

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.....	4
Lesson 1 LED.....	14
Lesson 2 Button &LED.....	27
Lesson 3 Flowing Water Light.....	34
Lesson 4 Analog & PWM.....	41
Lesson 5 RGBLED.....	48
Lesson 6 Active Buzzer.....	59
Lesson 7 PassiveBuzzer.....	66
Lesson 8 AD/DA Converter.....	74
Lesson 9 Potentiometer & LED.....	83
Lesson 10 Potentiometer & RGBLED.....	91
Lesson 11 Photoresistor & LED.....	98
Lesson 12 Thermistor.....	105
Lesson 13 Joystick.....	112
Lesson 14 Relay & Motor.....	119
Lesson 15 Servo.....	127
Lesson 16 Stepping Motor.....	138
Lesson 17 74HC595 & LEDBar Graph.....	149
Lesson 18 74HC595 & 7-Segment Display.....	161
Lesson 19 74HC595&4-Digit 7-Segment Display.....	170
Lesson 20 74HC595 & LED Matrix.....	184
Lesson 21 LCD1602.....	198
Lesson 22 DHT11.....	212
Lesson 23 Matrix Keypad.....	222
Lesson 24 Ultrasonic Ranging.....	230
Lesson 25 Infrared Sensor.....	239

Lesson 26 MPU6050.....	247
Lesson 27 Sound Sensor.....	256
Lesson 28 RFID RC522.....	263
Lesson 29 4N35.....	287
Lesson 30 NE555.....	296
Lesson 31 Infrared Remote Control.....	304
Lesson 32 Buzzer Welding.....	317
Lesson 33 Running Water Welding.....	321

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:

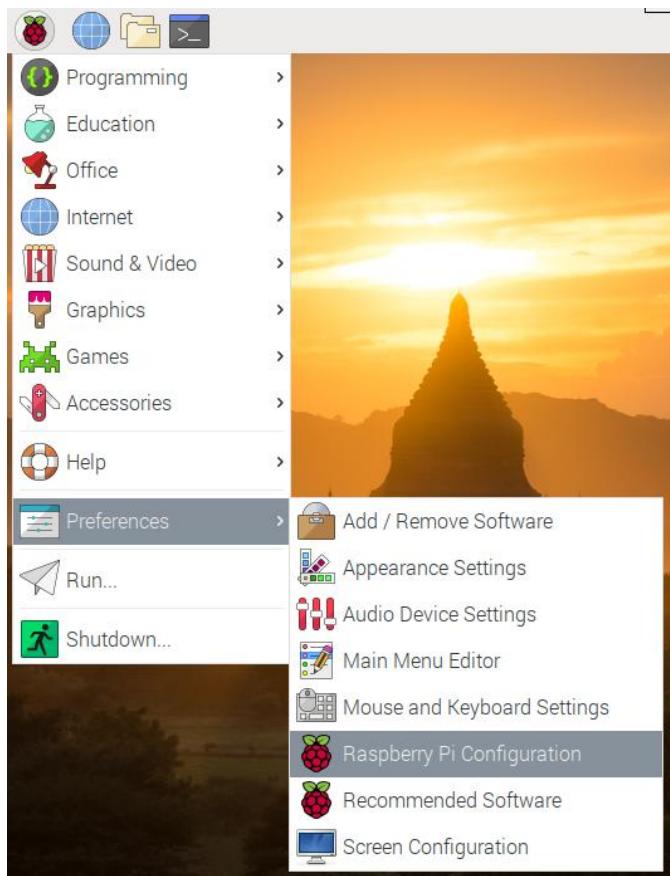
<https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up>

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

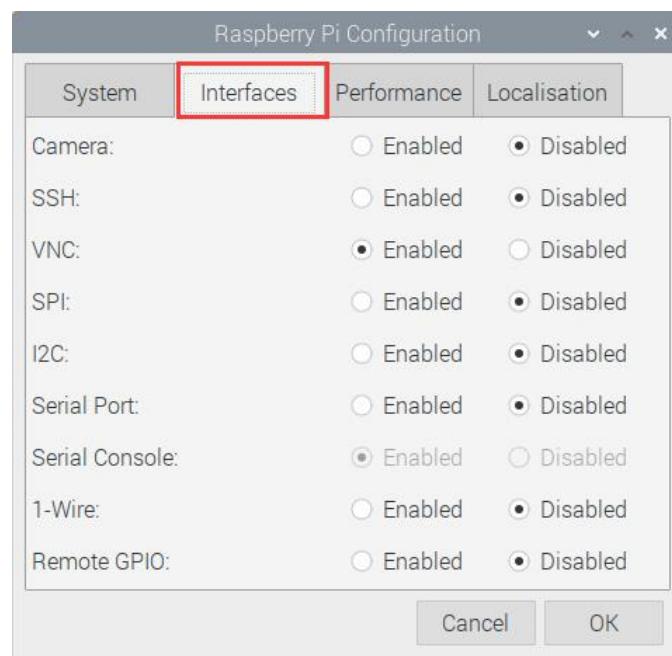
<https://projects.raspberrypi.org/en/projects/raspberry-pi-using>

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.

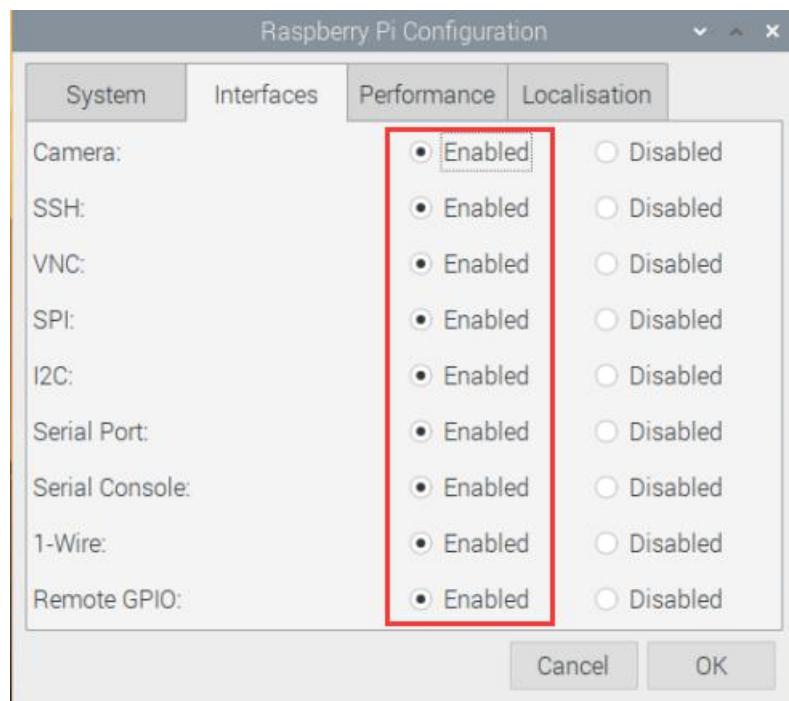


Select the ‘Interfaces’ option in the pop-up window as shown below:





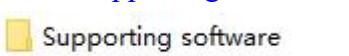
Enable all the interfaces as shown below:

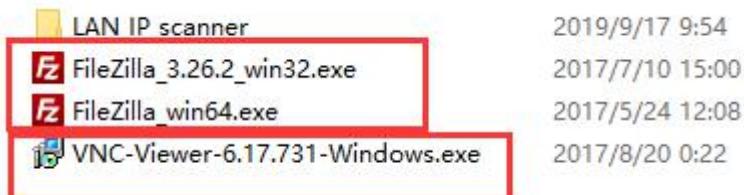


Click '**OK**' and restart the Raspberry Pi.

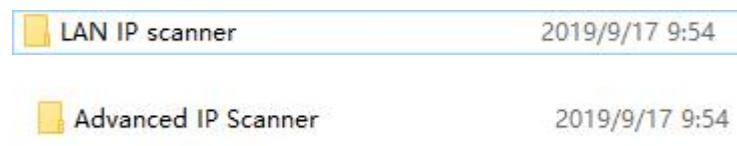
The Raspberry Pi configuration is completed, and then uses the computer to connect to the Raspberry Pi via Wi-Fi:

Steps

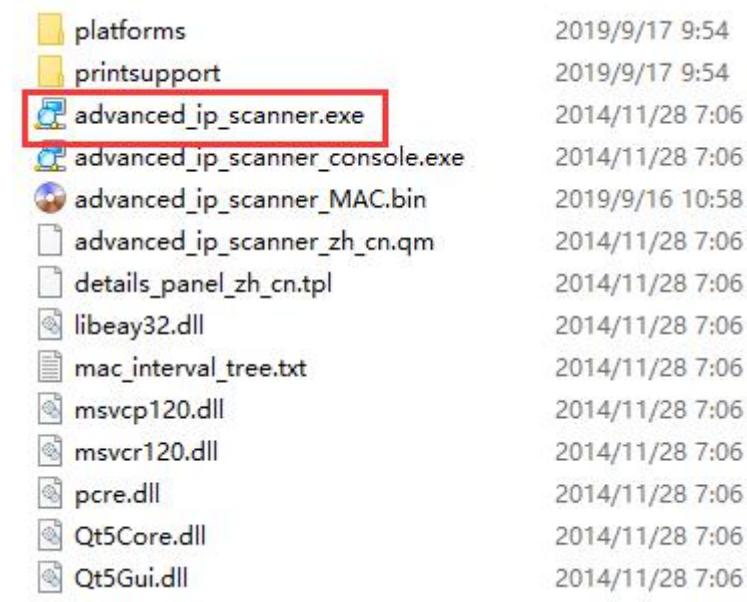
- 1) Open the '**Supporting software**' file in the companion file as shown below:
 2019/9/17 9:56
- 2) Install '**FileZilla**' and '**VNC**' software in the file:



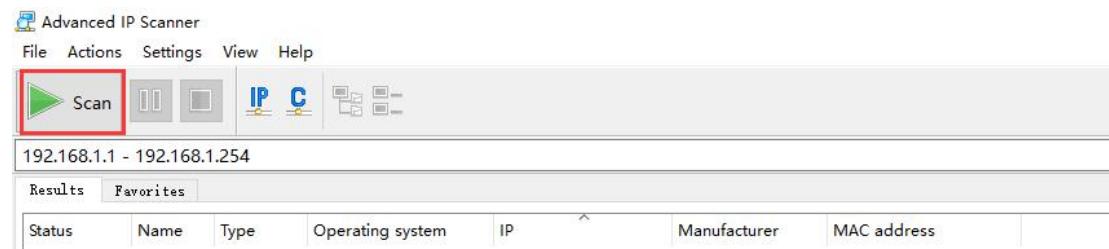
3) After installing the two software, open the [Advanced IP Scanner](#) file in the [LAN IP scanner](#) file as shown below:



4) Select '[advanced_ip_scanner.exe](#)' software as shown below:



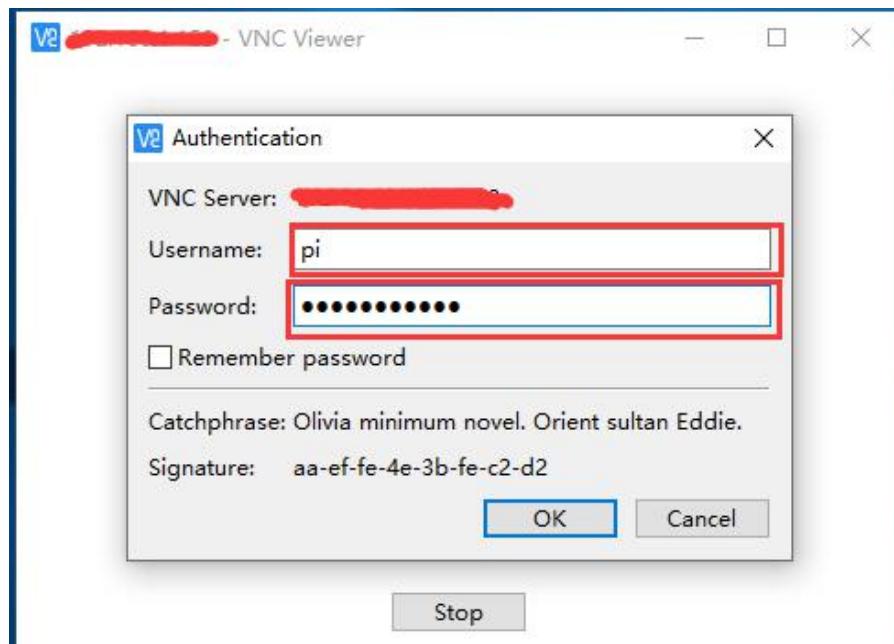
5) Click '[Scan](#)' to scan the Raspberry Pi IP (In the upper right corner of the desktop in the Raspberry Pi system, you can also view the IP address by placing the mouse on the wireless icon), as shown below:



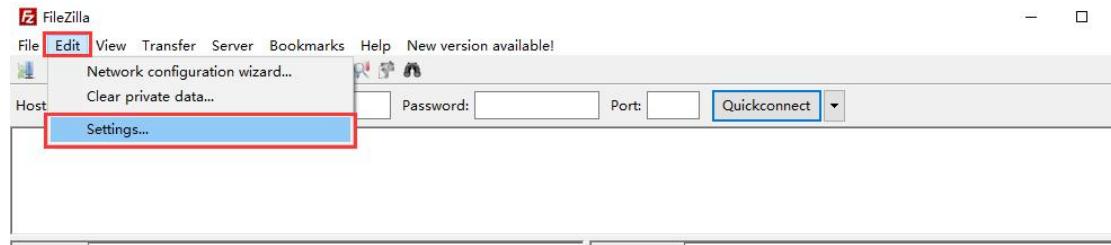
6) Record the IP address after scanning, as shown below:

DESKTOP...	[redacted]	50:7B:9D:A6:00:1F
001-PC	[redacted]	F4:4D:30:1C:3C:41
SKY-201...	[redacted]	1C:1B:0D:A5:D8:C8
192.168...	[redacted]	68:3E:34:E0:05:19
192.168...	[redacted]	DC:A6:32:0A:38:F5
192.168...	[redacted]	7C:03:AB:3F:82:13
192.168...	[redacted]	Raspberry Pi Four... B8:27:EB:CA:AF:DC
192.168...	[redacted]	88:40:3B:A2:5F:CF
192.168...	[redacted]	8C:25:05:94:85:DB
192.168...	[redacted]	14:D0:0D:92:C3:D7
192.168...	[redacted]	8C:16:45:44:63:79
PS2019E...	[redacted]	00:CF:E0:53:8F:A1
192.168...	[redacted]	60:EE:5C:76:C2:AB
PS2019P...	[redacted]	00:CF:E0:53:90:AD
192.168...	[redacted]	70:EF:00:29:54:70
192.168...	[redacted]	5C:C3:07:BF:87:AB
192.168...	[redacted]	DC:A6:32:15:5E:F4
-----		-----

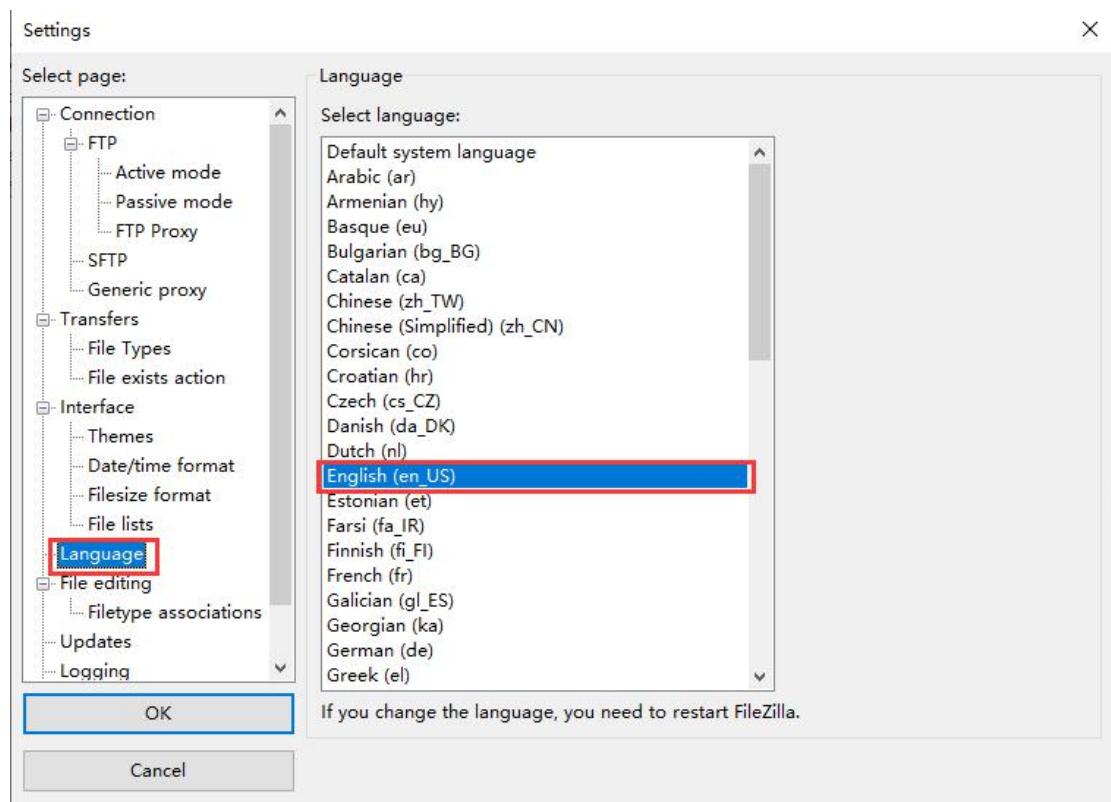
Open the VNC software after recording the IP address; enter the IP address in the input box, then press the Enter key, enter the user name and password in the pop-up window, click 'ok' to connect to the Raspberry Pi. As shown below:



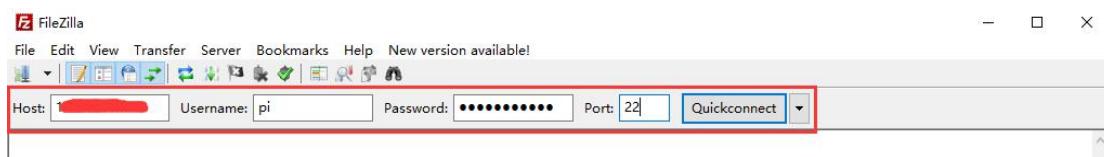
8) Use '[FileZilla](#)' software to achieve wireless transmission of files. Open the '[FileZilla](#)' software and set the language. As shown below, click the second button in the menu '[Edit](#)', then select the '[Settings](#)' button in the drop-down box.



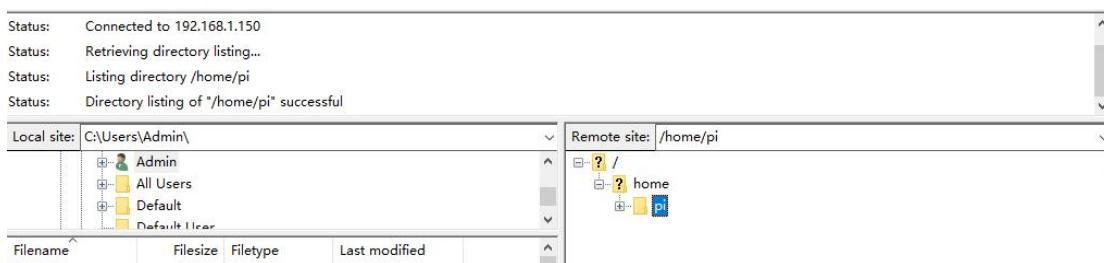
9) This tutorial takes English as an example. Click '[Language](#)', select '[English](#)', and then click '[ok](#)'. After setting, close the software and re-open it to set it successfully.



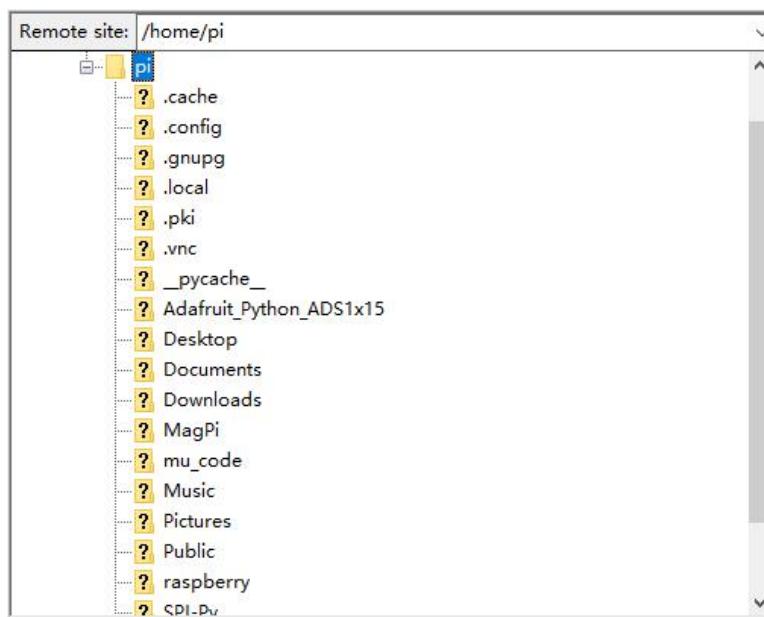
10) After the display language setting is completed, start connecting to the Raspberry Pi, enter the previously recorded IP address in the first input box, enter the user name in the second box, enter the password in the third box, and enter the port number default in the fourth box. 22, then click the '[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	GPIO.21	29 30	0V	-	-
22	6	GPIO.22	31 32	GPIO.26	12	26
23	13	GPIO.23	33 34	0V	-	-
24	19	GPIO.24	35 36	GPIO.27	16	27
25	26	GPIO.25	37 38	GPIO.28	20	28
0V			39 40	GPIO.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 3 Model B												
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM		
		3.3v			1 2			5v				
2	8	SDA.1	ALTO	1	3 4			5V				
3	9	SCL.1	ALTO	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	ALTO	0	19 20			0v				
9	13	MISO	ALTO	0	21 22	0	IN	GPIO. 6	6	25		
11	14	SCLK	ALTO	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 Model B												

If you are using Raspberry Pi 4B, there will 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 cannot 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
+-----+-----+-----+-----+-----+-----+-----+-----+
| 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 | 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 |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

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

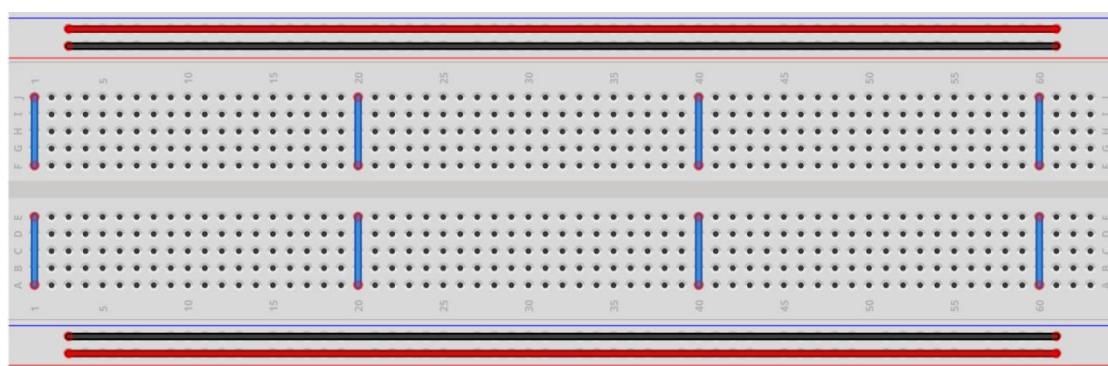
1 x breadboard

Product Introduction

BREADBOARD

A breadboard enables you to prototype circuits quickly without soldering.

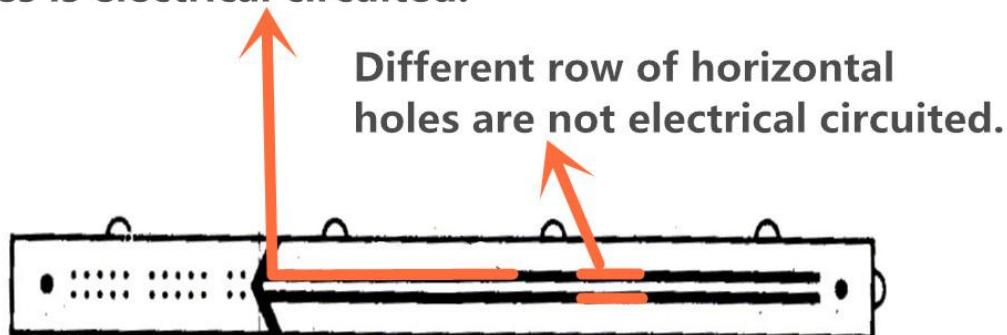
The specific principle is as follows:



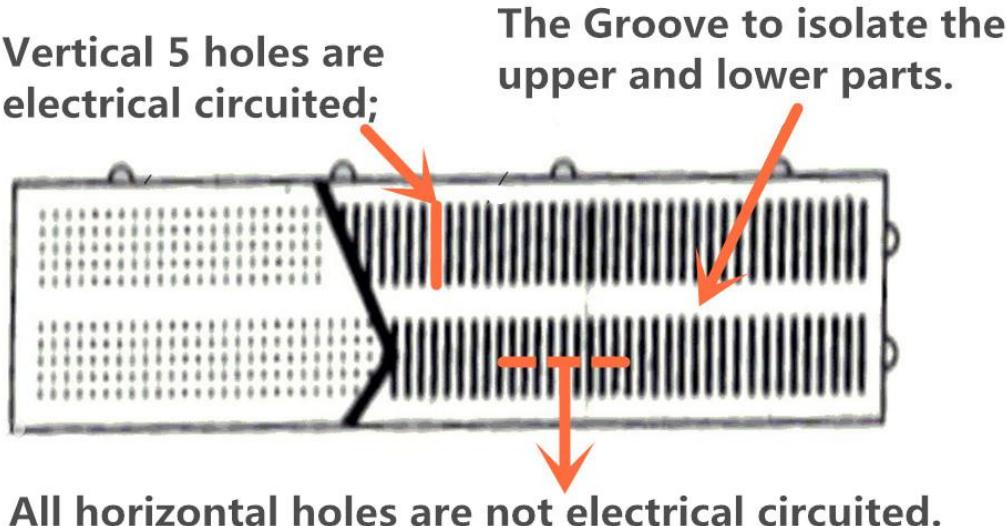


The inside of the breadboard is made up of metal strips. When inserted into the holes of the board, it can be in contact with the metal strip to achieve the purpose of conduction. Usually 5 holes are linked by a metal strip. There are two rows of vertical holes on both sides of the breadboard board,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

The same row of horizontal holes is electrical circuited.



Different row of horizontal holes are not electrical circuited.



Vertical 5 holes are electrical circuited;

The Groove to isolate the upper and lower parts.

All horizontal holes are not electrical circuited.

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 $k\Omega$ (1,000 Ω) and $M\Omega$ (1,000,000 Ω). These are called kilo-ohms and mega-ohms.

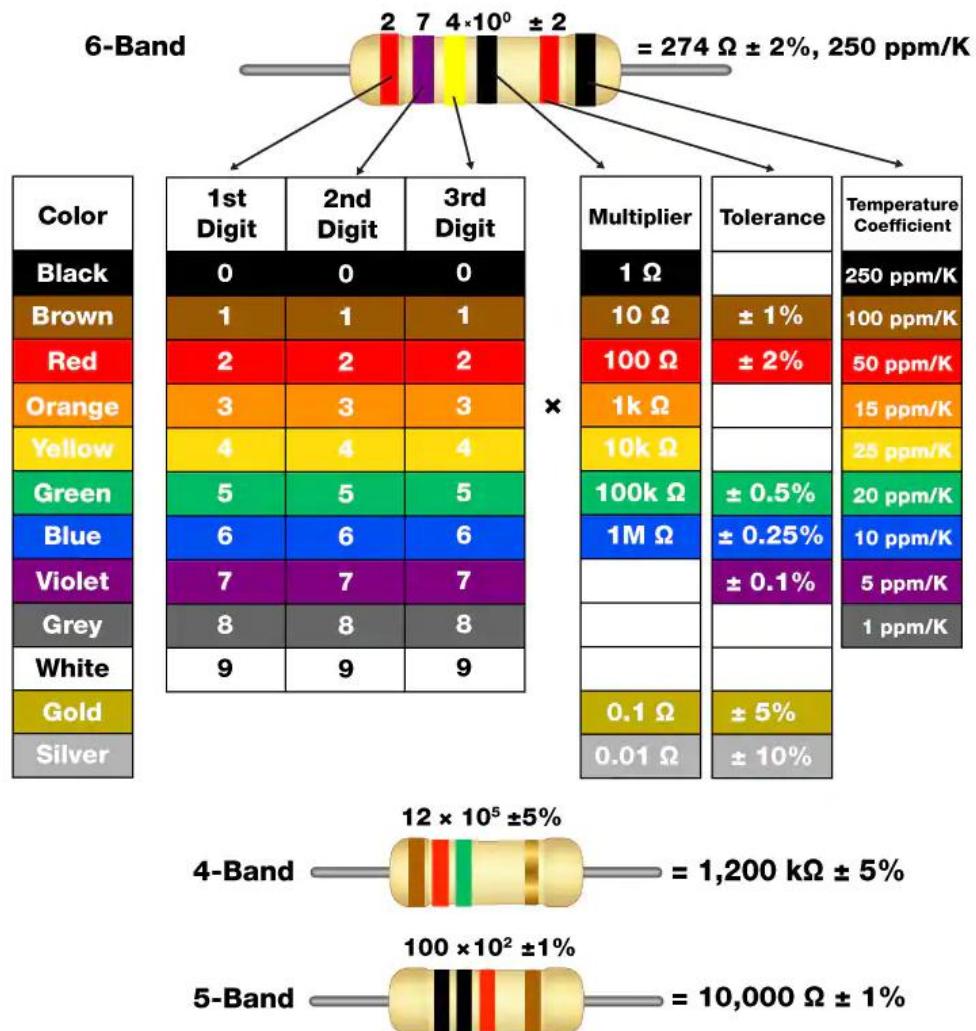
In this lesson, we mainly use a 220Ω resistor to protect the LED. You can also use a larger resistance value, but a larger resistance will make the LED's luminous intensity

weaker.

220 Ω , 1k Ω and 10k Ω .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.

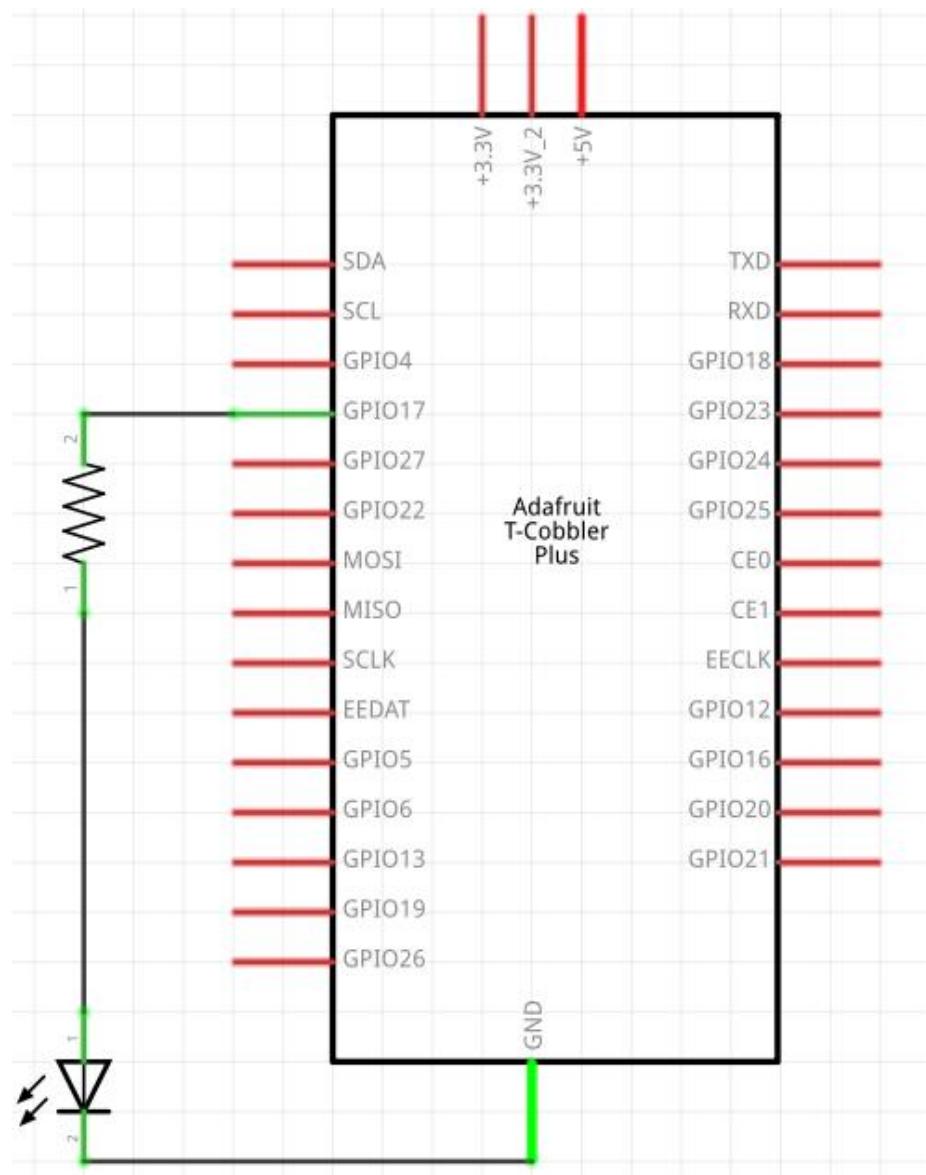
How to Read Resistor Color Codes



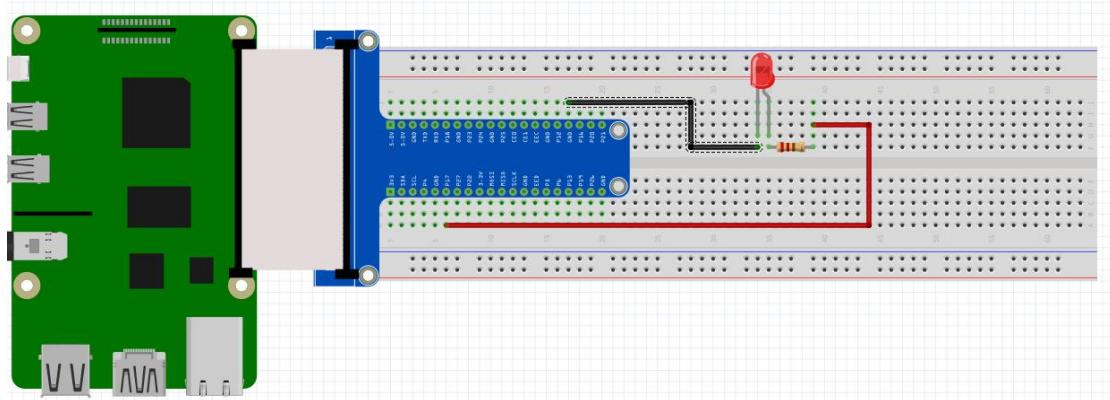
If you find that this discrimination method is too complicated, you can also use a multi-meter to measure the value of the resistance. The resistance is different from

LED, it has no positive and negative poles, so it can work normally by connecting it to the negative or positive pole of LED.

