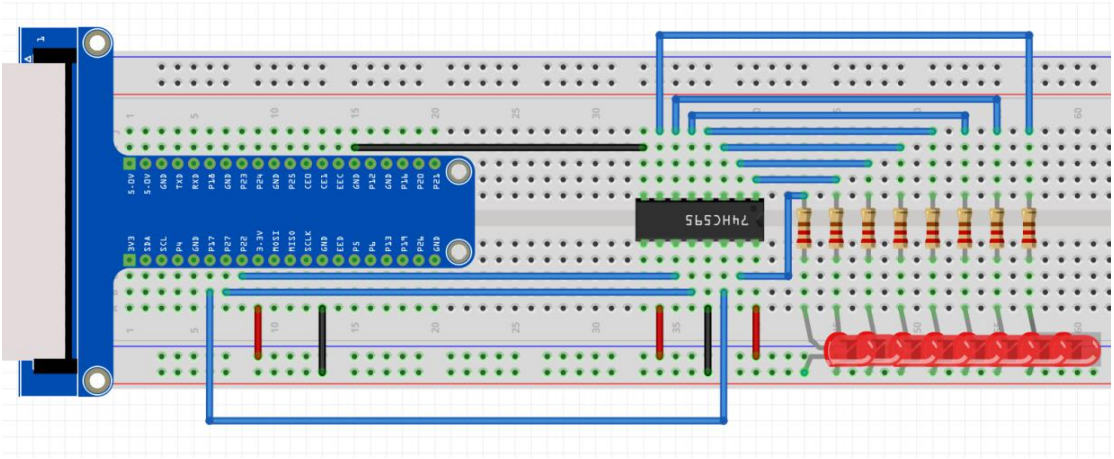
The ports of 74HC595 are described as follows:

Pin	Pin No.	Description
Q0-Q7	15,1-7	Parallel data output
VCC	16	The positive electrode of power supply, the voltage is 2~6V
GND	8	The negative electrode of power supply
DS	14	Serial data Input
OE	13	Enable output, When this pin is in high level, Q0-Q7is in high resistance state When this pin is in low level, Q0-Q7is in output mode
ST_CP	12	Parallel update output: when its electrical level is rising, it will update the parallel data
SH_CP	11	Serial shift clock: when its electrical level is rising, serial data input register will do a shift.
MR	10	Remove the shift register: When this pin is in low level, the content in shift register will be cleared.
Q7'	9	Serial data output: it can be connected

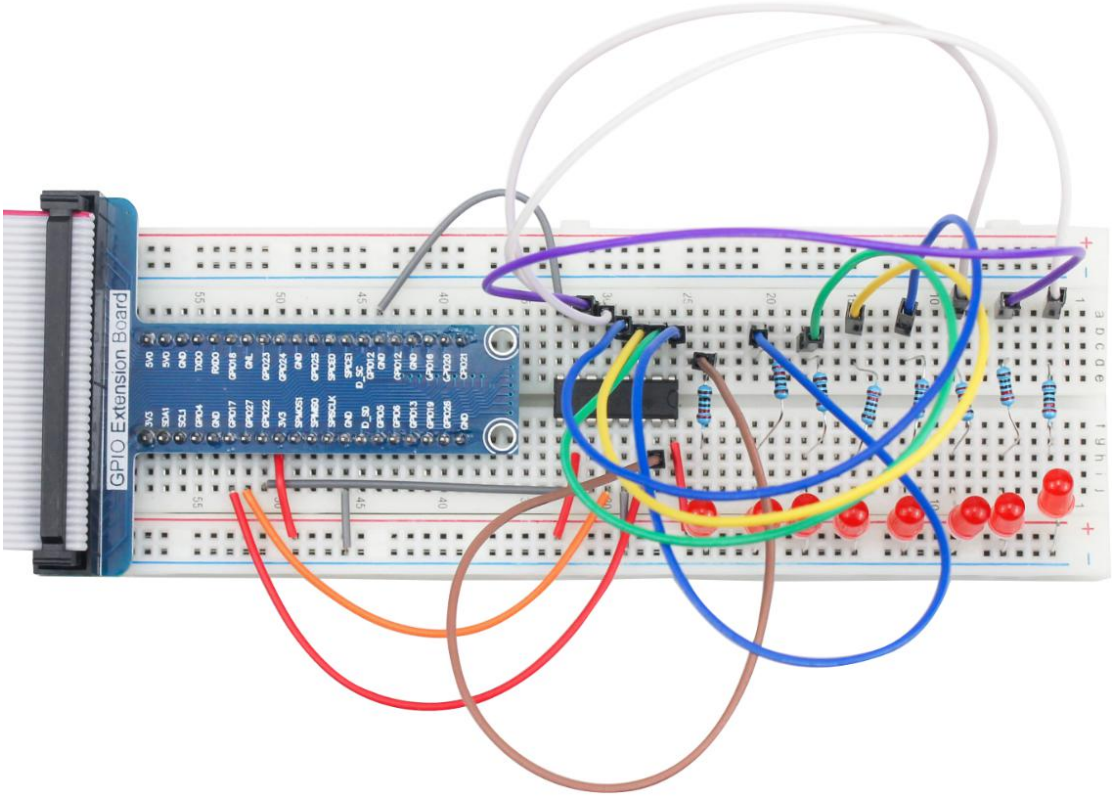
Connection Diagram



Wiring Diagram



Example Figure



C code

Open terminal and enter "cd code / C / 17.74HC595LED /"command to enter "74HC595LED" code directory;

Enter the "ls"command to view the file "74HC595LED.c" in the directory;

Enter "gcc 74HC595LED.c -o 74HC595LED -lwiringPi" command to generate "74HC595LED.c" executable file "74HC595LED", enter "ls" command to view; enter "sudo ./74HC595LED" command to run the code,

the results are shown :

Press "Ctrl + c" to end the program. The following is the program code:

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

#define     dataPin     0     //DS Pin of 74HC595(Pin14)
#define     latchPin    2     //ST_CP Pin of 74HC595(Pin12)
#define     clockPin 3        //CH_CP Pin of 74HC595(Pin11)

void sendOut(int dPin,int cPin,int order,int val){
      int i;
       for(i = 0; i < 8; i++){
            digitalWrite(cPin,LOW);
            if(order == LSBFIRST){
                  digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
                  delayMicroseconds(10);
            }
            else {
                  digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
                  delayMicroseconds(10);
            }
            digitalWrite(cPin,HIGH);
            delayMicroseconds(10);
      }
}
int main(void)
{
      int i;
      unsigned char x;
      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
            printf("setup wiringPi failed !");
            return 1;
      }
```

```
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        x=0x01;
        for(i=0;i<8;i++){
            digitalWrite(latchPin,LOW);         // Output low level to latchPin
            sendOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
            digitalWrite(latchPin,HIGH); // Output high level to latchPin, and
74HC595 will update the data to the parallel output port.
            x<<=1; // make the variable move one bit to left once, then the bright
LED move one step to the left once.
            delay(100);
        }
        x=0x80;
        delay(500);
        for(i=0;i<8;i++){
            digitalWrite(latchPin,LOW);
            sendOut(dataPin,clockPin,LSBFIRST,x);
            digitalWrite(latchPin,HIGH);
            x>>=1;
            delay(100);
        }
        delay(500);
    }
    return 0;
}
```

Code Interpretation

In the code, we define the "sendout (dPin, cPin, order, val)" function, which is used to output values in order. "dPin" represents the data pin and "cPin" represents the clock, in order from high to low or low to high. This function complies with the 74HC595 operating mode.

```
void sendOut(int dPin,int cPin,int order,int val){
    int i;
     for(i = 0; i < 8; i++){
          digitalWrite(cPin,LOW);
          if(order == LSBFIRST){
```

```
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
```

In the code, we configure three pins to control the 74HC595. And define a one-byte variable to control the state of 8 LEDs through the 8 bits of the variable. The LED lights on when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED on.

```
x=0x01;
```

In the “while” cycle of main function, use “for” cycle to send x to 74HC595 output pin to control the LED. In “for” cycle, x will be shift one bit to left in one cycle, then in the next round when data of x is sent to 74HC595, the LED turned on will move one bit to left once.

```
while(1){
    x=0x01;
    for(i=0;i<8;i++){
        digitalWrite(latchPin,LOW);
        sendOut(dataPin,clockPin,LSBFIRST,x);
        digitalWrite(latchPin,HIGH);
        x<<=1;
        delay(100);
    }
    x=0x80;
    delay(500);
    for(i=0;i<8;i++){
        digitalWrite(latchPin,LOW);
        sendOut(dataPin,clockPin,LSBFIRST,x);
        digitalWrite(latchPin,HIGH);
        x>>=1;
        delay(100);
    }
    delay(500);

}
```

Python code

1. Use the "cd code / python / 17.74HC595LED /" command to enter the directory of "74HC595LED".

2. Use "python 74HC95LED.py"command to execute "74HC595LED.py" code.

Press "Ctrl + c" to end the program. The following is the program code:

```
import RPi.GPIO as GPIO
import time

LWAY = 1
MWAY = 2

dataPin = 11            #DS Pin of 74HC595(Pin14)
latchPin = 13           #ST_CP Pin of 74HC595(Pin12)
clockPin = 15           #CH_CP Pin of 74HC595(Pin11)

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(dataPin,GPIO.OUT)
    GPIO.setup(latchPin,GPIO.OUT)
    GPIO.setup(clockPin,GPIO.OUT)

def sendout(dPin,cPin,way,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW)
        if(way == LWAY):
            GPIO.output(dPin,(0x01 & (val>>i)==0x01) and GPIO.HIGH or
GPIO.LOW)
        elif(way == MWAY):
            GPIO.output(dPin,(0x80 & (val<<i)==0x80) and GPIO.HIGH or
GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH)
```

```
def loop():
    while True:
        x=0x01
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)          #Output low level to
latchPin
            sendout(dataPin,clockPin,LWAY,x)       #Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)         #Output high level to
latchPin, and 74HC595 will update the data to the parallel output port.
            x<<=1                                  # make the variable move one bit to left
once, then the bright LED move one step to the left once.
            time.sleep(0.1)
        x=0x80
        time.sleep(0.5)
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)          #Output low level to
latchPin
            sendout(dataPin,clockPin,LWAY,x)       #Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)         #Output high level to
latchPin, and 74HC595 will update the data to the parallel output port.
            x>>=1                                  # make the variable move one bit to left
once, then the bright LED move one step to the left once.
            time.sleep(0.1)
        time.sleep(0.5)

def destroy():          # When 'Ctrl+C' is pressed, the function is executed.
    GPIO.cleanup()

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

Code Interpretation

In the code, we define the sendout(dPin,cPin,way,val) function, which is used to output values in sequence. “DPin” means data pin, “cPin” means clock, the order is from high to low or from low to high. This function conforms to the 74HC595 operating mode. “LWAY” and “MWAY” are two different directions.

```
def sendout(dPin,cPin,way,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW)
        if(way == LWAY):
            GPIO.output(dPin,(0x01 & (val>>i)==0x01) and GPIO.HIGH or
GPIO.LOW)
        elif(way == MWAY):
            GPIO.output(dPin,(0x80 & (val<<i)==0x80) and GPIO.HIGH or
GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH)
```

In the loop() function, we use two "for" cycle to achieve the target. First, define a variable "x=0x01", binary00000001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then a LED is turned on. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED turned on will be shifted. Repeat the operation, the effect of flowing water light will be formed. If the direction of the shift operation for x is different, the flowing direction is different.

```
def loop():
    while True:
        x=0x01
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)          #Output low level to
latchPin
            sendout(dataPin,clockPin,LWAY,x)       #Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)         #Output high level to
latchPin, and 74HC595 will update the data to the parallel output port.
            x<<=1                                  # make the variable move one bit to left
once, then the bright LED move one step to the left once.
            time.sleep(0.1)
        x=0x80
        time.sleep(0.5)
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)          #Output low level to
latchPin
            sendout(dataPin,clockPin,LWAY,x)       #Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)         #Output high level to
latchPin, and 74HC595 will update the data to the parallel output port.
            x>>=1                                  # make the variable move one bit to left
once, then the bright LED move one step to the left once.
            time.sleep(0.1)
        time.sleep(0.5)
```

Lesson 16 74HC595 & 7-Segment Display.