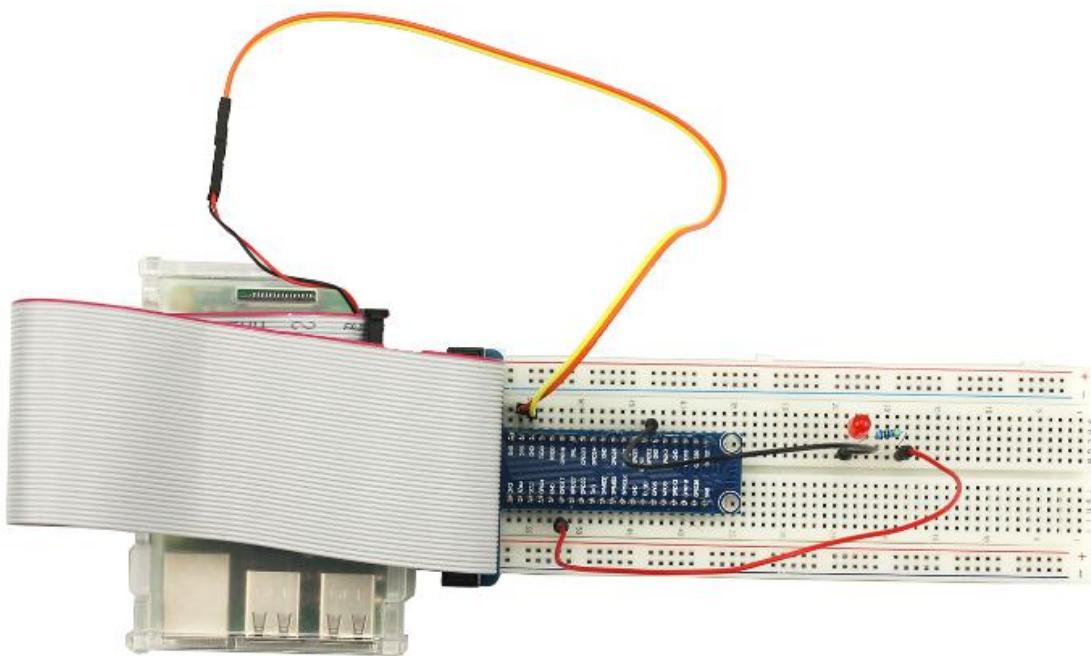
Connection Schematic



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.



```
pi@raspberrypi:~/code/C/1.LED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/1.LED/
pi@raspberrypi:~/code/C/1.LED $ ls
LED.c
pi@raspberrypi:~/code/C/1.LED $
```

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:



```
pi@raspberrypi:~/code/C/1.LED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/1.LED/
pi@raspberrypi:~/code/C/1.LED $ ls
LED.c
pi@raspberrypi:~/code/C/1.LED $ gcc LED.c -o LED -lwiringPi
pi@raspberrypi:~/code/C/1.LED $ ls
LED LED.c
pi@raspberrypi:~/code/C/1.LED $ sudo ./LED
wiringPi initialize successfully, GPIO 0(wiringPi pin)
led on...
...led off
led on...
pi@raspberrypi:~/code/C/1.LED $
```

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);
```

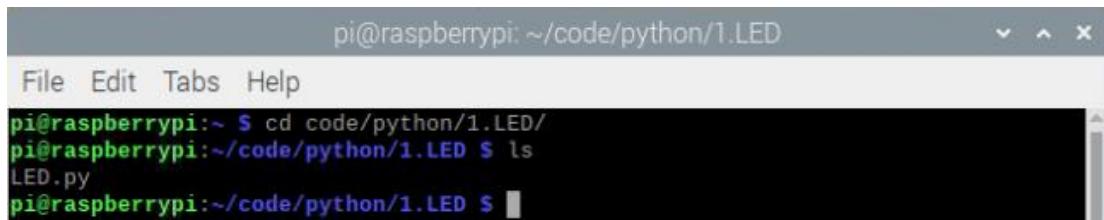
Write the value HIGH or LOW (1 or 0) to the pin preset to the output.

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.)

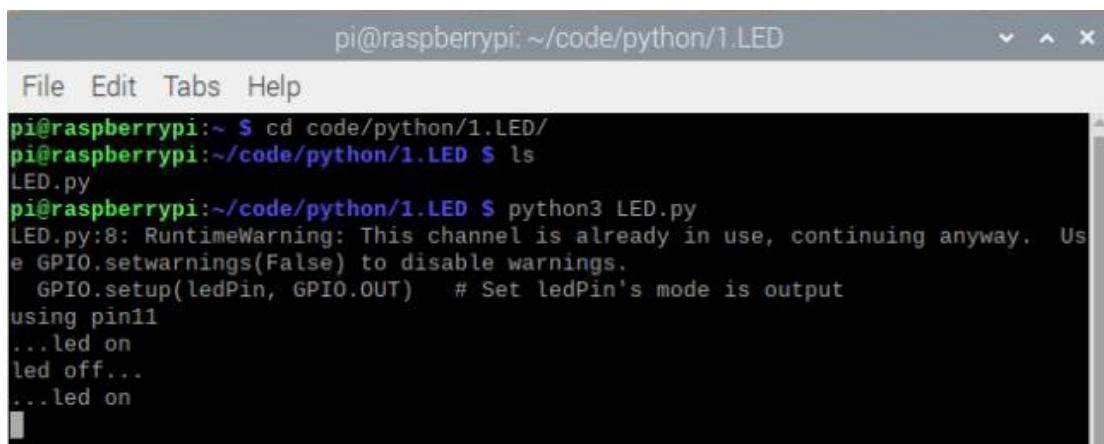
Open the terminal and enter `cd code/python/1.LED/` command to enter the LED.py code directory.

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



```
pi@raspberrypi: ~/code/python/1.LED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/1.LED/
pi@raspberrypi:~/code/python/1.LED $ ls
LED.py
pi@raspberrypi:~/code/python/1.LED $
```

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



```
pi@raspberrypi: ~/code/python/1.LED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/1.LED/
pi@raspberrypi:~/code/python/1.LED $ ls
LED.py
pi@raspberrypi:~/code/python/1.LED $ python3 LED.py
LED.py:8: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(ledPin, GPIO.OUT)  # Set ledPin's mode is output
using pin11
...led on
led off...
...led on
|
```

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.BARD)

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.BUILD)

 GPIO.setup(ledPin, GPIO.OUT)

 GPIO.output(ledPin, GPIO.LOW)

Print ('using pin%d'%ledPin)

In 'Setup()', 'GPIO.setmode(GPIO.BUILD)' 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

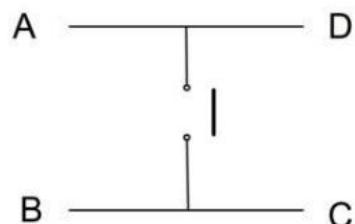
1 x Breadboard

1 x Button

Product Introduction

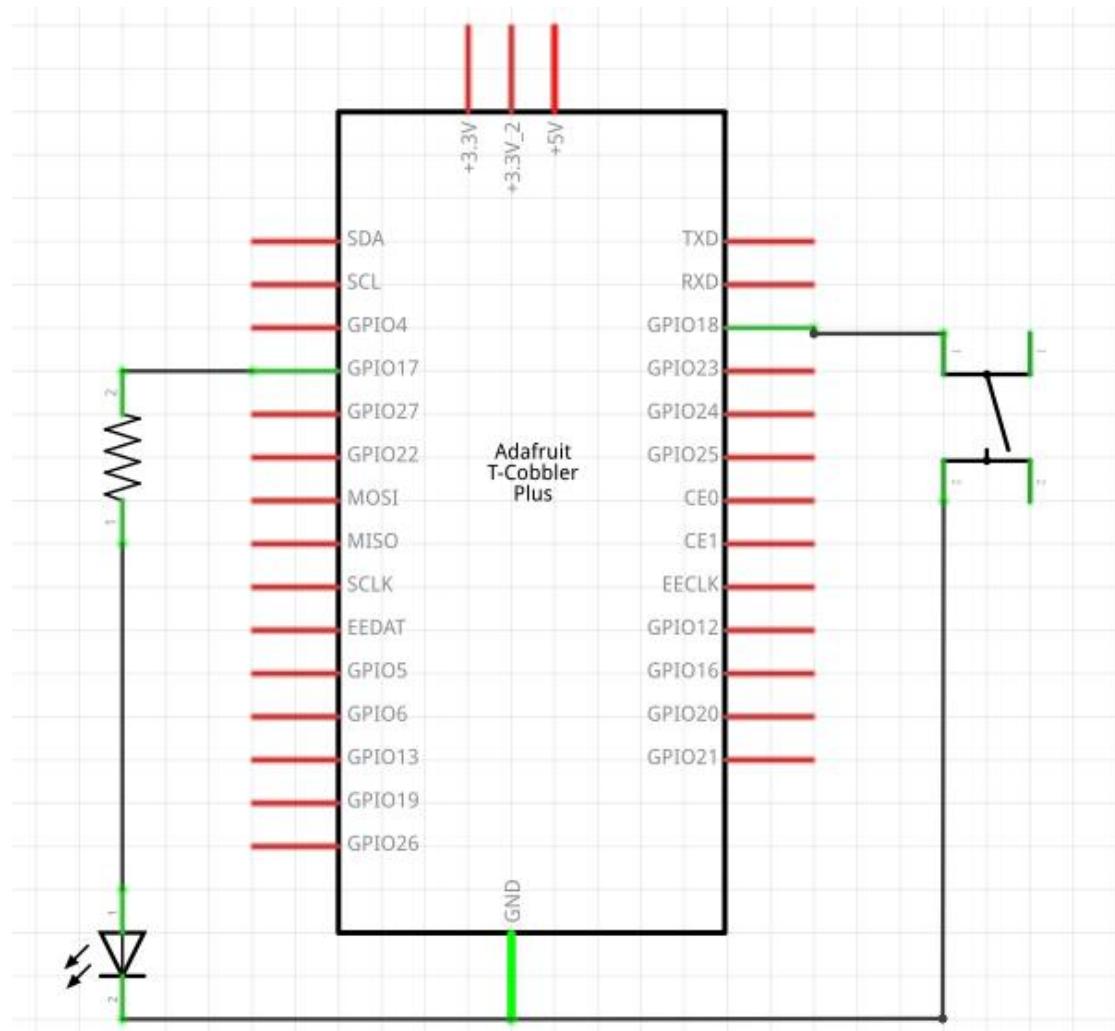
Push Button

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

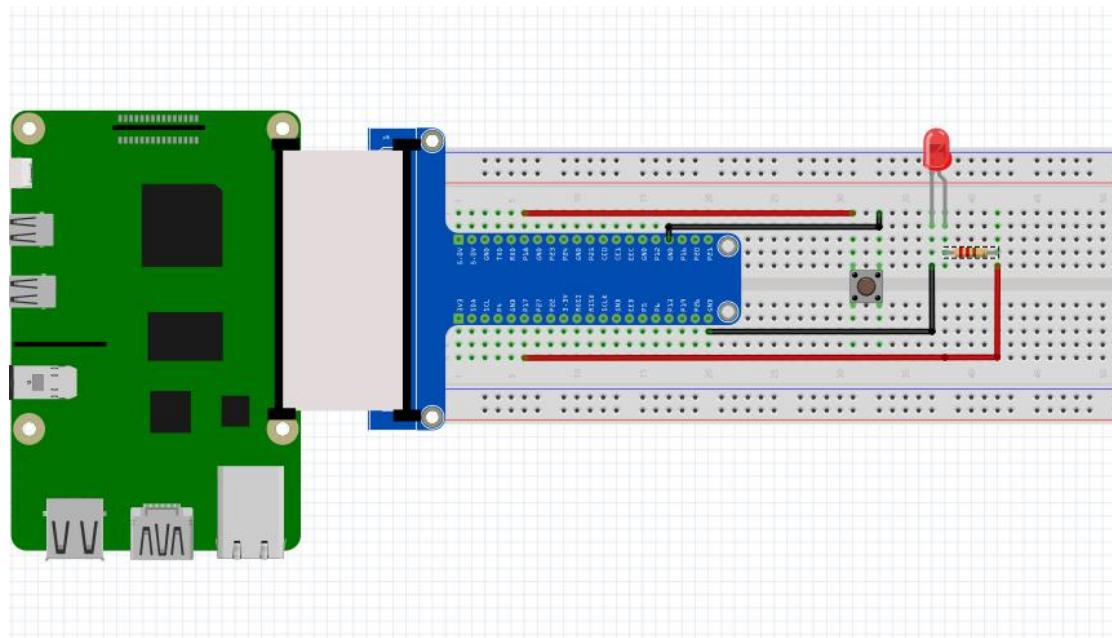


In fact only two pin connections are used. Inside the button, pins B and C are connected together, as are A and D.

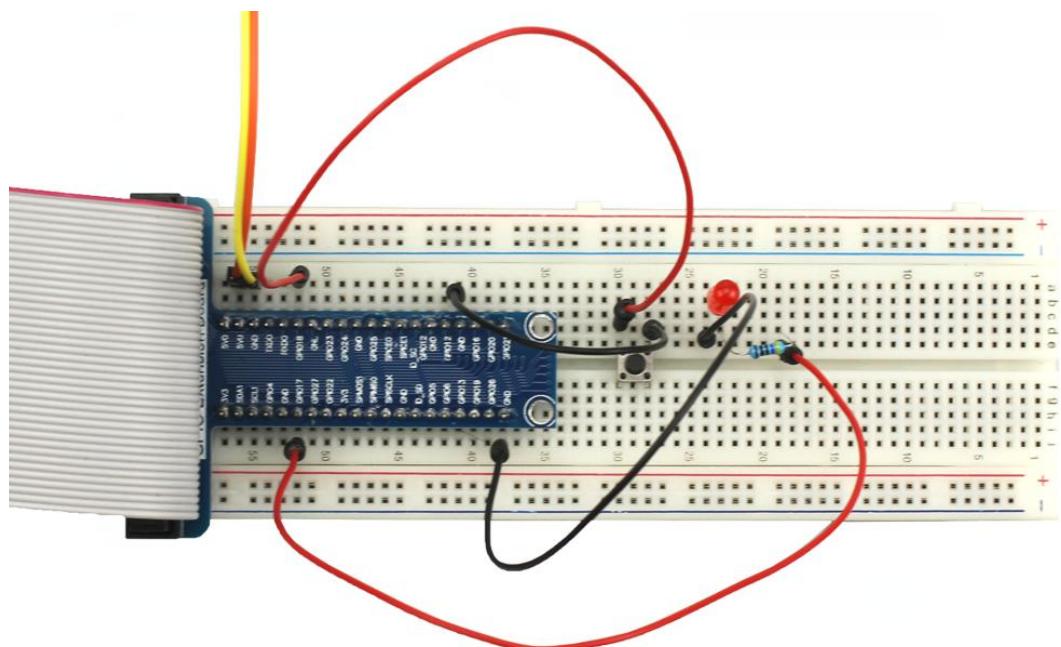
Connection Diagram



Wiring Diagram



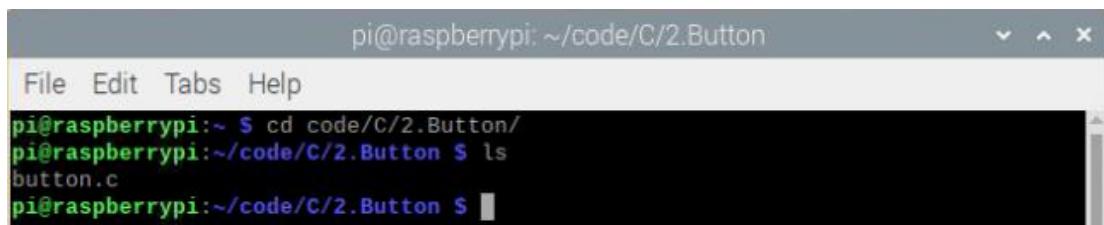
Example Figure



C code

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

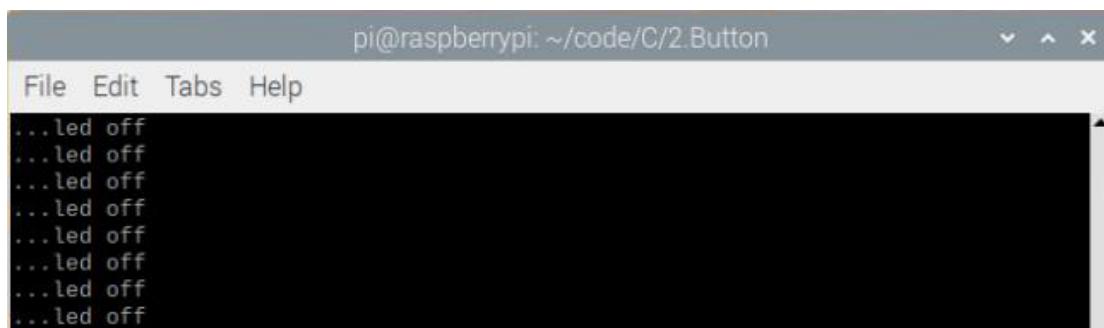
Enter the `ls` command to view the file button.c in the directory.



```
pi@raspberrypi:~/code/C/2.Button
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/2.Button/
pi@raspberrypi:~/code/C/2.Button $ ls
button.c
pi@raspberrypi:~/code/C/2.Button $
```

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:



```
pi@raspberrypi:~/code/C/2.Button
File Edit Tabs Help
...led off
```

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

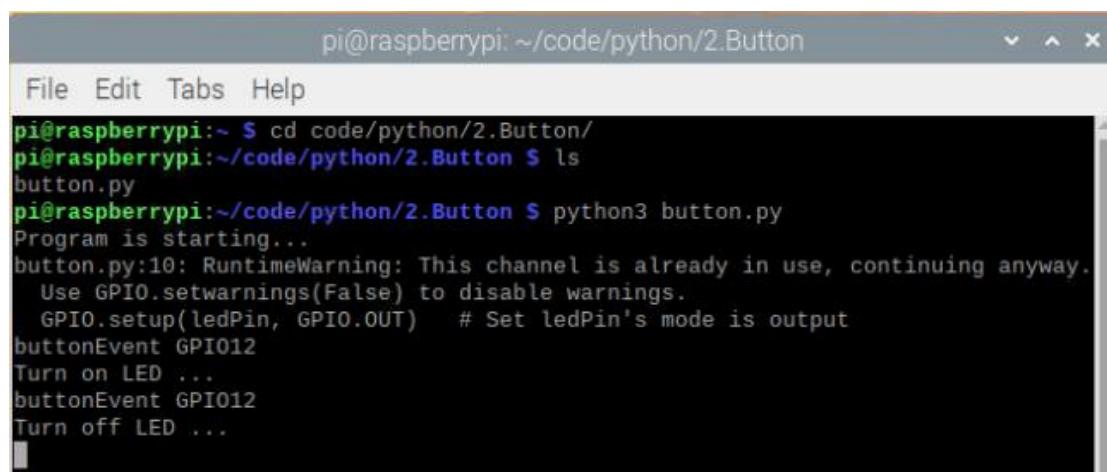
Open the terminal and enter the cd code/python/2.Button/ command to enter the code directory.

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



```
pi@raspberrypi:~/code/python/2.Button
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/2.Button/
pi@raspberrypi:~/code/python/2.Button $ ls
button.py
pi@raspberrypi:~/code/python/2.Button $
```

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



```
pi@raspberrypi:~/code/python/2.Button
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/2.Button/
pi@raspberrypi:~/code/python/2.Button $ ls
button.py
pi@raspberrypi:~/code/python/2.Button $ python3 button.py
Program is starting...
button.py:10: RuntimeWarning: This channel is already in use, continuing anyway.
  Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(ledPin, GPIO.OUT)    # Set ledPin's mode is output
buttonEvent GPIO12
Turn on LED ...
buttonEvent GPIO12
Turn off LED ...
|
```

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

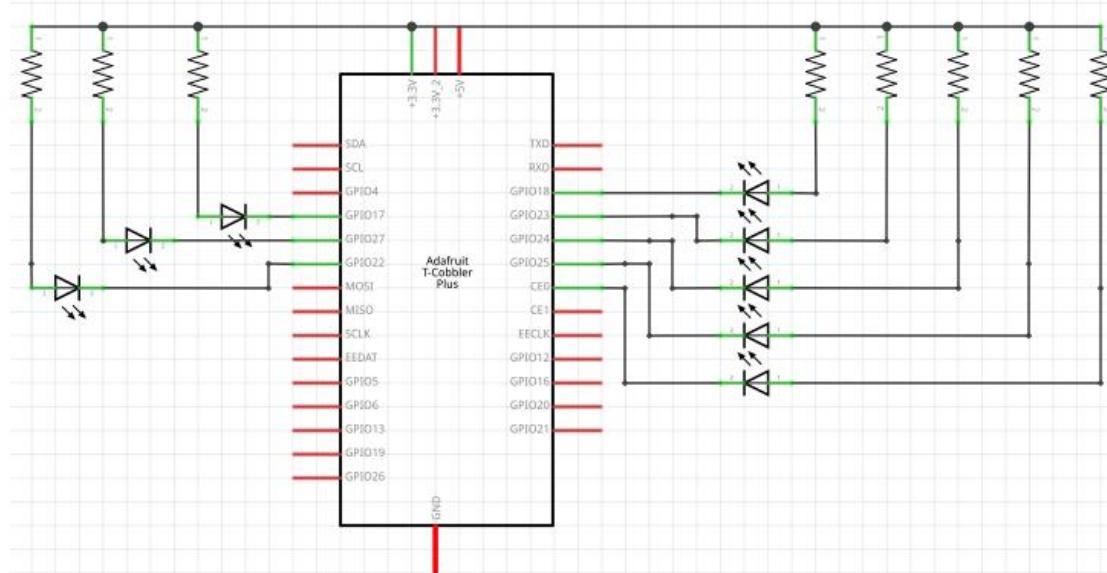
Overview

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

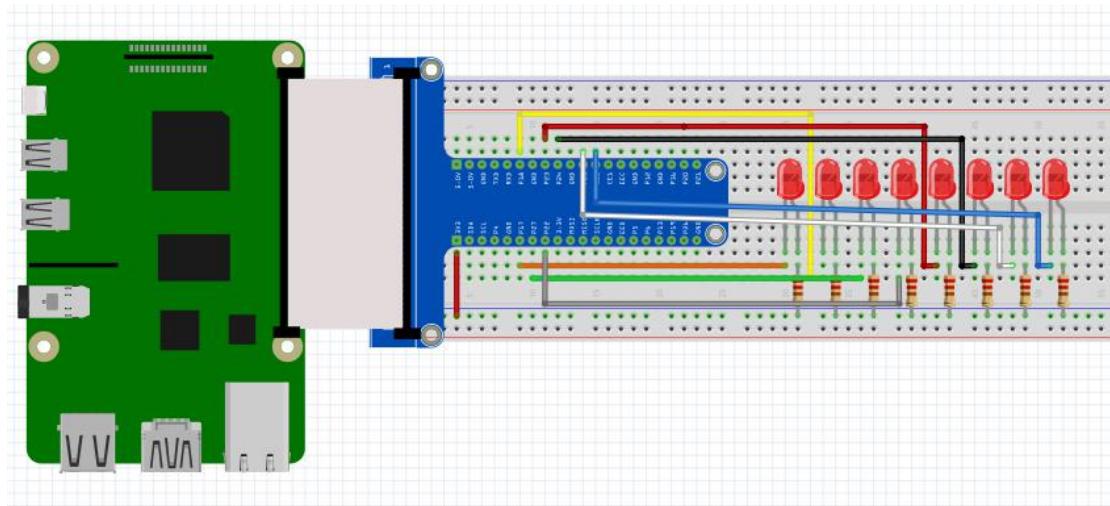
Parts Required

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

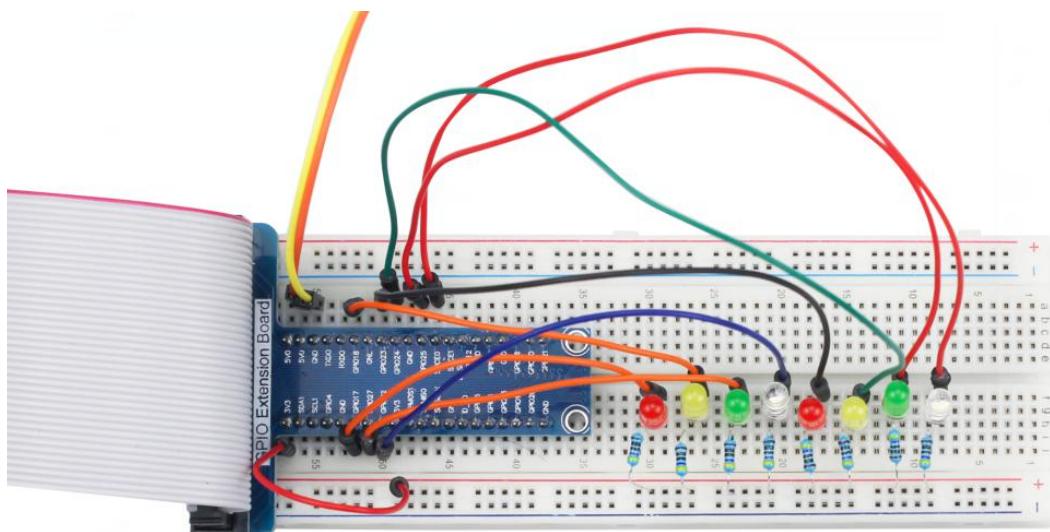
Connection diagram



Wiring diagram



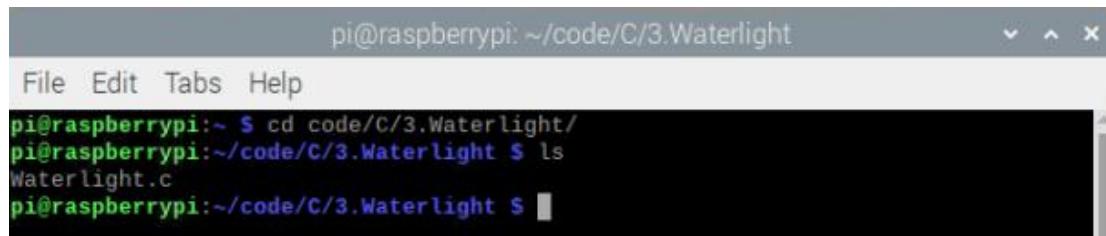
Example Figure



C code

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

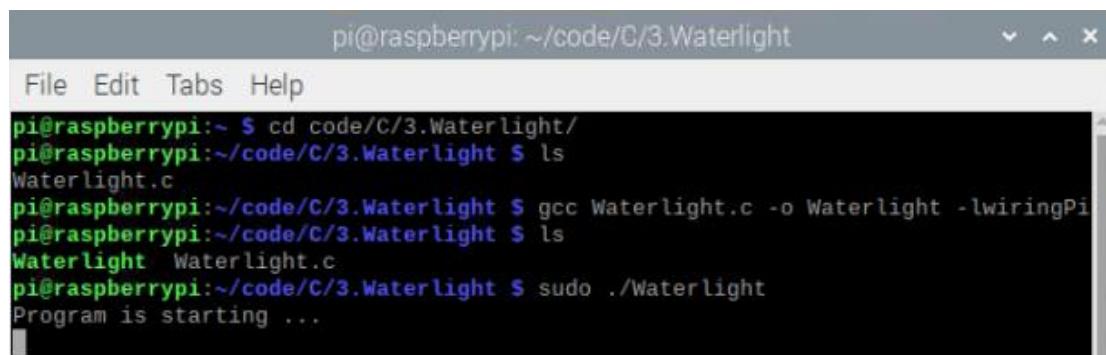
Enter the `ls` command to view the file in the directory Waterlight.c;



```
pi@raspberrypi:~/code/C/3.Waterlight
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/3.Waterlight/
pi@raspberrypi:~/code/C/3.Waterlight $ ls
Waterlight.c
pi@raspberrypi:~/code/C/3.Waterlight $
```

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:



```
pi@raspberrypi:~/code/C/3.Waterlight
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/3.Waterlight/
pi@raspberrypi:~/code/C/3.Waterlight $ ls
Waterlight.c
pi@raspberrypi:~/code/C/3.Waterlight $ gcc Waterlight.c -o Waterlight -lwiringPi
pi@raspberrypi:~/code/C/3.Waterlight $ ls
Waterlight Waterlight.c
pi@raspberrypi:~/code/C/3.Waterlight $ sudo ./Waterlight
Program is starting ...

```

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

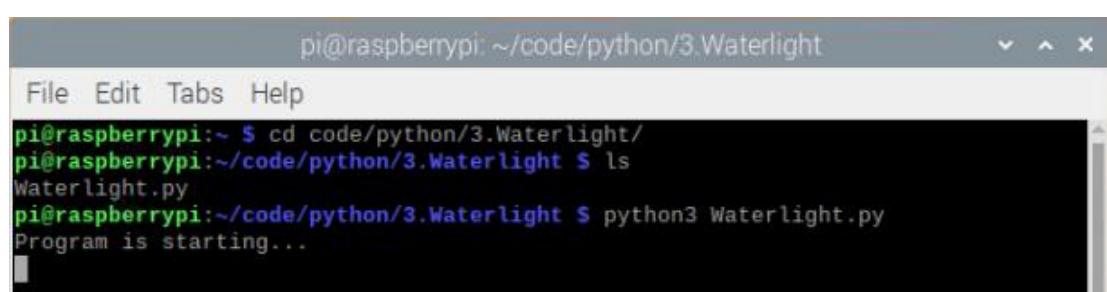
Open the terminal and use the `cd code/python/3.Waterlight/` command to enter the code directory;

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



```
pi@raspberrypi: ~code/python/3.Waterlight  
File Edit Tabs Help  
pi@raspberrypi:~ $ cd code/python/3.Waterlight/  
pi@raspberrypi:~/code/python/3.Waterlight $ ls  
Waterlight.py  
pi@raspberrypi:~/code/python/3.Waterlight $
```

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



```
pi@raspberrypi: ~code/python/3.Waterlight  
File Edit Tabs Help  
pi@raspberrypi:~ $ cd code/python/3.Waterlight/  
pi@raspberrypi:~/code/python/3.Waterlight $ ls  
Waterlight.py  
pi@raspberrypi:~/code/python/3.Waterlight $ python3 Waterlight.py  
Program is starting...  
|
```

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.BARD)

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

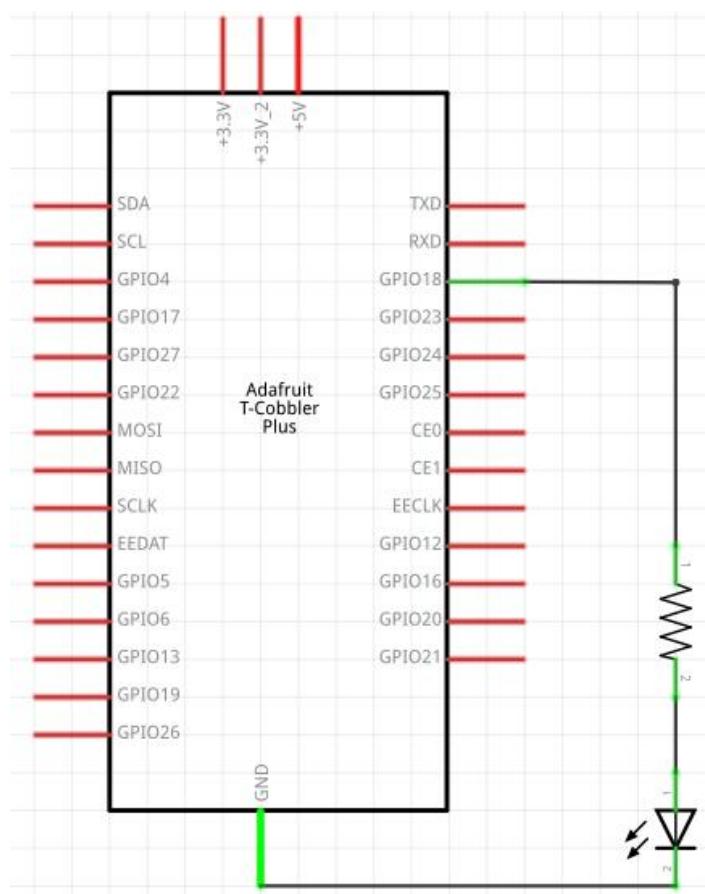
1 x LED

1 x 220 ohm Resistor

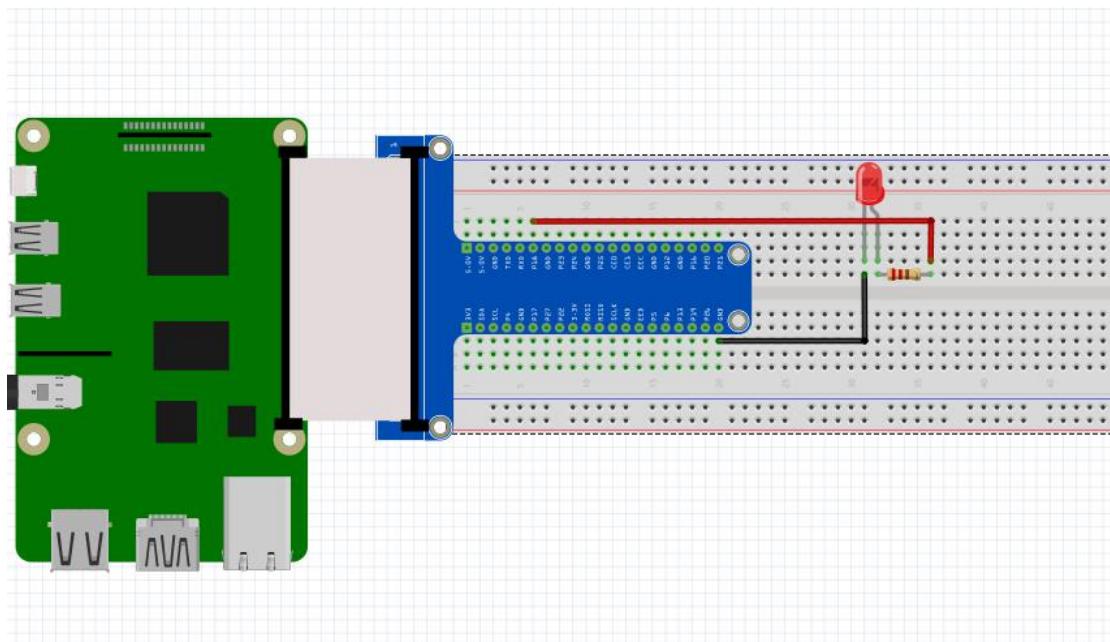
2 x Jumper Wires

1 x Breadboard

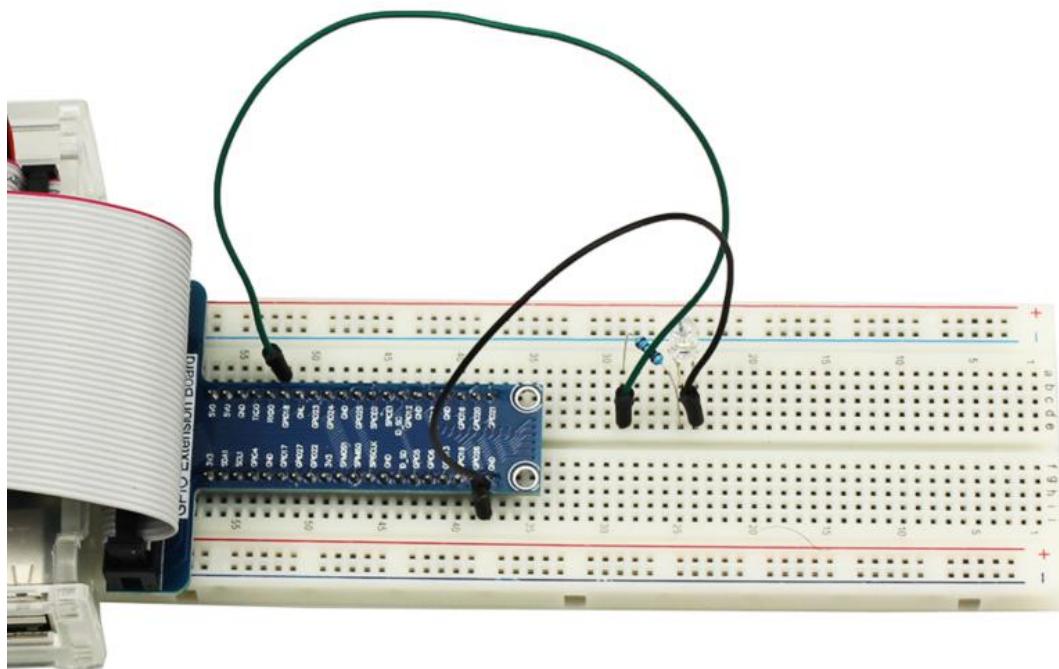
Connection Diagram



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.



```
pi@raspberrypi: ~/code/C/4.PWMLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/4.PWMLED/
pi@raspberrypi:~/code/C/4.PWMLED $ ls
pwmLED.c
pi@raspberrypi:~/code/C/4.PWMLED $
```

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:



```
pi@raspberrypi: ~/code/C/4.PWMLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/4.PWMLED/
pi@raspberrypi:~/code/C/4.PWMLED $ ls
pwmLED.c
pi@raspberrypi:~/code/C/4.PWMLED $ gcc pwmLED.c -o pwm -lwiringPi
pi@raspberrypi:~/code/C/4.PWMLED $ ls
pwm  pwmLED.c
pi@raspberrypi:~/code/C/4.PWMLED $ sudo ./pwm
pi@raspberrypi:~/code/C/4.PWMLED $
```

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.

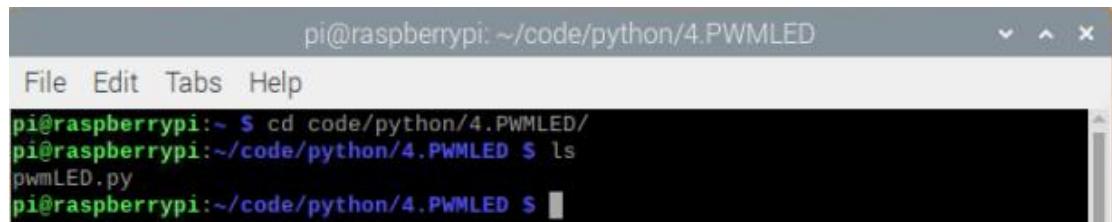
```
void softPwmWrite (int pin, int value) ;
```

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.



```
pi@raspberrypi: ~/code/python/4.PWMLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/4.PWMLED/
pi@raspberrypi:~/code/python/4.PWMLED $ ls
pwmLED.py
pi@raspberrypi:~/code/python/4.PWMLED $
```

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



```
pi@raspberrypi: ~/code/python/4.PWMLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/4.PWMLED/
pi@raspberrypi:~/code/python/4.PWMLED $ ls
pwmLED.py
pi@raspberrypi:~/code/python/4.PWMLED $ python3 pwmLED.py
|
```

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

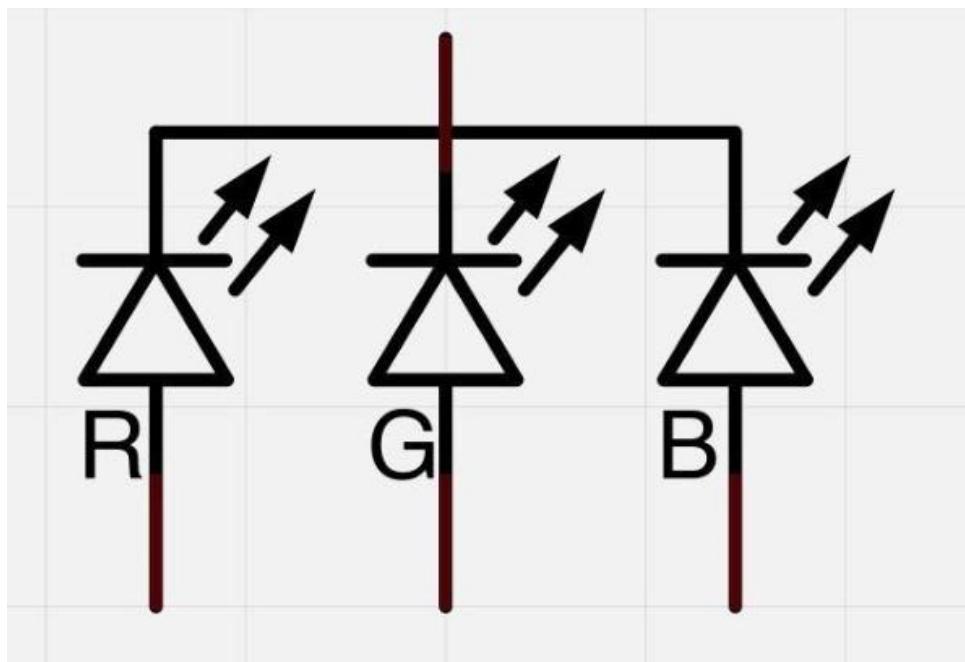
The RGB LED looks like a normal LED from the surface, except that the RGB LED has four pins, and the regular LED only has two pins. In fact, the RGB LED is internally packaged with three LEDs, one red, one green, and one blue.

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 “~”.

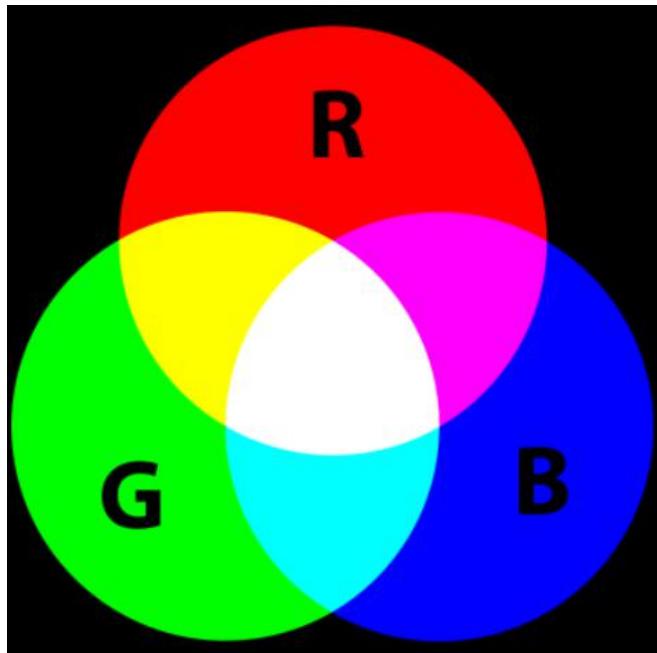


This pin does not require a resistor to connect directly to GND. However, the other three pins require resistors to prevent excessive current from flowing. The other three pins also become positive pins. The left side of the CATHODE is the GREEN and BLUE pins in turn, and the right side is the RED pin.



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.

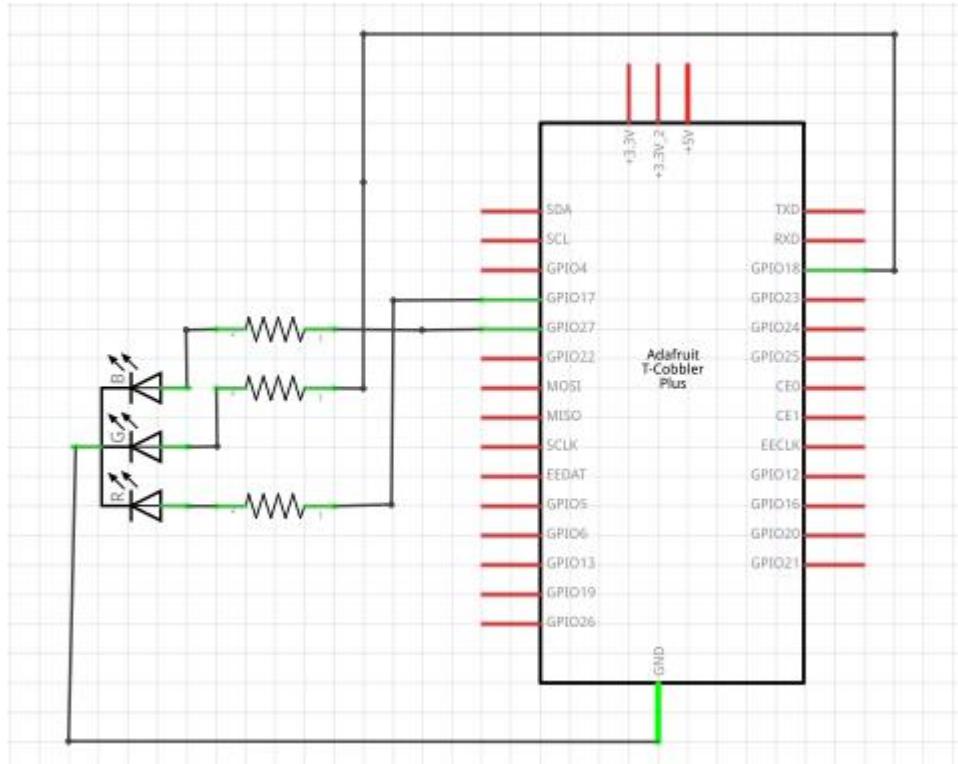


For example, if we set the three LEDs to the same brightness, the overall light color will be displayed in white. If we turn off the blue LED, only the red and green LEDs with the same brightness, then the LED will be color yellow.

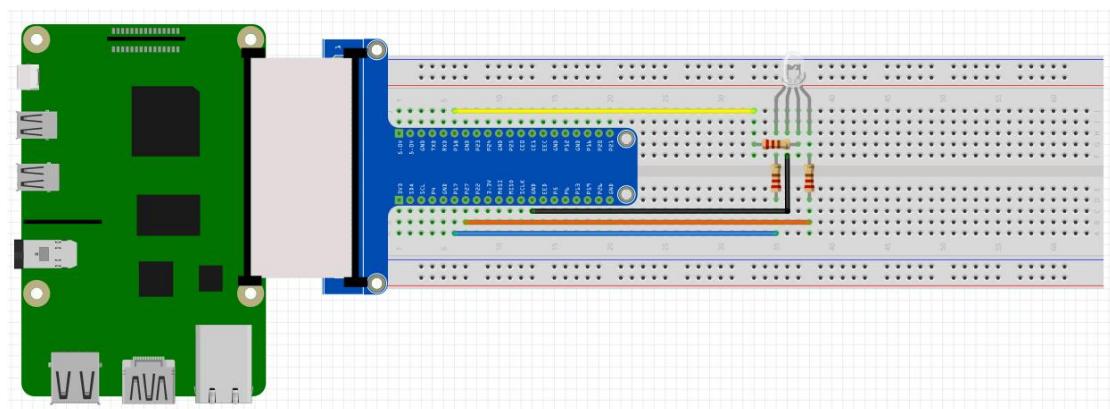
We can easily control the brightness of red, green and blue of each RGB LED, which makes it easy to control different colors.

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.

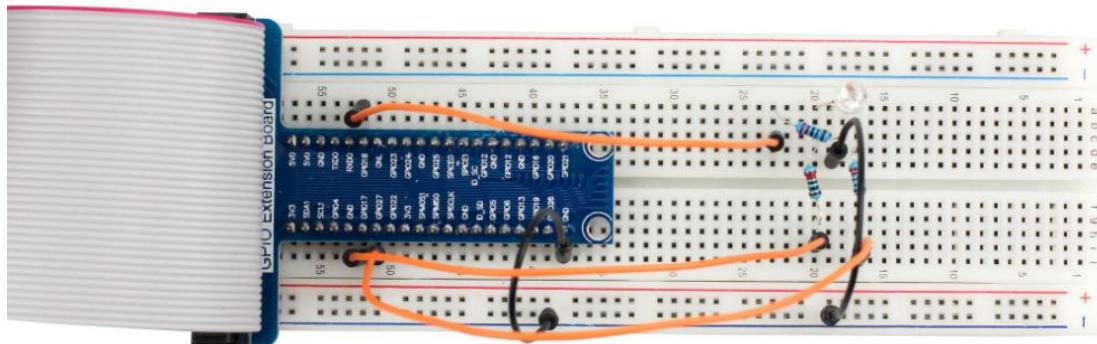
Connection diagram



Wiring Diagram



Example Figure



C code

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;