Overview

In this course, you will learn how to use 74HC595 to control 7-segment display. and make it display sixteen decimal character "0-F".

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

1 x 74HC595

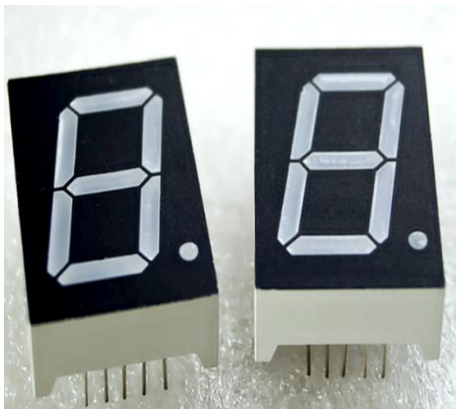
1 x 220Ω Resistor

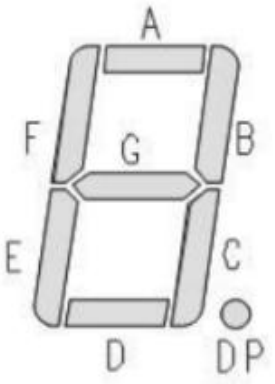
1 x 7 Segment Display

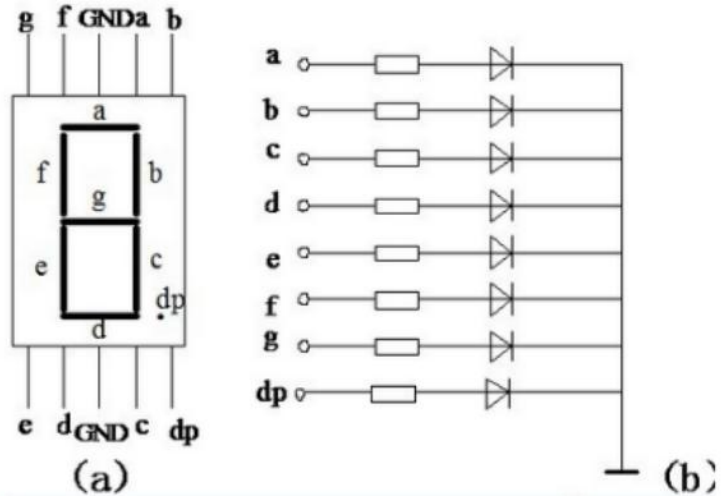
Product Introduction

7-segment display

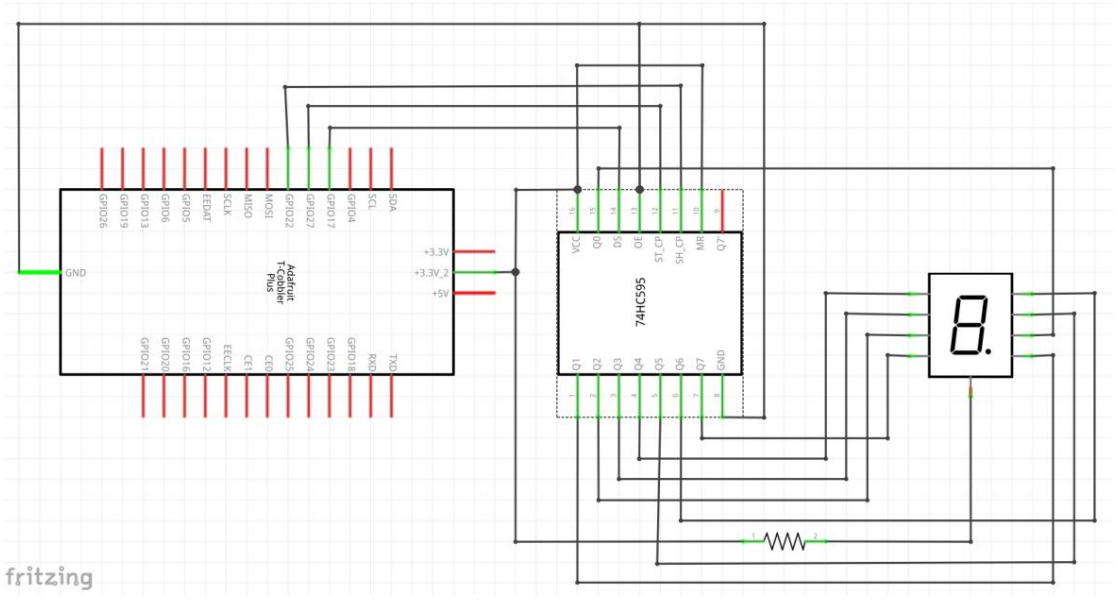
The 7-segment digital tube is composed of multiple light-emitting diodes packaged together to form an "8"-shaped device. The wires have been connected internally, and only their respective pins and common electrodes need to be led out. The digital tube is actually composed of seven light-emitting LEDs, plus eight decimal points. These segments are represented by the letters a, b, c, d, e, f, g, and dp. When voltage is applied to specific segments of the digital tube, these specific segments will light up to form what our eyes see. The words are gone. Such as: display a "2", then it should be “A“bright “B’ bright ‘G’ bright ‘E’ bright ‘D” bright “F’ not bright ‘C’ not bright “DP” not bright. The 7-segment digital tube has different points such as general brightness and super bright, and also has different sizes such as 0.5 inch and 1 inch. The display strokes of small-sized digital tubes are usually composed of one light-emitting diode, while large-sized digital tubes are composed of two or more light-emitting diodes. In general, the voltage drop of a single light-emitting diode is about 1.8V, and the current does not exceed 30mA. The anode of the light-emitting diode is connected to the anode of the power supply together is called the common anode digital tube, and the cathode of the light-emitting diode is connected to the cathode of the power supply together is called the common cathode digital tube. The numbers and characters displayed by commonly used LED digital tubes are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. This lesson uses a common anode digital tube.