```
pi@raspberrypi:~/code/C/5.RGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/5.RGBLED/
pi@raspberrypi:~/code/C/5.RGBLED $ ls
RGBLED.c
pi@raspberrypi:~/code/C/5.RGBLED $
```

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:

```
pi@raspberrypi:~/code/C/5.RGBLED
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/5.RGBLED/
pi@raspberrypi:~/code/C/5.RGBLED $ ls
RGBLED.c
pi@raspberrypi:~/code/C/5.RGBLED $ gcc RGBLED.c -o RGBLED -lwiringPi -lpthread
pi@raspberrypi:~/code/C/5.RGBLED $ ls
RGBLED RGBLED.c
pi@raspberrypi:~/code/C/5.RGBLED $ sudo ./RGBLED
Program is starting ...
r=83, g=86, b=77
r=15, g=93, b=35
r=86, g=92, b=49
r=21, g=62, b=27
r=90, g=59, b=63
r=26, g=40, b=26
r=72, g=36, b=11
```

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

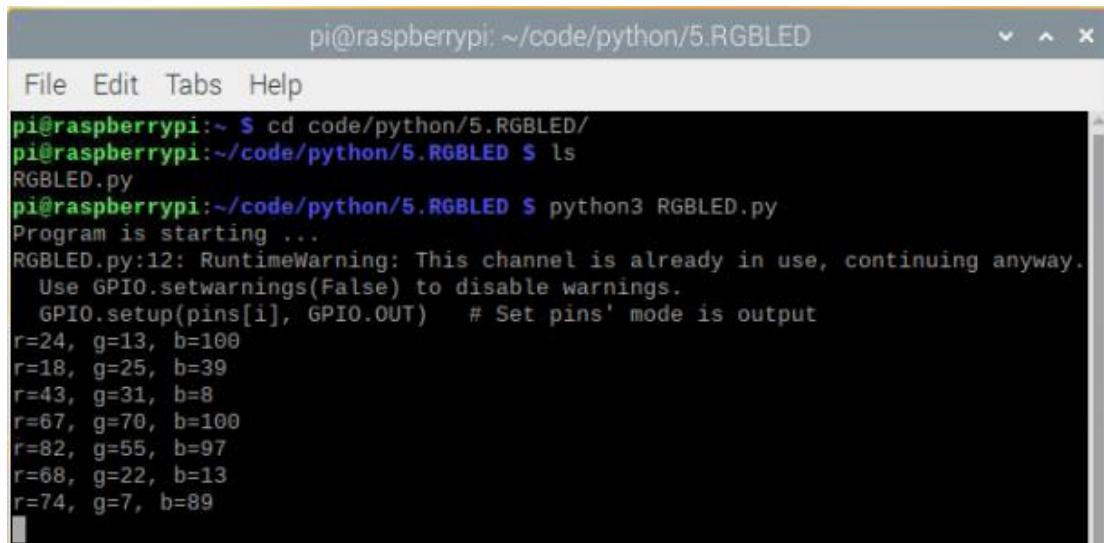
Open the terminal and use `cd code/python/5.RGBLED/` command to enter the code directory;

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



```
pi@raspberrypi:~/code/python/5.RGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/5.RGBLED/
pi@raspberrypi:~/code/python/5.RGBLED $ ls
RGBLED.py
pi@raspberrypi:~/code/python/5.RGBLED $
```

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



```
pi@raspberrypi:~/code/python/5.RGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/5.RGBLED/
pi@raspberrypi:~/code/python/5.RGBLED $ ls
RGBLED.py
pi@raspberrypi:~/code/python/5.RGBLED $ python3 RGBLED.py
Program is starting ...
RGBLED.py:12: RuntimeWarning: This channel is already in use, continuing anyway.
  Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(pins[i], GPIO.OUT)    # Set pins' mode is output
r=24, g=13, b=100
r=18, g=25, b=39
r=43, g=31, b=8
r=67, g=70, b=100
r=82, g=55, b=97
r=68, g=22, b=13
r=74, g=7, b=89

```

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.



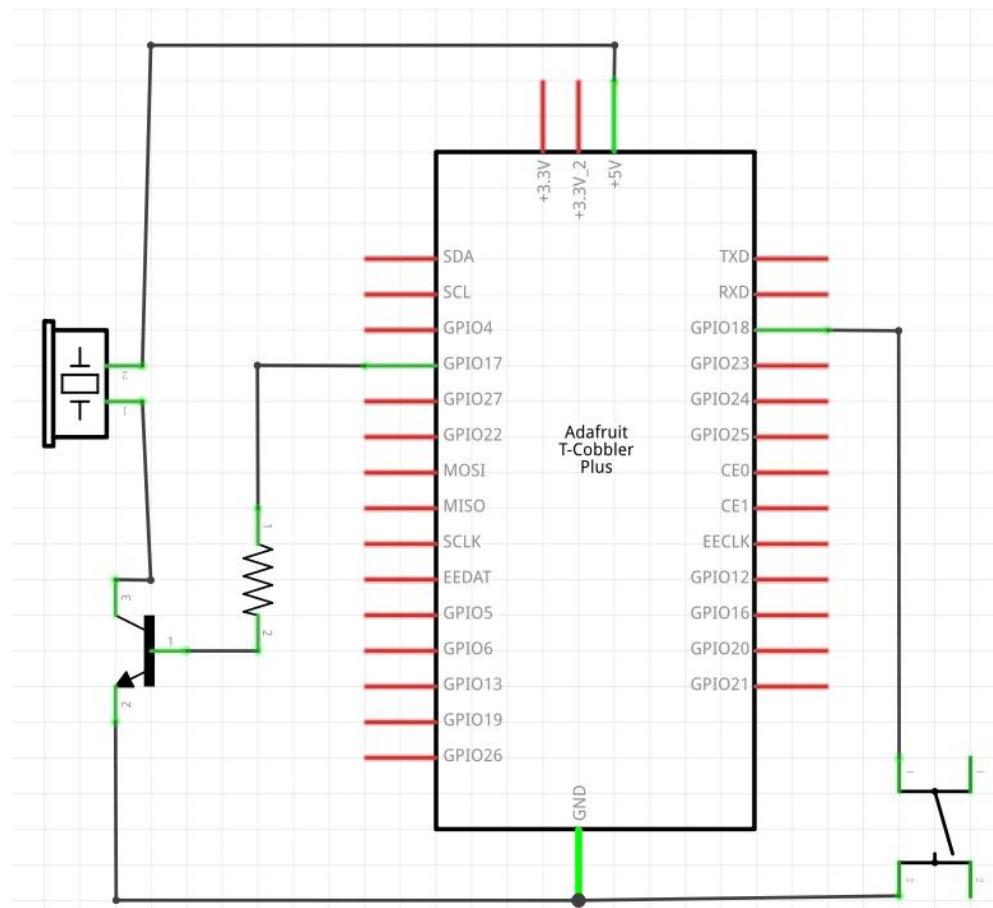
S8050 Triode

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

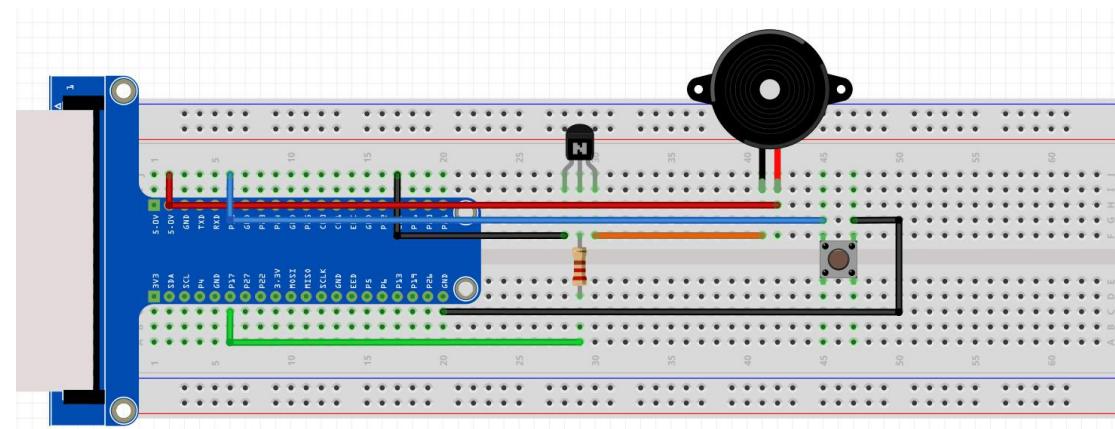
It is often seen in various amplifier circuits, and has a wide range of applications, mainly for high-frequency amplification. Can also be used as a switching circuit. From left to right, the pins are the emitter, base, and collector.



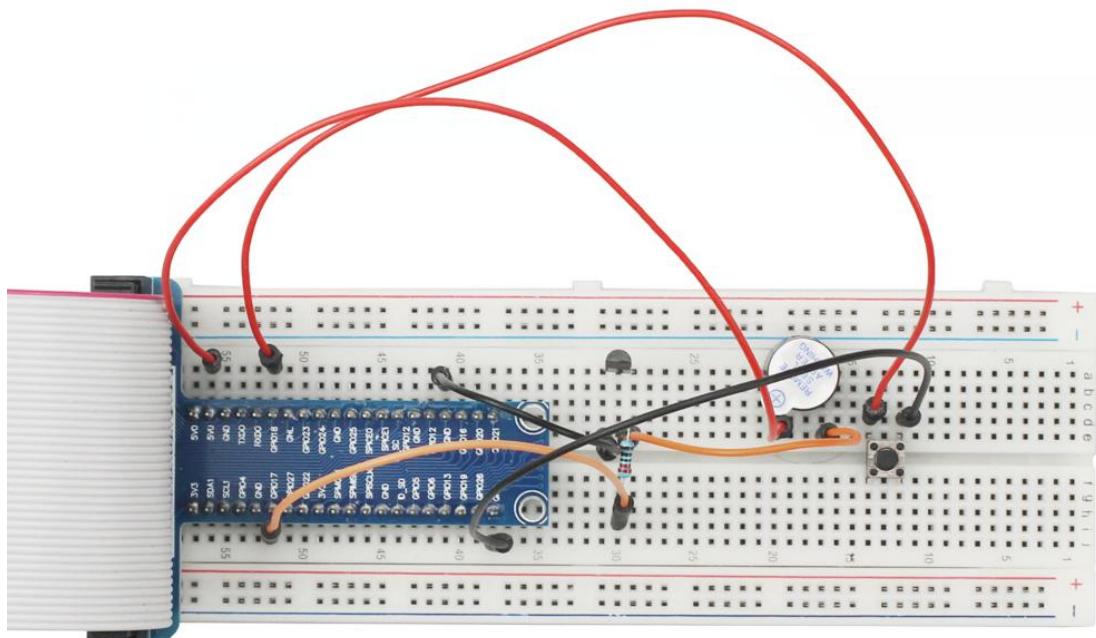
Connection Diagram



Wiring Diagram



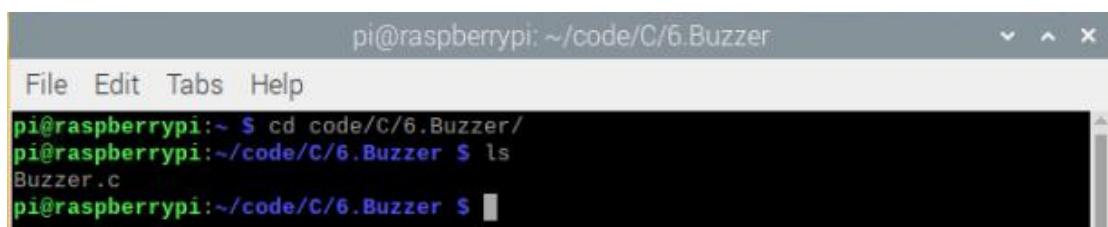
Example picture



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;



```
pi@raspberrypi:~/code/C/6.Buzzer
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/6.Buzzer/
pi@raspberrypi:~/code/C/6.Buzzer $ ls
Buzzer.c
pi@raspberrypi:~/code/C/6.Buzzer $
```

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:



A terminal window titled "pi@raspberrypi: ~ /code/C/6.Buzzer". The window shows the command "sudo ./buzzer" being run and its output. The output consists of nine lines of text, each ending with a backslash and a new line character, indicating a continuous loop or a bug in the code.

```
...buzzer off
```

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");
        }
    }
}
```

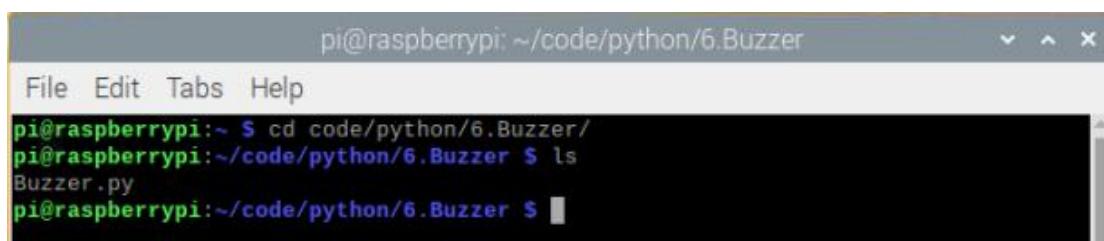
```
    }
}

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.



A screenshot of a terminal window titled "pi@raspberrypi: ~/code/python/6.Buzzer". The window shows the following text:

```
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/6.Buzzer/
pi@raspberrypi:~/code/python/6.Buzzer $ ls
Buzzer.py
pi@raspberrypi:~/code/python/6.Buzzer $
```

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



A screenshot of a terminal window titled "pi@raspberrypi: ~/code/python/6.Buzzer". The window shows the following text:

```
File Edit Tabs Help
buzzer off ...
```

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

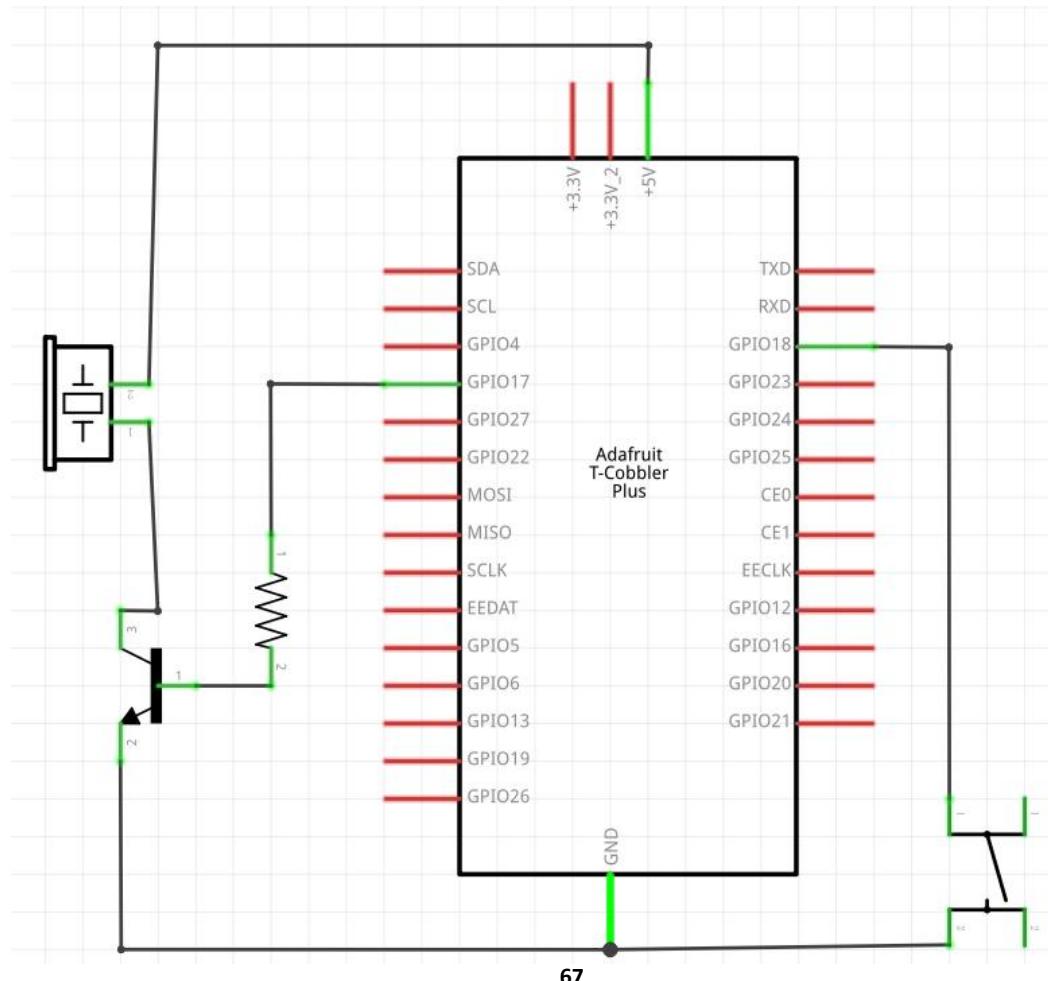
Product Introduction

Passive Buzzer

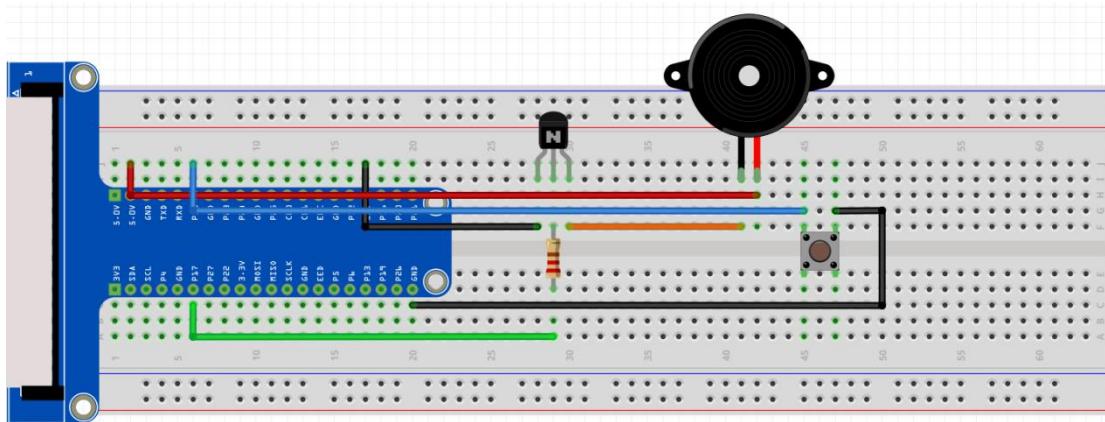
The passive buzzer works by using PWM to generate sound by achieving air vibration. As long as the vibration frequency is changed, different sounds can be produced. For example, send a 523Hz pulse, which can generate Do, pulse 587Hz, it can generate IF Re, 659Hz pulse, it can generate Mi.



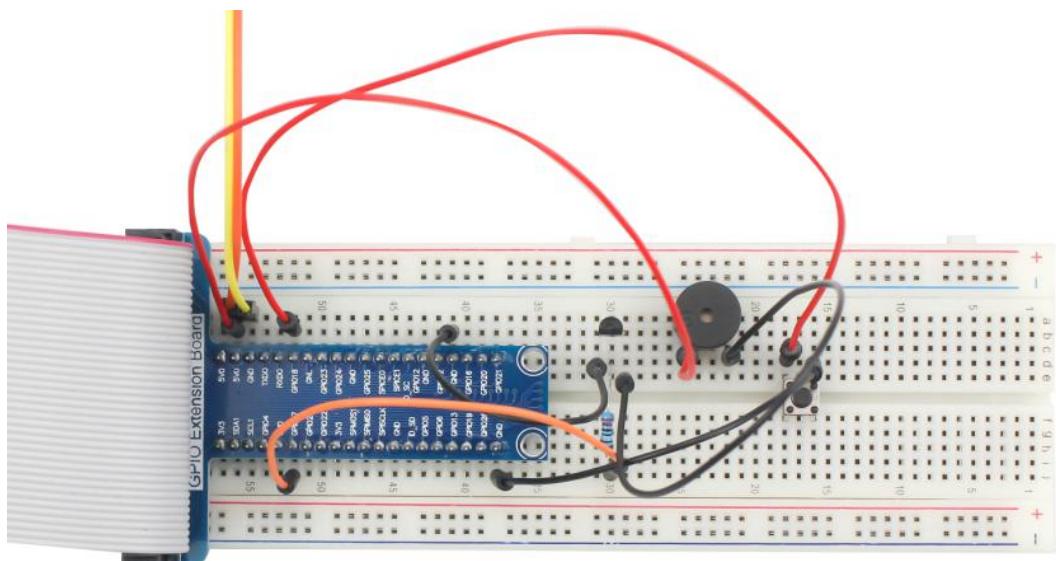
Connection Diagram



Wiring Diagram



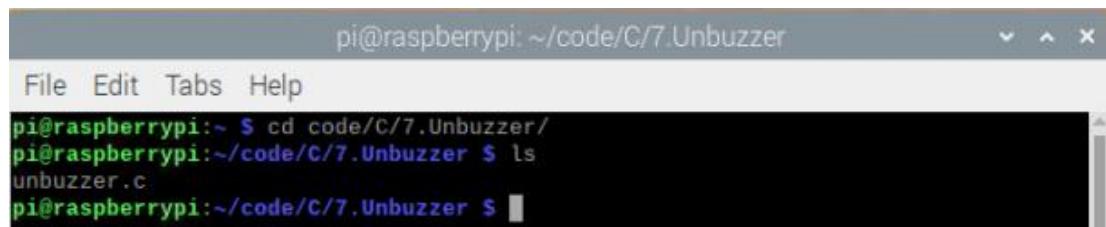
Example picture



C code

Open the terminal and enter the [cd code / C / 7.Unbuzzer](#) / command to enter the unbuzzer.c code directory;

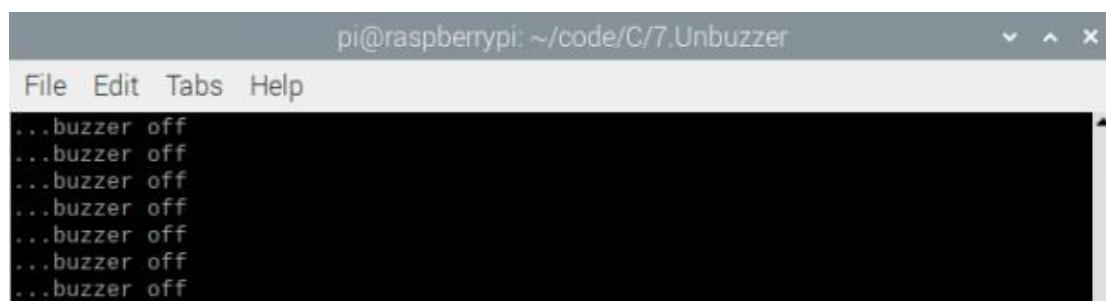
Enter the `ls` command to view the file unbuzzer.c in the directory;



```
pi@raspberrypi: ~/code/C/7.Unbuzzer
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/7.Unbuzzer/
pi@raspberrypi:~/code/C/7.Unbuzzer $ ls
unbuzzer.c
pi@raspberrypi:~/code/C/7.Unbuzzer $
```

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:



```
pi@raspberrypi: ~/code/C/7.Unbuzzer
File Edit Tabs Help
...buzzer off
```

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 buzzRPin 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

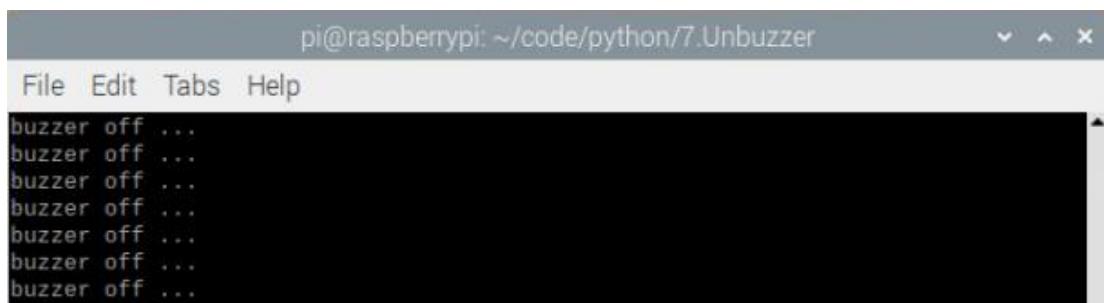
Open the terminal and use the `cd code / python / 7.Unbuzzer /` command to enter the code directory

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



```
pi@raspberrypi:~/code/python/7.Unbuzzer
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/7.Unbuzzer/
pi@raspberrypi:~/code/python/7.Unbuzzer $ ls
unbuzzer.py
pi@raspberrypi:~/code/python/7.Unbuzzer $
```

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



```
pi@raspberrypi:~/code/python/7.Unbuzzer
File Edit Tabs Help
buzzer off ...
```

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 you need to create a software PWM pin through 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.

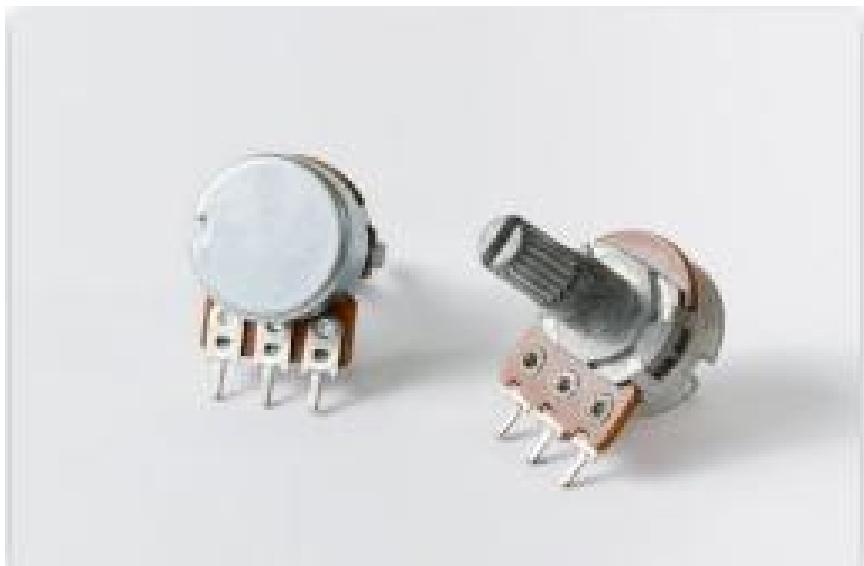
Parts Required

- 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

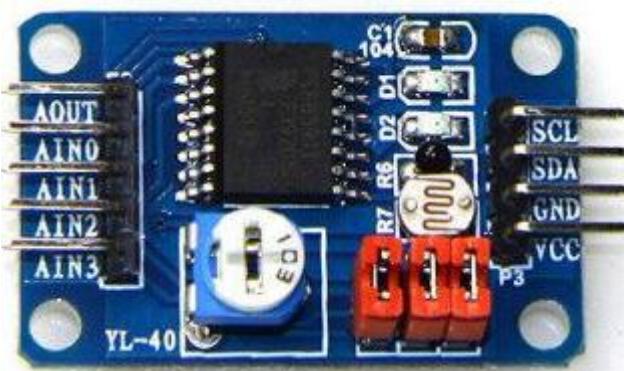
Potentiometer

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.

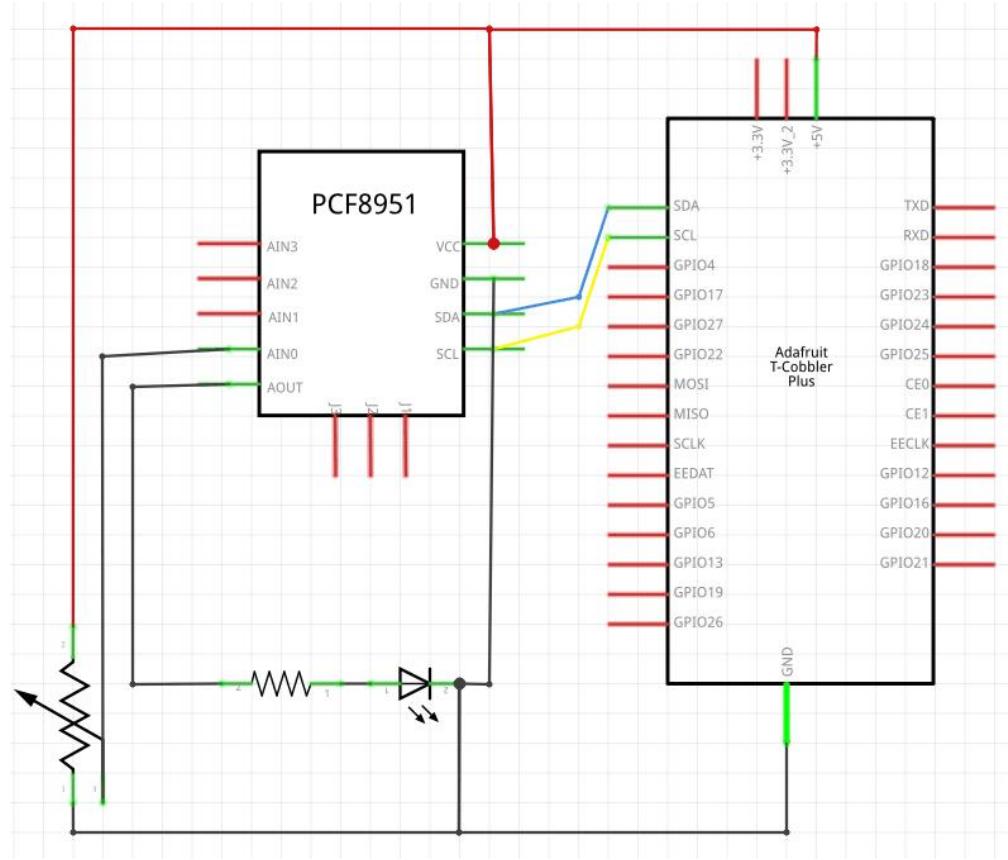


PCF8591 module

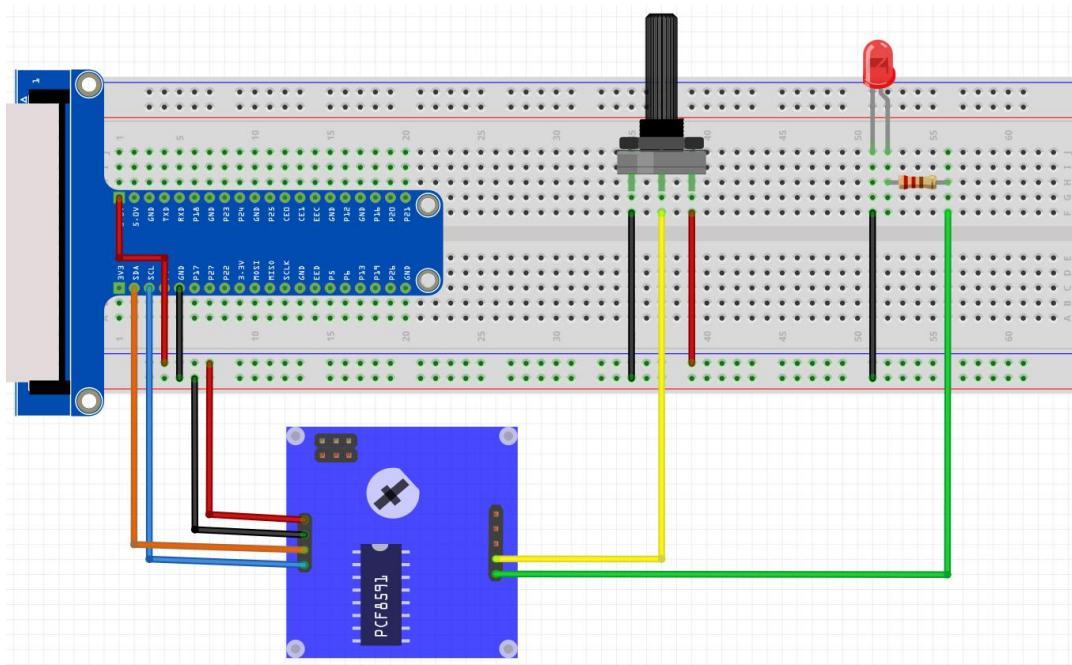
The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. PCF8591's three address pins A0, A1, and A2 can be used for hardware address programming, allowing access to eight PCF8591 devices on the 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.



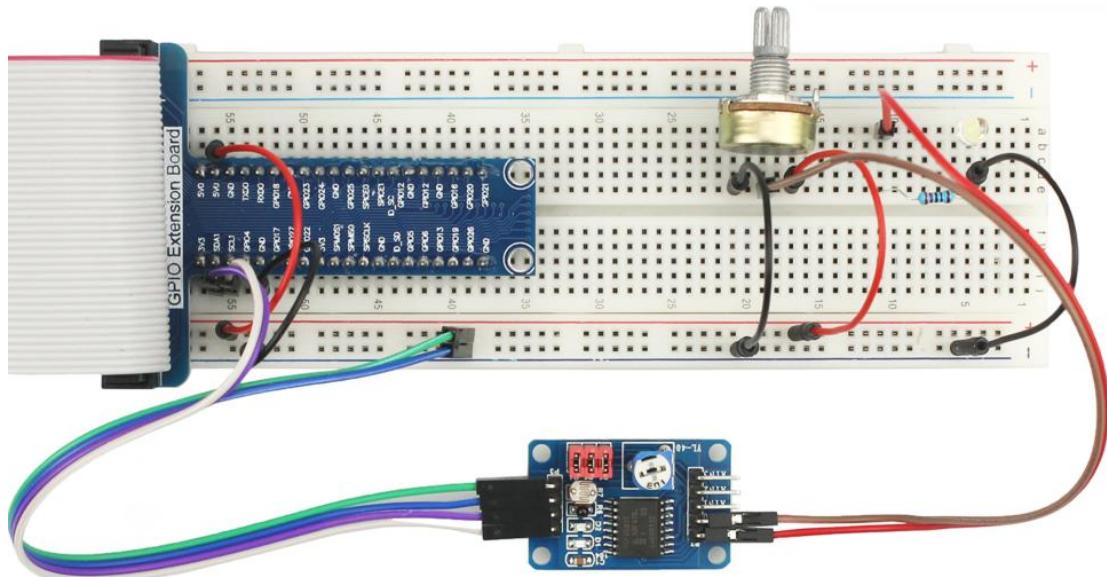
Connection Diagram



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 libsунпин3v5 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;

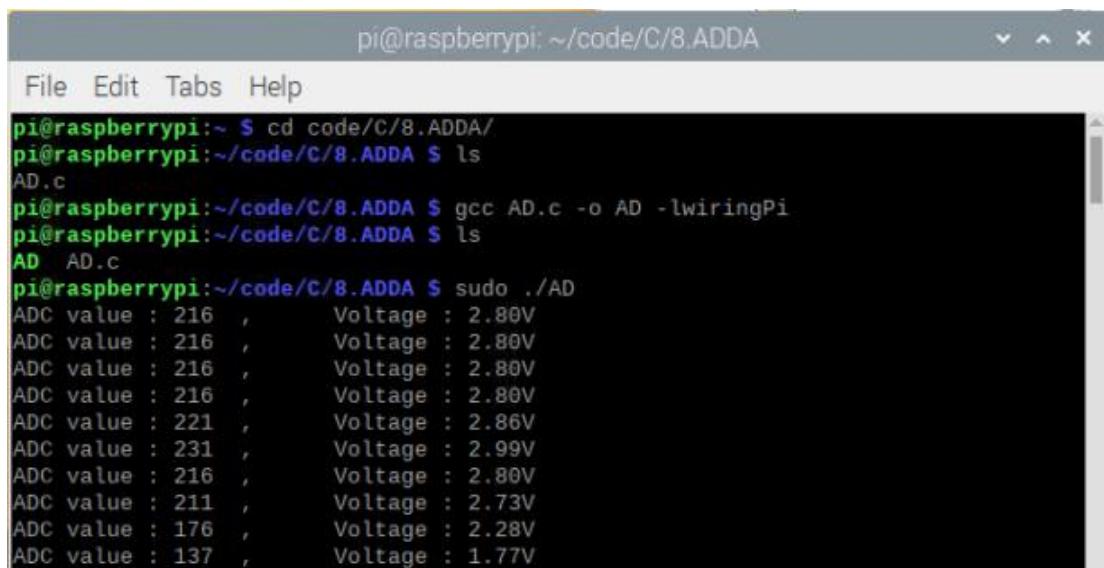
Enter the `ls` command to view the file AD.c in the directory;



```
pi@raspberrypi: ~/code/C/8.ADDA
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/8.ADDA/
pi@raspberrypi:~/code/C/8.ADDA $ ls
AD.c
pi@raspberrypi:~/code/C/8.ADDA $
```

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:



```
pi@raspberrypi: ~/code/C/8.ADDA
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/8.ADDA/
pi@raspberrypi:~/code/C/8.ADDA $ ls
AD.c
pi@raspberrypi:~/code/C/8.ADDA $ gcc AD.c -o AD -lwiringPi
pi@raspberrypi:~/code/C/8.ADDA $ ls
AD AD.c
pi@raspberrypi:~/code/C/8.ADDA $ sudo ./AD
ADC value : 216 , Voltage : 2.80V
ADC value : 221 , Voltage : 2.86V
ADC value : 231 , Voltage : 2.99V
ADC value : 216 , Voltage : 2.80V
ADC value : 211 , Voltage : 2.73V
ADC value : 176 , Voltage : 2.28V
ADC value : 137 , Voltage : 1.77V
```

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
#define nbase + 0
#define A1 pinbase + 1
#define A2 pinbase + 2
#define A3 pinbase + 3

```

The default I_C 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);
}

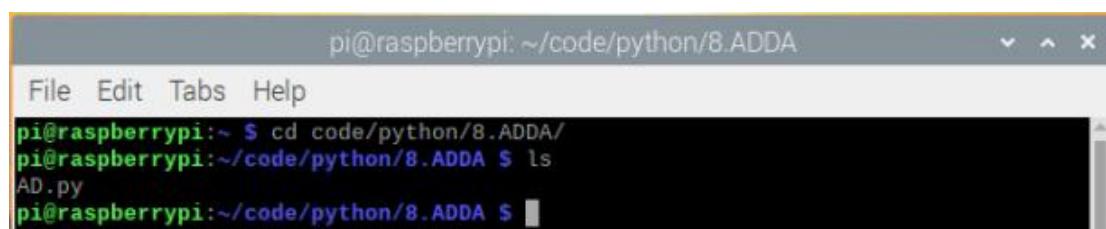
```

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

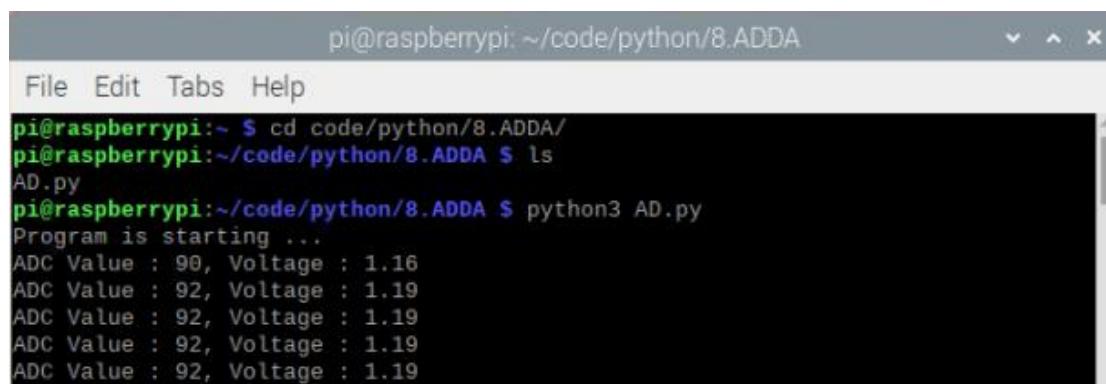
Open the terminal and use the `cd code / python / 8.ADDA /` command to enter the code directory

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