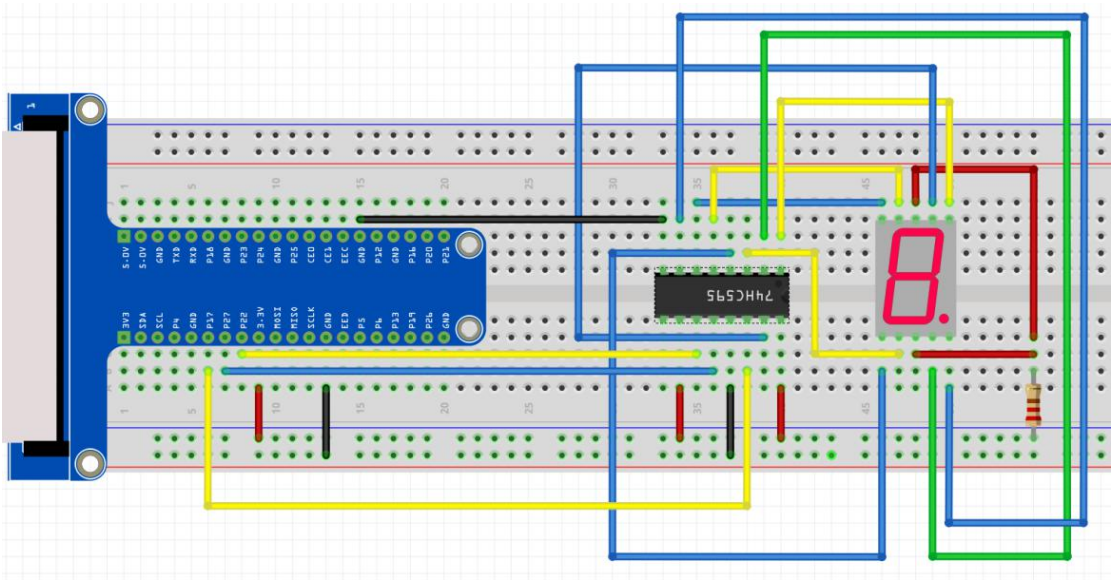




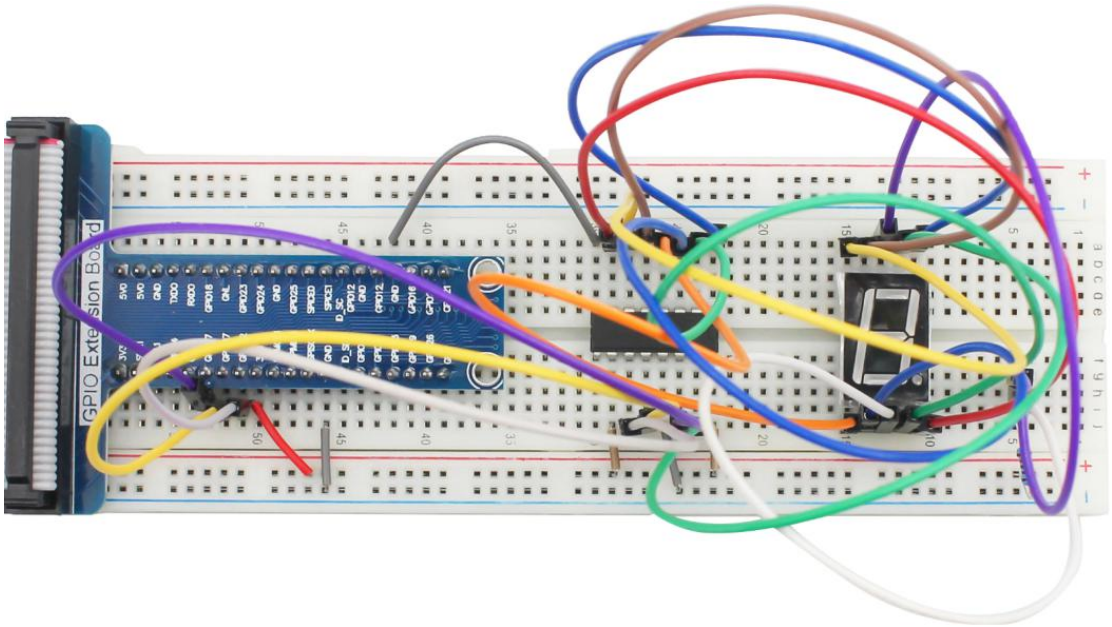
Connection Diagram



Wiring Diagram



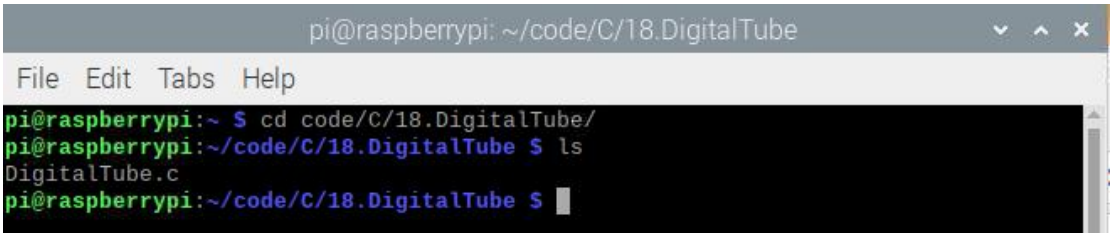
Example picture



C code

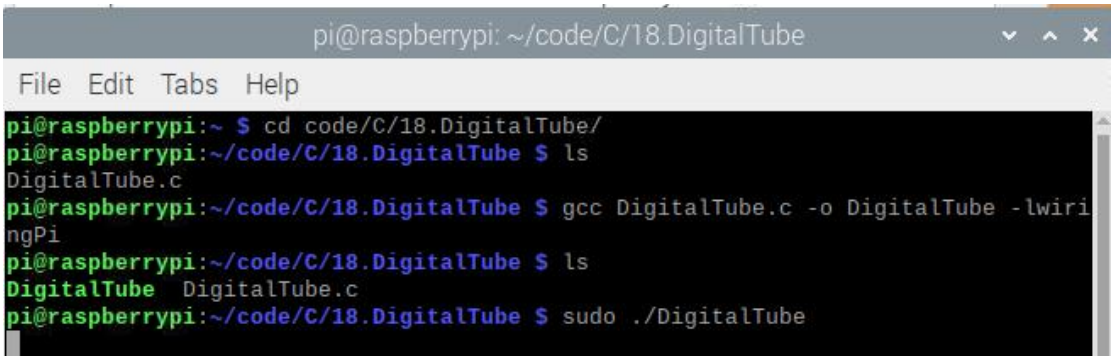
Open the terminal and enter the"cd code / C / 18.DigitalTube /" command to enter the "DigitalTube" code directory;

Enter the "ls"command to view the file "DigitalTube.c" in the directory;



Enter "gcc DigitalTube.c -o DigitalTube -lwiringPi"command to generate "DigitalTube.c" executable file "DigitalTube", enter "ls" command to view; enter "sudo ./DigitalTube" command to run the code,

the results are shown below :



Press "Ctrl + c" to end the program. The following is the program code:

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

#define     dataPin     0     //DS Pin of 74HC595(Pin14)
#define     latchPin    2     //ST_CP Pin of 74HC595(Pin12)
#define     clockPin 3        //CH_CP Pin of 74HC595(Pin11)
//encoding for character 0-F of common anode SevenSegmentDisplay.
```

```
unsigned char
num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,
0x86,0x8e};

void sendOut(int dPin,int cPin,int order,int val){
    int i;
     for(i = 0; i < 8; i++){
          digitalWrite(cPin,LOW);
          if(order == LSBFIRST){
               digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
               delayMicroseconds(10);
          }
          else {
               digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
               delayMicroseconds(10);
          }
          digitalWrite(cPin,HIGH);
          delayMicroseconds(10);
    }
}

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            sendOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and
the highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
        for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            sendOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);//Use the
"&0x7f" to display the decimal point.
            digitalWrite(latchPin,HIGH);
```

```
            delay(500);
        }
    }
    return 0;
}
```

Code Interpretation

First, put encoding of “0”-“F” into the”num” array.

```
Unsigned char
num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,
0x86,0x8e};
```

In the “for” cycle of loop() function, the contents of the array "num" are output in order. The digital tube can display the corresponding characters correctly. Note that in the "sendOut(int dPin, int cPin, int order, int val)" function, flag bit highest bit will be transmitted preferentially.

```
for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            sendOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and
the highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
```

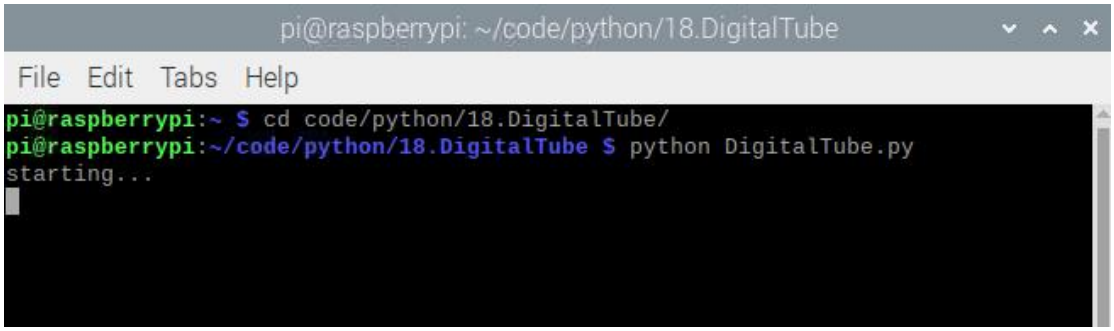
If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

```
sendOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);
```

Python code

1. Use the "cd code / python / 18.DigitalTube /"command to enter the directory of DigitalTube.

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



Press "Ctrl + c" to end the program. The following is the program code:

```
import RPi.GPIO as GPIO
import time

LWAY = 1
MWAY = 2
#define the pins connect to 74HC595
dataPin     = 11        #DS Pin of 74HC595(Pin14)
latchPin    = 13        #ST_CP Pin of 74HC595(Pin12)
clockPin = 15           #CH_CP Pin of 74HC595(Pin11)

num =
[0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x
8e]

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(dataPin,GPIO.OUT)
    GPIO.setup(latchPin,GPIO.OUT)
    GPIO.setup(clockPin,GPIO.OUT)

def sendout(dPin,cPin,way,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW)
        if(way == LWAY):
            GPIO.output(dPin,(0x01 & (val>>i)==0x01) and GPIO.HIGH or
GPIO.LOW)
        elif(way == MWAY):
            GPIO.output(dPin,(0x80 & (val<<i)==0x80) and GPIO.HIGH or
GPIO.LOW)
```

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

def loop():
    while True:
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            sendout(dataPin,clockPin,MWAY,num[i])    #Output the figures and
the highest level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            sendout(dataPin,clockPin,MWAY,num[i]&0x7f)    #Use "&0x7f"to
display the decimal point.
GPIO.output(latchPin,GPIO.HIGH)
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)

def destroy():          # When 'Ctrl+C' is pressed, the function is executed.
    GPIO.cleanup()

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

Code Interpretation

First, put encoding of “0”-“F” into the“num” array.

```
num =
[0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x
8e]
```

In the “for” cycle of loop() function, the contents of the array "num" are output in order. The digital tube can display the corresponding characters correctly. Note that in the "sendOut(int dPin,int cPin,int order,int val)" function, flag bit highest bit will be transmitted preferentially.

```
for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
```

```
            sendout(dataPin,clockPin,MWAY,num[i])      #Output the figures and
the highest level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
```

If you want to display the decimal point, make the highest bit of each array become 0, which can be implemented easily by num[i]&0x7f.