```
pi@raspberrypi:~/code/python/8.ADDA
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/8.ADDA/
pi@raspberrypi:~/code/python/8.ADDA $ ls
AD.py
pi@raspberrypi:~/code/python/8.ADDA $
```

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



```
pi@raspberrypi:~/code/python/8.ADDA
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/8.ADDA/
pi@raspberrypi:~/code/python/8.ADDA $ ls
AD.py
pi@raspberrypi:~/code/python/8.ADDA $ python3 AD.py
Program is starting ...
ADC Value : 90, Voltage : 1.16
ADC Value : 92, Voltage : 1.19
```

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

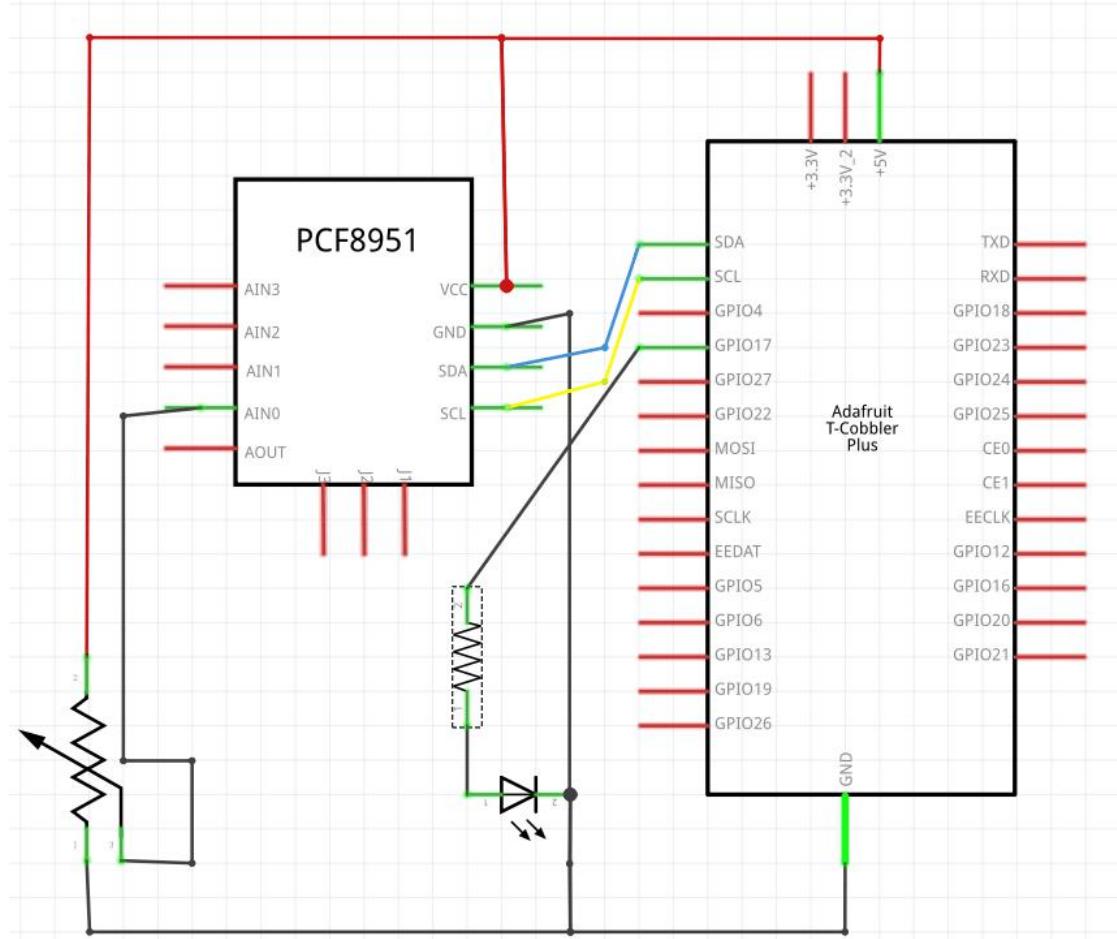
11 x DuPont

1 x Breadboard

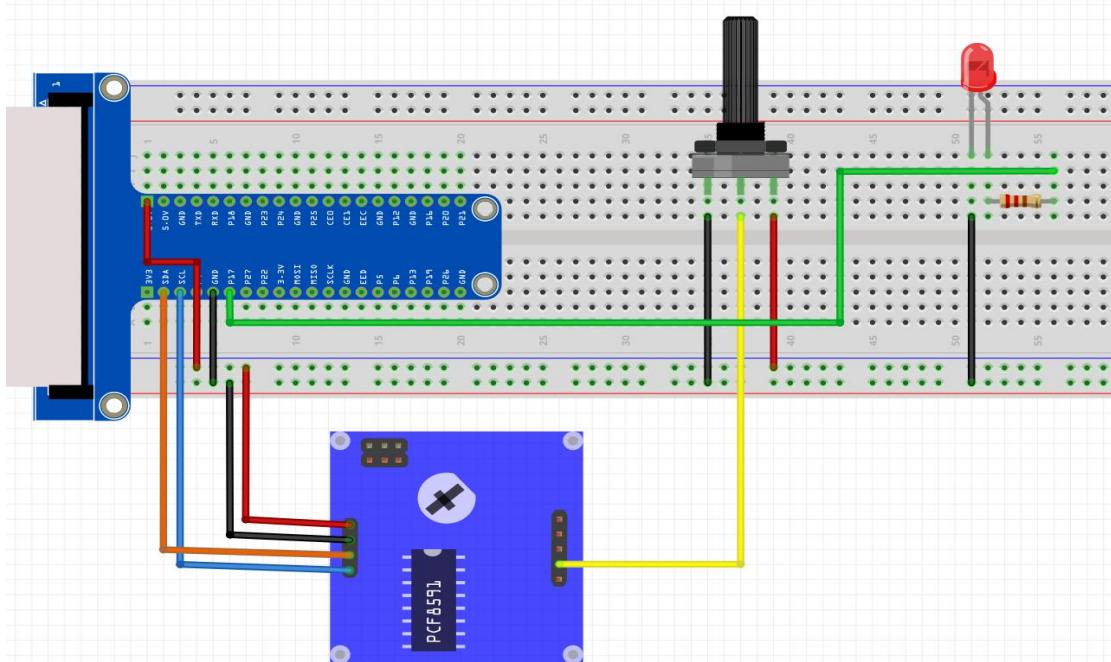
1 x Potentiometer

1 x LED

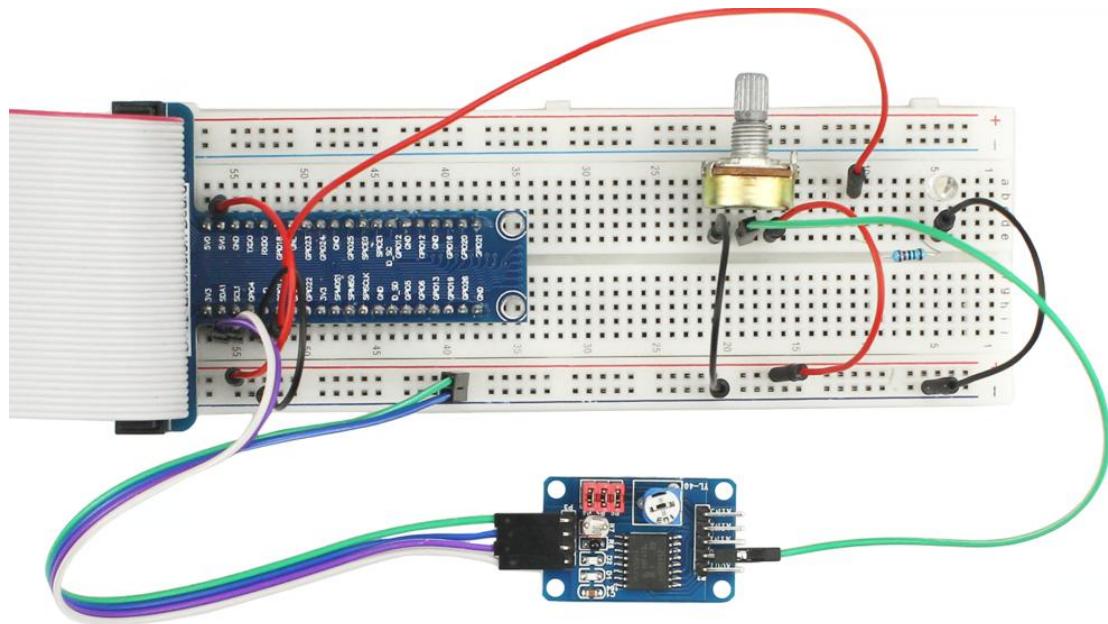
Connection Diagram



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;

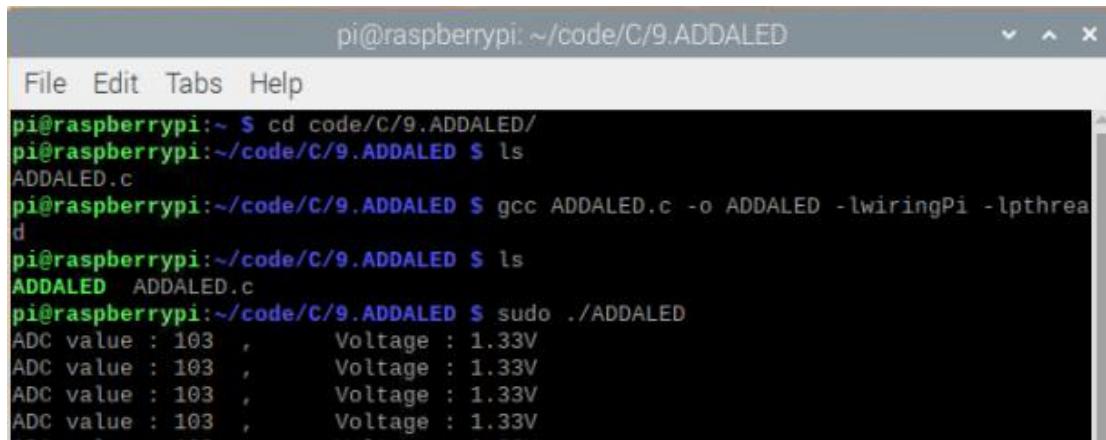
Enter the `ls` command to view the file ADDALED.c in the directory;



```
pi@raspberrypi: ~code/C/9.ADDALED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/9.ADDALED/
pi@raspberrypi:~/code/C/9.ADDALED $ ls
ADDALED.c
pi@raspberrypi:~/code/C/9.ADDALED $
```

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:



The screenshot shows a terminal window titled "pi@raspberrypi: ~ /code/C/9.ADDALED". The window contains the following text:

```
pi@raspberrypi:~ $ cd code/C/9.ADDALED/
pi@raspberrypi:~/code/C/9.ADDALED $ ls
ADDALED.c
pi@raspberrypi:~/code/C/9.ADDALED $ gcc ADDALED.c -o ADDALED -lwiringPi -lpthread
pi@raspberrypi:~/code/C/9.ADDALED $ ls
ADDALED ADDALED.c
pi@raspberrypi:~/code/C/9.ADDALED $ sudo ./ADDALED
ADC value : 103 , Voltage : 1.33V
```

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;
}

```

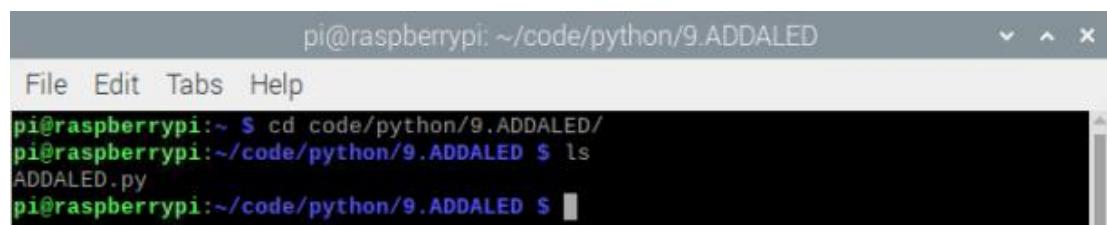
Code Interpretation

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.

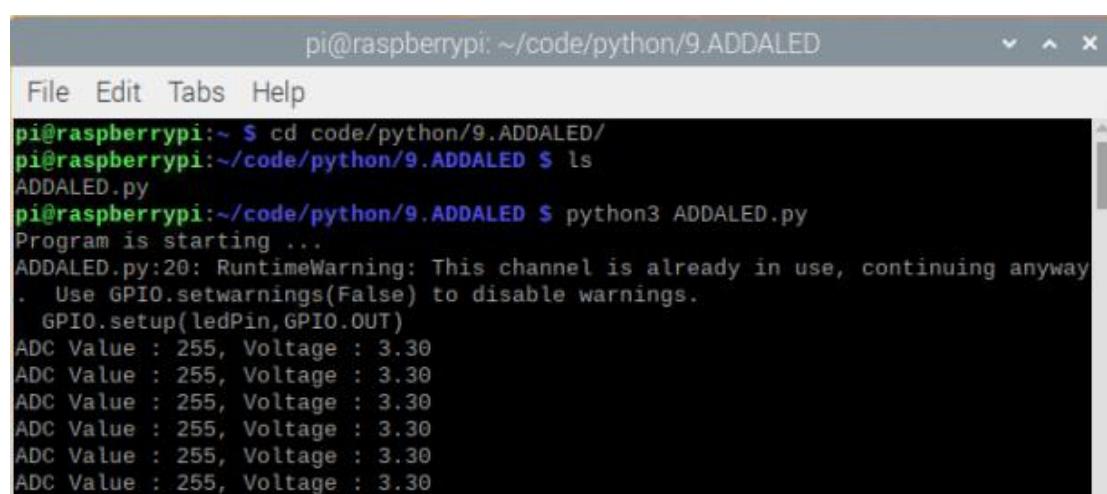
Python code

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

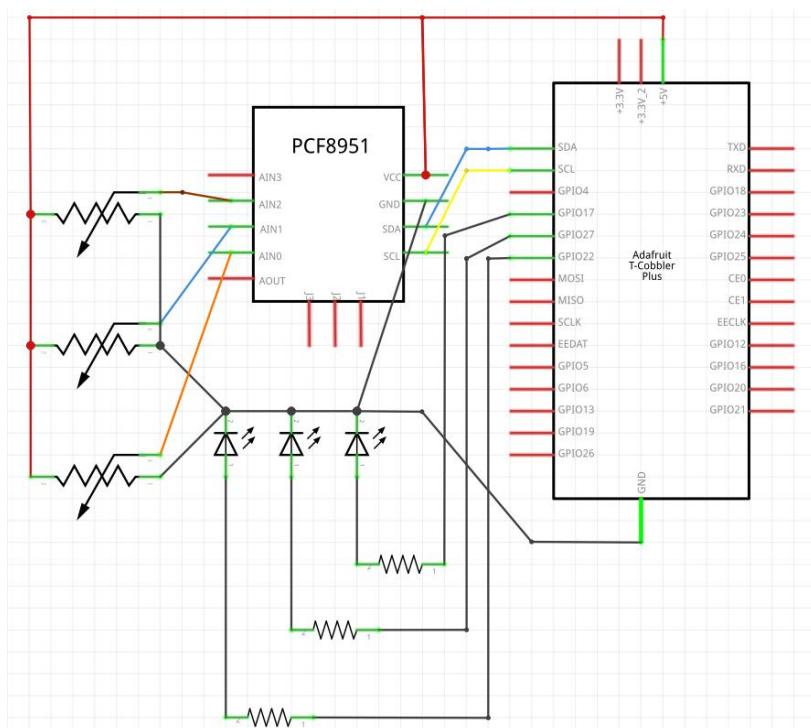
21 x DuPont

1 x Breadboard

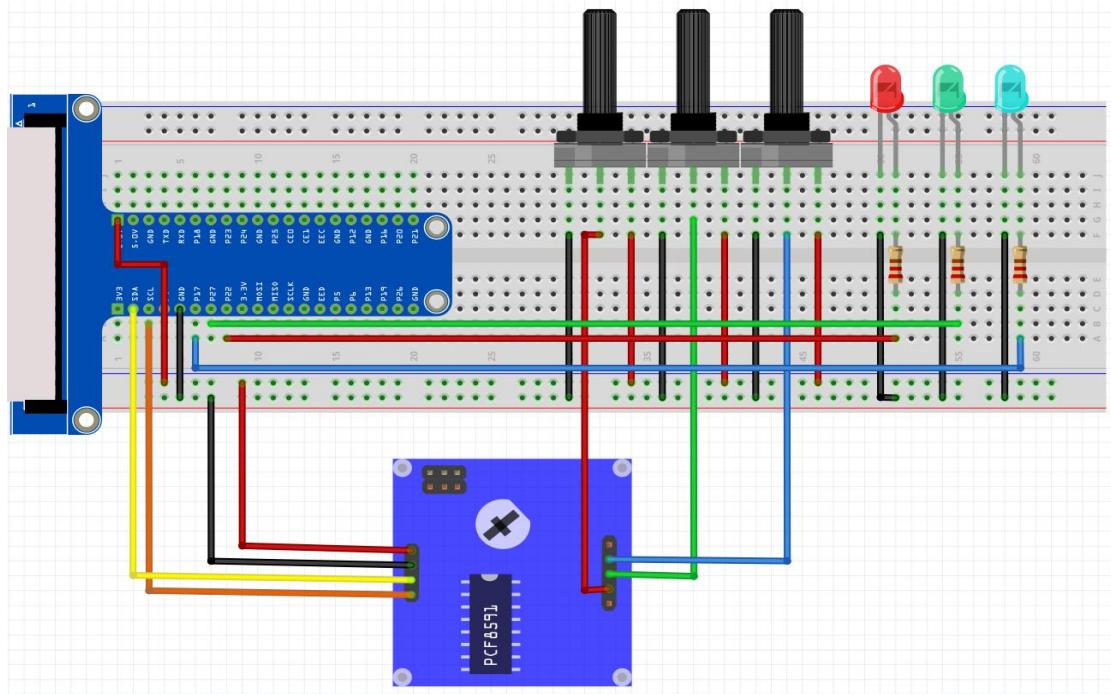
1 x RGBLED

3 x Potentiometer

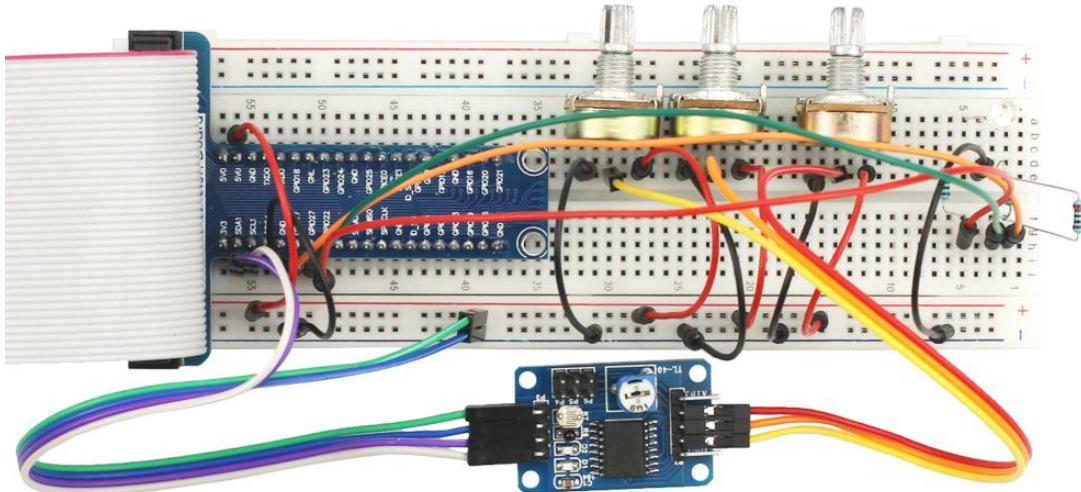
Connection Diagram



Wiring Diagram



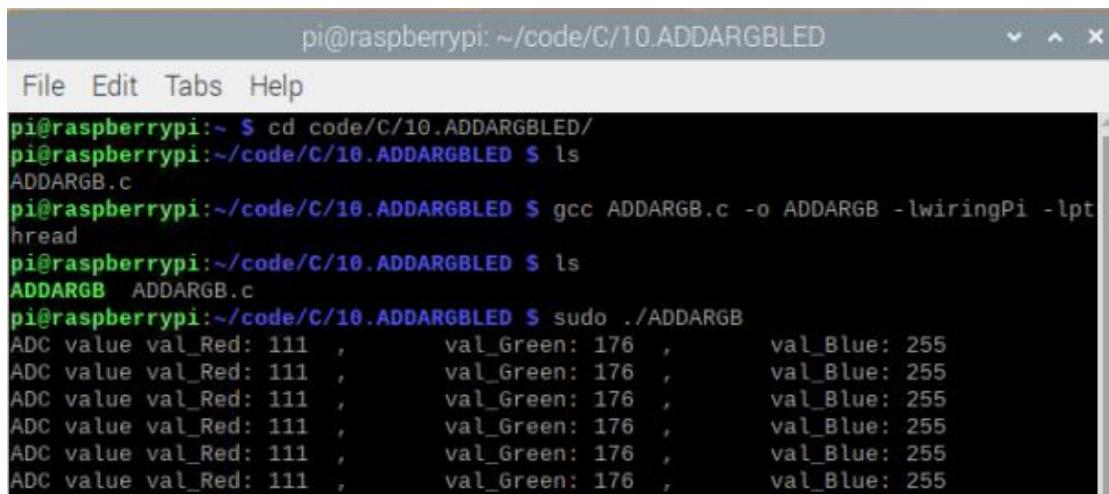
Example picture



C code

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

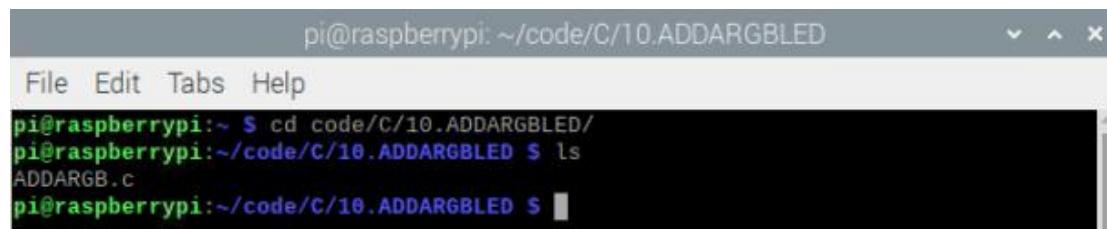
Enter the `ls` command to view the file ADDARGB.c in the directory;



```
pi@raspberrypi:~/code/C/10.ADDARGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/10.ADDARGBLED/
pi@raspberrypi:~/code/C/10.ADDARGBLED $ ls
ADDARGB.c
pi@raspberrypi:~/code/C/10.ADDARGBLED $ gcc ADDARGB.c -o ADDARGB -lwiringPi -lpthread
pi@raspberrypi:~/code/C/10.ADDARGBLED $ ls
ADDARGB ADDARGB.c
pi@raspberrypi:~/code/C/10.ADDARGBLED $ sudo ./ADDARGB
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
ADC value val_Red: 111 , val_Green: 176 , val_Blue: 255
```

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:



```
pi@raspberrypi:~/code/C/10.ADDARGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/10.ADDARGBLED/
pi@raspberrypi:~/code/C/10.ADDARGBLED $ ls
ADDARGB.c
pi@raspberrypi:~/code/C/10.ADDARGBLED $ █
```

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 ,val_Green: %d ,val_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

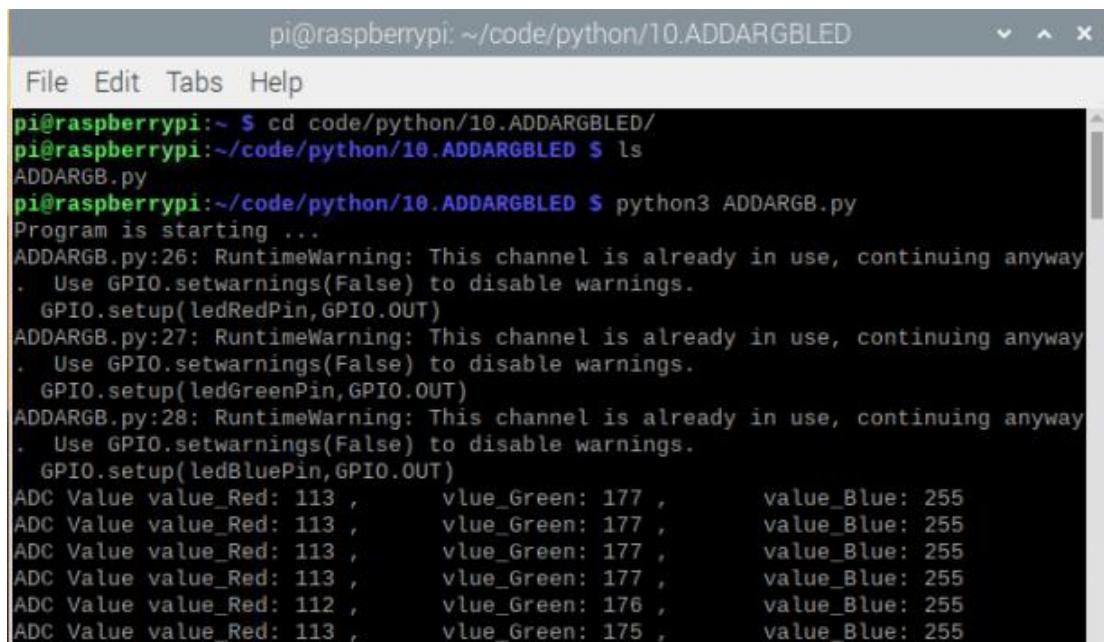
Open the terminal and use the `cd code / python / 10.ADDARGBLED /` command to enter the code directory;

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



```
pi@raspberrypi:~/code/python/10.ADDARGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/10.ADDARGBLED/
pi@raspberrypi:~/code/python/10.ADDARGBLED $ ls
ADDARGB.py
pi@raspberrypi:~/code/python/10.ADDARGBLED $
```

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



```
pi@raspberrypi:~/code/python/10.ADDARGBLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/10.ADDARGBLED/
pi@raspberrypi:~/code/python/10.ADDARGBLED $ ls
ADDARGB.py
pi@raspberrypi:~/code/python/10.ADDARGBLED $ python3 ADDARGB.py
Program is starting ...
ADDARGB.py:26: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(ledRedPin,GPIO.OUT)
ADDARGB.py:27: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(ledGreenPin,GPIO.OUT)
ADDARGB.py:28: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(ledBluePin,GPIO.OUT)
ADC Value value_Red: 113 ,      vluue_Green: 177 ,      value_Blue: 255
ADC Value value_Red: 113 ,      vluue_Green: 177 ,      value_Blue: 255
ADC Value value_Red: 113 ,      vluue_Green: 177 ,      value_Blue: 255
ADC Value value_Red: 113 ,      vluue_Green: 177 ,      value_Blue: 255
ADC Value value_Red: 112 ,      vluue_Green: 176 ,      value_Blue: 255
ADC Value value_Red: 113 ,      vluue_Green: 175 ,      value_Blue: 255
```

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 ,\tvalue_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 ,\tvalue_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 to use 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

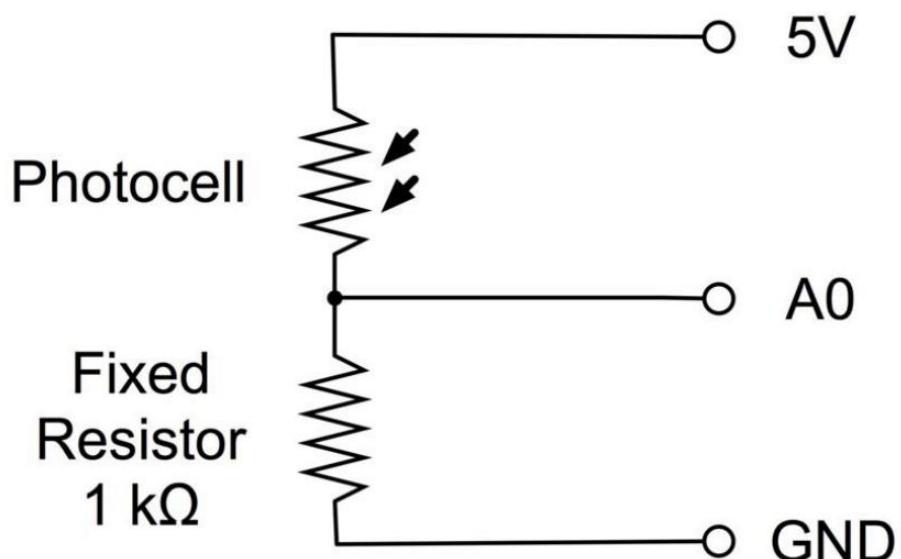
1 x LED

1 x Photoresistor

Product Introduction

Photosensitive Resistor

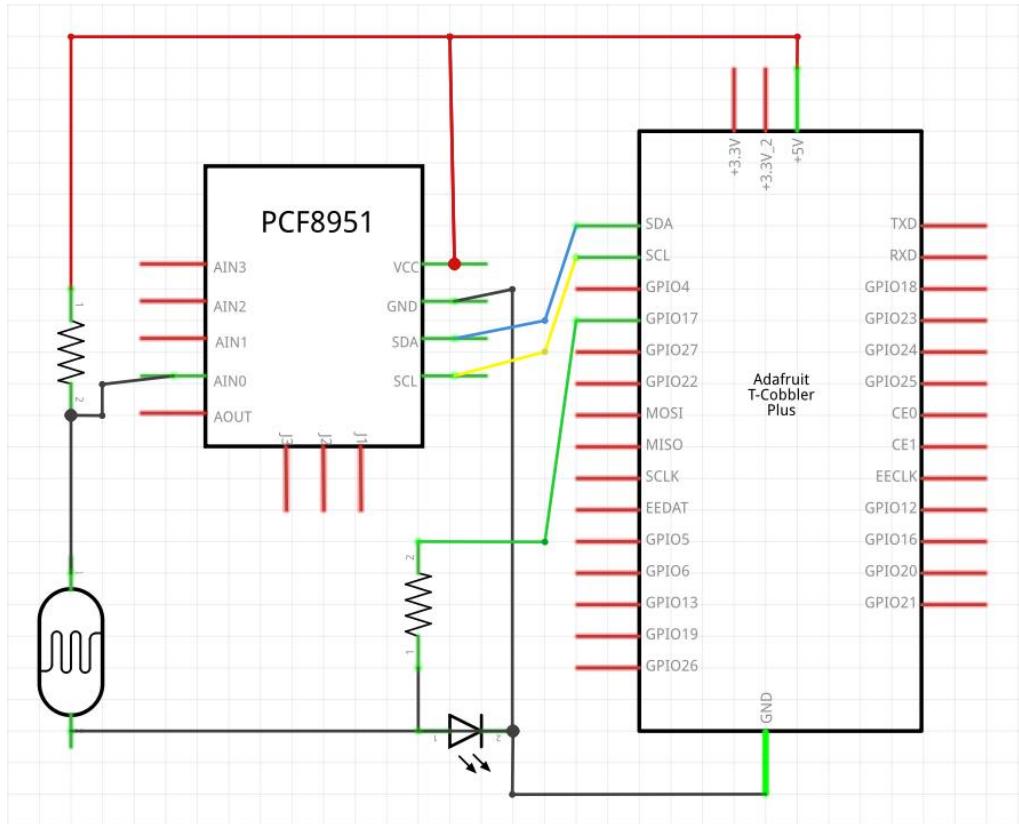
Photosensitive resistors, also known as LDRs. Photoresistors are like ordinary resistors, except that the resistance of the resistor changes as the light falls on them. This has a resistance of about $50\text{ k}\Omega$ in the near dark and 500Ω in the strong light. To convert this changed resistance 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.



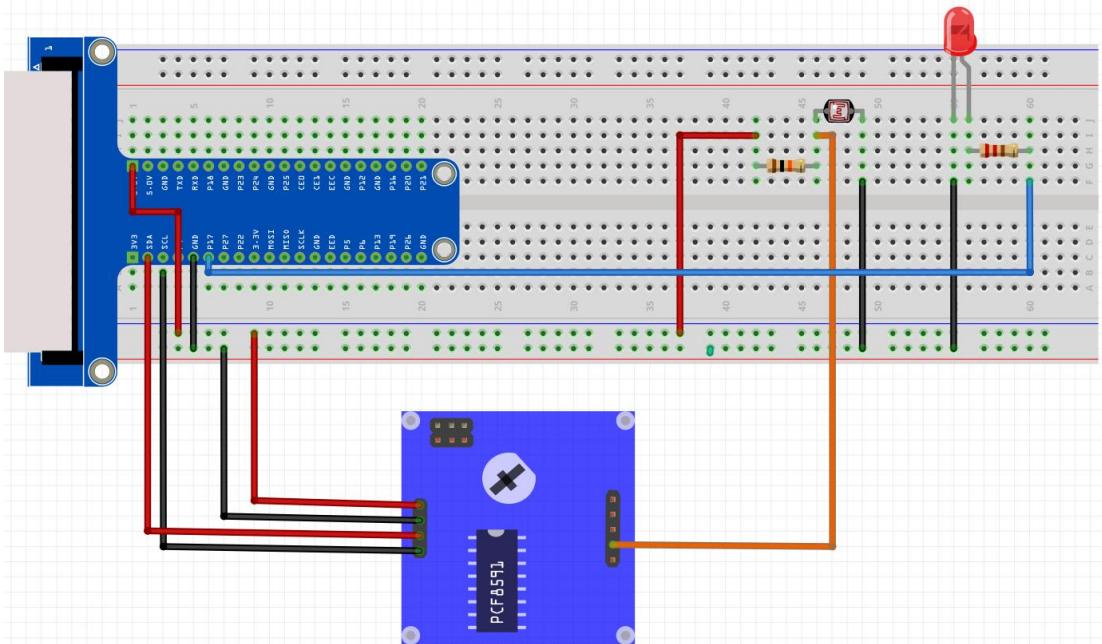
The values of the resistor and the photoresistor are put together to behave like a single data. When the light is very bright, the resistance of the photoresistor is very low compared to a fixed value resistor, so it is as if the potentiometer is set to maximum. When the photoresistor is in dim light, the resistance becomes greater than a fixed $1\text{k}\Omega$ 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

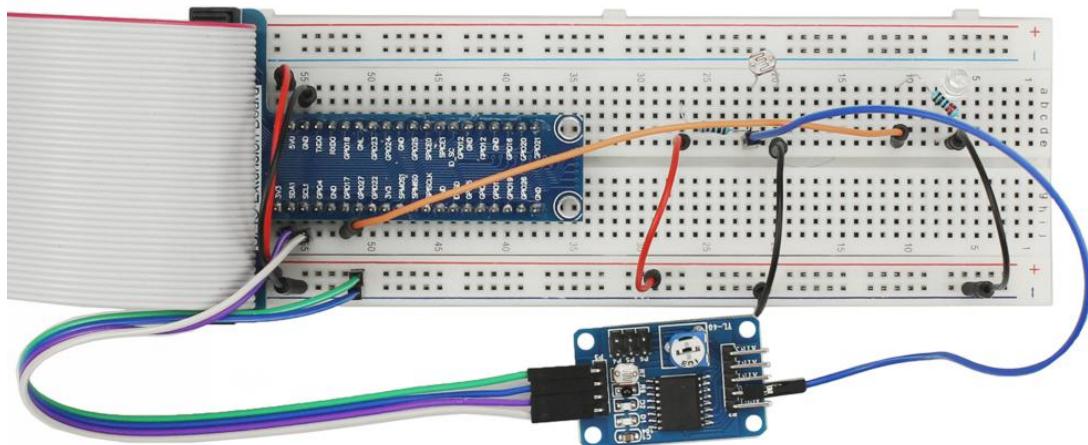


Wiring Diagram





Example Figure



C code

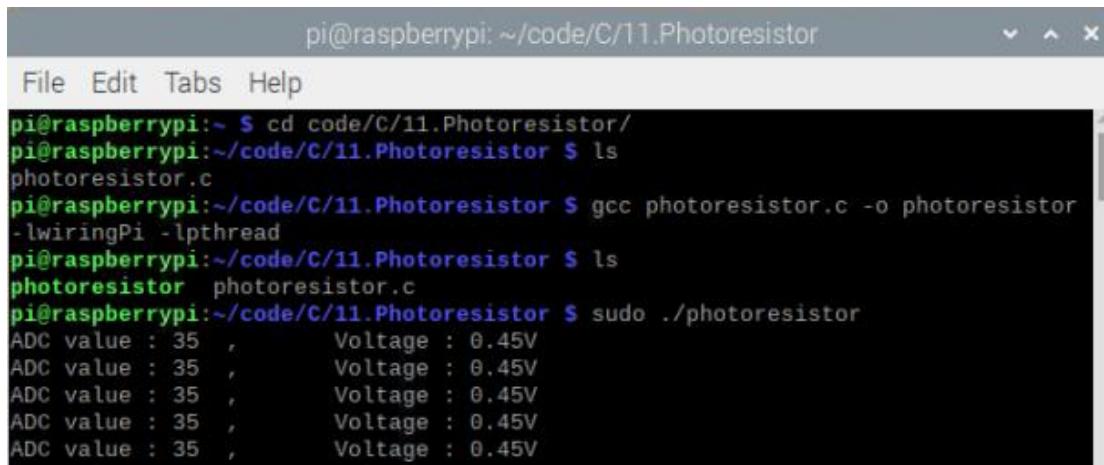
Open the terminal and enter the `cd code / C / 11.Photoresistor /` command to enter the photoresistor.c code directory;

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

```
pi@raspberrypi: ~/code/C/11.Photoresistor
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/11.Photoresistor/
pi@raspberrypi:~/code/C/11.Photoresistor $ ls
photoresistor.c
pi@raspberrypi:~/code/C/11.Photoresistor $
```

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:



```
pi@raspberrypi:~/code/C/11.Photoresistor
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/11.Photoresistor/
pi@raspberrypi:~/code/C/11.Photoresistor$ ls
photoresistor.c
pi@raspberrypi:~/code/C/11.Photoresistor$ gcc photoresistor.c -o photoresistor
-lwiringPi -lpthread
pi@raspberrypi:~/code/C/11.Photoresistor$ ls
photoresistor photoresistor.c
pi@raspberrypi:~/code/C/11.Photoresistor$ sudo ./photoresistor
ADC value : 35 , Voltage : 0.45V
```

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.

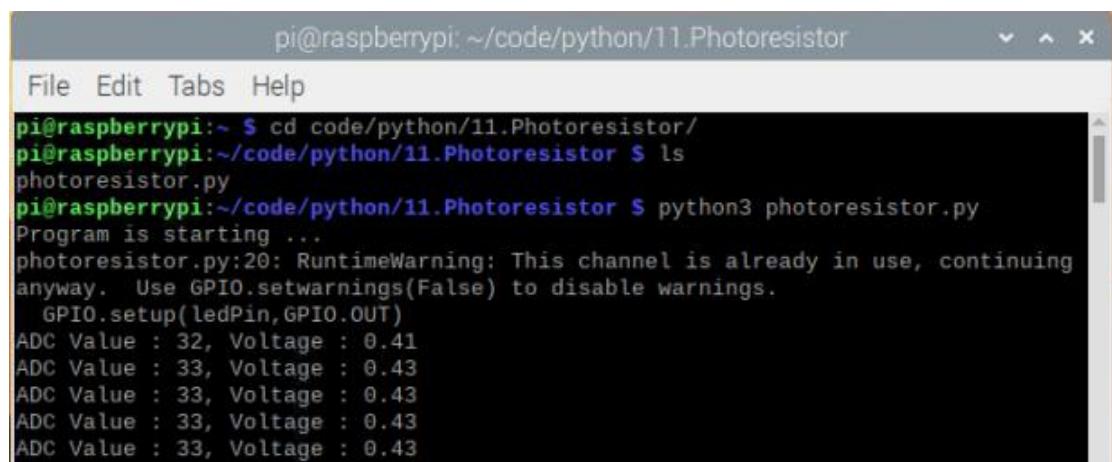


```

pi@raspberrypi: ~/code/python/11.Photoresistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/11.Photoresistor/
pi@raspberrypi:~/code/python/11.Photoresistor $ ls
photoresistor.py
pi@raspberrypi:~/code/python/11.Photoresistor $ 

```

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



```

pi@raspberrypi: ~/code/python/11.Photoresistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/11.Photoresistor/
pi@raspberrypi:~/code/python/11.Photoresistor $ ls
photoresistor.py
pi@raspberrypi:~/code/python/11.Photoresistor $ python3 photoresistor.py
Program is starting ...
photoresistor.py:20: RuntimeWarning: This channel is already in use, continuing
anyway.  Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(ledPin,GPIO.OUT)
ADC Value : 32, Voltage : 0.41
ADC Value : 33, Voltage : 0.43

```

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

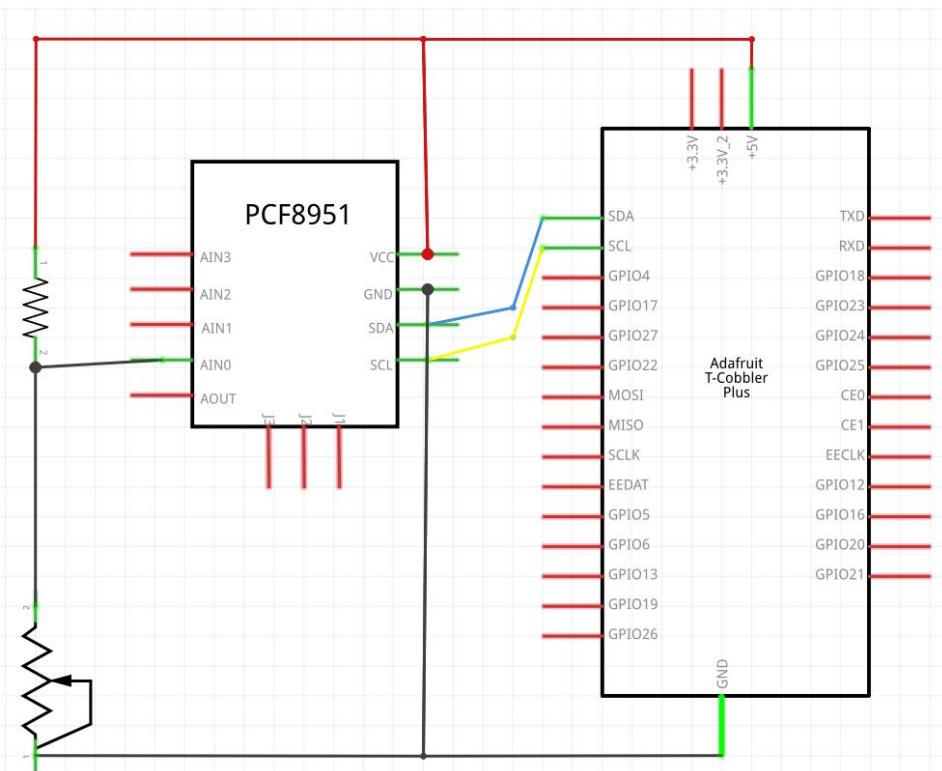
Product Introduction

Thermistor

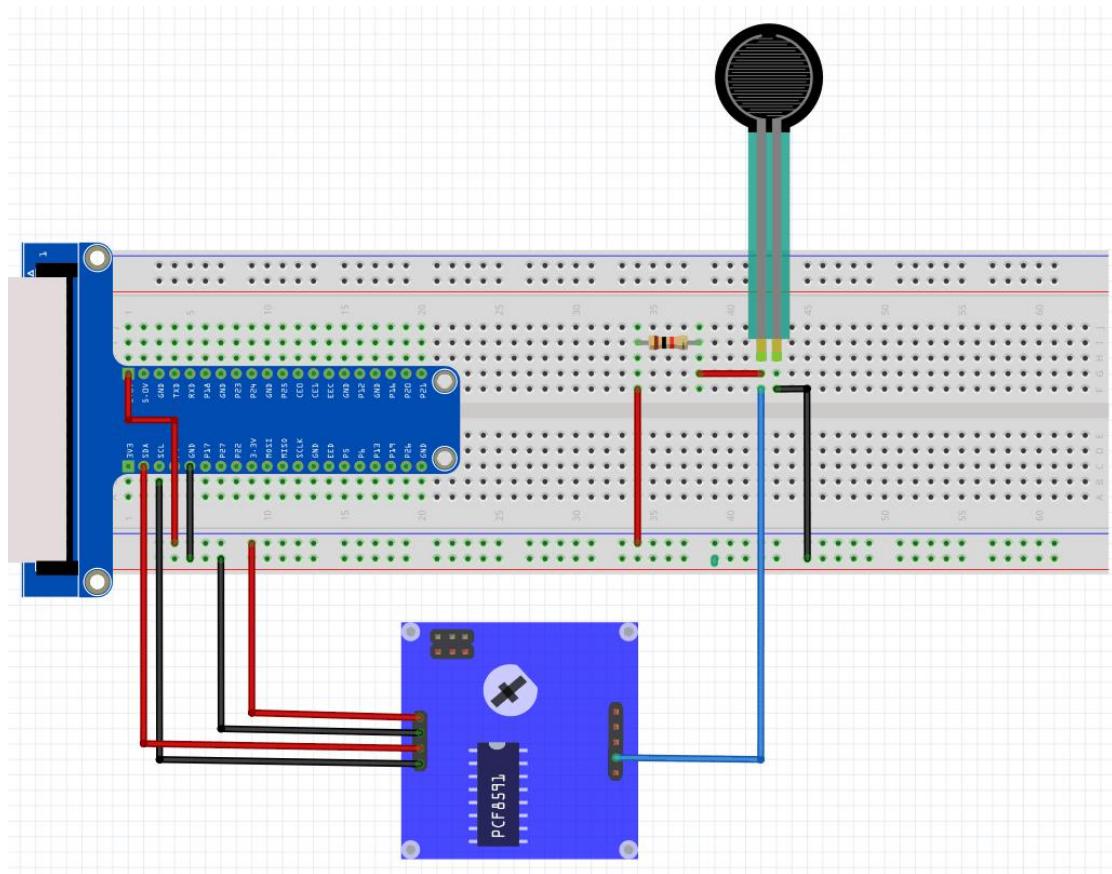
Thermistors are a type of sensitive components. They are divided into positive temperature coefficient thermistors (PTC) and negative temperature coefficient thermistors (NTC) according to different temperature coefficients. The typical characteristic of a thermistor is that it is sensitive to temperature 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.



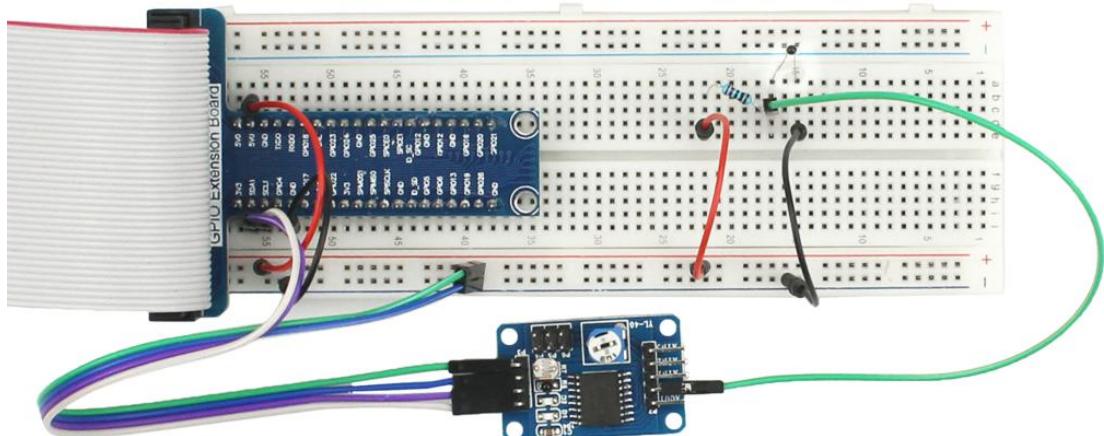
Connection Diagram



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;

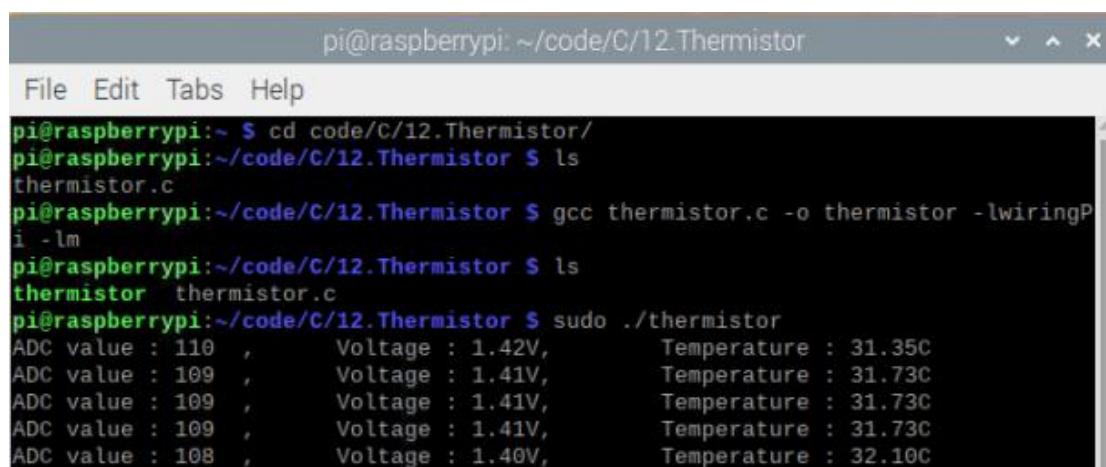
Enter the [ls](#) command to view the thermistor.c file in the directory;



```
pi@raspberrypi:~/code/C/12.Thermistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/12.Thermistor/
pi@raspberrypi:~/code/C/12.Thermistor $ ls
thermistor.c
pi@raspberrypi:~/code/C/12.Thermistor $
```

Enter the [gcc thermistor.c -o thermistor -lwiringPi -lm](#) command to generate the thermistor.c executable file thermistor, and enter the ls command to view;

Enter the [sudo ./thermistor](#) command to run the code. The result is as follows:



```
pi@raspberrypi:~/code/C/12.Thermistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/12.Thermistor/
pi@raspberrypi:~/code/C/12.Thermistor $ ls
thermistor.c
pi@raspberrypi:~/code/C/12.Thermistor $ gcc thermistor.c -o thermistor -lwiringPi -lm
pi@raspberrypi:~/code/C/12.Thermistor $ ls
thermistor thermistor.c
pi@raspberrypi:~/code/C/12.Thermistor $ sudo ./thermistor
ADC value : 110 , Voltage : 1.42V, Temperature : 31.35C
ADC value : 109 , Voltage : 1.41V, Temperature : 31.73C
ADC value : 109 , Voltage : 1.41V, Temperature : 31.73C
ADC value : 109 , Voltage : 1.41V, Temperature : 31.73C
ADC value : 108 , Voltage : 1.40V, Temperature : 32.10C
```

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);
}
```

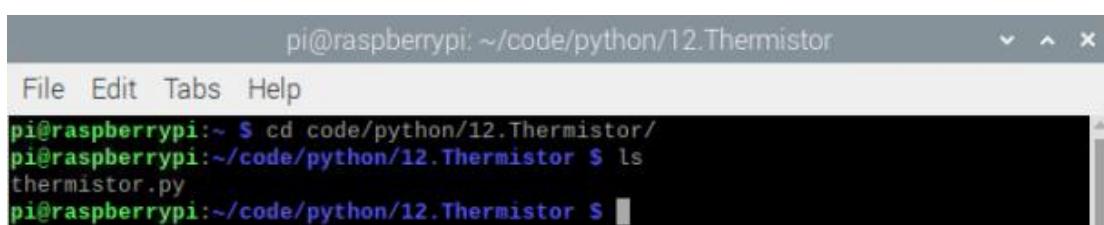
In the code, read the ADC value of the PCF8591 A0 port, then calculate the voltage and resistance.

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

Python code

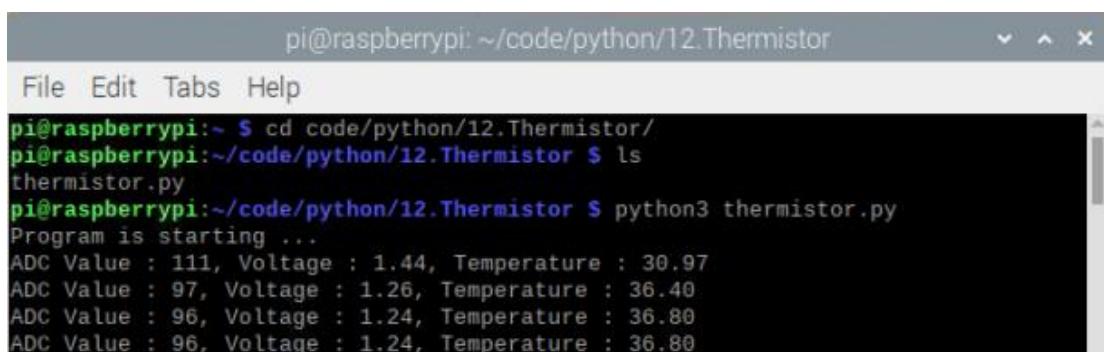
Open the terminal and use the `cd code / python / 12.Thermistor /` command to enter the code directory;

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



```
pi@raspberrypi: ~code/python/12.Thermistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/12.Thermistor/
pi@raspberrypi:~/code/python/12.Thermistor $ ls
thermistor.py
pi@raspberrypi:~/code/python/12.Thermistor $
```

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



```
pi@raspberrypi: ~code/python/12.Thermistor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/12.Thermistor/
pi@raspberrypi:~/code/python/12.Thermistor $ ls
thermistor.py
pi@raspberrypi:~/code/python/12.Thermistor $ python3 thermistor.py
Program is starting ...
ADC Value : 111, Voltage : 1.44, Temperature : 30.97
ADC Value : 97, Voltage : 1.26, Temperature : 36.40
ADC Value : 96, Voltage : 1.24, Temperature : 36.80
ADC Value : 96, Voltage : 1.24, Temperature : 36.80
```

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 Joystick

Overview

In this lesson you will learn how to use a Raspberry Pi to control a joystick sensor.

Parts Required

- 1 x Raspberry Pi
- 1 x PCF8591 Module
- 11 x Jumper Wires
- 1 x Breadboard
- 1 x Joystick

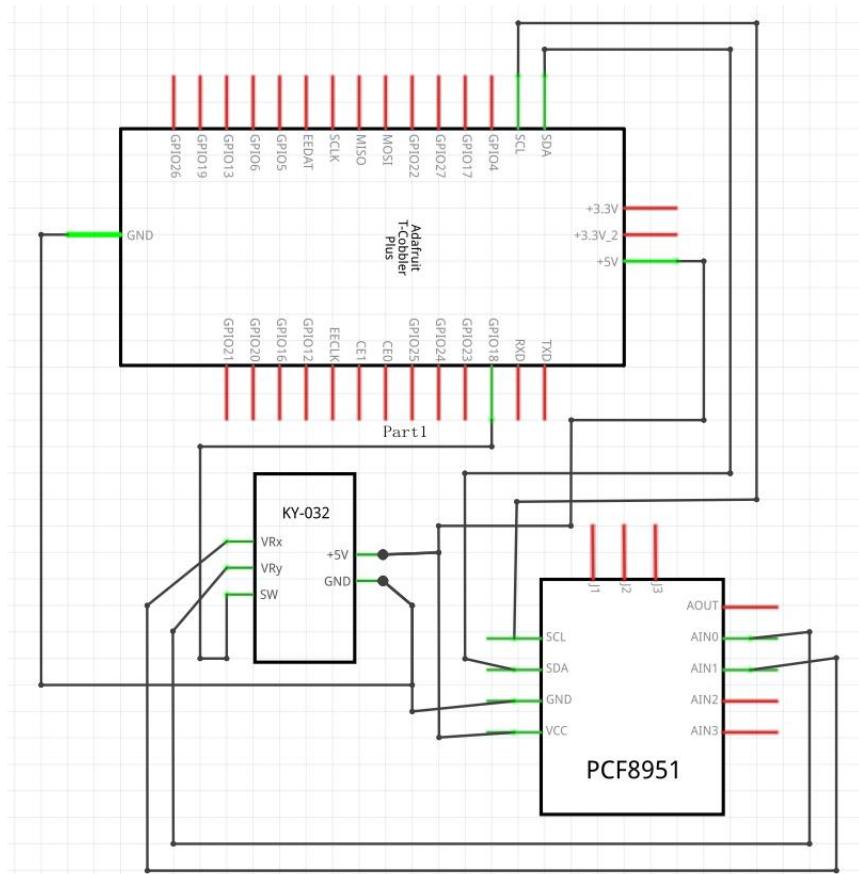
Product Introduction

Joystick

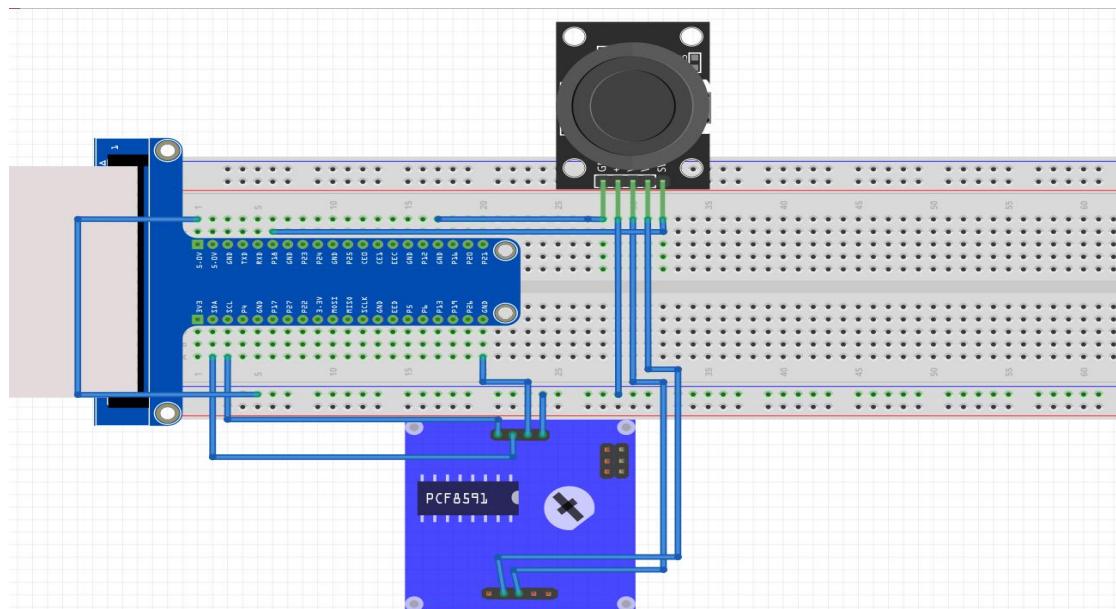
Joystick is a kind of sensor used with your fingers, which is widely used in gamepad and remote controller. It can shift in direction Y or direction X at the same time. And it can also be pressed in direction Z.



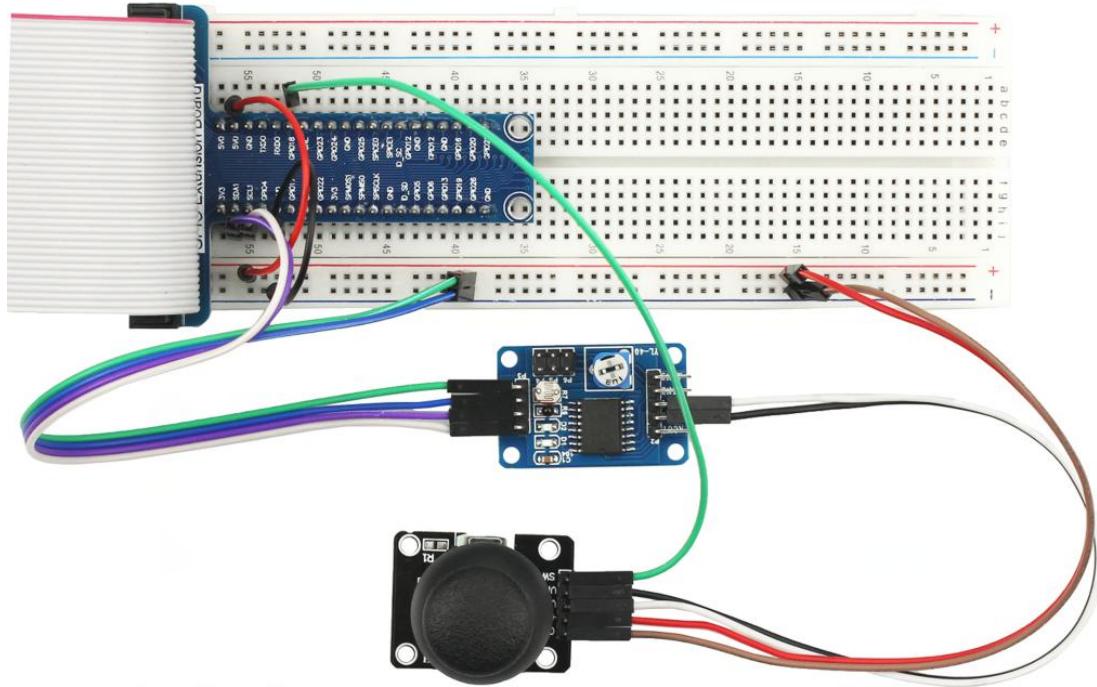
Connection diagram



Wiring diagram



Example Picture



C code

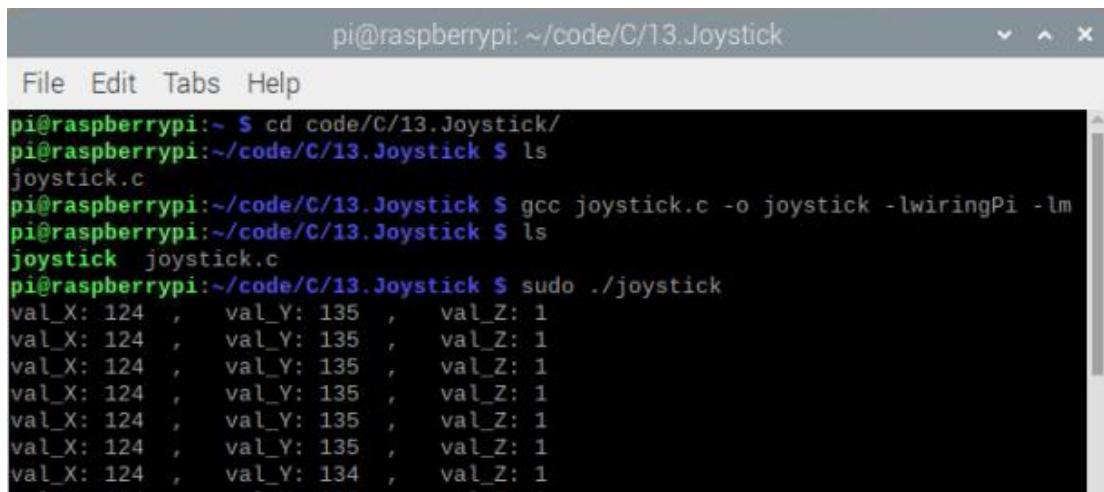
Open the terminal and enter the `cd code / C / 13.Joystick /` command to enter the joystick.c code directory;

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

```
pi@raspberrypi:~/code/C/13.Joystick
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/13.Joystick/
pi@raspberrypi:~/code/C/13.Joystick $ ls
joystick.c
pi@raspberrypi:~/code/C/13.Joystick $
```

Enter the `gcc joystick.c -o joystick -lwiringPi -lm` command to generate `joystick.c` as an executable file `joystick`, and enter the `ls` command to view;

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



The screenshot shows a terminal window titled "pi@raspberrypi: ~/code/C/13.Joystick". The window contains the following text:

```
pi@raspberrypi:~$ cd code/C/13.Joystick/
pi@raspberrypi:~/code/C/13.Joystick $ ls
joystick.c
pi@raspberrypi:~/code/C/13.Joystick $ gcc joystick.c -o joystick -lwiringPi -lm
pi@raspberrypi:~/code/C/13.Joystick $ ls
joystick joystick.c
pi@raspberrypi:~/code/C/13.Joystick $ sudo ./joystick
val_X: 124 , val_Y: 135 , val_Z: 1
val_X: 124 , val_Y: 134 , val_Z: 1
```

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 Z_Pin 1

int main(void){
    int val_X,val_Y,val_Z;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    pinMode(Z_Pin,INPUT);
    pullUpDnControl(Z_Pin,PUD_UP);
    pcf8591Setup(pinbase,address);
```

```
while(1){  
    val_Z = digitalRead(Z_Pin);  
    val_Y = analogRead(A0);  
    val_X = analogRead(A1);  
    printf("val_X: %d \tval_Y: %d \tval_Z: %d  
\n",val_X,val_Y,val_Z);  
    delay(100);  
}  
return 0;  
}
```

Code Interpretation

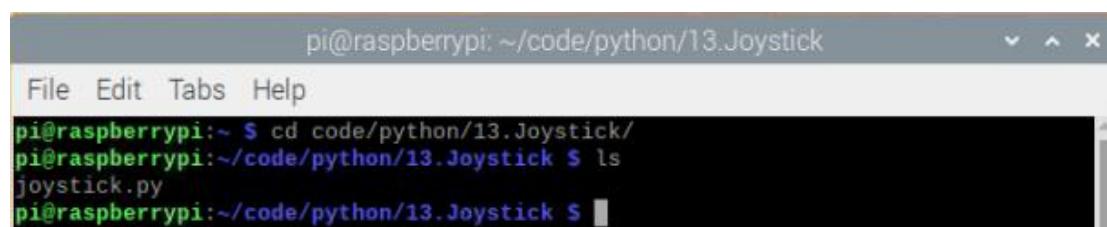
```
while(1){  
    val_Z = digitalRead(Z_Pin);  
    val_Y = analogRead(A0);  
    val_X = analogRead(A1);  
    printf("val_X: %d \tval_Y: %d \tval_Z: %d  
\n",val_X,val_Y,val_Z);  
    delay(100);  
}
```

In the code, configure Z_Pin as a pull-up input mode. In the while loop of the main function, use AnalogRead () to read the values of the X and Y axes, and digitalRead () to read the values of the Z axis, and then print them out.

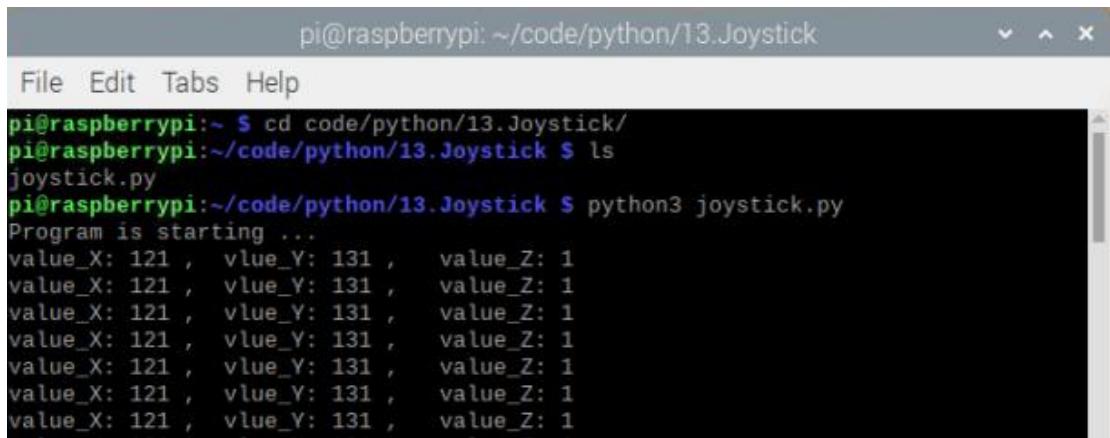
Python code

Open the terminal and use the `cd code / python / 13.Joystick /` command to enter the code directory

Enter the command `ls` to view the file joystick.py in the directory.



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



```
pi@raspberrypi: ~ /code/python/13.Joystick
File Edit Tabs Help
pi@raspberrypi: ~ $ cd code/python/13.Joystick/
pi@raspberrypi: ~/code/python/13.Joystick $ ls
joystick.py
pi@raspberrypi:~/code/python/13.Joystick $ python3 joystick.py
Program is starting ...
value_X: 121 ,  value_Y: 131 ,  value_Z: 1
```

The following is the code:

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

address = 0x48
bus=smbus.SMBus(1)
cmd=0x40
Z_Pin = 12
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():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(Z_Pin,GPIO.IN,GPIO.PUD_UP)
def loop():
    while True:
        val_Z = GPIO.input(Z_Pin)
        val_Y = analogRead(0)
        val_X = analogRead(1)
```

```
        print
('value_X: %d ,\tvalue_Y: %d ,\tvalue_Z: %d' %(val_X,val_Y,val_Z))
        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 loop():
    while True:
        val_Z = GPIO.input(Z_Pin)
        val_Y = analogRead(0)
        val_X = analogRead(1)
        print
('value_X: %d ,\tvalue_Y: %d ,\tvalue_Z: %d' %(val_X,val_Y,val_Z))
        time.sleep(0.01)
```

In the while loop, use 'analogRead ()' to read the values of the X and Y axes, and use 'GPIO.input ()' to read the values of the Z axis, and then use 'print (' value_X:% d, \ tvlue_Y :% d, \ tvalue_Z:% d ' % (val_X, val_Y, val_Z))' output.

Lesson 14 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 $1k\Omega$ 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

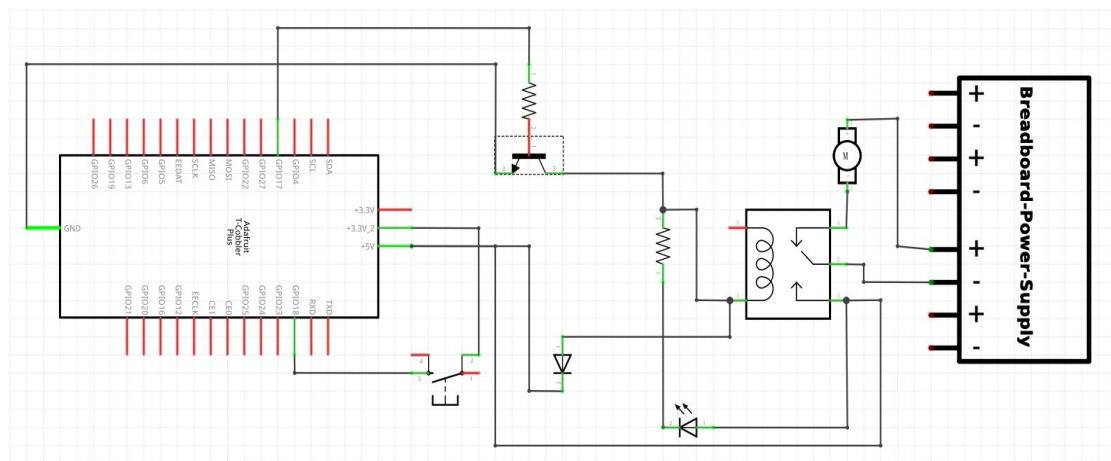
Motor is a device that converts electrical energy into mechanical energy. Motor consists of two parts: stator and rotor. When motor works, the stationary part is stator, and the rotating part is rotor. Stator is usually the outer case of motor, and it has terminals to connect to the power. Rotor is usually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



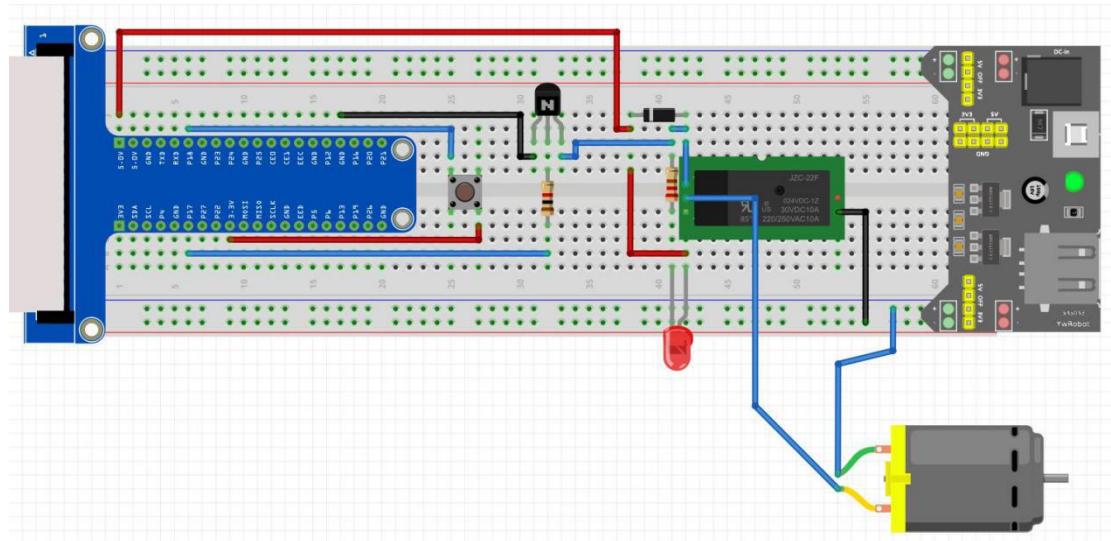
Circuit

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.

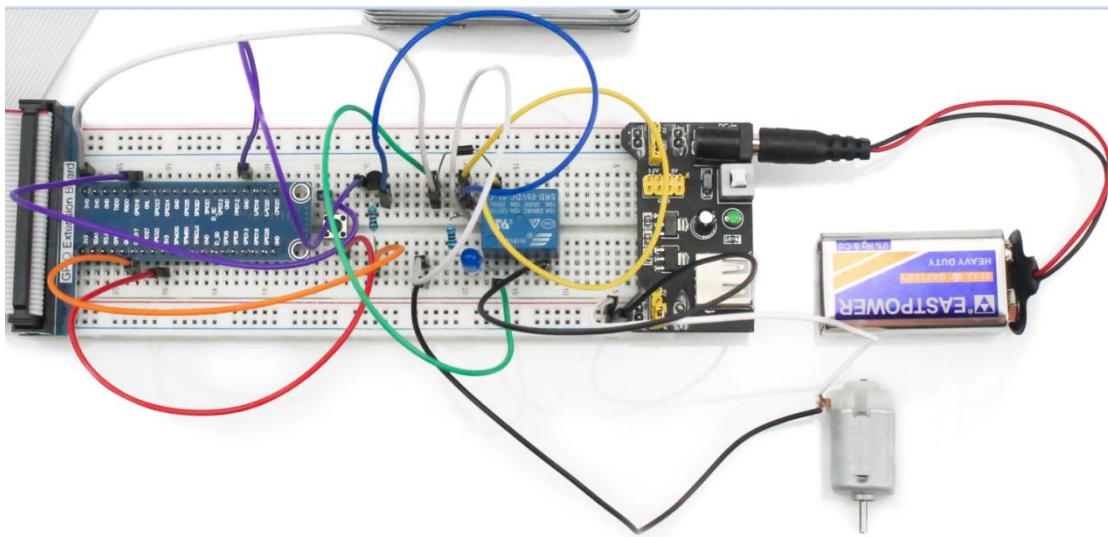
Connection diagram



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;

```
pi@raspberrypi: ~/code/C/14.Relay
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/14.Relay/
pi@raspberrypi:~/code/C/14.Relay $ ls
Relay.c
pi@raspberrypi:~/code/C/14.Relay $
```

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:

```
pi@raspberrypi: ~/code/C/14.Relay
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/14.Relay/
pi@raspberrypi:~/code/C/14.Relay $ ls
Relay.c
pi@raspberrypi:~/code/C/14.Relay $ gcc Relay.c -o Relay -lwiringPi
pi@raspberrypi:~/code/C/14.Relay $ ls
Relay Relay.c
pi@raspberrypi:~/code/C/14.Relay $ sudo ./Relay
starting...
pi@raspberrypi:~/code/C/14.Relay $
```

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 message to 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.

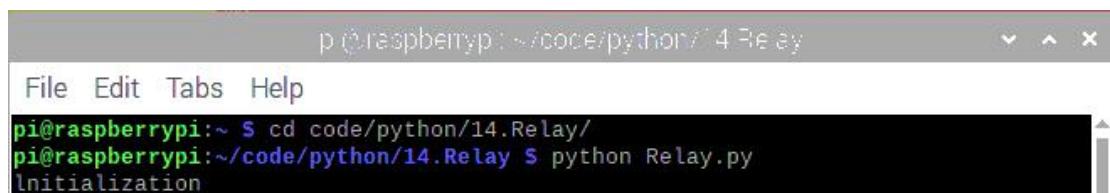
```

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.



```

pi@raspberrypi:~$ cd code/python/14.Relay/
pi@raspberrypi:~/code/python/14.Relay $ python Relay.py
Initialization

```

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__':
    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 15 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, sensor and 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.