```
sendout(dataPin,clockPin,MWAY,num[i]&0x7f)    #Use "&0x7f"to display the
decimal point.
```

Lesson 17 LCD1602

Overview

In this lesson you will learn learn a display screen, LCD1602.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

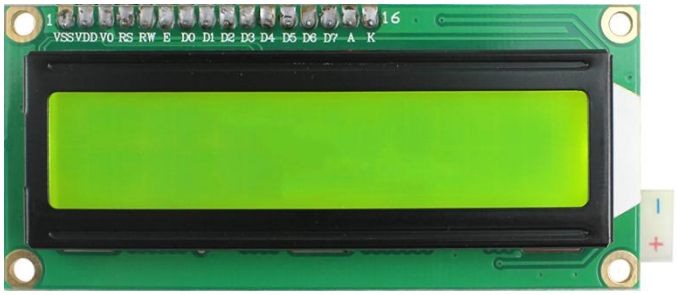
1 x LCD1602

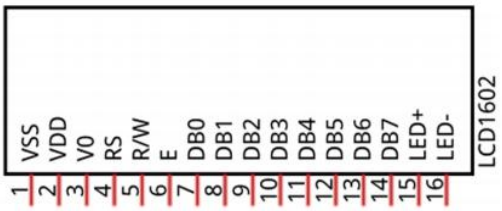
1 x PCF8574

Product Introduction

LCD1602

LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. I2C LCD1602 integrates a I2C interface, which connects the serial-input &parallel-output module to LCD1602. We just use 4 lines to the operate LCD1602 easily.As shown below is a monochrome LCD1602 display screen， and its circuit pin diagram：

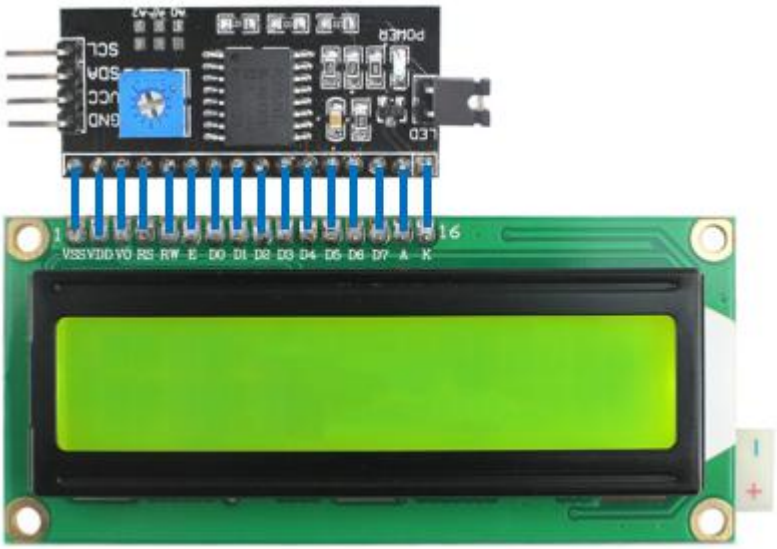




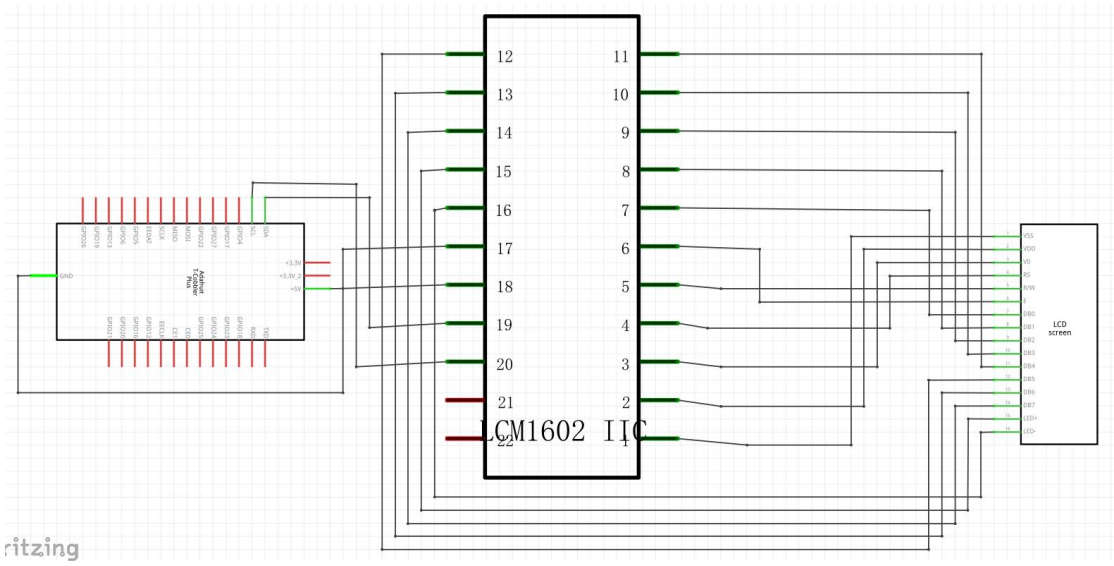
PCF8574

The serial-to-parallel chip used in this module is PCF8574(PCF8574A), and its default I2C address is 0x27(0x3F), and you can view all the RPI bus on your I2C device address through command "i2cdetect -y 1" to. PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each other: (refer to the "configuration I2C" section below) below is the PCF8574 pin schematic diagram and the block pin diagram:

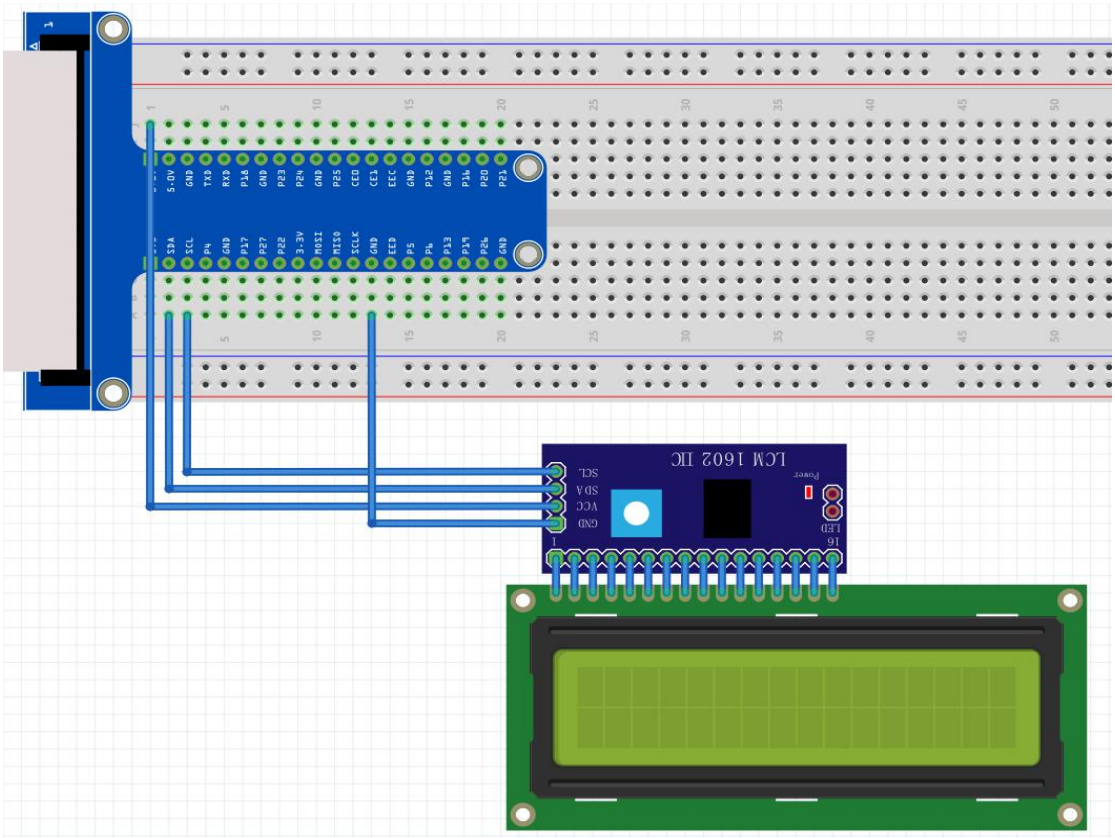
LCD Display



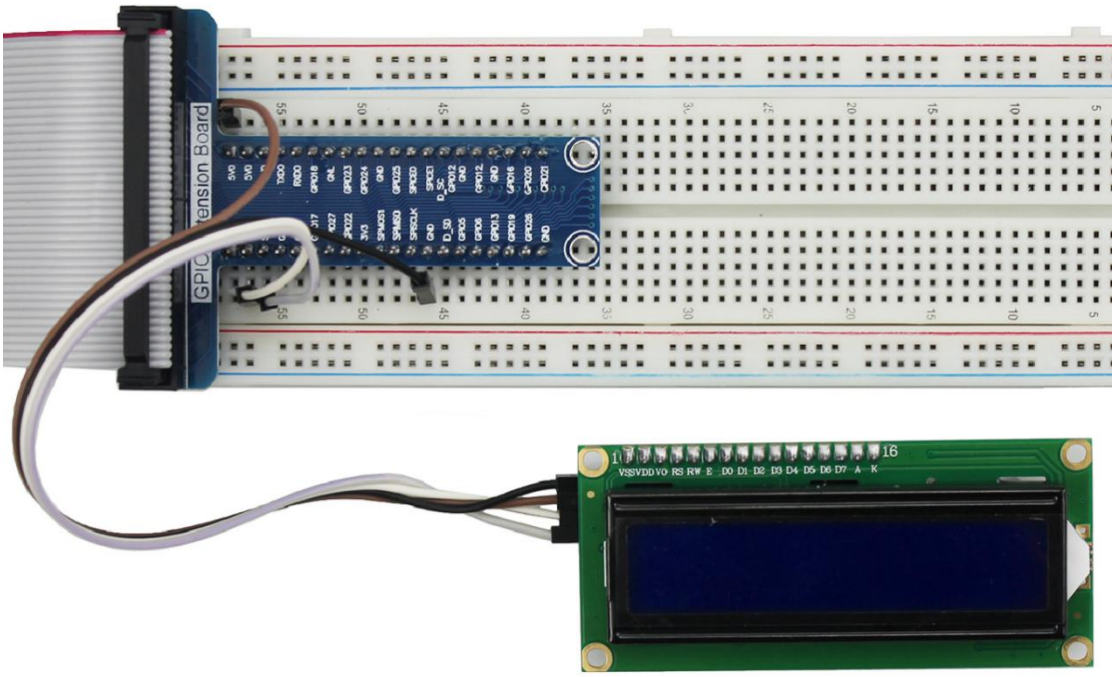
Connection Diagram



Wiring Diagram



Example Figure



Configure I2C

The I2C interface raspberry pi is closed by default. When the VNC interface is opened in the remote Raspberry pi interface in the front, we also open the I2C interface. We can enter a command to check whether the I2C interface is open:

```
lsmod | grep i2c
```

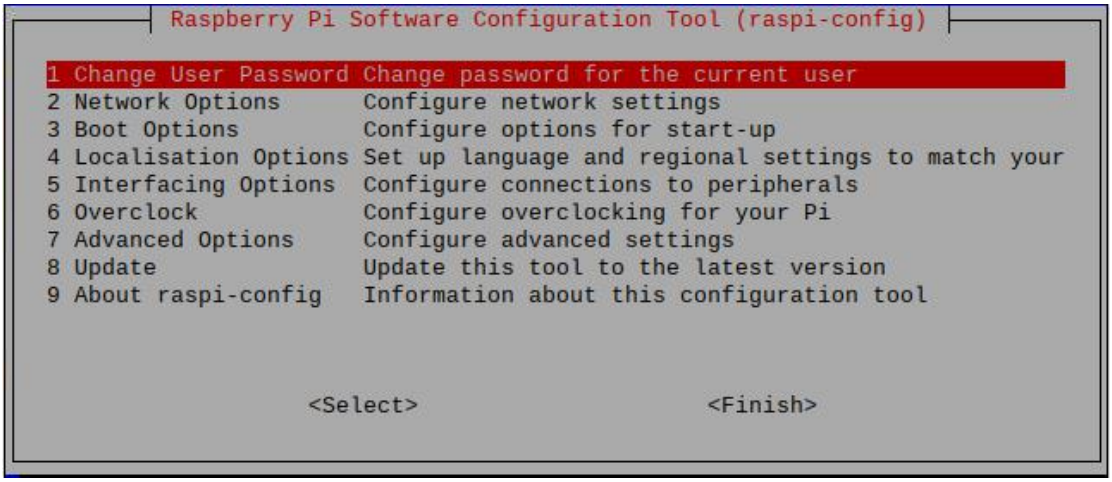
If the I2C interface is open, the following will be displayed:

```
pi@raspberrypi:~ $ lsmod | grep i2c
i2c_bcm2835            16384  0
i2c_dev                20480  0
pi@raspberrypi:~ $
```

The following numbers 16384 and 20480 are different for each Raspberry pi, so as long as the displayed content is roughly the same as the above content, the I2C interface of the Raspberry pi has been opened. If the I2C interface is not open, we need to open it manually. We can enter the command in the terminal:

```
sudo raspi-config
```

Then open the following dialog



Select "5 Interfacing Options"->"P5 I2C"->"<Yes>"->"<OK>"->"<Finish>", and then restart the Raspberry pi. Now you can check whether the I2C interface Open successfully.

Install I2C-Tools

Enter the following command to install I2C-Tools:

sudo apt-get install i2c-tools

Enter the following command for I2C device address detection:

i2cdetect -y 1

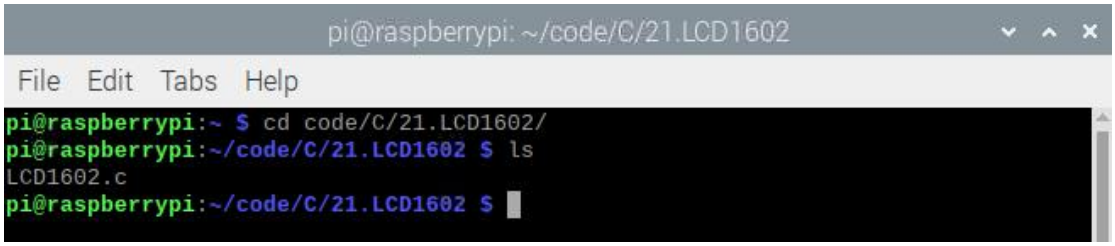
```
pi@raspberrypi:~ $ sudo apt-get install i2c-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
i2c-tools is already the newest version (4.1-1).
i2c-tools set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 131 not upgraded.
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

27 is the I2C address of PCF8574.

C code

Open the terminal and enter the "cd code / C / 21.LCD1602 /" command to enter the "LCD1602" code directory;

Enter the"ls"command to view the file "LCD1602.c" in the directory;



Enter "gcc LCD1602.c -o LCD1602 -lwiringPi -lwiringPiDev" command to generate "LCD1602.c" executable file "LCD1602", enter "ls" command to view; enter "sudo ./LCD1602" command to run the code.