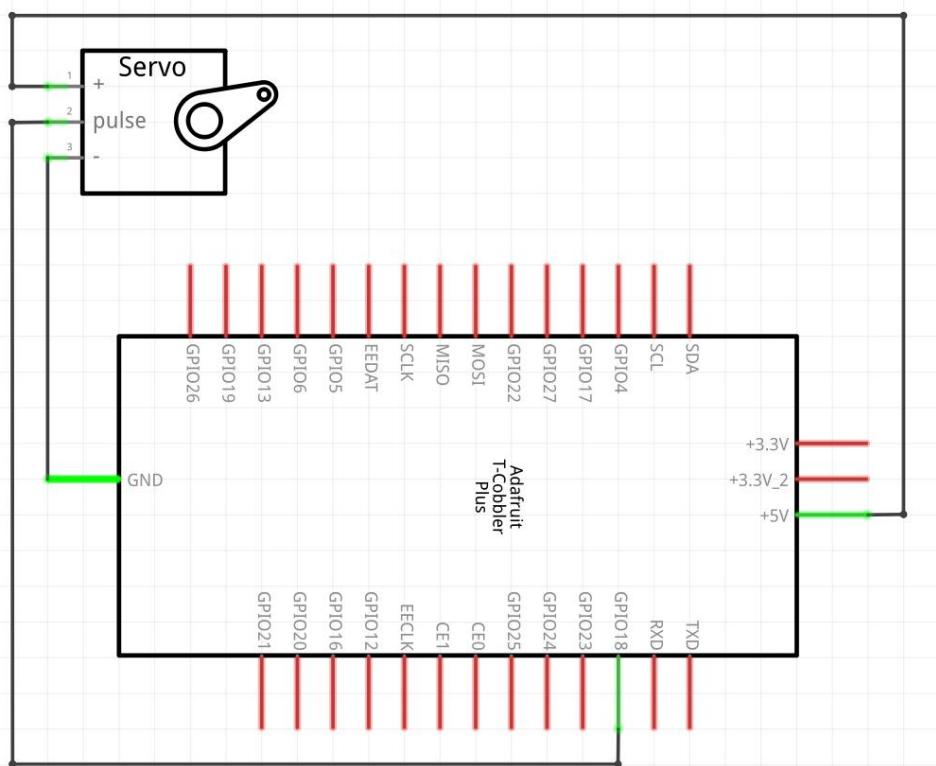


Part of the corresponding values are as follows:

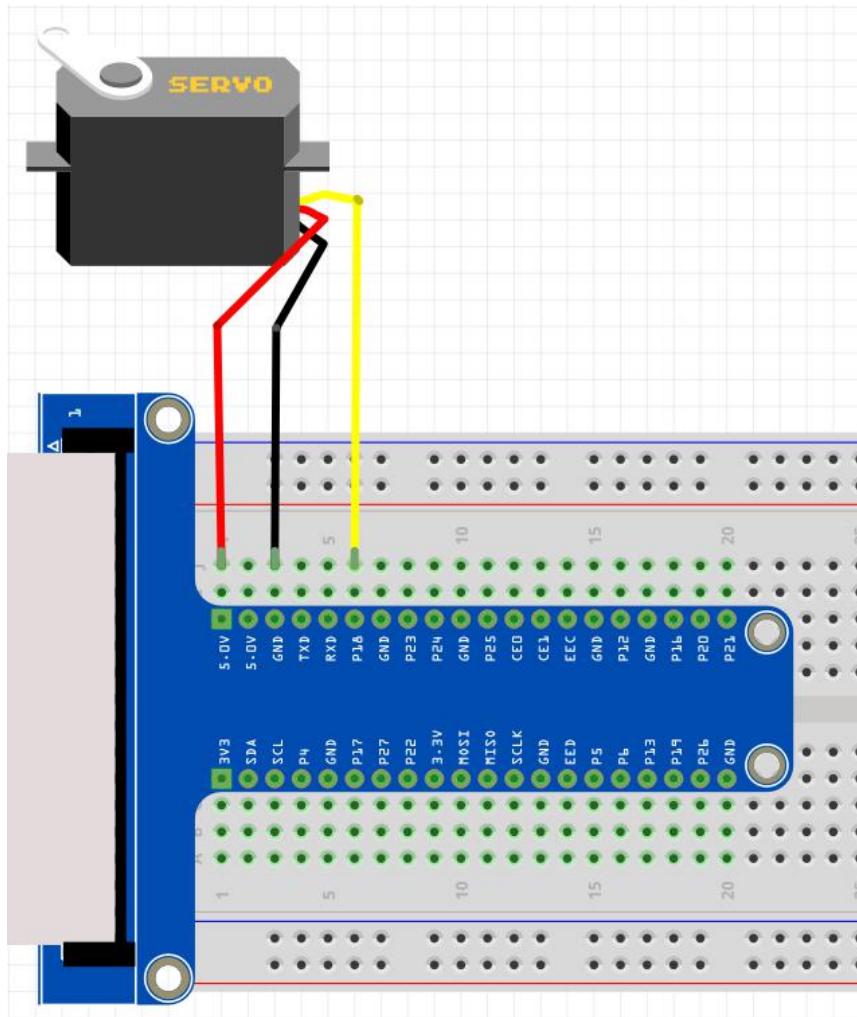
High level time	Servo angle
0.5ms	0 degree
1ms	45 degree
1.5ms	90 degree
2ms	135 degree
2.5ms	180 degree

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

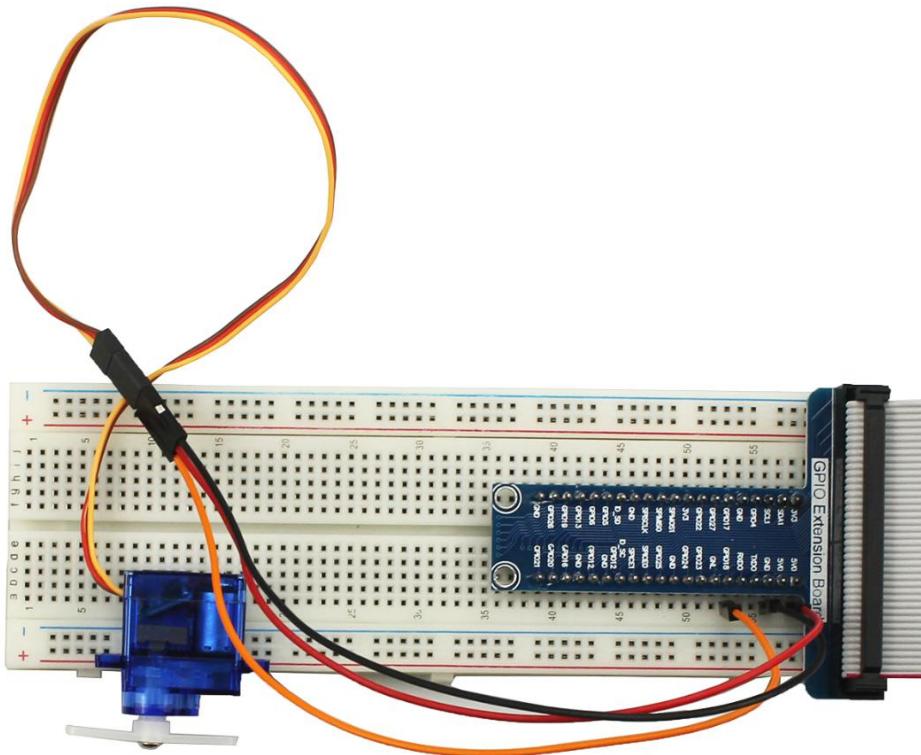
Connection Diagram



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;



```
pi@raspberrypi:~/code/C/15.Servo
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/15.Servo/
pi@raspberrypi:~/code/C/15.Servo $ ls
Servo.c
pi@raspberrypi:~/code/C/15.Servo $
```

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:



```
pi@raspberrypi:~ $ cd code/C/15.Servo/  
pi@raspberrypi:~/code/C/15.Servo $ ls  
Servo.c  
pi@raspberrypi:~/code/C/15.Servo $ gcc Servo.c -o Servo -lwiringPi  
pi@raspberrypi:~/code/C/15.Servo $ ls  
Servo  Servo.c  
pi@raspberrypi:~/code/C/15.Servo $ sudo ./Servo  
starting ...
```

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 message to
screen
        printf("setup wiringPi faiservo !");
        return 1;
    }
    printf("starting ...\\n");
    servoInit(servoPin);           //initialize PMW pin of servo
    while(1){
        for(i=SEROV_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=SEROV_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 order to get the PWM with cycle of 20ms, the pwmRange should 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 written 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));
}
```

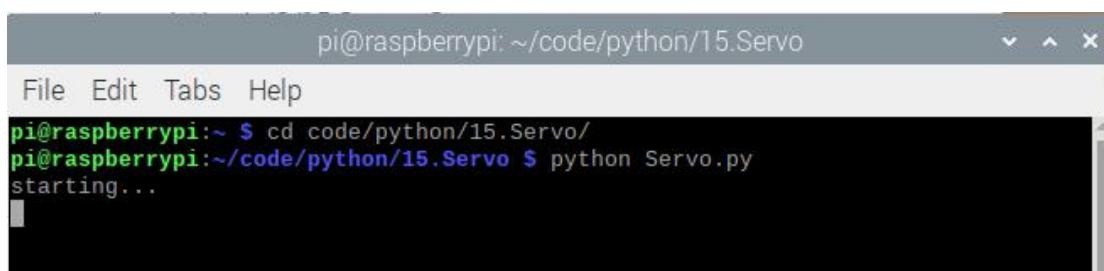
Finally, in the "while" cycle of main function, use two "for" cycle to make servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
while(1){
    for(i=SEROVO_MIN_MS;i<SERVO_MAX_MS;i++){
        servoWriteMS(servoPin,i);
        delay(10);
    }
    delay(1000);
    for(i=SEROVO_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.



A screenshot of a terminal window titled "pi@raspberrypi: ~/code/python/15.Servo". The window shows the following command-line session:

```
pi@raspberrypi:~ $ cd code/python/15.Servo/
pi@raspberrypi:~/code/python/15.Servo $ python Servo.py
starting...
```

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 16 Stepping Motor

Overview

In this lesson you will learn how to control the rotation of a stepper motor by Raspberry pi, and how to control the angle of rotation of a stepper motor. We have learned DC motor and servo before: the DC motor can rotate constantly but we cannot make it rotate to a specific angle. On the contrary, the ordinary servo can rotate to a certain angle but can not rotate constantly. In this chapter, we will learn a motor which can rotate not only constantly, but also to a specific angle, stepping motor. Using stepping motor can achieve higher accuracy of mechanical motion easily.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board

1 x Breadboard

1 x Stepper Motor

1 x ULN2003 Stepper Motor Driver

Product Introduction

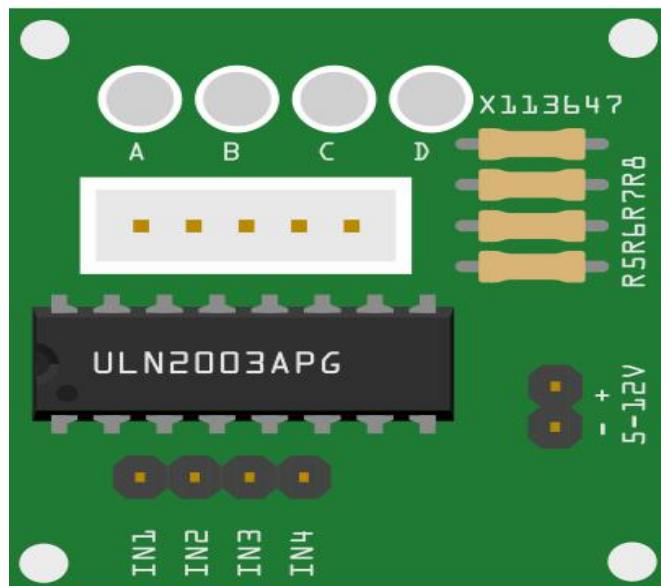
Stepping Motor

Stepping motor is an open-loop control device which converts the electric pulse signal into angular displacement or linear displacement. In non-overload condition, the speed of the motor and the location of the stop depends only on the pulse signal frequency and pulse number, and not affected by the load changes. The outside piece is the stator and the inside is the rotor of the motor. There are a certain number of coils, usually integer multiple of phases number, in the stator and when powered on, an electromagnet will be formed to attract a convex part (usually iron or permanent magnet) of the rotor. Therefore, the electric motor can be driven by conducting the coils on stator orderly. By controlling the number of rotation steps, you can control the stepping motor rotation angle. By controlling the time between two steps, you can control the stepping motor rotation speed. When rotating clockwise, the order of coil powered on is: A B C D A..... . And the rotor will rotate in accordance with the order, step by step down, called four steps four pats. If the coils are powered on

in the reverse order, D C B A D..., the rotor will rotate in anti-clockwise direction. Stepping motor has other control methods, such as connect A phase, then connect A B phase, the stator will be located in the middle of the A B, only a half-step. This way can improve the stability of stepping motor, and reduce noise, the sequence of coil powered on is: A AB B BC C CD D DA A....., the rotor will rotate in accordance with the order, a half step by a half step, called four step eight pat. Equally, if the coil is powered on in reverse order, the stepping motor will rotate in reverse rotation. The stator of stepping motor we use has 32 magnetic poles, so a circle needs 32 steps. The output shaft of the stepping motor is connected with a reduction gear set, and the reduction ratio is 1/64. So the final output shaft rotates a circle requiring a $32 \times 64 = 2048$ step.

ULN2003 Stepping motor driver

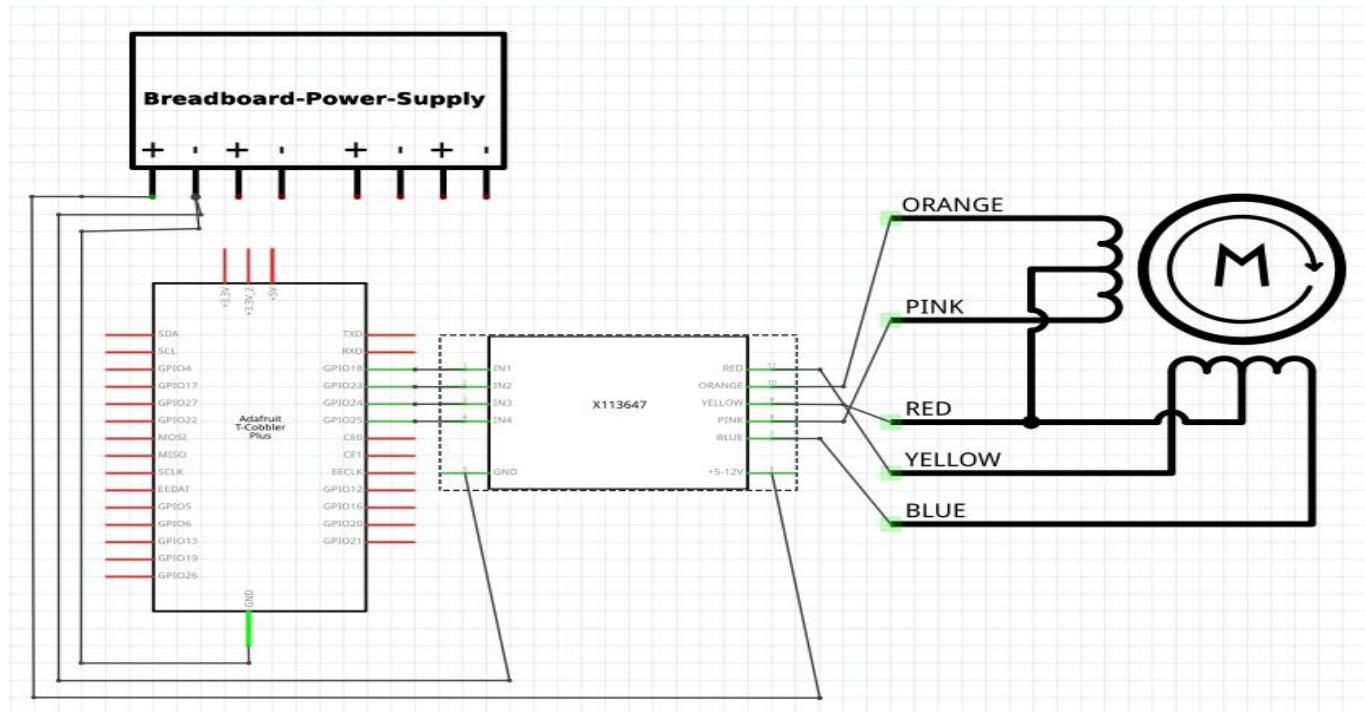
ULN2003 stepping motor driver is used to convert the weak signal into powerful control signal to drive the stepping motor. The input signal IN1-IN4 corresponds to the output signal A-D, and 4 LED is integrated in the board to indicate the state of signals. The PWR interface can be used as a power supply for stepping motor. By default, PWR and VCC are connected by a short circuit.



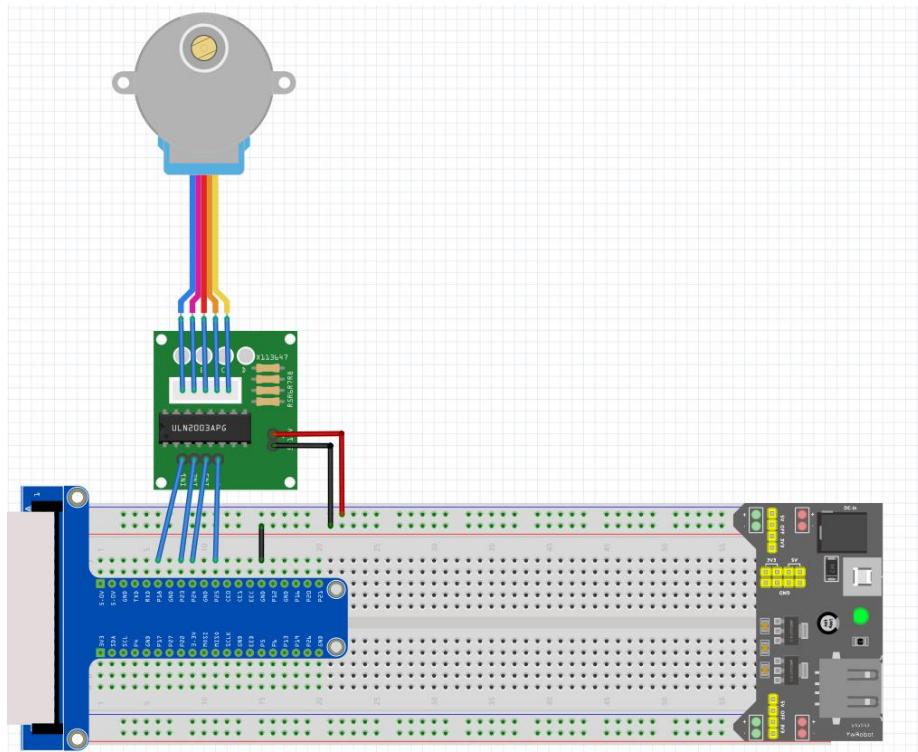
Circuit

When building the circuit, the rated voltage of the stepping motor 5V, and use the breadboard power supply independently, and do not use the RPi power supply. Additionally, breadboard power supply needs to share Ground with RPi.

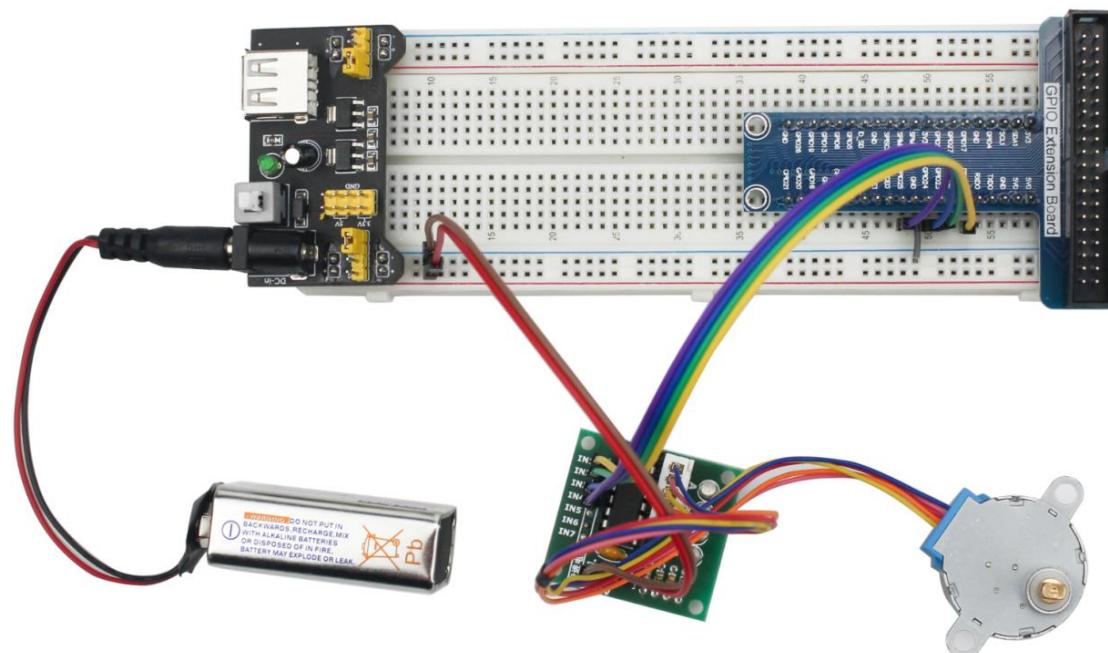
Connection Diagram



Wiring Diagram



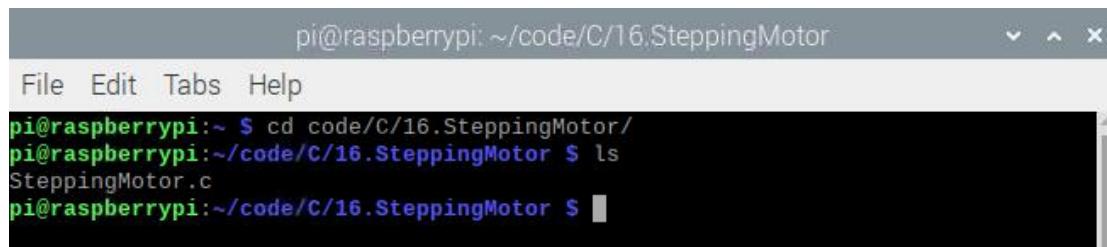
Example Figure



C code

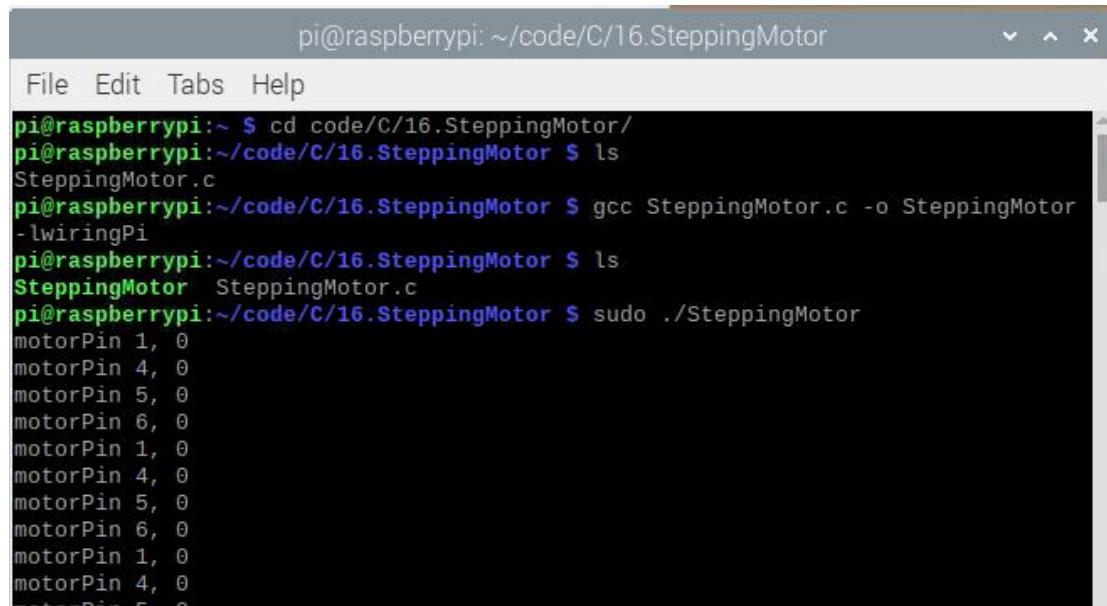
Open terminal and enter "cd code / C / 16.SteppingMotor /" command to enter "SteppingMotor" code directory;

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



```
pi@raspberrypi:~/code/C/16.SteppingMotor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/16.SteppingMotor/
pi@raspberrypi:~/code/C/16.SteppingMotor $ ls
SteppingMotor.c
pi@raspberrypi:~/code/C/16.SteppingMotor $
```

Enter "gcc SteppingMotor.c -o SteppingMotor -lwiringPi" command to generate "SteppingMotor.c" executable file "SteppingMotor", enter "ls" command to view; enter "sudo ./SteppingMotor" command to run the code, the results are shown below :



```
pi@raspberrypi:~/code/C/16.SteppingMotor
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/16.SteppingMotor/
pi@raspberrypi:~/code/C/16.SteppingMotor $ ls
SteppingMotor.c
pi@raspberrypi:~/code/C/16.SteppingMotor $ gcc SteppingMotor.c -o SteppingMotor -lwiringPi
pi@raspberrypi:~/code/C/16.SteppingMotor $ ls
SteppingMotor SteppingMotor.c
pi@raspberrypi:~/code/C/16.SteppingMotor $ sudo ./SteppingMotor
motorPin 1, 0
motorPin 4, 0
motorPin 5, 0
motorPin 6, 0
motorPin 1, 0
motorPin 4, 0
motorPin 5, 0
motorPin 6, 0
motorPin 1, 0
motorPin 4, 0
motorPin 5, 0
```

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

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

const int motorPins[]={1,4,5,6};      //define pins connected to four phase ABCD of
stepper motor
const int CCWStep[]={0x01,0x02,0x04,0x08}; //define power supply order for coil
```

```
for rotating anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};    //define power supply order for coil
for rotating clockwise
//as for four phase stepping motor, four steps is a cycle. the function is used to drive
the stepping motor clockwise or anticlockwise to take four steps
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order
        for (i=0;i<4;i++){ //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH :
LOW);
            else          //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH :
LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        if(ms<3)          //the delay can not be less than 3ms, otherwise it will
exceed speed limit of the motor
            ms=3;
        delay(ms);
    }
}
//continuous rotation function, the parameter steps specifies the rotation cycles, every
four steps is a cycle
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
int main(void){
    int i;

    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    for(i=0;i<4;i++){
        pinMode(motorPins[i],OUTPUT);
    }
```

```

while(1){
    moveSteps(1,3,512);      //rotating 360° clockwise, a total of 2048
steps in a circle, namely, 512 cycles.
    delay(1000);
    moveSteps(0,3,512);      //rotating 360° anticlockwise
    delay(1000);
}
return 0;
}

```

Code Interpretation

In the code, define four pins of stepper motor and coil power supply order of four steps rotation mode.

```

const int motorPins[]={1,4,5,6};
const int CCWStep[]={0x01,0x02,0x04,0x08};
const int CWStep[]={0x08,0x04,0x02,0x01};

```

Subfunction “moveOnePeriod (int dir, int ms)” will drive the stepping motor rotating four step clockwise oranticlockwise, four step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate forward, otherwise it rotates to reverse direction. Parameter "ms" indicates the time between eachtwo steps. The "ms" of stepping motor used in this project is 3ms (the shortest time), less than 3ms will exceed the speed limit of stepping motor resulting in that motor can not rotate.

```

void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){
        for (i=0;i<4;i++){
            if(dir == 1)
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH :
LOW);
            else
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH :
LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        if(ms<3)
            ms=3;
    }
}

```

```
    delay(ms);
}
}

Subfunction moveSteps (int dir, int ms, int steps) is used to specific cycle number of
stepping motor.

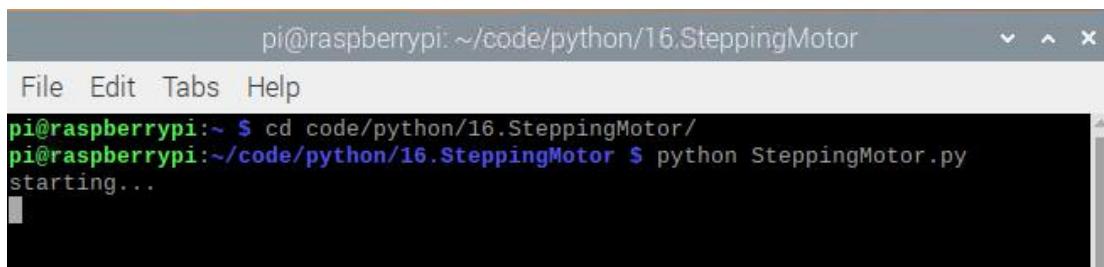
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
```

Finally, in the while cycle of main function, rotate one circle clockwise, and then one circle anticlockwise. According to the previous knowledge of the stepping motor, it can be known that the stepping motor rotation for one circle requires 2048 steps, that is, $2048/4=512$ cycle.

```
while(1){
    moveSteps(1,3,512);
    delay(1000);
    moveSteps(0,3,512);
    delay(1000);
}
```

Python code

1. Use the "cd code / python / 16.StepMotor /" command to enter the directory of the SteppingMotor code.
2. Use "python SteppingMotor.py" command to execute "SteppingMotor.py" code.



A terminal window titled "pi@raspberrypi: ~ /code/python/16.StepMotor". The window shows the following command sequence:

```
pi@raspberrypi:~ $ cd code/python/16.StepMotor/
pi@raspberrypi:~/code/python/16.StepMotor $ python SteppingMotor.py
starting...
```

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

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

```
import time
```

```
motorpins = (12,16,18,22)      #define pins connected to four phase ABCD of stepper  
motor
```

```
Reverse = (0x01,0x02,0x04,0x08) #define power supply order for coil for rotating  
anticlockwise
```

```
Forward = (0x08,0x04,0x02,0x01) #define power supply order for coil for rotating  
clockwise
```

```
def setup():
```

```
    print("starting...")  
    GPIO.setmode(GPIO.BRD)  
    for pin in motorpins:  
        GPIO.setup(pin,GPIO.OUT)
```

```
def motorang(dir,ms):
```

```
    for j in range(0,4,1):  
        for i in range(0,4,1):  
            if (dir == 1):  
                GPIO.output(motorpins[i],((Reverse[j] == 1<<i) and GPIO.HIGH  
or GPIO.LOW))  
            else:  
                GPIO.output(motorpins[i],((Forward[j] == 1<<i) and  
GPIO.HIGH or GPIO.LOW))  
            if(ms<3):  
                ms = 3  
            time.sleep(ms*0.001)
```

```
def motordirang(dir,ms,steps):
```

```
    for i in range(steps):  
        motorang(dir,ms)
```

```
def loop():
```

```
    while True:  
        motordirang(1,3,512)  
        time.sleep(0.5)  
        motordirang(0,3,512)  
        time.sleep(0.5)
```

```
def destroy():
```

```
    GPIO.cleanup()
```

```

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

Code Interpretation

In the code, define four pins of the stepper motor and coil power supply order of four steps rotation mode.

```

motorpins = (12,16,18,22)      #define pins connected to four phase ABCD of stepper
motor
Reverse = (0x01,0x02,0x04,0x08) #define power supply order for coil for rotating
anticlockwise
Forward = (0x08,0x04,0x02,0x01) #define power supply order for coil for rotating
clockwise
    
```

Subfunction motorang(dir,ms) will drive the stepping motor rotating four step clockwise or anticlockwise, four steps as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate forward, otherwise it rotates to reverse direction. Parameter "ms" indicates the time between each two steps. The "ms" of stepping motor used in this project is 3ms (the shortest time), less than 3ms will exceed the speed limit of stepping motor resulting in that motor can not rotate.

```

def motorang(dir,ms):
    for j in range(0,4,1):
        for i in range(0,4,1):
            if (dir == 1):
                GPIO.output(motorpins[i],((Reverse[j] == 1<<i) and GPIO.HIGH
or GPIO.LOW))
            else:
                GPIO.output(motorpins[i],((Forward[j] == 1<<i) and
GPIO.HIGH or GPIO.LOW))
            if(ms<3):
                ms = 3
            time.sleep(ms*0.001)
    
```

Subfunction motordirang (dir, ms, steps) is used for the number of times of the stepper motor and is used to control the rotation angle of the stepper motor.

```
def motordirang(dir,ms,steps):
    for i in range(steps):
        motorang(dir,ms)
```

In the 'while' loop of the main function 'loop()', first call the function 'motordirang(1,3,512)' to rotate clockwise, call the function 'time.sleep(0.5)' to stop 0.5m, and then call the function 'Motordirang(0,3,512)' rotate counterclockwise once, and then call the function 'time.sleep(0.5)' to stop 0.5m. According to the understanding of stepping motors, it can be known that one rotation of stepping motors requires 2048 steps, that is, $2048/4 = 512$ cycles.

```
def loop():
    while True:
        motordirang(1,3,512)
        time.sleep(0.5)
        motordirang(0,3,512)
        time.sleep(0.5)
```

Lesson 17 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

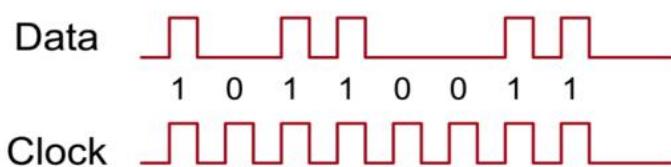
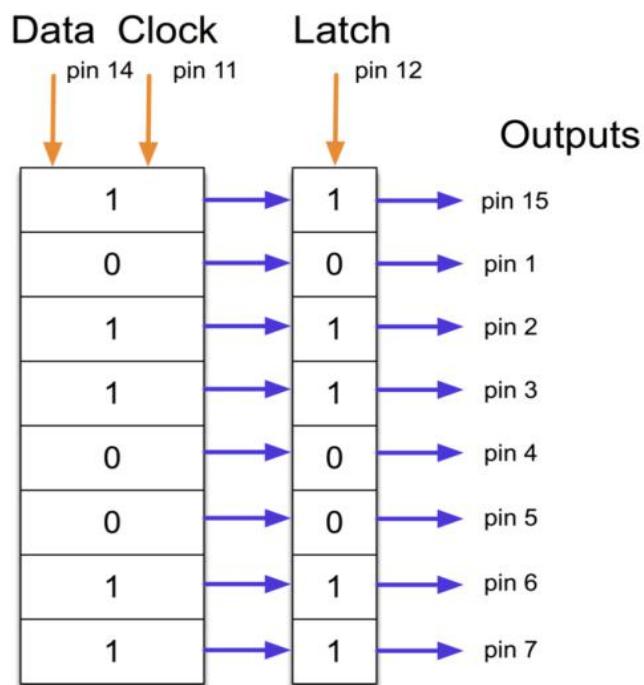
8 x 220Ω Resistor

8 x LED

Product Introduction

74HC595

74HC595 chip is used to convert serial data into parallel data. 74HC595 can convert the serial data of one 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.



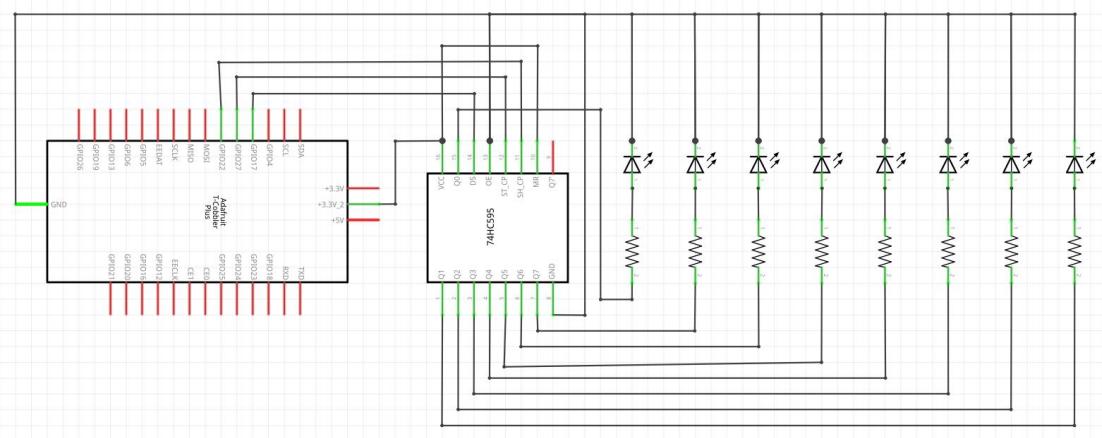
The clock pin needs to receive 8 pulses. In each pulse, if the data pin is high, 1 is pushed into the shift register; when the data pin is low, it is directly 0. When all 8 pulses are received, enabling the "Latch" pin will copy these 8 values to the shift register; this is necessary; otherwise, the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

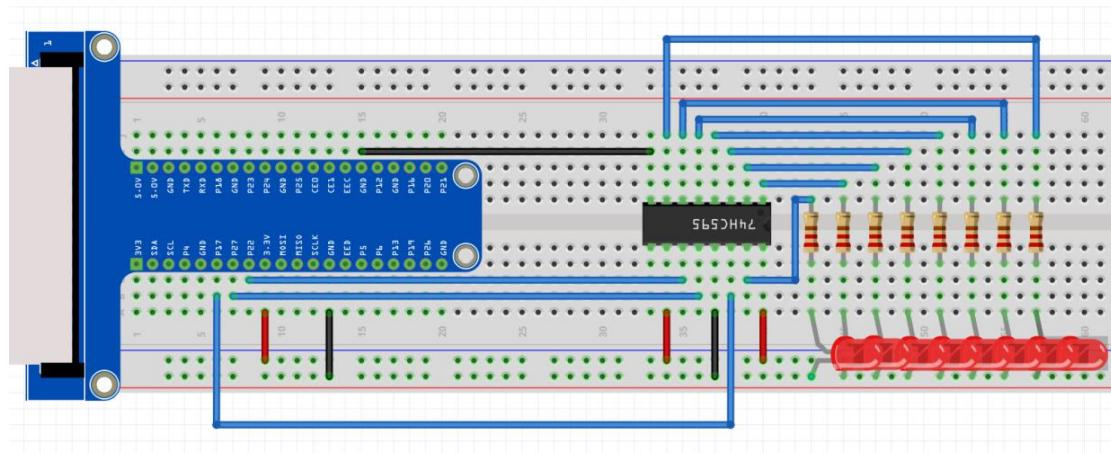
The ports of 74HC595 are described as follows:

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

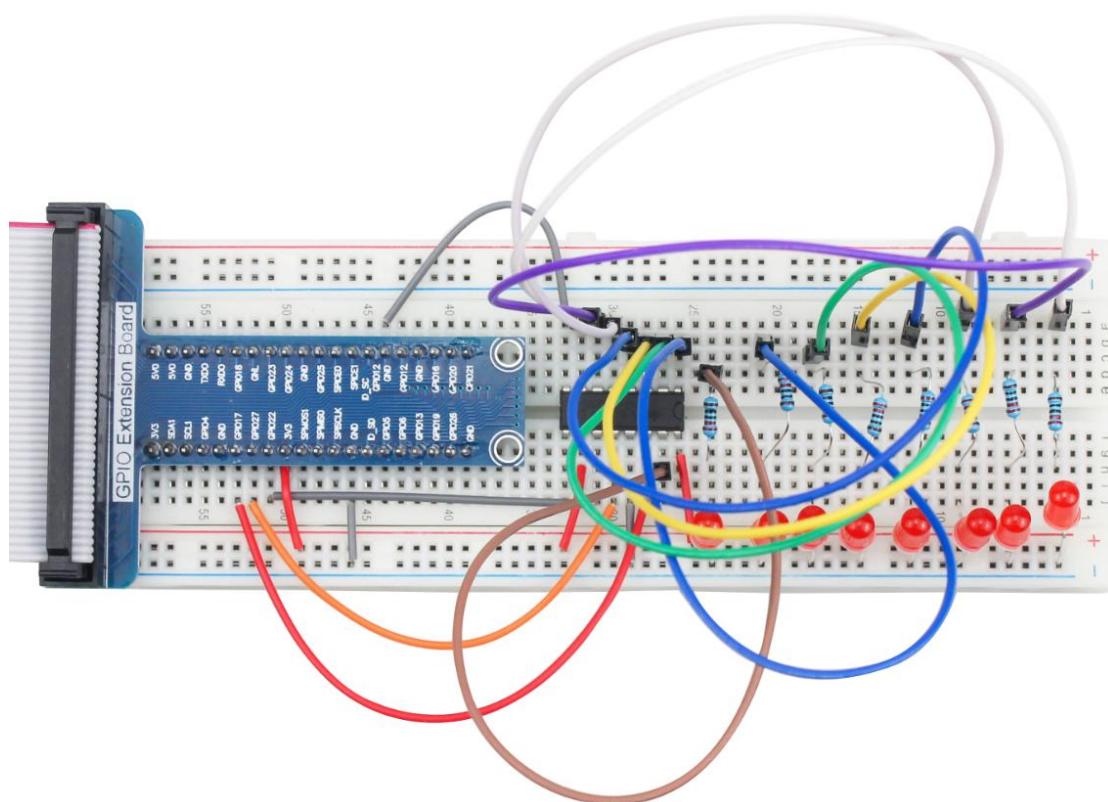
Connection Diagram



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;

```
pi@raspberrypi:~/code/C/17.74HC595LED
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/17.74HC595LED/
pi@raspberrypi:~/code/C/17.74HC595LED $ ls
74HC595LED.c
pi@raspberrypi:~/code/C/17.74HC595LED $
```

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 :