The results are as shown in below:

Press "Ctrl + c" to end the program. The following is the program code:

```
#include <stdlib.h>
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8574.h>
#include <lcd.h>
#include <time.h>

#define pcf8574_address 0x27        // default I2C address of Pcf8574
```

```
//#define pcf8574_address 0x3F              // default I2C address of Pcf8574A
#define BASE 64                 // BASE is not less than 64
/////// Define the output pins of the PCF8574, which are directly connected to the
LCD1602 pin.
#define RS          BASE+0
#define RW          BASE+1
#define EN          BASE+2
#define LED         BASE+3
#define D4          BASE+4
#define D5          BASE+5
#define D6          BASE+6
#define D7          BASE+7

int lcdhd;// used to handle LCD
void printCPUTemperature(){// sub function used to print CPU temperature
        FILE *fp;
        char str_temp[15];
        float CPU_temp;
        // CPU temperature data is stored in this directory.
        fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
        fgets(str_temp,15,fp);          // read file temp
        CPU_temp = atof(str_temp)/1000.0;      // convert to Celsius degrees
        printf("CPU's temperature : %.2f \n",CPU_temp);
        lcdPosition(lcdhd,0,0);         // set the LCD cursor position to (0,0)
        lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp);// Display CPU temperature on LCD
        fclose(fp);
}
void printDataTime(){//used to print system time
        time_t rawtime;
        struct tm *timeinfo;
        time(&rawtime);// get system time
        timeinfo = localtime(&rawtime);// convert to local time
        printf("%s \n",asctime(timeinfo));
        lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)

lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->t
m_sec);
//Display system time on LCD
}
int main(void){
        int i;

        if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
                printf("failed !");
```

```
            return 1;
        }
        pcf8574Setup(BASE,pcf8574_address);// initialize PCF8574
        for(i=0;i<8;i++){
            pinMode(BASE+i,OUTPUT);         // set PCF8574 port to output mode
        }
        digitalWrite(LED,HIGH);         // turn on LCD backlight
        digitalWrite(RW,LOW);             // allow writing to LCD
lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);// initialize LCD and return
“handle” used to handle LCD
        if(lcdhd == -1){
            printf("lcdInit failed !");
            return 1;
        }
        while(1){
            printCPUTemperature();// print CPU temperature
            printDataTime();              // print system time
            delay(1000);
        }
        return 0;
}
```

Code Interpretation

It can be seen from the code that PCF8591 and PCF8574 have a lot of similarities, they are through the I2C interface to expand the GPIO RPI. First defines the I2C address of the PCF8574 and the extension of the GPIO pin, which is connected to the GPIO pin of the LCD1602.

```
#define pcf8574_address 0x27              // default I2C address of Pcf8574
//#define pcf8574_address 0x3F              // default I2C address of Pcf8574A
#define BASE 64                 // BASE is not less than 64
//////// Define the output pins of the PCF8574, which are directly connected to the
LCD1602 pin.
#define RS          BASE+0
#define RW          BASE+1
#define EN          BASE+2
#define LED         BASE+3
#define D4          BASE+4
#define D5          BASE+5
#define D6          BASE+6
#define D7          BASE+7
```

Then, in main function, initialize the PCF8574, set all the pins to output mode, and turn on the LCD1602 backlight.

```
pcf8574Setup(BASE,pcf8574_address);// initialize PCF8574
    for(i=0;i<8;i++){
        pinMode(BASE+i,OUTPUT);        // set PCF8574 port to output mode
    }
    digitalWrite(LED,HIGH);        // turn on LCD backlight
```

Then use lcdInit() to initialize LCD1602 and set the RW pin of LCD1602 to 0 (namely, can be write) according to requirements of this function. The return value of the function called "Handle" is used to handle LCD1602"next.

```
lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);
```

In the main function “while”, two subfunctions are called to display the CPU temperature and the time. First look atsubfunction “printCPUTemperature()”. The CPU temperature data isstored in the "/sys/class/thermal/thermal_zone0/temp"file. We need read contents of the file, and convert it to temperature value stored in variable “CPU_temp”, and use “lcdPrintf()” to display it on LCD.

```
void printCPUTemperature(){// sub function used to print CPU temperature
    FILE *fp;
    char str_temp[15];
    float CPU_temp;
    // CPU temperature data is stored in this directory.
    fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
    fgets(str_temp,15,fp);          // read file temp
    CPU_temp = atof(str_temp)/1000.0;     // convert to Celsius degrees
    printf("CPU's temperature : %.2f \n",CPU_temp);
    lcdPosition(lcdhd,0,0);         // set the LCD cursor position to (0,0)
    lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp);// Display CPU temperature on LCD
    fclose(fp);
}
```

Next is subfunction “printDataTime()” used to print system time. First, got the standard time and store it into variable rawtime, and then converted it to the local time , and finally display the time information on LCD1602.

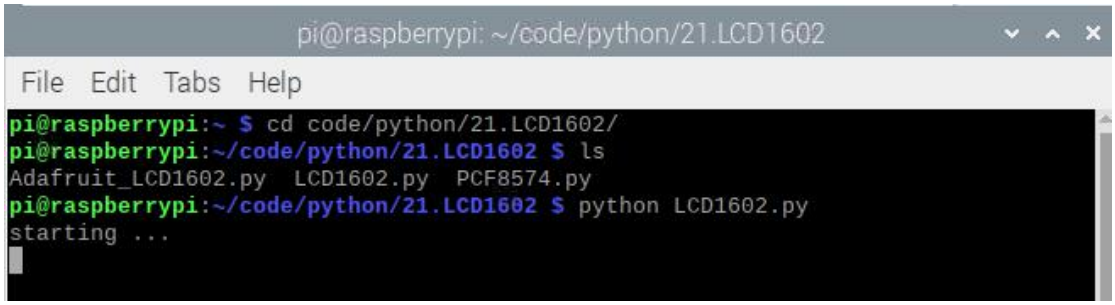
```
void printDataTime(){//used to print system time
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);// get system time
    timeinfo = localtime(&rawtime);// convert to local time
```

```
        printf("%s \n",asctime(timeinfo));
        lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)

lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->t
m_sec);
//Display system time on LCD
}
```

Python code

1. Use the "cd code / python / 21.LCD1602 /" command to enter the directory of LCD1602.

2. Use"python LCD1602.py" command to execute "LCD1602.py" code.



Press "Ctrl + c" to end the program. The following is the program code:

```
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

from time import sleep, strftime
from datetime import datetime

def get_cpu_temp():         # get CPU temperature and store it into file
"/sys/class/thermal/thermal_zone0/temp"
        tmp = open('/sys/class/thermal/thermal_zone0/temp')
        cpu = tmp.read()
        tmp.close()
        return '{:.2f}'.format( float(cpu)/1000 ) + ' C'

def get_time_now():         # get system time
        return datetime.now().strftime('       %H:%M:%S')
```

```
def loop():
    mcp.output(3,1)         # turn on LCD backlight
    lcd.begin(16,2)         # set number of LCD lines and columns
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)    # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( get_time_now() )      # display the time
        sleep(1)

def destroy():# When 'Ctrl+C' is pressed, the function is executed.
    lcd.clear()

PCF8574_address = 0x27    # I2C address of the PCF8574 chip.
PCF8574A_address = 0x3F    # I2C address of the PCF8574A chip.
# Create PCF8574 GPIO adapter.
try:
    mcp = PCF8574_GPIO(PCF8574_address)
except:
    try:
        mcp = PCF8574_GPIO(PCF8574A_address)
    except:
        print ('I2C Address Error !')
        exit(1)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)

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

Code Interpretation

The code uses two files, ‘PCF8574.py’ and ‘Adafruit_LCD1602.py’. These two files and code files are stored in the same directory, they are both required, please do not delete them. ‘PCF8574.py’ is used to provide I2C communication modes and operating methods for certain ports forRaspberry pi and PCF8574 chips.
‘Adafruit_LCD1602.py’ is used to provide some function operation methods for LCD1602.
‘PCF8574.py’: This module provides two classes PCF8574_I2C and PCF8574_GPIO.
PCF8574_I2C class: Provides PCF8574 read and write methods. PCF8574_GPIO

class: provides a set of standardized GPIO functions.
'Adafruit_LCD1602.py': This module provides the basic operation method of LCD1602, including Adafruit_CharLCD class.
In the code, first'mcp = PCF8574_GPIO(PCF8574_address)' gets the object used to operate the PCF8574 port, then'lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO= mcp)' Get the object used to operate the LCD1602.

```
PCF8574_address = 0x27    # I2C address of the PCF8574 chip.
PCF8574A_address = 0x3F    # I2C address of the PCF8574A chip.
# Create PCF8574 GPIO adapter.
try:
    mcp = PCF8574_GPIO(PCF8574_address)
except:
    try:
        mcp = PCF8574_GPIO(PCF8574A_address)
    except:
        print ('I2C Address Error !')
        exit(1)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
```

According to the circuit connection, port 3 of PCF8574 is connected to positive pole of LCD1602 backlight. Then in the loop () function, use of mcp.output(3,1) to turn on LCD1602 backlight, and set number of LCD lines and columns.

```
def loop():
    mcp.output(3,1)     # turn on LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
```

In the next while cycle, set the cursor position, and display the CPU temperature and time.

```
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)    # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( get_time_now() )     # display the time
        sleep(1)
```

The CPU temperature is stored in the file "/sys/class/thermal/thermal_zone0/temp". "tmp = open('/sys/class/thermal/thermal_zone0/temp')" is used to open the file, and "cpu = tmp.read()"is used to read the content of the file, then convert it to degrees Celsius and return.

```
def get_cpu_temp():                    # get CPU temperature and store it into
file
      #"/sys/class/thermal/thermal_zone0/temp"
       tmp = open('/sys/class/thermal/thermal_zone0/temp')
       cpu = tmp.read()
       tmp.close()
      return '{:.2f}'.format( float(cpu)/1000 ) + ' C'
```

‘def get_time_now()’ is a sub-function for getting time:

```
def get_time_now():          # get system time
      return datetime.now().strftime('       %H:%M:%S')
```

Lesson 18 DHT11

Overview

In this lesson you will learn how to use DHT11 to measure temperature and humidity.

Parts Required

1 x Raspberry pi

1 x 10K Resistor

1 x Breadboard

1 x DHT11 Module

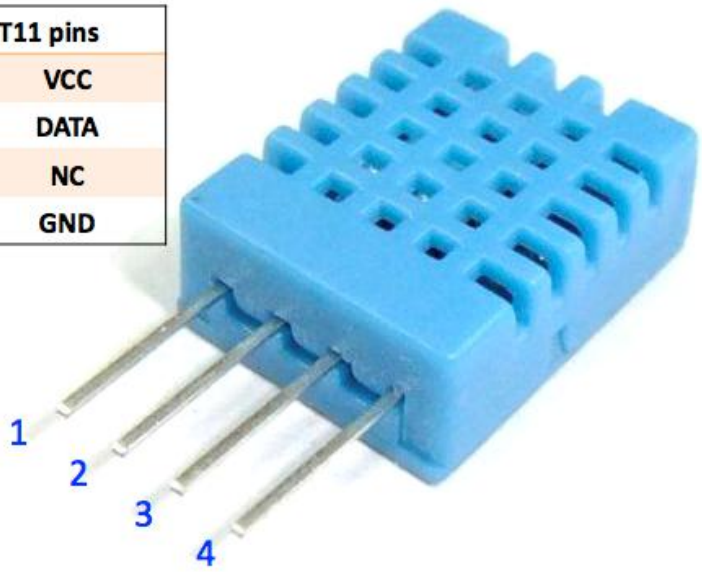
Some Jumper Wiress

Product Introduction

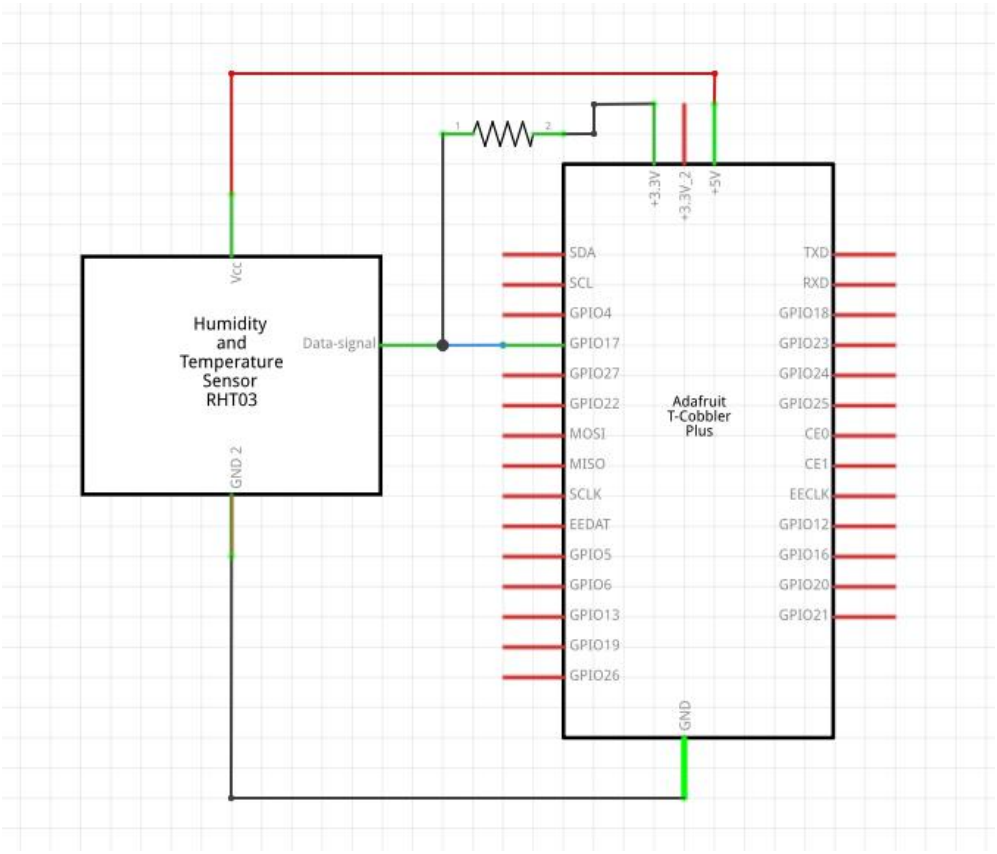
Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated inside.It has 1S's initialization time after powered up. The operating voltage is within the range of 3.3V-5.5V. SDA pin is a data pin, which is used to communicate with other device.NC pin (Not Connected Pin) are pins found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have unexposed functionality).

Those pins should not be connected to any of the circuit connections.

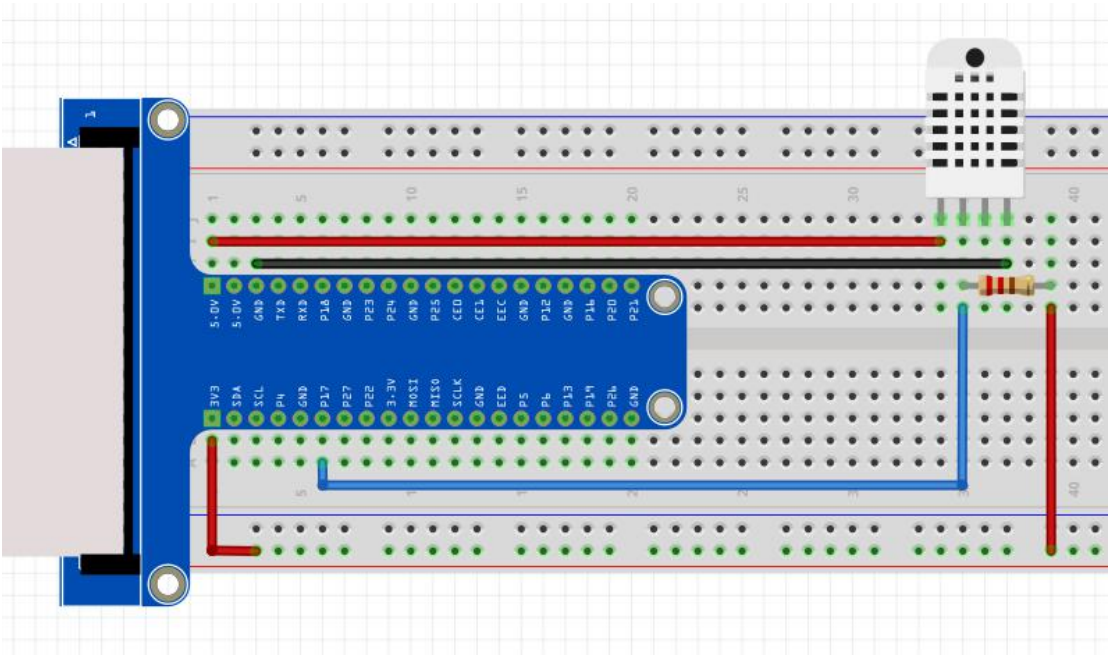
DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



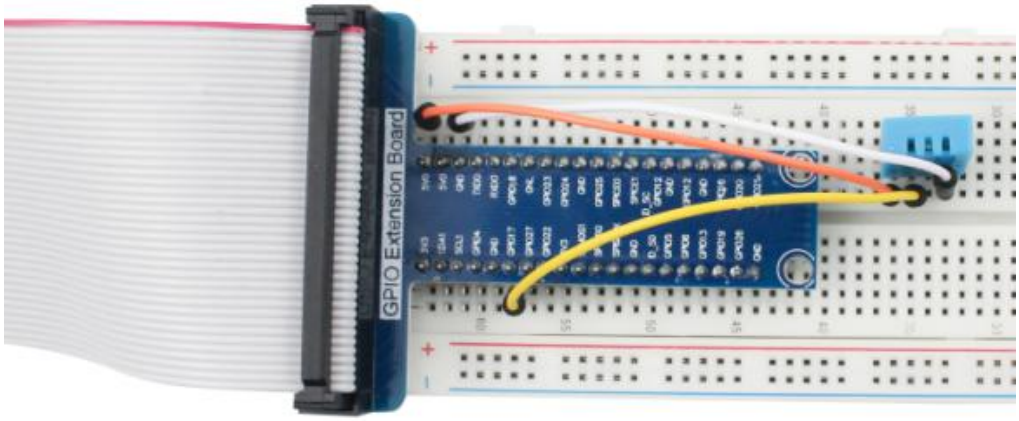
Connection Diagram



Wiring Diagram



Example Figure



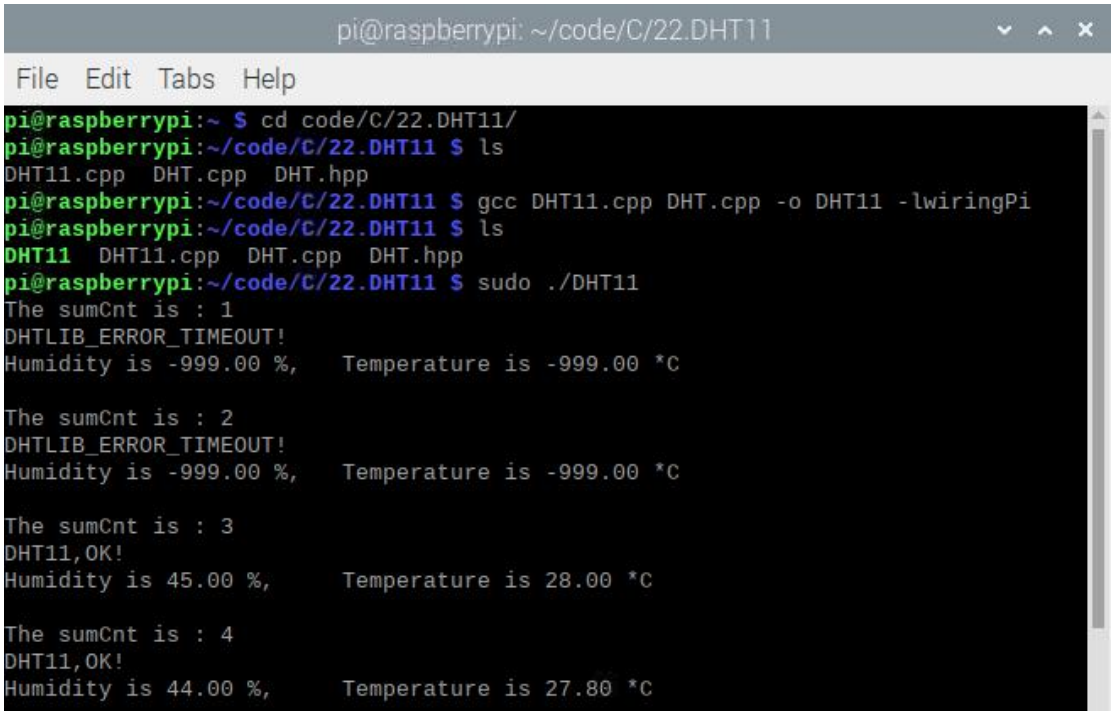
C code

Open the terminal and enter the "cd code / C / 22.DHT11 /" command to enter the "DHT11" code directory;

Enter the "ls"command to view the files "DHT11.cpp", "DHT.cpp", and "DHT.hpp" in the directory;

Enter "gcc DHT11.cpp DHT.cpp -o DHT11 -lwiringPi" command to generate "DHT11.cpp" and "DHT.cpp" executable file "DHT11", enter"ls" command to view; enter "sudo ./DHT11" command to run the code.

The results are as follows:



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdint.h>
#include "DHT.hpp"

#define DHT11_Pin    0       //define the pin of sensor

int main(){
    DHT dht;              //create a DHT class object
    int chk,sumCnt;//chk:read the return value of sensor; sumCnt:times of reading
sensor
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
        printf("failed !");
        return 1;
    }
    while(1){
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value.
Then determine whether data read is normal according to the return value.
        sumCnt++;            //counting number of reading times
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:         //if the return value is DHTLIB_OK, the data
is normal.
```

```
                printf("DHT11,OK! \n");
                break;
            case DHTLIB_ERROR_CHECKSUM:         //data check has errors
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:          //reading DHT times out
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:          //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f
*C\n\n",dht.humidity,dht.temperature);
        delay(2000);
    }
    return 1;
}
```

Code Interpretation

In this project code, we use a custom library file "DHT.hpp". It is located in the same directory with program files "DHT11.cpp" and "DHT.cpp", and methods for reading DHT sensor are provided in the library file. By using this library, we can easily read the DHT sensor. First create a DHT class object in the code.

```
int main(){
    DHT dht;            //create a DHT class object
```

And then in the "while" cycle, use chk = dht.readDHT11 (DHT11_Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk". And then use variable "sumCnt" to record number of reading times.

```
while(1){
        chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value.
Then determine whether data read is normal according to the return value.
        sumCnt++;           //counting number of reading times
        printf("The sumCnt is : %d \n",sumCnt);
        switch(chk){
            case DHTLIB_OK:         //if the return value is DHTLIB_OK, the data
is normal.
                printf("DHT11,OK! \n");
                break;
            case DHTLIB_ERROR_CHECKSUM:         //data check has errors
```

```
                printf("DHTLIB_ERROR_CHECKSUM! \n");
                break;
            case DHTLIB_ERROR_TIMEOUT:          //reading DHT times out
                printf("DHTLIB_ERROR_TIMEOUT! \n");
                break;
            case DHTLIB_INVALID_VALUE:          //other errors
                printf("DHTLIB_INVALID_VALUE! \n");
                break;
        }
```

Finally print the results:

```
printf("Humidity is %.2f %%, \t Temperature is %.2f
*C\n\n",dht.humidity,dht.temperature);
```

Library file "DHT.hpp" contains a DHT class and his public member functions int readDHT11 (int pin) is used to read sensor DHT11 and store the temperature and humiditydata read to member variables doublehumidity and temperature. The implementation method of the function is included in the file "DHT.cpp".