```
pi@raspberrypi: ~/code/C/17.74HC595LED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/17.74HC595LED/
pi@raspberrypi:~/code/C/17.74HC595LED $ ls
74HC595LED.c
pi@raspberrypi:~/code/C/17.74HC595LED $ gcc 74HC595LED.c -o 74HC595LED -lwiringPi
pi@raspberrypi:~/code/C/17.74HC595LED $ ls
74HC595LED  74HC595LED.c
pi@raspberrypi:~/code/C/17.74HC595LED $ sudo ./74HC595LED
```

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 message to 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 74HC595LED.py" command to execute "74HC595LED.py" code.



A screenshot of a terminal window on a Raspberry Pi. The title bar says "pi@raspberrypi: ~ /code/python/17.74HC595LED". The menu bar includes "File", "Edit", "Tabs", and "Help". The main window shows a command-line interface with the following text:
pi@raspberrypi:~ \$ cd code/python/17.74HC595LED/
pi@raspberrypi:~/code/python/17.74HC595LED \$ python 74HC595LED.py
starting...

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__':
    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 18 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

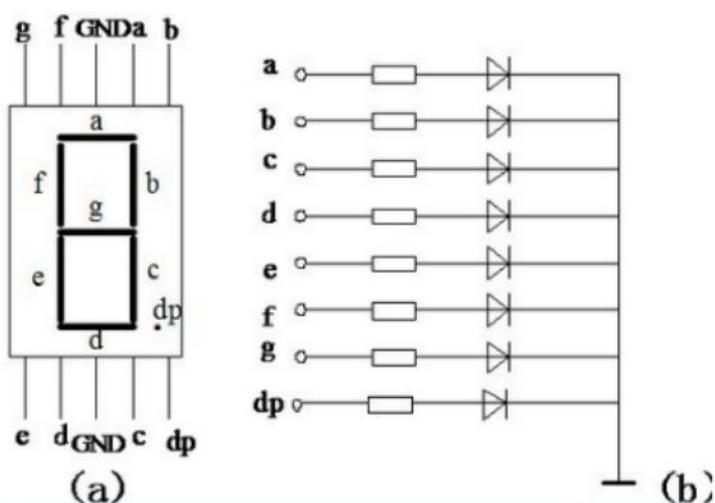
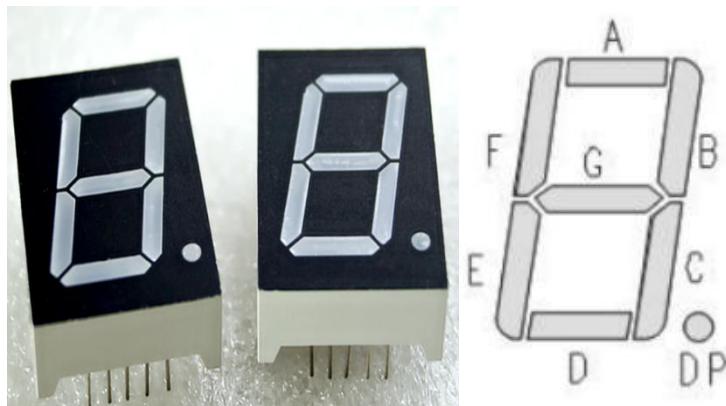
1 x 7 Segment Display

Product Introduction

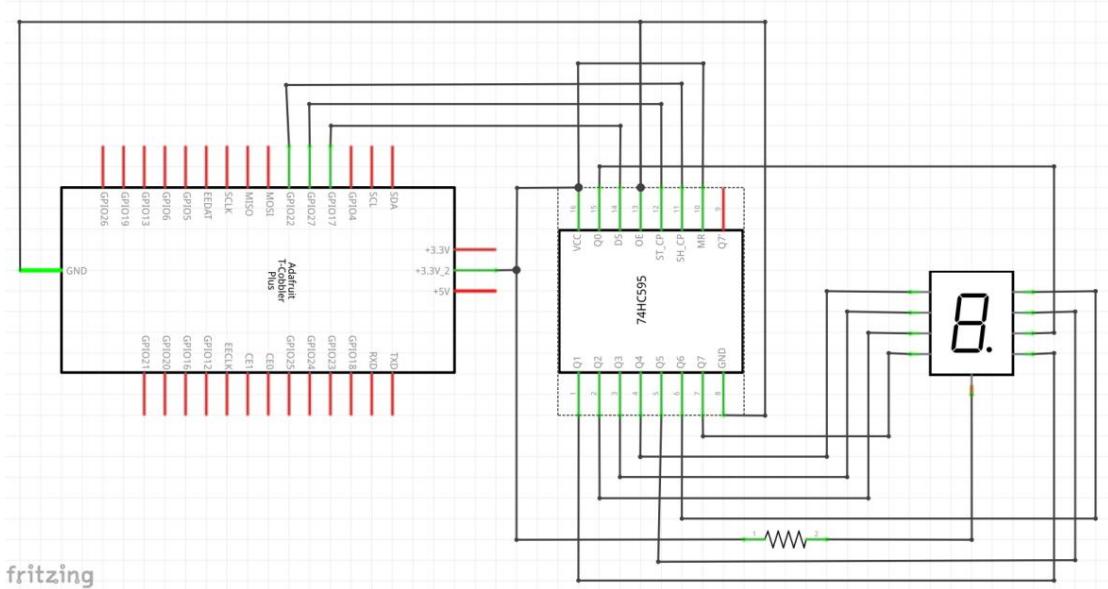
7-segment display

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

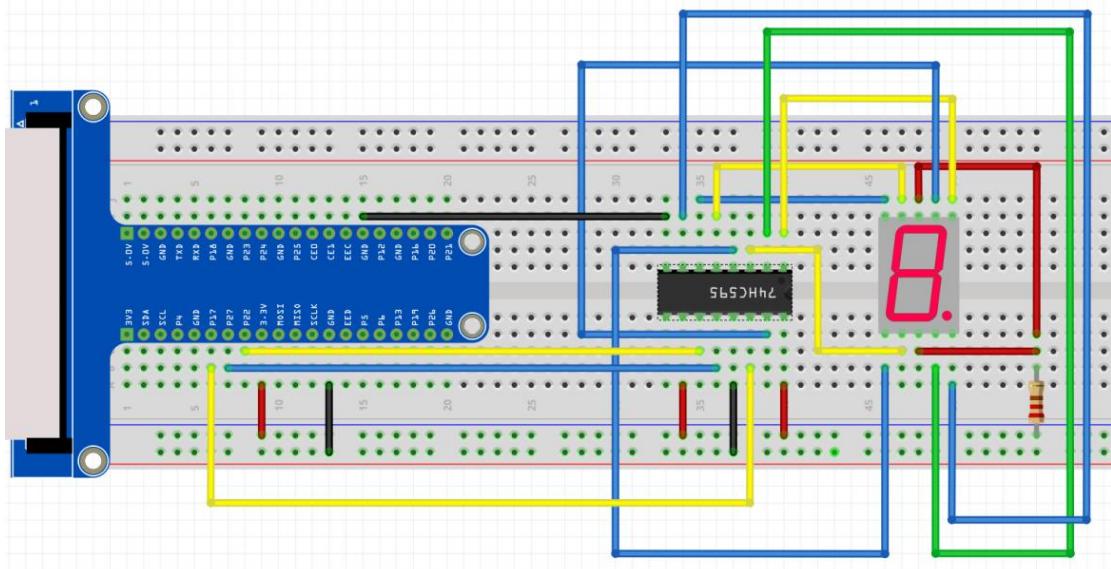
anode digital tube, and the cathode of the light-emitting diode is connected to the cathode of the power supply together is called the common cathode digital tube. The numbers and characters displayed by commonly used LED digital tubes are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. This lesson uses a common anode digital tube.



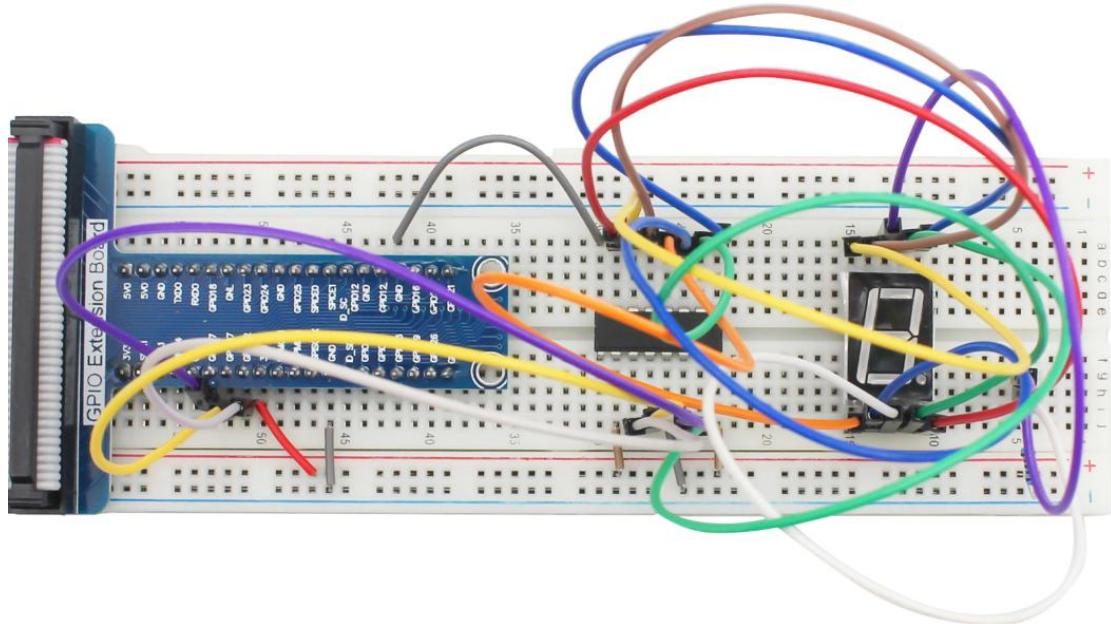
Connection Diagram



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;

```
pi@raspberrypi: ~code/C/18.DigitalTube
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/18.DigitalTube/
pi@raspberrypi:~/code/C/18.DigitalTube $ ls
DigitalTube.c
pi@raspberrypi:~/code/C/18.DigitalTube $
```

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 :



```
pi@raspberrypi:~/code/C/18.DigitalTube
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/18.DigitalTube/
pi@raspberrypi:~/code/C/18.DigitalTube $ ls
DigitalTube.c
pi@raspberrypi:~/code/C/18.DigitalTube $ gcc DigitalTube.c -o DigitalTube -lwiringPi
pi@raspberrypi:~/code/C/18.DigitalTube $ ls
DigitalTube DigitalTube.c
pi@raspberrypi:~/code/C/18.DigitalTube $ sudo ./DigitalTube
```

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 message to 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 transferred 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.

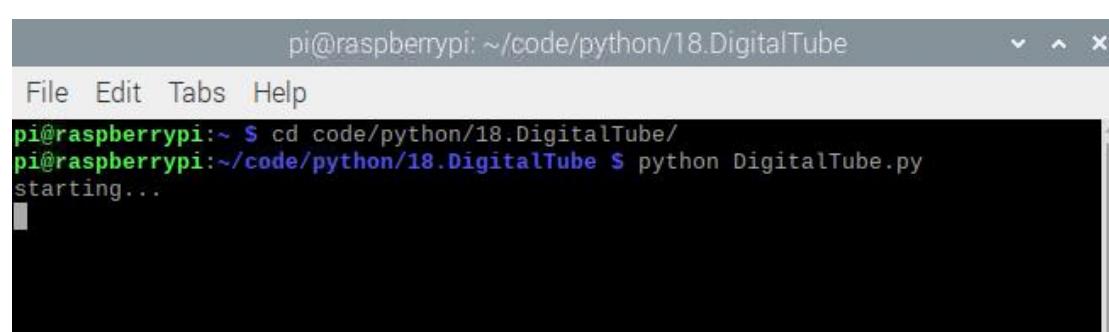
```
for(i=0;i<sizeof(num);i++){
    digitalWrite(latchPin,LOW);
    sendOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and
the highest level is transferred preferentially.
    digitalWrite(latchPin,HIGH);
    delay(500);
}
```

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.



A terminal window titled "pi@raspberrypi: ~ /code/python/18.DigitalTube". The window shows the following command sequence:

```
pi@raspberrypi:~ $ cd code/python/18.DigitalTube/
pi@raspberrypi:~/code/python/18.DigitalTube $ python DigitalTube.py
starting...
```

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 transferred 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 transferred 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 19 74HC595&4-Digit 7-Segment Display

Overview

In this lesson you will learn how to use a 4-digit 7-segment display. Based on the previous lesson, we will change the 1-digit 7-segment display into a 4-digit 7-segment display.

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 4-digit 7-segment Display

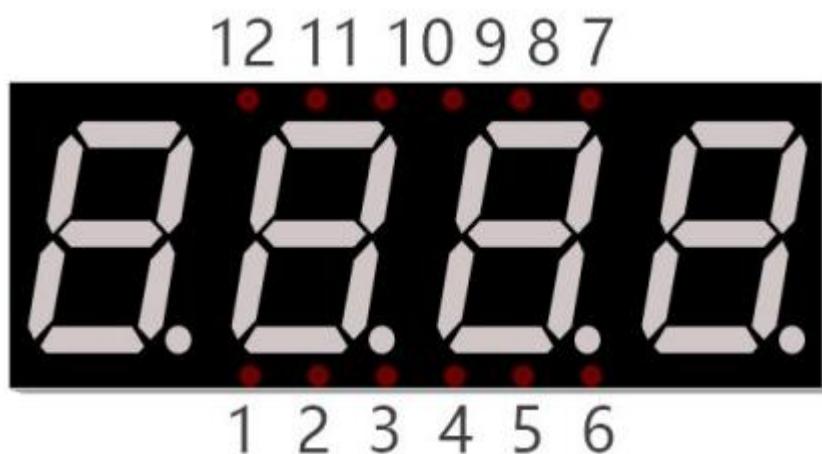
Product Introduction

4 Digit 7-Segment Display

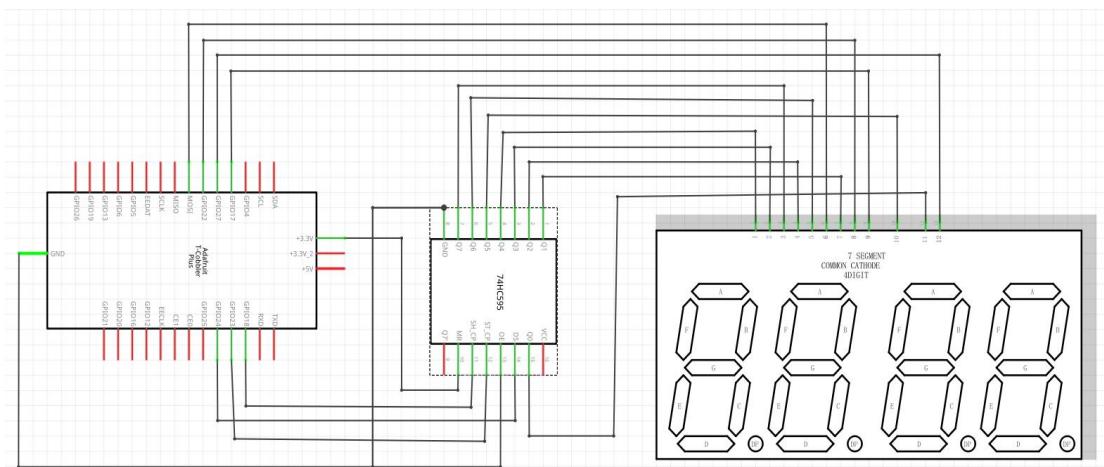
4 Digit 7-segment display integrates four 7-segment display, so it can display more numbers. Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that 4-Digit display in turn, one by one, not together. First send high level to common end of the first tube, and send low level to the rest of the three common end, and then send content to 8 LED cathode pins of the first tube. At this time, the first 7-segment display will display content and the rest three one in closed state. Similarly, the second, third, fourth 7-segment display the content in turn, namely, scan display. Although the four numbers are displayed in turn separately, but this process is very fast, and due to the optical afterglow effect and

people in vision persistence effect, we can see all 4 numbers at the same time. On the contrary, if each figure is displayed for a long time, you can see that the numbers are displayed separately. According to the difference about common cathode and anode.

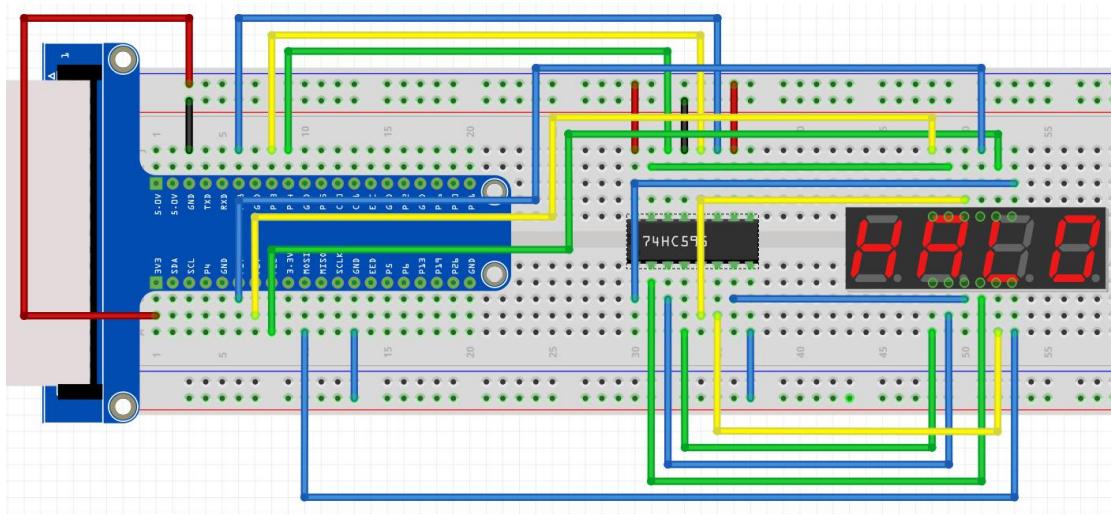
Its internal structure and pins diagram is shown below:



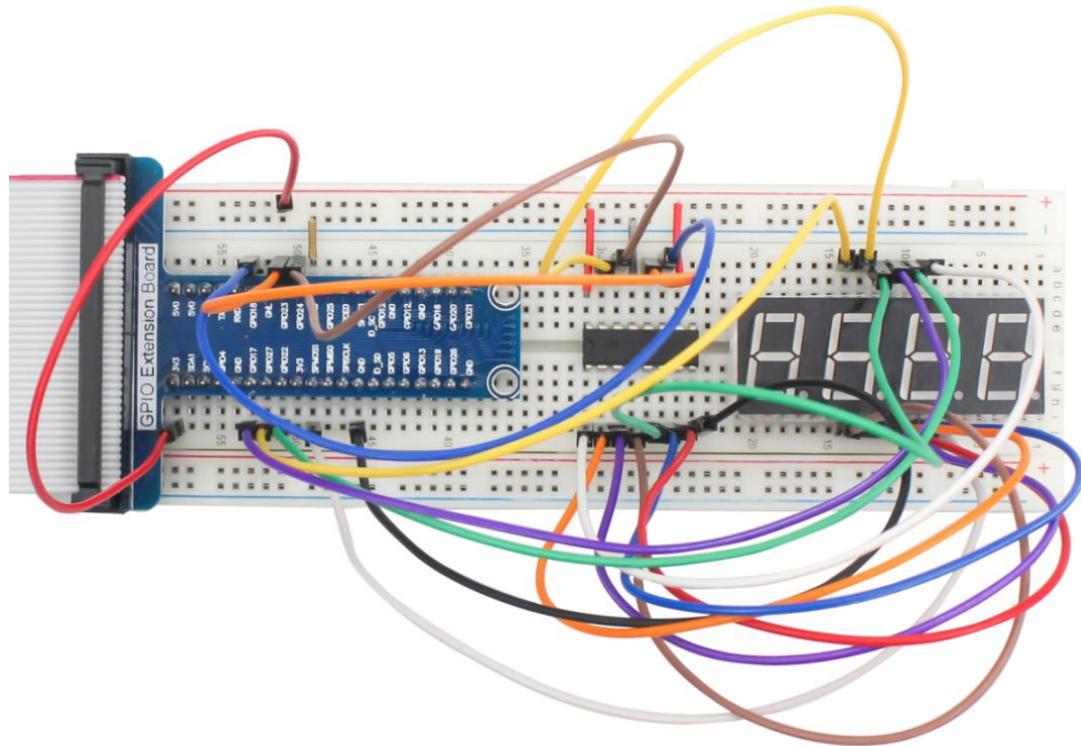
Connection Diagram



Wiring Diagram



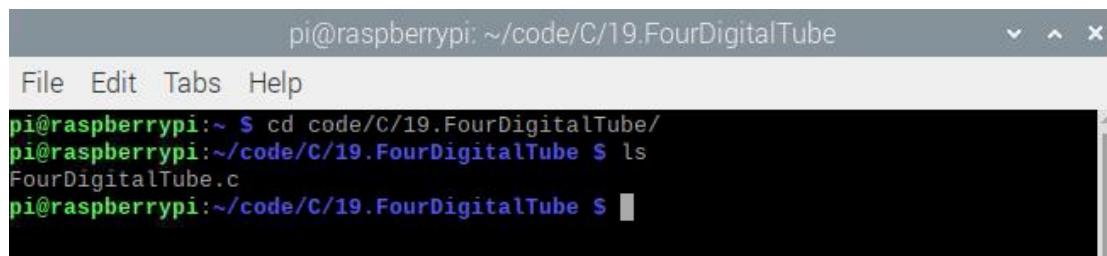
Example Figure



C code

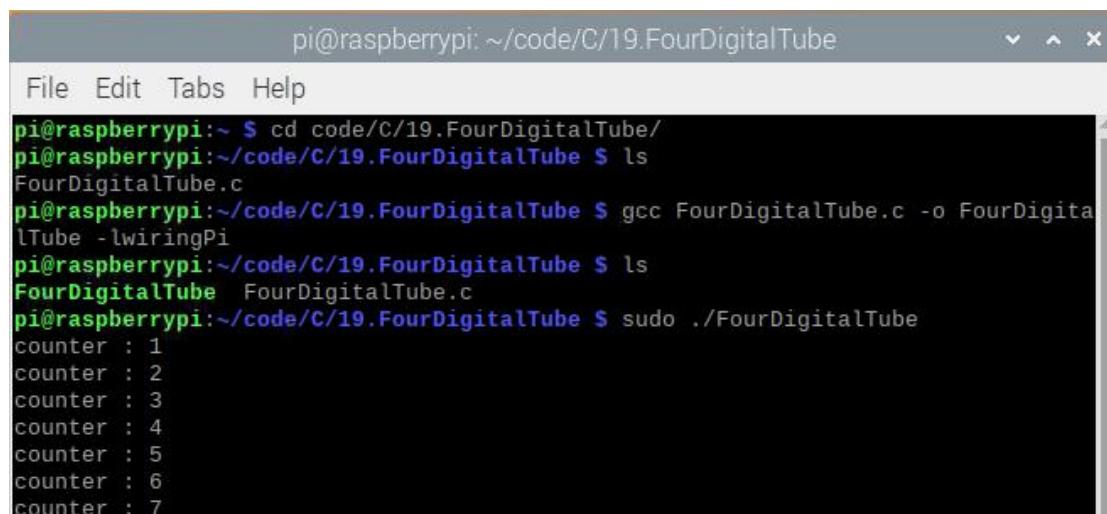
Open the terminal and enter the "`cd code / C / 19.FourDigitalTube /`" command to enter the "FourDigitalTube" code directory;

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



```
pi@raspberrypi: ~/code/C/19.FourDigitalTube
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/19.FourDigitalTube/
pi@raspberrypi:~/code/C/19.FourDigitalTube $ ls
FourDigitalTube.c
pi@raspberrypi:~/code/C/19.FourDigitalTube $
```

Enter "`gcc FourDigitalTube.c -o FourDigitalTube -lwiringPi`" command to generate "FourDigitalTube.c" executable file "FourDigitalTube", enter "`ls`" command to view; enter "`sudo ./FourDigitalTube`" command to run the code, the results are shown below :



```
pi@raspberrypi: ~/code/C/19.FourDigitalTube
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/19.FourDigitalTube/
pi@raspberrypi:~/code/C/19.FourDigitalTube $ ls
FourDigitalTube.c
pi@raspberrypi:~/code/C/19.FourDigitalTube $ gcc FourDigitalTube.c -o FourDigitalTube -lwiringPi
pi@raspberrypi:~/code/C/19.FourDigitalTube $ ls
FourDigitalTube FourDigitalTube.c
pi@raspberrypi:~/code/C/19.FourDigitalTube $ sudo ./FourDigitalTube
counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
counter : 6
counter : 7
```

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

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define      dataPin      5 //DS Pin of 74HC595(Pin14)
#define      latchPin     4 //ST_CP Pin of 74HC595(Pin12)
#define      clockPin     1 //CH_CP Pin of 74HC595(Pin11)
```

```
const int digitPin[]={0,2,3,12};           // Define 7-segment display common pin
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;      //variable counter,the number will be displayed by 7-segment
display
//Open one of the 7-segment display and close the remaining three, the parameter
digit is optional for 1,2,4,8
void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) != 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) != 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) != 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) != 0x01) ? LOW : HIGH);
}
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);
    }
}

void outData(int8_t data){      //function used to output data for 74HC595
    digitalWrite(latchPin,LOW);
    sendOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
void display(int dec){ //display function for 7-segment display
    int delays = 1;
    outData(0xff);
    selectDigit(0x01);      //select the first, and display the single digit
    outData(num[dec%10]);
    delay(delays);          //display duration
}
```

```
outData(0xff);
selectDigit(0x02);      //select the second, and display the tens digit
outData(num[dec%100/10]);
delay(delays);

outData(0xff);
selectDigit(0x04);      //select the third, and display the hundreds digit
outData(num[dec%1000/100]);
delay(delays);

outData(0xff);
selectDigit(0x08);      //select the fourth, and display the thousands digit
outData(num[dec%10000/1000]);
delay(delays);

}

void timer(int sig){      //Timer function
    if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus
        1, and update the number displayed by 7-segment display
        counter++;
        alarm(1);          //set the next timer time
        printf("counter : %d \n",counter);
    }
}

int main(void)
{
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);      //set the pin connected to 74HC595 for
output mode
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    //set the pin connected to 7-segment display common end to output mode
    for(i=0;i<4;i++){
        pinMode(digitPin[i],OUTPUT);
        digitalWrite(digitPin[i],HIGH);
    }
    signal(SIGALRM,timer); //configure the timer
```

```

alarm(1);           //set the time of timer to 1s
while(1){
    display(counter); //display the number counter
}
return 0;
}

```

Code Interpretation

First, define the pin of 74HC595 and 7-segment display common end, character encoding and a variable "counter" to be displayed counter.

```

#define      dataPin      5
#define      latchPin     4
#define      clockPin     1
const int digitPin[]={0,2,3,12};
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;

num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;

```

Subfunction "selectDigit(int digit)" is used to open one of the 7-segment digital tubes and turn off the other three 7-segment digital tubes. The parameter digital value can be 1, 2, 4, 8.

```

void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) != 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) != 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) != 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) != 0x01) ? LOW : HIGH);
}

```

Subfunction "outData(int8_t data)" is used to make 74HC595 output a 8-bit data.

```

void outData(int8_t data){
    digitalWrite(latchPin,LOW);
    sendOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}

```

Subfunction `display (int dec)` is used to make 4-Digit 7-segment display a 4-bit integer. First open the common end of first 7-segment display and close to the other three, at this time, it can be used as 1-Digit 7- segment display. The first is used for displaying single digit of "dec", the second for tens digit, third for hundreds digit and fourth for thousands digit respectively. Each digit will be displayed for a period of time through using `delay ()`. The time in this code is set very short, so you will see different digit is in a mess. If the time is set long enough, you will see that every digit is display independent.

```
void display(int dec){
    int delays = 1;
    outData(0xff);
    selectDigit(0x01);
    outData(num[dec%10]);
    delay(delays);
    outData(0xff);
    selectDigit(0x02);
    outData(num[dec%100/10]);
    delay(delays);
    outData(0xff);
    selectDigit(0x04);
    outData(num[dec%1000/100]);
    delay(delays);

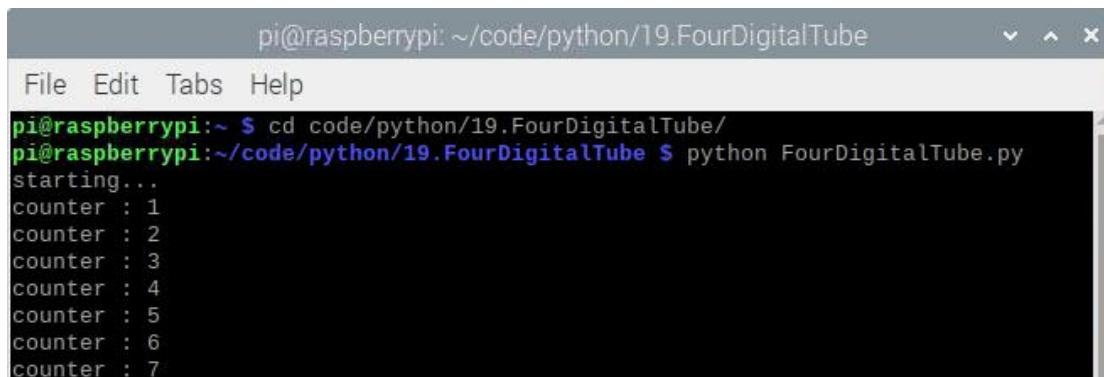
    outData(0xff);
    selectDigit(0x08);          outData(num[dec%10000/1000]);
    delay(delays);
}
```

Subfunction `timer (int sig)` is the timer function, which will set an alarm signal. This function will be executed once at set intervals. Accompanied by the execution, the variable counter will be added 1, and then reset the time of timer to 1s.

```
void timer(int sig){      //Timer function
    if(sig == SIGALRM){  //If the signal is SIGALRM, the value of counter plus
        1, and update the number displayed by 7-segment display
        counter++;
        alarm(1);           //set the next timer time
        printf("counter : %d \n",counter);
    }
}
```

Python code

1. Use the "cd code / python / 19.FourDigitalTube /" command to enter the "FourDigitalTube" directory.
2. Use "python FourDigitalTube.py" command to execute "FourDigitalTube.py" code.



A terminal window titled "pi@raspberrypi: ~ / code / python / 19. FourDigitalTube". The window shows the following command-line session:

```
pi@raspberrypi:~ $ cd code/python/19.FourDigitalTube/
pi@raspberrypi:~/code/python/19.FourDigitalTube $ python FourDigitalTube.py
starting...
counter : 1
counter : 2
counter : 3
counter : 4
counter : 5
counter : 6
counter : 7
```

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

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

LWAY = 1
MWAY = 2
#define the pins connect to 74HC595
dataPin  = 18      #DS Pin of 74HC595(Pin14)
latchPin = 16      #ST_CP Pin of 74HC595(Pin12)
clockPin = 12      #CH_CP Pin of 74HC595(Pin11)

counter =0         #Variable counter, the number will be displayed by 7-segment
display
t=0               # define the Timer object

num = [0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90]
digitPin = (11,13,15,19)    # Define the pin of 7-segment display common end

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

```
GPIO.setup(clockPin,GPIO.OUT)
for Pin in digitPin:
    GPIO.setup(Pin,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 outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
    sendout(dataPin,clockPin,MWAY,data)
    GPIO.output(latchPin,GPIO.HIGH)

def selectDigit(digit): # Open one of the 7-segment display and close the remaining
three, the parameter digit is optional for 1,2,4,8
    GPIO.output(digitPin[0],GPIO.LOW if ((digit&0x08) != 0x08) else
GPIO.HIGH)
    GPIO.output(digitPin[1],GPIO.LOW if ((digit&0x04) != 0x04) else
GPIO.HIGH)
    GPIO.output(digitPin[2],GPIO.LOW if ((digit&0x02) != 0x02) else
GPIO.HIGH)
    GPIO.output(digitPin[3],GPIO.LOW if ((digit&0x01) != 0x01) else
GPIO.HIGH)

def display(dec):      #display function for 7-segment display
    outData(0xff)    #eliminate residual display
    selectDigit(0x01)  #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.001)  #display duration
    outData(0xff)
    selectDigit(0x02)  # Select the second, and display the tens digit
    outData(num[dec%100//10])
    time.sleep(0.001)
    outData(0xff)
```

```
selectDigit(0x04)    # Select the third, and display the hundreds digit
outData(num[dec%1000//100])
time.sleep(0.001)
outData(0xff)
selectDigit(0x08)    # Select the fourth, and display the thousands digit
outData(num[dec%10000//1000])
time.sleep(0.001)

def timer():          #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                         #Start timing
    counter+=1
    print ("counter : %d"%counter)

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

def destroy(): # When 'Ctrl+C' is pressed, the function is executed.
    global t
    GPIO.cleanup()
    t.cancel() # cancel the timer

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

Code Interpretation

First, define the pins of 74HC595 and the common end of the 7-segment digital tube, the character code of the numbers "0-8" and the variable "counter" for timing.

```

dataPin = 18      #DS Pin of 74HC595(Pin14)
latchPin = 16    #ST_CP Pin of 74HC595(Pin12)
clockPin = 12    #CH_CP Pin of 74HC595(Pin11)
counter = 0       #Variable counter, the number will be displayed by 7-segment
display
t=0              # define the Timer object
num = [0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90]
digitPin = (11,13,15,19)  # Define the pin of 7-segment display c

```

Subfunction “selectDigit(digit)” function is used to open one of the 7-segment display and close the other 7- segment display, where the parameter digit value can be 1,2,4,8.

```

def selectDigit(digit): # Open one of the 7-segment display and close the remaining
three, the parameter digit is optional for 1,2,4,8
    GPIO.output(digitPin[0],GPIO.LOW if ((digit&0x08) != 0x08) else
    GPIO.HIGH)
    GPIO.output(digitPin[1],GPIO.LOW if ((digit&0x04) != 0x04) else
    GPIO.HIGH)
    GPIO.output(digitPin[2],GPIO.LOW if ((digit&0x02) != 0x02) else
    GPIO.HIGH)
    GPIO.output(digitPin[3],GPIO.LOW if ((digit&0x01) != 0x01) else GPIO.HIGH)

```

Subfunction “outData (data)” is used to make the 74HC595 output a 8-bit data immediately.

```

def outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
    sendout(dataPin,clockPin,MWAY,data)
    GPIO.output(latchPin,GPIO.HIGH)

```

Subfunction “display (dec)” is used to make 4-Digit 7-segment display a 4-bit integer. First open the commonend of first 7-segment display and close to the other three, at this time, it can be used as 1-Digit 7-segment display. The first is used for displaying single digit of "dec", the second for tens digit, third for hundreds digit and fourth for thousands digit respectively. Each digit will be displayed for a period of time through using delay (). The time in this code is set very short, so you will see different digit is

in a mess. If the time is set long enough, you will see that every digit is display independent.

```
def display(dec):      #display function for 7-segment display
    outData(0xff)    #eliminate residual display
    selectDigit(0x01)  #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.001)  #display duration
    outData(0xff)
    selectDigit(0x02)  # Select the second, and display the tens digit
    outData(num[dec%100//10])
    time.sleep(0.001)
    outData(0xff)
    selectDigit(0x04)  # Select the third, and display the hundreds digit
    outData(num[dec%1000//100])
    time.sleep(0.001)
    outData(0xff)
    selectDigit(0x08)  # Select the fourth, and display the thousands digit
    outData(num[dec%10000//1000])
    time.sleep(0.001)
```

Subfunction “timer ()” is the timer callback function. When the time is up, this function will be executed. Accompanied by the execution, the variable counter will be added 1, and then reset the time of timer to 1s.1s later, the function will be executed again.

```
def timer():          #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                      #Start timing
    counter+=1
    print ("counter : %d"%counter)
```

Subfunction “setup()”, configure all input output modes for the GPIO pin used.Finally, in loop function, make the digital tube display variable counter value in the “while” cycle. The value will change in function timer (), so the content displayed by 7-segment display will change accordingly.

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

```
t.start()          # Start timing  
while True:  
    display(counter)      # display the number
```

After the program is executed, press "Ctrl+C", then subfunction `destroy()` will be executed, and GPIO resources and timers will be released in this subfunction.

```
def destroy(): # When 'Ctrl+C' is pressed, the function is executed.  
    global t  
    GPIO.cleanup()  
    t.cancel() # cancel the timer
```

Lesson 20 74HC595 & LED Matrix

Overview

In this lesson you will learn how to continue to use the 74HC595 to control more LED, LEDMatrix, we will use two 74HC595 to control a monochrome LEDMatrix (8*8) to make it display some graphics and characters.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board

1 x Breadboard

2 x 74HC595

8 x 220Ω Resistor

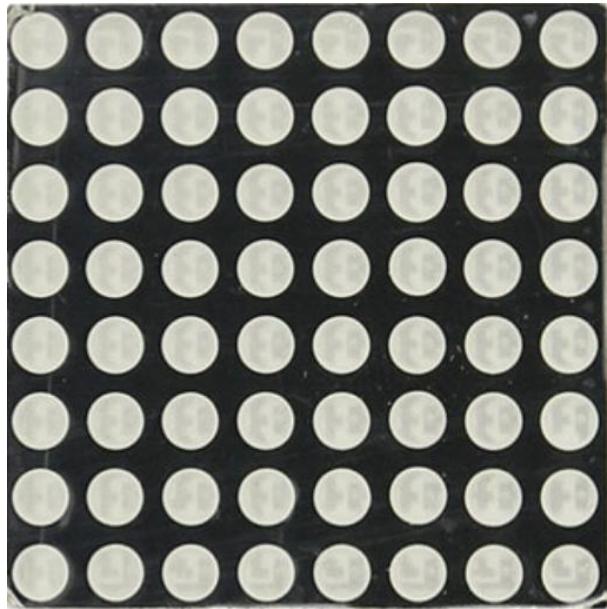
1 x LED Matrix (8 * 8)

Product Introduction

LED matrix

LED matrix is a rectangular display module that consists of several LEDs.

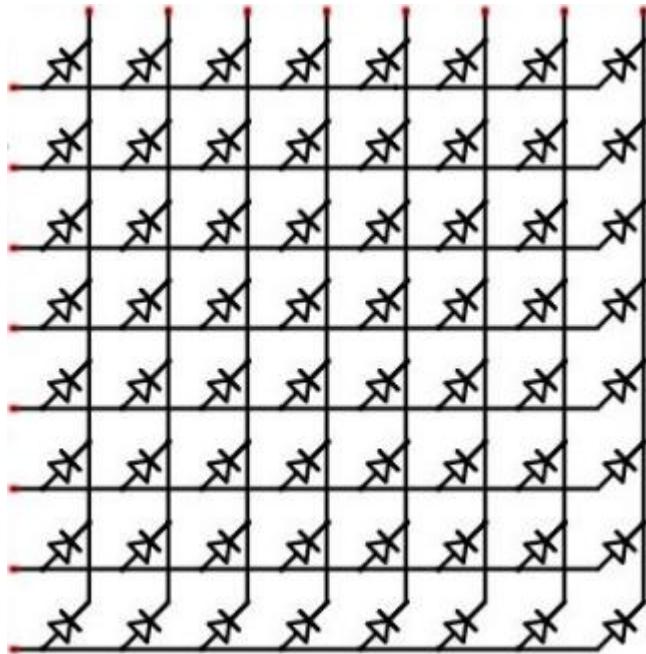
The following is an 8*8 monochromeLED matrix with 64 LEDs (8 rows and 8 columns).



In order to facilitate the operation and save the ports, positive pole of LEDs in each row and negative pole of LEDs in each column are respectively connected together inside LED matrix module, which is called Common Anode. There is another form. Negative pole of LEDs in each row and positive pole of LEDs in each column are respectively connected together, which is called Common Cathode.

The one we use in this project is a common anode LEDMatrix.

Connection mode of common anode

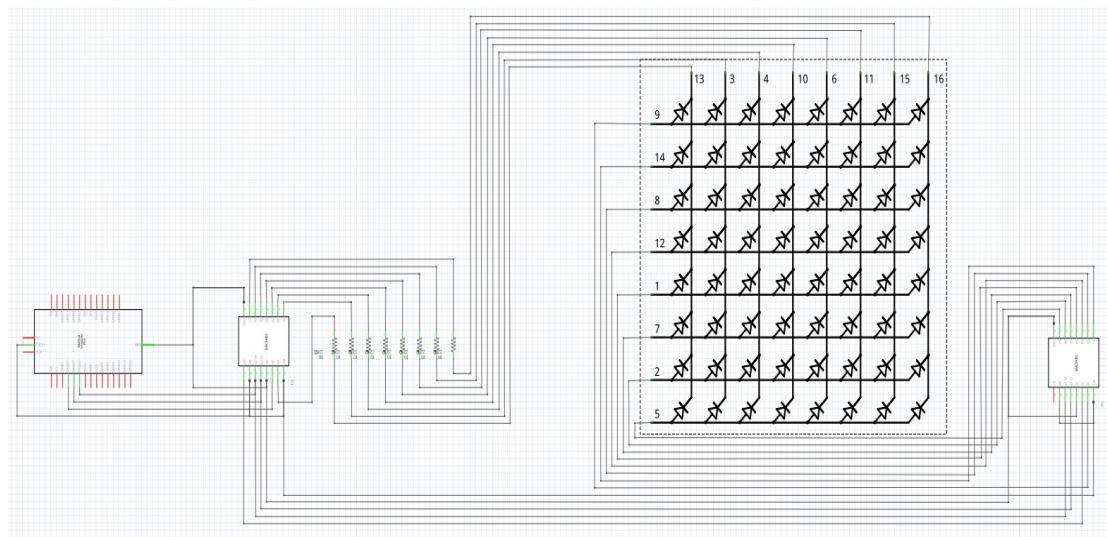


Connection mode of common cathode

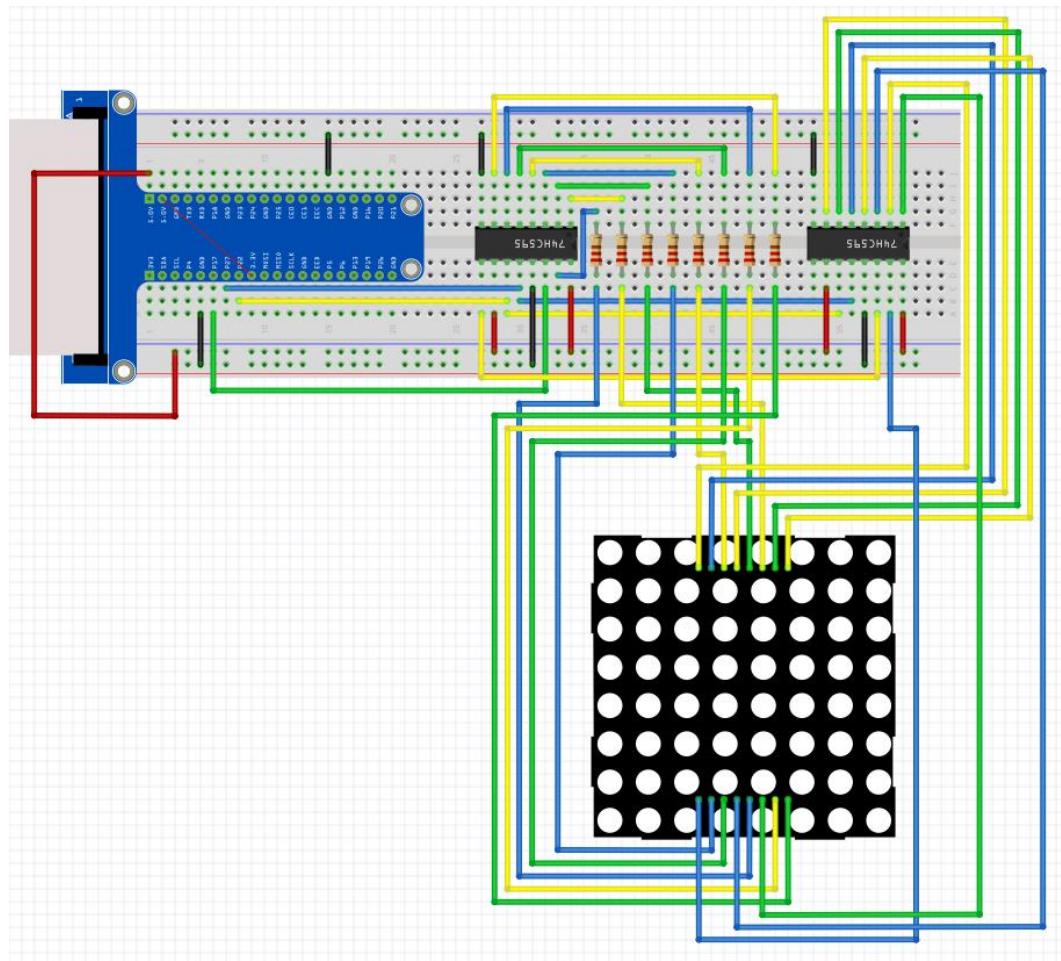


Scanning rows is another display way of dot matrix. Whether scanning line or column, 16 GPIO are required. In order to save GPIO of control board, two 74HC595 is used. Every piece of 74HC595 has eight parallel output ports, so two pieces has 16 ports in total, just enough. The control line and data line of two 74HC595 are not all connected to the RPi, but connect Q7 pin of first stage 74HC595 to data pin of second one, namely, two 74HC595 are connected in series. It is the same to using one "74HC595" with 16 parallel output ports.

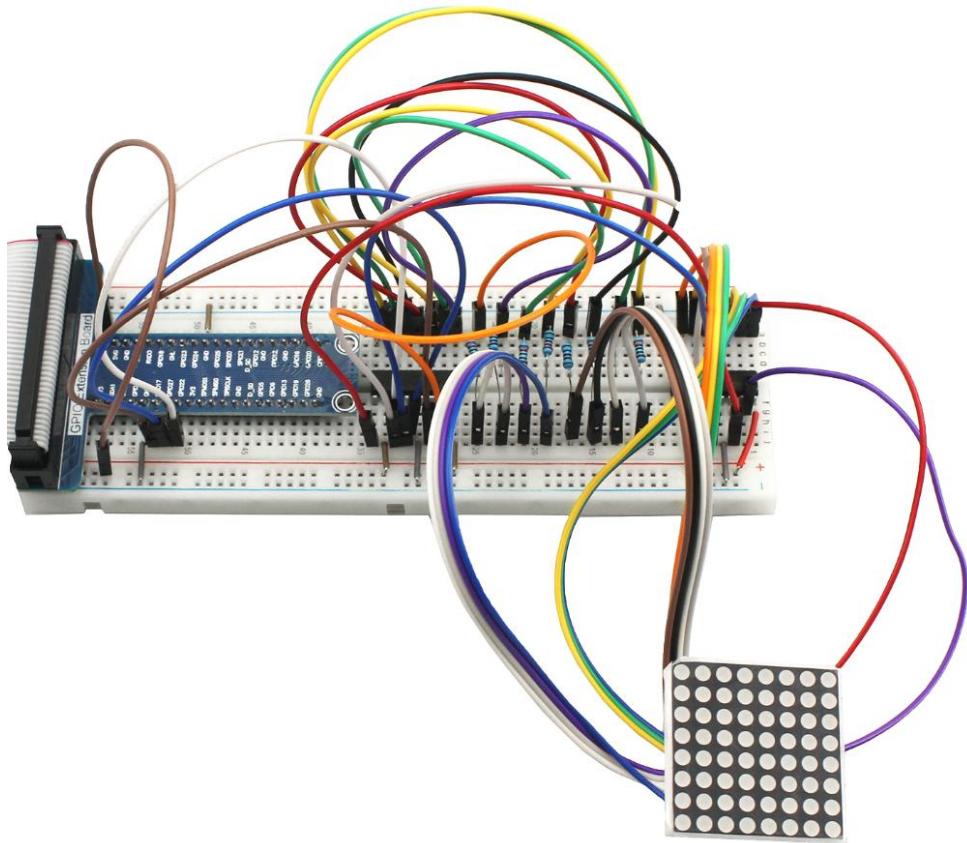
Connection Diagram



Wiring Diagram



Example Figure



C code

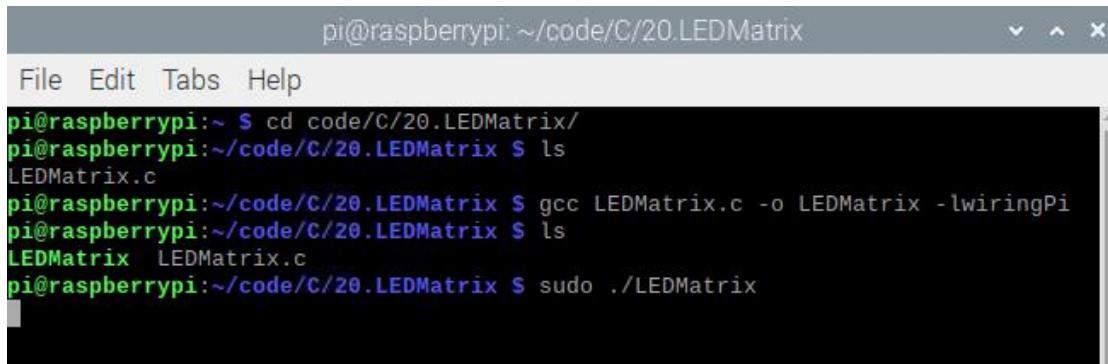
Open the terminal and enter the "cd code / C / 20.LEDMatrix /" command to enter the "LEDMatrix" code directory;

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