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

////read return flag of sensor
#define DHTLIB_OK                       0
#define DHTLIB_ERROR_CHECKSUM     -1
#define DHTLIB_ERROR_TIMEOUT      -2
#define DHTLIB_INVALID_VALUE      -999

#define DHTLIB_DHT11_WAKEUP       18
#define DHTLIB_DHT_WAKEUP         1

#define DHTLIB_TIMEOUT                100

class DHT{
    public:
        double humidity,temperature;      //use to store temperature and humidity
data read
        int readDHT11(int pin);       //read DHT11
    private:
        uint8_t bits[5];      //Buffer to receiver data
        int readSensor(int pin,int wakeupDelay);      //
```

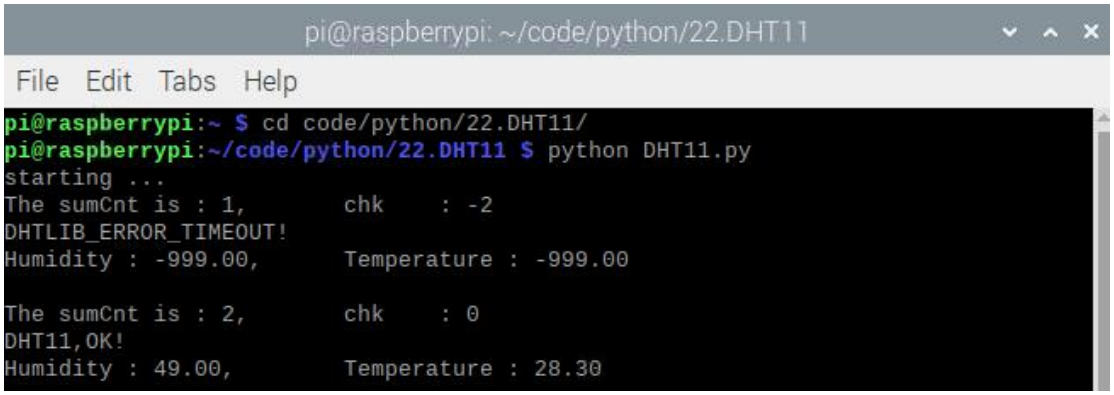
```
};
```

Python code

1. Use the "cd code / python / 22.DHT11 /" command to enter the directory of the temperature and humidity sensor.

2. Use "python DHT11.py"command to execute "DHT11.py" code.

After the program is executed, the terminal window will display the current reading temperature and humidity, as well as the temperature and humidity values, as shown below:



Press "Ctrl + c" to end the program. The following is the program code:

```
import RPi.GPIO as GPIO
import time
import DHT as DHT
DHTPin = 11         #define the pin of DHT11
def loop():
        dht = DHT.DHT(DHTPin)      #create a DHT class object
        sumCnt = 0                          #number of reading times
        while(True):
                sumCnt += 1                 #counting number of reading times
                chk = dht.readDHT11()         #read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
                print ("The sumCnt is : %d, \t chk        : %d"%(sumCnt,chk))
                if (chk is dht.DHTLIB_OK):           #read DHT11 and get a return value.
```

Then determine whether data read is normal according to the return value.

```
                print("DHT11,OK!")
            elif(chk is dht.DHTLIB_ERROR_CHECKSUM): #data check has errors
                print("DHTLIB_ERROR_CHECKSUM!!")
            elif(chk is dht.DHTLIB_ERROR_TIMEOUT):    #reading DHT times out
                print("DHTLIB_ERROR_TIMEOUT!")
            else:                       #other errors
                print("Other error!")

            print("Humidity : %.2f, \t Temperature : %.2f
\n"%(dht.humidity,dht.temperature))
            time.sleep(3)

if __name__ == '__main__':
    print ('starting ... ')
    try:
        loop()
    except KeyboardInterrupt:
        GPIO.cleanup()
        exit()
```

Code Interpretation

In this project code, we use a module "DHT.py", which provides a method to read the sensor DHT. It is located in the same directory as the program file "DHT11.py". By using this library, we can easily read DHT sensors.

‘DHT.py’: This is a Python module for reading temperature and humidity data from DHT sensors. Local functions and variables are described as follows:

Variable humidity: stores the humidity data read from the sensor;

Variable temperature: stores temperature data read from the sensor;

def readDHT11 (pin): read the temperature and humidity of the sensor DHT11 and return the value used to determine whether the data is normal.

The module "DHT.py" contains DHT classes. And the class function def readDHT11 (pin) is used to read the sensor DHT11 and store the read temperature and humidity data to the member variable humidity and temperature.

First create a DHT class object in the code.

```
dht = DHT.DHT(DHTPin)      #create a DHT class object
```

And then in the "while" cycle, use “chk = dht.readDHT11 (DHT11Pin)” to read the DHT11, and determine whether the data read is normal according to the return value "chk". And then use variable “sumCnt” to record number of reading times.

```
while(True):
        sumCnt += 1                #counting number of reading times
        chk = dht.readDHT11()        #read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
        print ("The sumCnt is : %d, \t chk      : %d"%(sumCnt,chk))
        if (chk is dht.DHTLIB_OK):          #read DHT11 and get a return value.
Then determine whether data read is normal according to the return value.
            print("DHT11,OK!")
        elif(chk is dht.DHTLIB_ERROR_CHECKSUM): #data check has errors
            print("DHTLIB_ERROR_CHECKSUM!!")
        elif(chk is dht.DHTLIB_ERROR_TIMEOUT):    #reading DHT times out
            print("DHTLIB_ERROR_TIMEOUT!")
        else:                       #other errors
            print("Other error!")
```

Finally print the results:

```
print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
```

Lesson 19 Ultrasonic Ranging

Overview

In this lesson you will learn how to use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Parts Required

1 x Raspberry Pi Development Board

1 x Ultrasonic Sensor

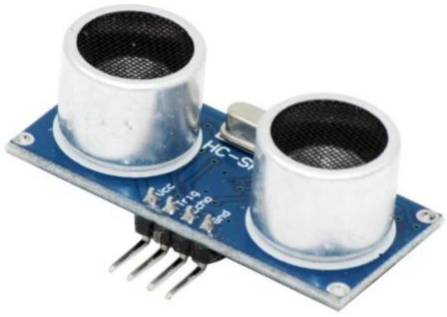
4 x Double Male Jumper Wires

1 x Breadboard

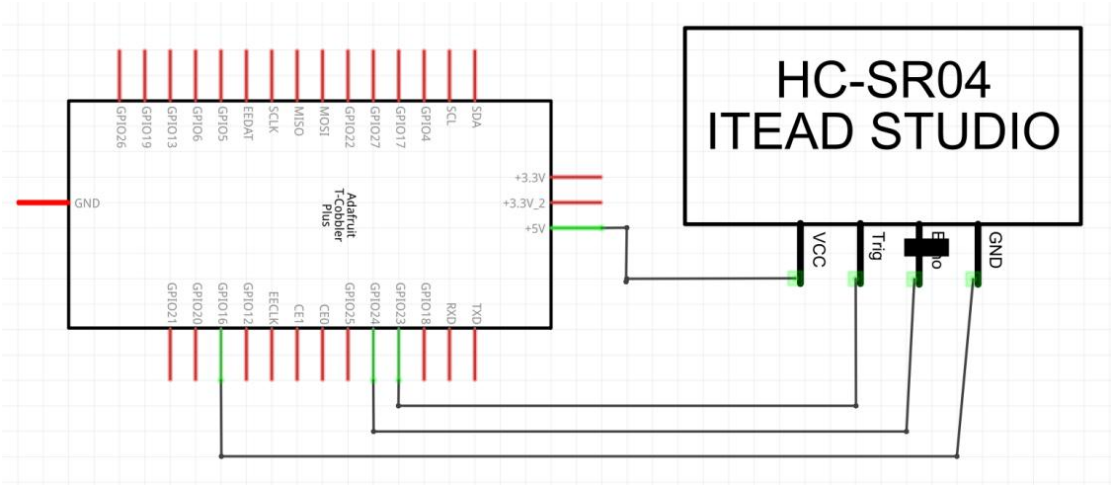
Product Introduction

Ultrasonic ranging

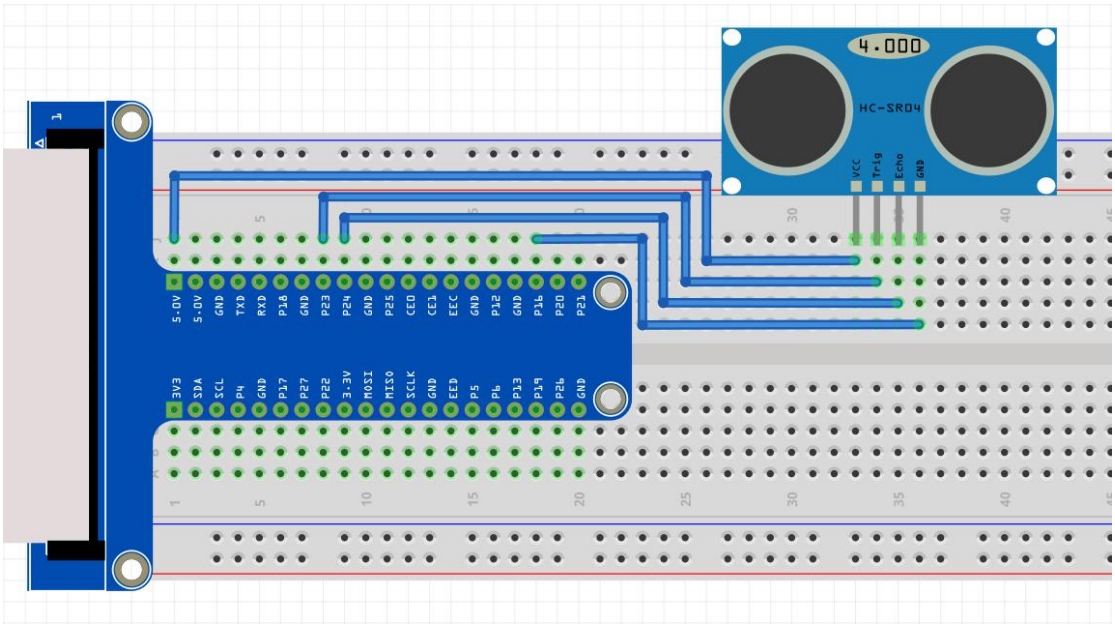
Ultrasonic ranging module use the principle that ultrasonic will reflect when it encounters obstacles. Start counting the time when ultrasonic is transmitted. And when ultrasonic encounters an obstacle, it will reflect back. The counting will end after ultrasonic is received, and the time difference is the total time of ultrasonic from transmitting to receiving.Ultrasonic module integrates a transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into sound waves (mechanical energy) and the function of the receiver is opposite.



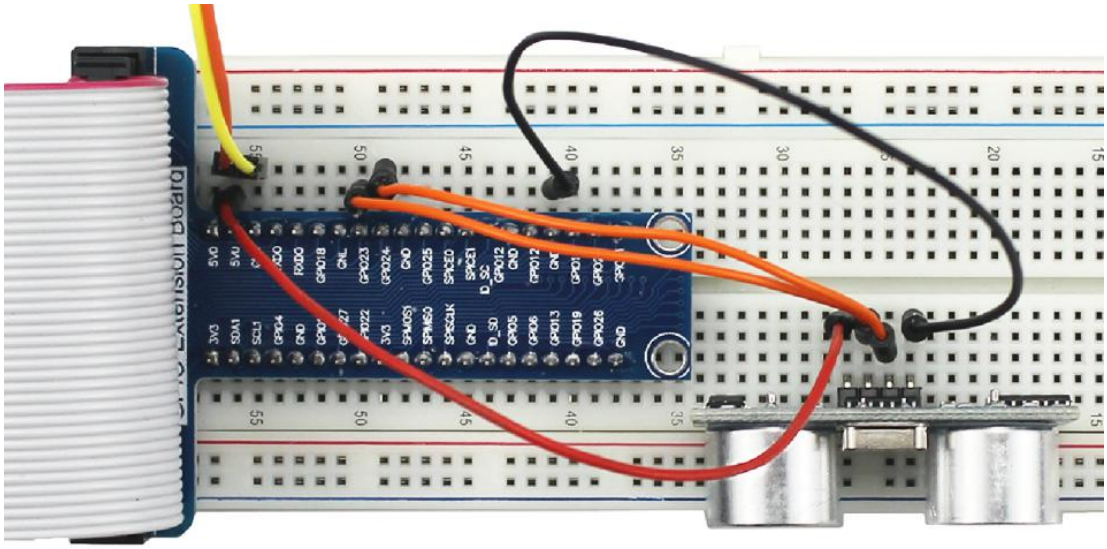
Connection Diagram



Wiring Diagram



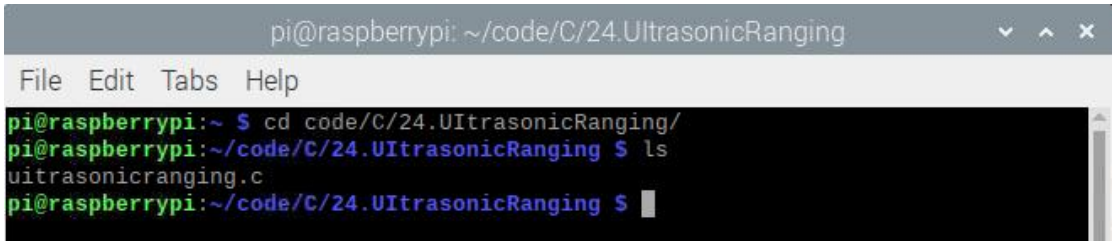
Example Figure



C code

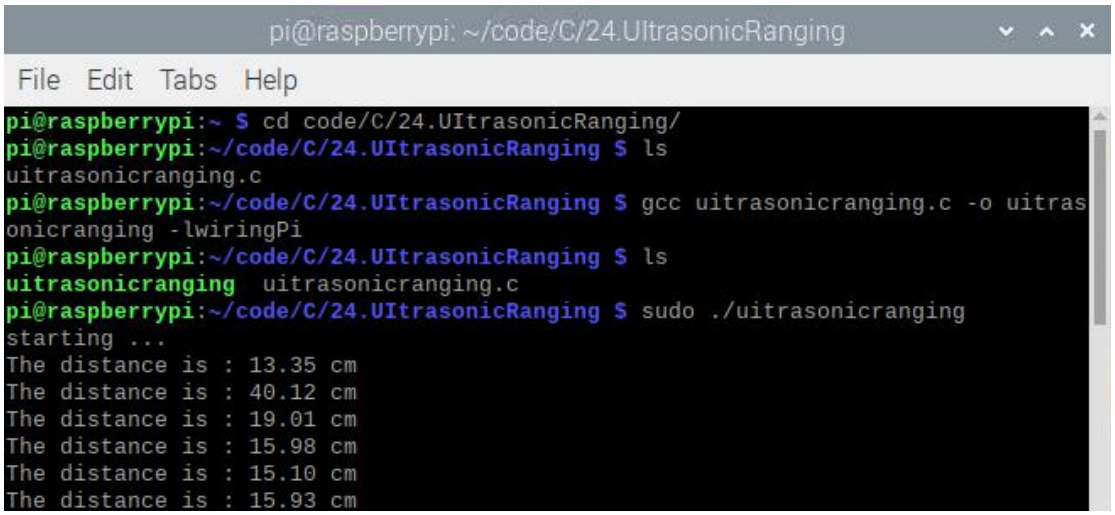
Open terminal and enter "cd code / C / 24.UItrasonicRanging /"command to enter "UItrasonicRanging" code directory;

Enter the "ls" command to view the file "uitrasonicranging.c" in the directory;



Enter "gcc uitrasonicranging.c -o uitrasonicranging -lwiringPi"command to generate "uitrasonicranging .c" executable file "uitrasonicranging", enter "ls" command to view; enter "sudo ./uitrasonicranging" command to run the code.

The results are shown below :



Press "Ctrl + c" to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define trigPin 4
```

```
#define echoPin 5
#define MAX_DISTANCE 220              // define the maximum measured distance
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the
maximum measured distance
//function pulseIn: obtain pulse time of a pin
int pulseIn(int pin, int level, int timeout);
float getSonar(){      // get the measurement results of ultrasonic module,with unit: cm
      long pingTime;
      float distance;
      digitalWrite(trigPin,HIGH); //trigPin send 10us high level
      delayMicroseconds(10);
      digitalWrite(trigPin,LOW);
      pingTime = pulseIn(echoPin,HIGH,timeOut);      //read plus time of echoPin
      distance = (float)pingTime * 340.0 / 2.0 / 10000.0; // the sound speed is
340m/s,and calculate distance
      return distance;
}

int main(){
      printf("starting ... \n");
      if(wiringPiSetup() == -1){ //when initialize wiring failed,print messageto screen
            printf("failed !");
            return 1;
      }
      float distance = 0;
      pinMode(trigPin,OUTPUT);
      pinMode(echoPin,INPUT);
      while(1){
            distance = getSonar();
            printf("The distance is : %.2f cm\n",distance);
            delay(1000);
      }
      return 1;
}

int pulseIn(int pin, int level, int timeout)
{
    struct timeval tn, t0, t1;
    long micros;
    gettimeofday(&t0, NULL);
    micros = 0;
    while (digitalRead(pin) != level)
    {
        gettimeofday(&tn, NULL);
```

```
        if (tn.tv_sec > t0.tv_sec) micros = 1000000L; else micros = 0;
        micros += (tn.tv_usec - t0.tv_usec);
        if (micros > timeout) return 0;
    }
    gettimeofday(&t1, NULL);
    while (digitalRead(pin) == level)
    {
        gettimeofday(&tn, NULL);
        if (tn.tv_sec > t0.tv_sec) micros = 1000000L; else micros = 0;
        micros = micros + (tn.tv_usec - t0.tv_usec);
        if (micros > timeout) return 0;
    }
    if (tn.tv_sec > t1.tv_sec) micros = 1000000L; else micros = 0;
    micros = micros + (tn.tv_usec - t1.tv_usec);
    return micros;
}
```

Code Interpretation

First, define the pins and the maximum measurement distance.

```
#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220
```

If the module does not return high level, we can not wait forever. So we need to calculate the lasting time over maximum distance, that is, time Out. timOut= 2*MAX_DISTANCE/100/340*1000000. The constant part behind is approximately equal to 58.8.

```
#define timeOut MAX_DISTANCE*60
```

Subfunction “getSonar ()” function isused to start the ultrasonic module for a measurement, and return the measured distance with unit cm. In this function, first let “trigPin” send 10us high level to start the ultrasonic module. Then use “pulseIn ()” to read ultrasonic module and return the duration of high level. Finally calculate the measured distance according to the time.

```
float getSonar(){      // get the measurement results of ultrasonic module,with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //trigPin send 10us high level
    delayMicroseconds(10);
```

```
        digitalWrite(trigPin,LOW);
        pingTime = pulseIn(echoPin,HIGH,timeOut);      //read plus time of echoPin
        distance = (float)pingTime * 340.0 / 2.0 / 10000.0; // the sound speed is
340m/s,and calculate distance
        return distance;
}
```

Finally, in the while loop of main function, get the measurement distance and print it out constantly.

```
while(1){
            distance = getSonar();
            printf("The distance is : %.2f cm\n",distance);
            delay(1000);
}
```

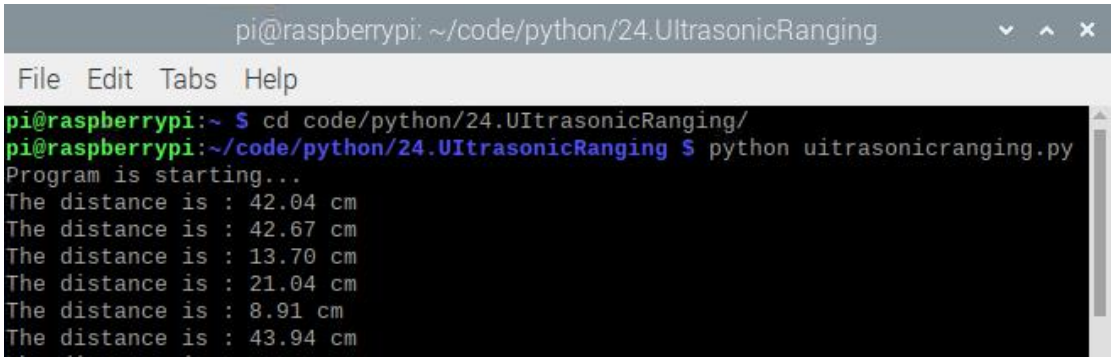
About function “pulseIn()”:Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

```
int pulseIn(int pin, int level, int timeout);
```

Python code

1. Use the "cd code / python / 24.UItrasonicRanging /" command to enter the "UItrasonicRanging" directory.

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



Press "Ctrl + c" to end the program. The following is the program code:

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

```
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220
timeOut = MAX_DISTANCE*60
def pulseIn(pin,level,timeOut):
    t0 = time.time()
    while(GPIO.input(pin) != level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    t0 = time.time()
    while(GPIO.input(pin) == level):
        if((time.time() - t0) > timeOut*0.000001):
            return 0;
    pulseTime = (time.time() - t0)*1000000
    return pulseTime
def getSonar():
    GPIO.output(trigPin,GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigPin,GPIO.LOW)
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)
    distance = pingTime * 340.0 / 2.0 / 10000.0
    return distance
def setup():
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(trigPin, GPIO.OUT)
    GPIO.setup(echoPin, GPIO.IN)
def loop():
    while(True):
        distance = getSonar()
        print ("The distance is : %.2f cm"%(distance))
        time.sleep(1)
if __name__ == '__main__':
    setup()
    try:
        loop()
    except KeyboardInterrupt:
    GPIO.cleanup()
```

Code Interpretation

```
import RPi.GPIO as GPIO
import time
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220
```

First, define the pins and the maximum measurement distance.

```
timeOut = MAX_DISTANCE*60
```

If the module does not return high level, we can not wait forever. So we need to calculate the lasting time over maximum distance, that is, “timeOut(μ s)” . “timeOut= 2*MAX_DISTANCE/100/340*1000000”. Theconstant part behind is approximately equal to 58.8.

```
def getSonar():
    GPIO.output(trigPin,GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigPin,GPIO.LOW)
  pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)
    distance = pingTime * 340.0 / 2.0 / 10000.0
    return distance
```

Subfunction “getSonar ()” function is used to start the ultrasonic module for a measurement, and return the measured distance with unit cm. In this function, first let “trigPin” send 10us high level to start the ultrasonic module. Then use “pulseIn ()” to read ultrasonic module and return the duration of high level. Finally calculate the measured distance according to the time.