```
pi@raspberrypi:~/code/C/20.LEDMatrix
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/20.LEDMatrix/
pi@raspberrypi:~/code/C/20.LEDMatrix $ ls
LEDMatrix.c
pi@raspberrypi:~/code/C/20.LEDMatrix $
```

Enter "gcc LEDMatrix.c -o LEDMatrix -lwiringPi" command to generate "LEDMatrix.c" executable file "LEDMatrix", enter "ls" command to view; enter "sudo ./LEDMatrix" command to run the code, the results are shown below :



```
pi@raspberrypi:~/code/C/20.LEDMatrix
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/20.LEDMatrix/
pi@raspberrypi:~/code/C/20.LEDMatrix $ ls
LEDMatrix.c
pi@raspberrypi:~/code/C/20.LEDMatrix $ gcc LEDMatrix.c -o LEDMatrix -lwiringPi
pi@raspberrypi:~/code/C/20.LEDMatrix $ ls
LEDMatrix LEDMatrix.c
pi@raspberrypi:~/code/C/20.LEDMatrix $ sudo ./LEDMatrix
```

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 //SH_CP Pin of 74HC595(Pin11)

// data of smiling face
unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
unsigned char data[]={ // data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // ""
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
```

```
0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // ""
};

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,j,k;
    unsigned char x;
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("failed !");
        return 1;
    }
    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(j=0;j<500;j++){// Repeat enough times to display the smiling face a
period of time
            x=0x80;
            for(i=0;i<8;i++){
                digitalWrite(latchPin,LOW);
                sendOut(dataPin,clockPin,MSBFIRST,pic[i]);// first shift data of

```

line information to the first stage 74HC959

```
sendOut(dataPin,clockPin,MSBFIRST,~x); //then shift data of
column information to the second stage 74HC959
```

```
digitalWrite(latchPin,HIGH); //Output data of two stage 74HC595
at the same time
x>>=1; // display the next column
delay(1);
}
}
for(k=0;k<sizeof(data)-8;k++){ //sizeof(data) total number of "0-F"
columns
for(j=0;j<20;j++){ // times of repeated displaying LEDMatrix in every
frame, the bigger the "j", the longer the display time
x=0x80; // Set the column information to start from the first
column
for(i=k;i<8+k;i++){
digitalWrite(latchPin,LOW);
sendOut(dataPin,clockPin,MSBFIRST,data[i]);
sendOut(dataPin,clockPin,MSBFIRST,~x);
digitalWrite(latchPin,HIGH);
x>>=1;
delay(1);
}
}
}
return 0;
}
```

Code Interpretation

The first “for” cycle in the “while” cycle is used to display a static smile. Display column information from leftto right, one column by one column, totally 8 columns. Repeat 500 times to ensure display time enough.

```
for(j=0;j<500;j++){ // Repeat enough times to display the smiling face a period of time
x=0x80;
for(i=0;i<8;i++){
digitalWrite(latchPin,LOW);
```

```
sendOut(dataPin,clockPin,MSBFIRST,pic[i]);
sendOut(dataPin,clockPin,MSBFIRST,~x);

digitalWrite(latchPin,HIGH);
x>>=1;
delay(1);
}

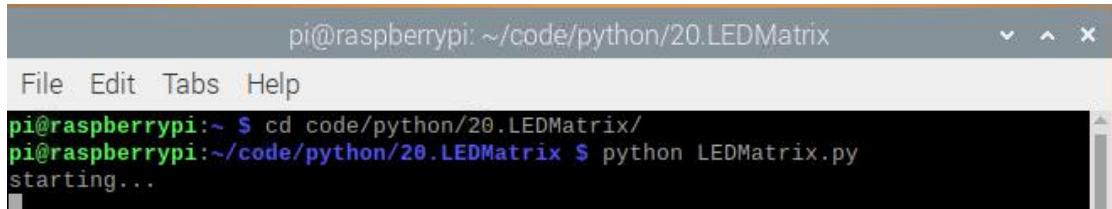
}
```

The second “for” cycle is used to display scrolling characters "0-F", totally $18*8=144$ columns. Display the 0-8 column, 1-9 column, 2-10 column..... 138-144 column in turn to achieve scrolling effect. The display of each frame is repeated a certain number of times, and the more times the number of repetitions, the longer the single frame display, the slower the rolling.

```
for(k=0;k<sizeof(data)-8;k++){
    for(j=0;j<20;j++){
        x=0x80;
        for(i=k;i<8+k;i++){
            digitalWrite(latchPin,LOW);
            sendOut(dataPin,clockPin,MSBFIRST,data[i]);
            sendOut(dataPin,clockPin,MSBFIRST,~x);
            digitalWrite(latchPin,HIGH);
            x>>=1;
            delay(1);
        }
    }
}
```

Python code

1. Use the "cd code / python / 20.LEDMatrix /" command to enter the "LEDMatrix" directory.
2. Use "python LEDMatrix.py" command to execute "LEDMatrix.py" code.



```
pi@raspberrypi:~/code/python/20.LEDMatrix
pi@raspberrypi:~/code/python/20.LEDMatrix $ python LEDMatrix.py
starting...
```

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)
```

```
pic = [0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c] #data of smiling face
```

```
data = [#data of "0-F"
```

```
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # ""
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
```

```

0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # ""
]

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 j in range(0,500):# Repeat enough times to display the smiling face a
period of time
            x=0x80
            for i in range(0,8):
                GPIO.output(latchPin,GPIO.LOW)
                sendout(dataPin,clockPin,MWAY,pic[i]) #first shift data of line
information to first stage 74HC959

                sendout(dataPin,clockPin,MWAY,~x) #then shift data of column
information to second stage 74HC959
                GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage
74HC595 at the same time
                time.sleep(0.001)# display the next column
                x>>=1
            for k in range(0,len(data)-8):#len(data) total number of "0-F" columns

```

for j in range(0,20):# times of repeated displaying LEDMatrix in every frame, the bigger the "j", the longer the display time.

```
x=0x80      # Set the column information to start from the first column
```

```
for i in range(k,k+8):
    GPIO.output(latchPin,GPIO.LOW)
    sendout(dataPin,clockPin,MWAY,data[i])
    sendout(dataPin,clockPin,MWAY,~x)
    GPIO.output(latchPin,GPIO.HIGH)
    time.sleep(0.001)
    x>>=1
```

```
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

The first “for” cycle in the “while” cycle is used to display a static smile. Display column information from left to right, one column by one column, totally 8 columns. Repeat 500 times to ensure display time enough.

```
for j in range(0,500):# Repeat enough times to display the smiling face a period of time
```

```
    x=0x80
```

```
    for i in range(0,8):
```

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

```
        sendout(dataPin,clockPin,MWAY,pic[i]) #first shift data of line information to first stage 74HC959
```

```
        sendout(dataPin,clockPin,MWAY,~x) #then shift data of column information to second stage 74HC959
```

```
        GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage 74HC595 at the same time
```

```
time.sleep(0.001)# display the next column  
x>>=1
```

The second “for” cycle is used to display scrolling characters "0-F", totally 18*8=144 columns. Display the 0-8 column, 1-9 column, 2-10 column..... 138-144 column in turn to achieve scrolling effect. The display of each frame is repeated a certain number of times, and the more times the number of repetitions, the longer the single frame display, the slower the rolling.

```
for k in range(0,len(data)-8):#len(data) total number of "0-F" columns  
    for j in range(0,20):# times of repeated displaying LEDMatrix in every  
    #frame, the bigger the "j", the longer the display time.  
        x=0x80          # Set the column information to start from the first  
        column
```

```
        for i in range(k,k+8):  
            GPIO.output(latchPin,GPIO.LOW)  
            sendout(dataPin,clockPin,MWAY,data[i])  
            sendout(dataPin,clockPin,MWAY,~x)  
            GPIO.output(latchPin,GPIO.HIGH)  
            time.sleep(0.001)  
        x>>=1
```

Lesson 21 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

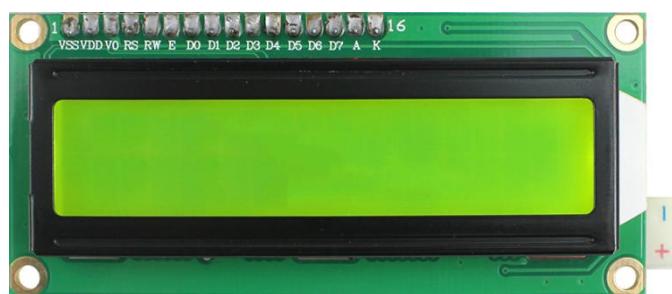
1 x LCD1602

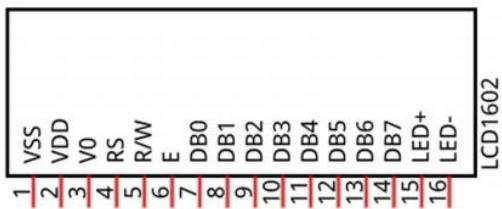
1 x PCF8574

Product Introduction

LCD1602

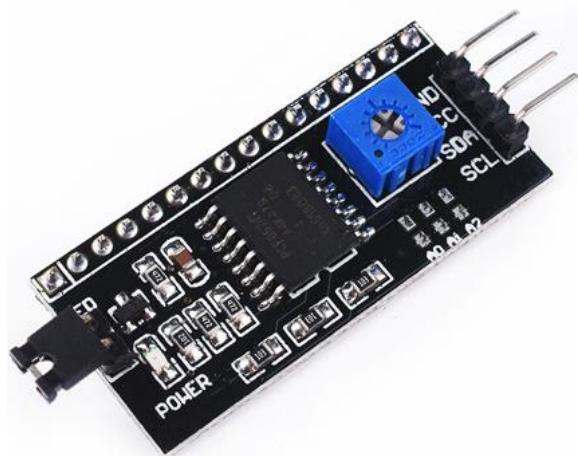
LCD1602 can display 2 lines of characters in 16 columns. It can display numbers, letters, symbols, ASCII code and so on. I2C LCD1602 integrates a I2C interface, which connects the serial-input ¶llel-output module to LCD1602. We just use 4 lines to the operate LCD1602 easily. As shown below is a monochrome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





PCF8574

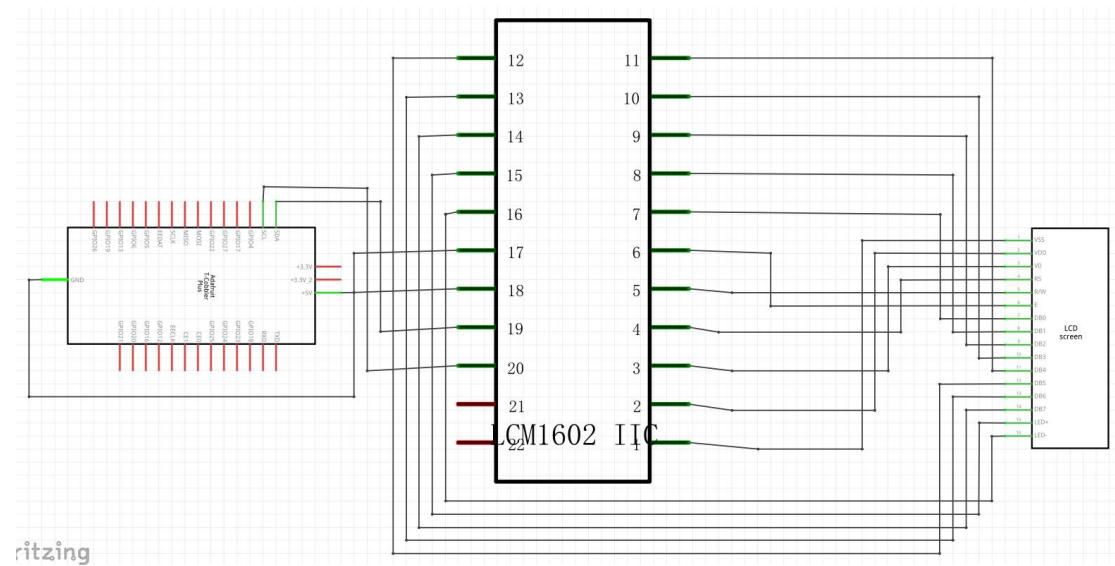
The serial-to-parallel chip used in this module is PCF8574(PCF8574A), and its default I2C address is 0x27(0x3F), and you can view all the RPI bus on your I2C device address through command "i2cdetect -y 1" to. PCF8574 module pin and LCD1602 pin are corresponding to each other and connected with each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



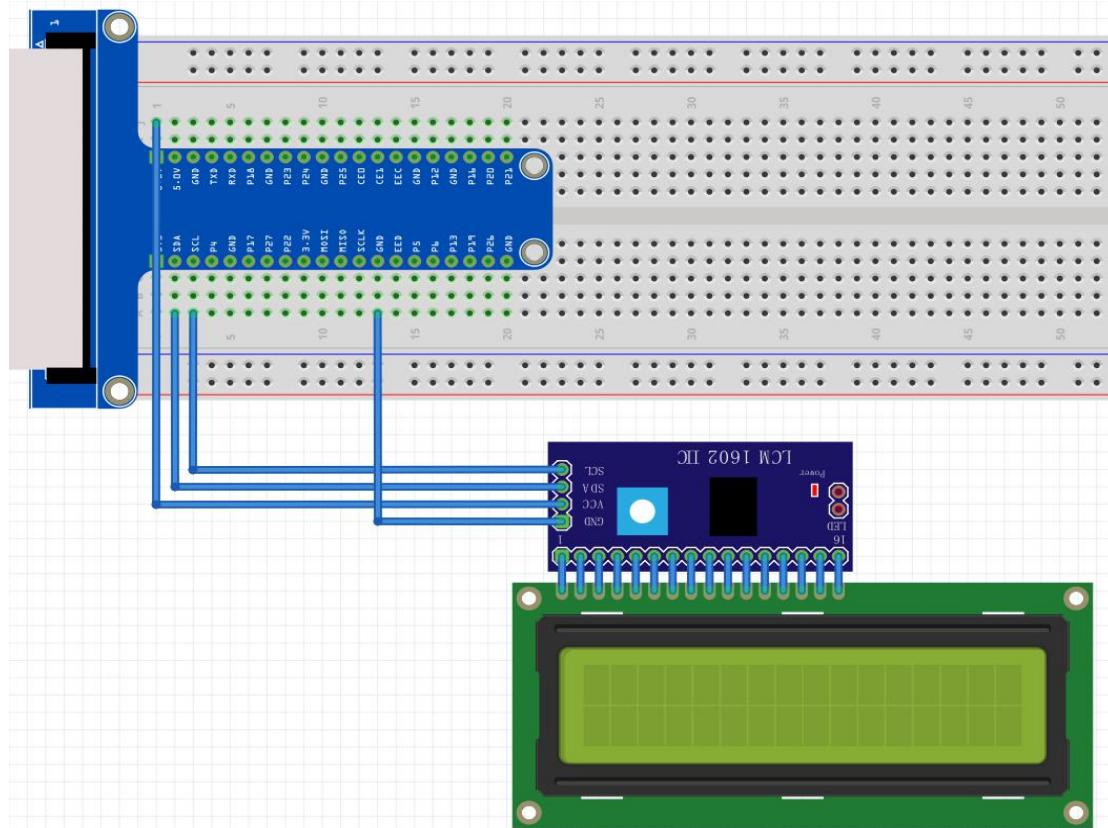
LCD Display



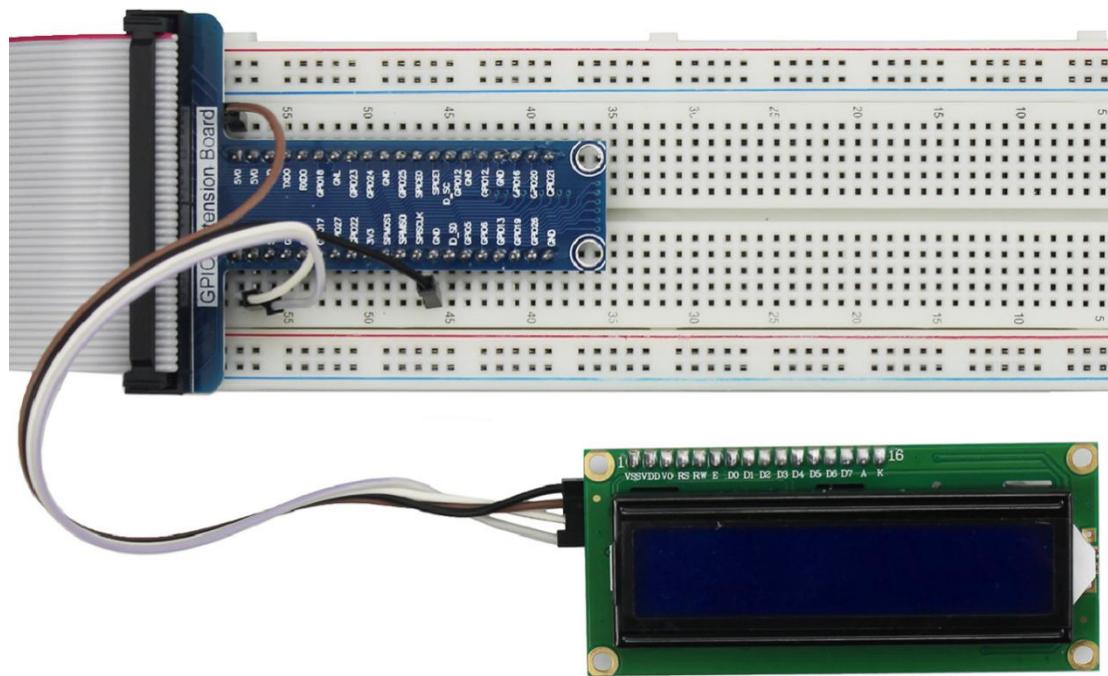
Connection Diagram



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

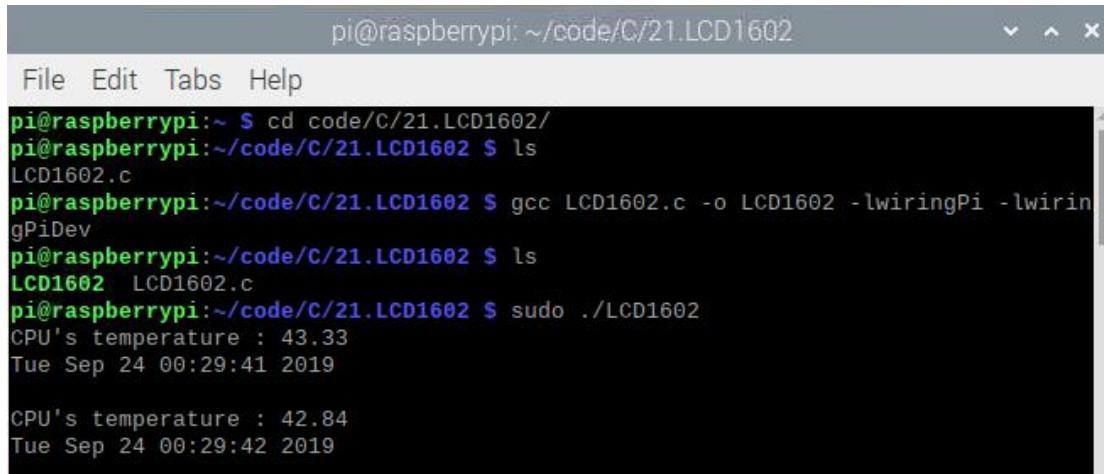
Open the terminal and enter the "cd code / C / 21.LCD1602 /" command to enter the "LCD1602" code directory;

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

```
pi@raspberrypi:~/code/C/21.LCD1602
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/21.LCD1602/
pi@raspberrypi:~/code/C/21.LCD1602 $ ls
LCD1602.c
pi@raspberrypi:~/code/C/21.LCD1602 $
```

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:



```
pi@raspberrypi:~/code/C/21.LCD1602
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/21.LCD1602/
pi@raspberrypi:~/code/C/21.LCD1602 $ ls
LCD1602.c
pi@raspberrypi:~/code/C/21.LCD1602 $ gcc LCD1602.c -o LCD1602 -lwiringPi -lwiringPiDev
pi@raspberrypi:~/code/C/21.LCD1602 $ ls
LCD1602  LCD1602.c
pi@raspberrypi:~/code/C/21.LCD1602 $ sudo ./LCD1602
CPU's temperature : 43.33
Tue Sep 24 00:29:41 2019

CPU's temperature : 42.84
Tue Sep 24 00:29:42 2019
```

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 message to 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 is stored 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.

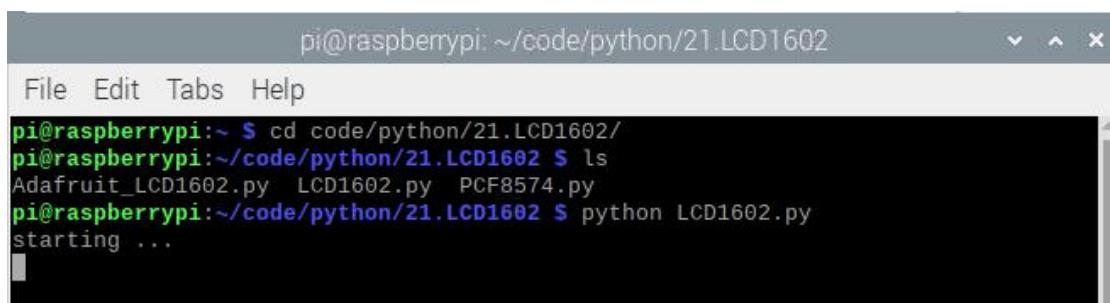
```
void printDataTime(){//used to print system time
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);// get system time
    timeinfo = localtime(&rawtime);// convert to local time
    printf("%s \n",asctime(timeinfo));
    lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)

    lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->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.



A screenshot of a terminal window titled "pi@raspberrypi: ~ /code/python/21.LCD1602". The window shows the following command-line session:

```
pi@raspberrypi:~ $ cd code/python/21.LCD1602/
pi@raspberrypi:~/code/python/21.LCD1602 $ ls
Adafruit_LCD1602.py  LCD1602.py  PCF8574.py
pi@raspberrypi:~/code/python/21.LCD1602 $ python LCD1602.py
starting ...
```

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 for Raspberry 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 22 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

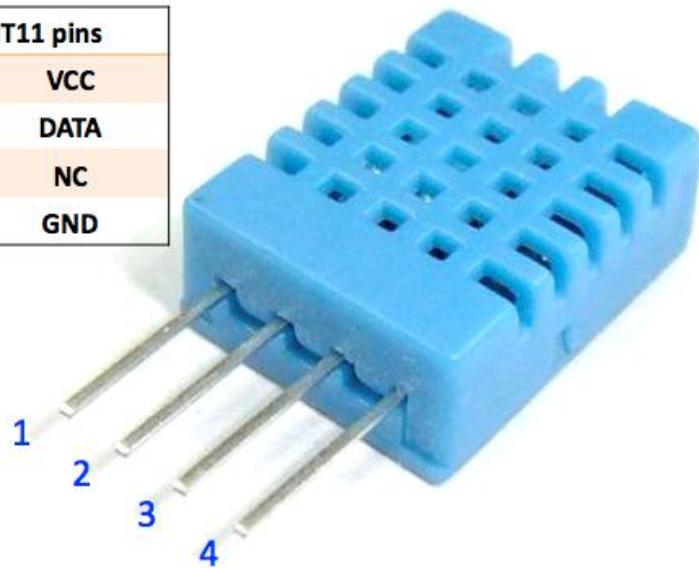
Some Jumper Wires

Product Introduction

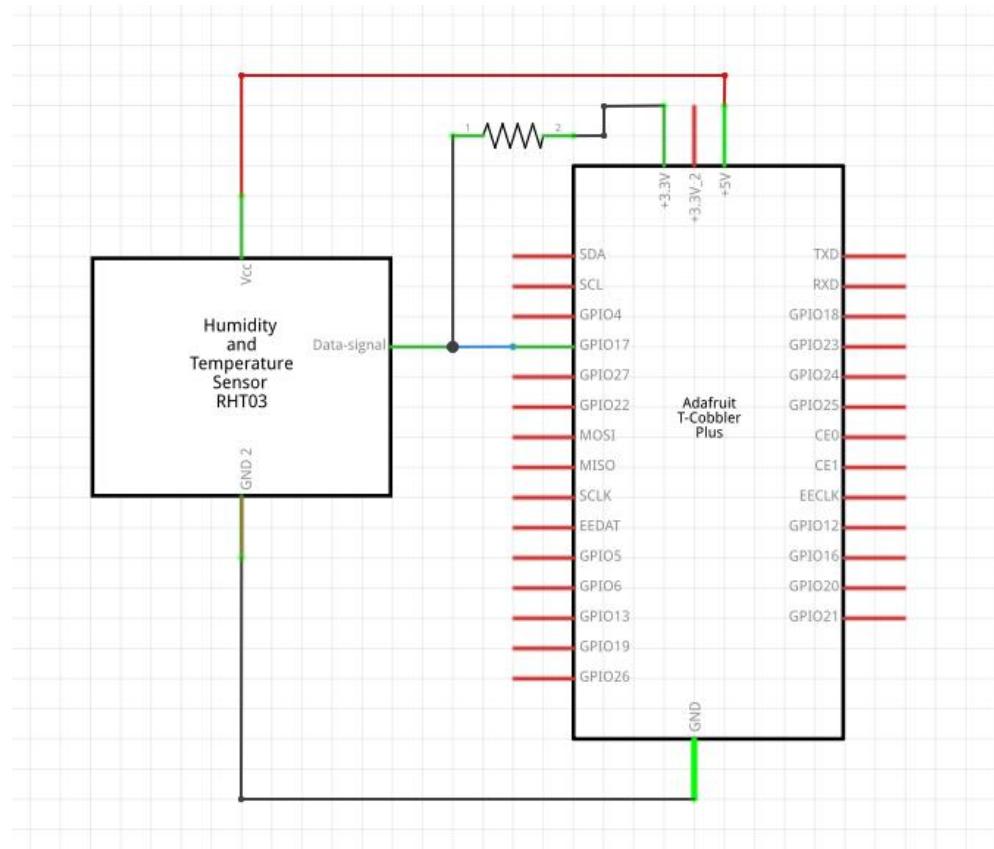
Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated inside. It has 1S's initialization time after powered up. The operating voltage is within the range of 3.3V-5.5V. SDA pin is a data pin, which is used to communicate with 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.

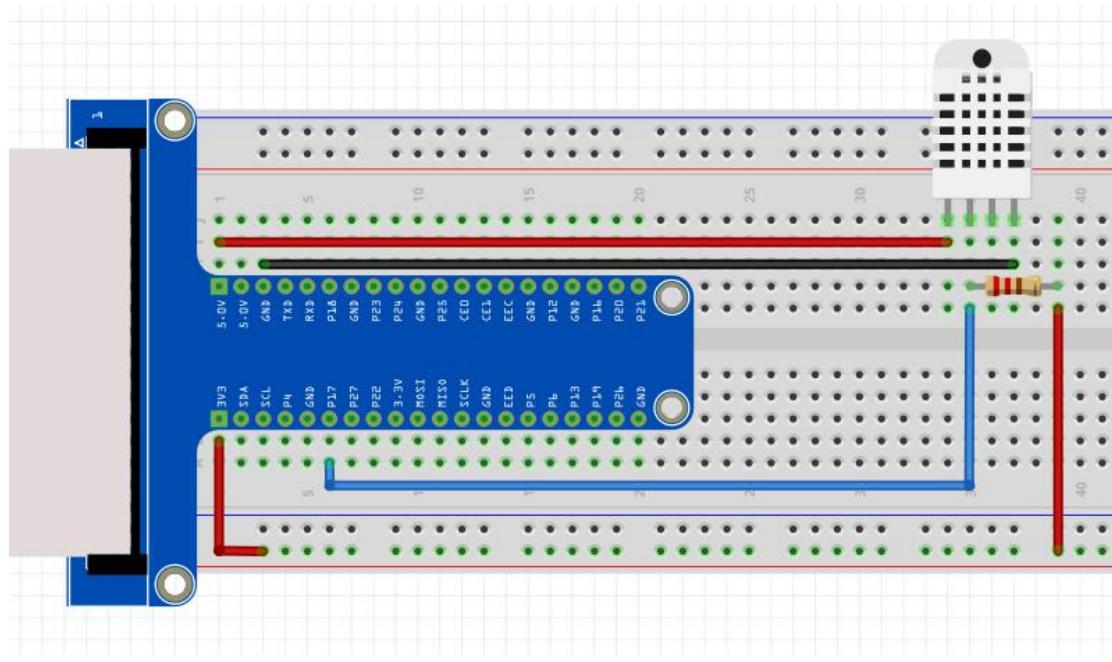
DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



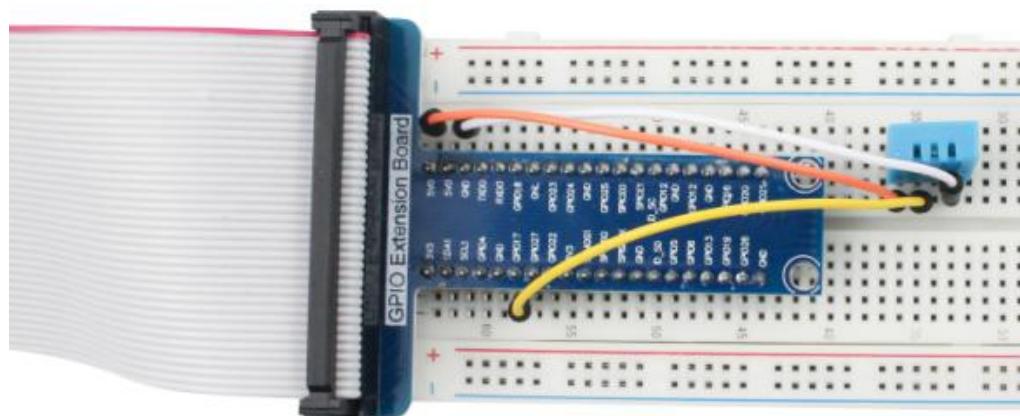
Connection Diagram



Wiring Diagram



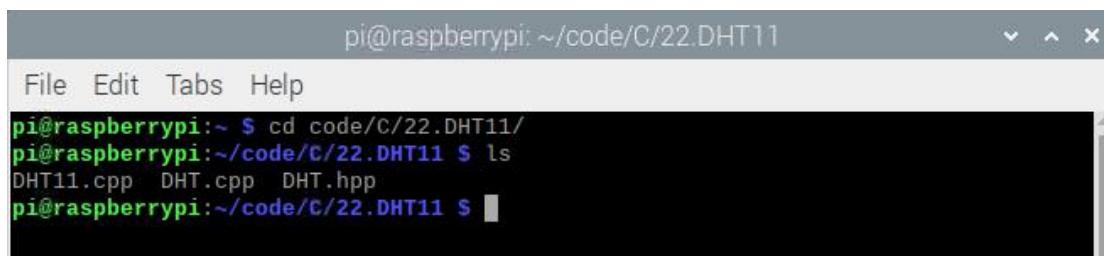
Example Figure



C code

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

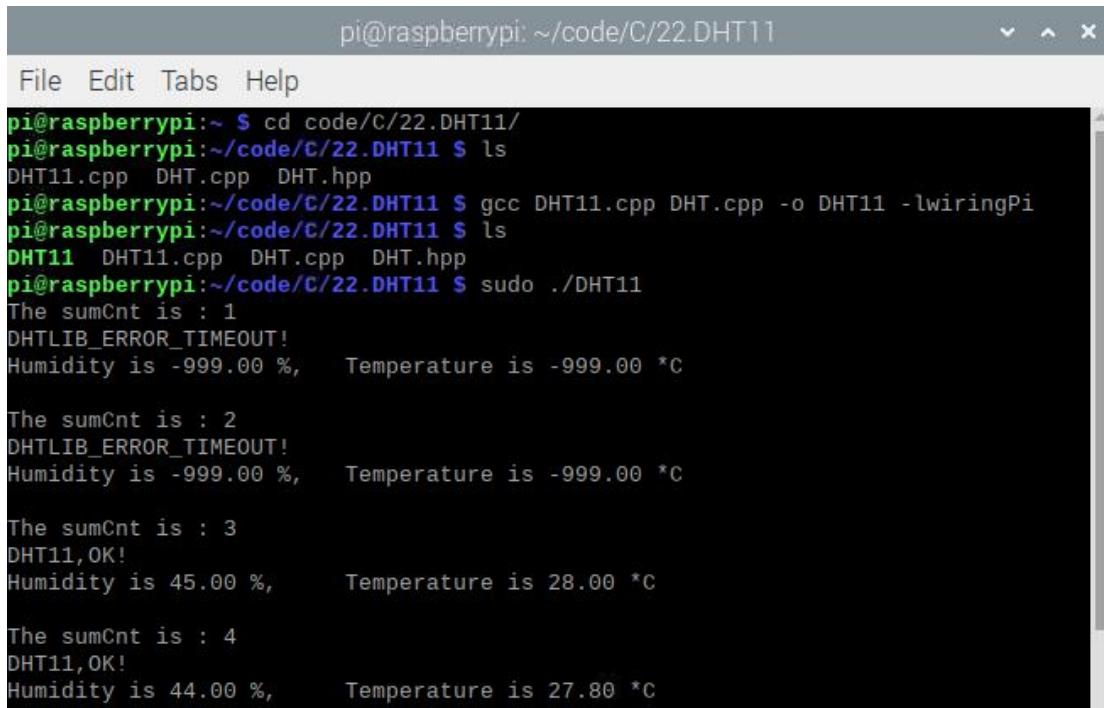
Enter the "ls" command to view the files "DHT11.cpp", "DHT.cpp", and "DHT.hpp" in the directory;



```
pi@raspberrypi:~/code/C/22.DHT11
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/22.DHT11/
pi@raspberrypi:~/code/C/22.DHT11 $ ls
DHT11.cpp  DHT.cpp  DHT.hpp
pi@raspberrypi:~/code/C/22.DHT11 $
```

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:



```
pi@raspberrypi:~/code/C/22.DHT11
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/22.DHT11/
pi@raspberrypi:~/code/C/22.DHT11 $ ls
DHT11.cpp  DHT.cpp  DHT.hpp
pi@raspberrypi:~/code/C/22.DHT11 $ gcc DHT11.cpp DHT.cpp -o DHT11 -lwiringPi
pi@raspberrypi:~/code/C/22.DHT11 $ ls
DHT11  DHT11.cpp  DHT.cpp  DHT.hpp
pi@raspberrypi:~/code/C/22.DHT11 $ sudo ./DHT11
The sumCnt is : 1
DHTLIB_ERROR_TIMEOUT!
Humidity is -999.00 %, Temperature is -999.00 *C

The sumCnt is : 2
DHTLIB_ERROR_TIMEOUT!
Humidity is -999.00 %, Temperature is -999.00 *C

The sumCnt is : 3
DHT11,OK!
Humidity is 45.00 %, Temperature is 28.00 *C

The sumCnt is : 4
DHT11,OK!
Humidity is 44.00 %, Temperature is 27.80 *C
```

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 message to 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 humidity data read to member variables double humidity 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); //
```

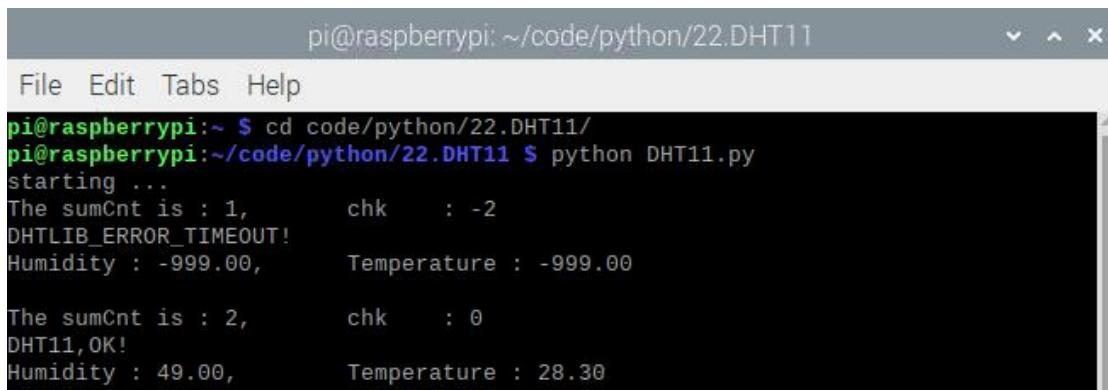
```
};
```

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:



The terminal window shows the following output:

```
pi@raspberrypi: ~/code/python/22.DHT11
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/22.DHT11/
pi@raspberrypi:~/code/python/22.DHT11 $ python DHT11.py
starting ...
The sumCnt is : 1,      chk    : -2
DHTLIB_ERROR_TIMEOUT!
Humidity : -999.00,    Temperature : -999.00

The sumCnt is : 2,      chk    : 0
DHT11,OK!
Humidity : 49.00,     Temperature : 28.30
```

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 23 Matrix Keypad

Overview

In this lesson you will learn how to use the Raspberry Pi to manipulate the matrix keypad.

Parts Required

1 x Raspberry Pi

1 x Matrix keyboard

8 x Double male Jumper Wires

1 x Breadboard

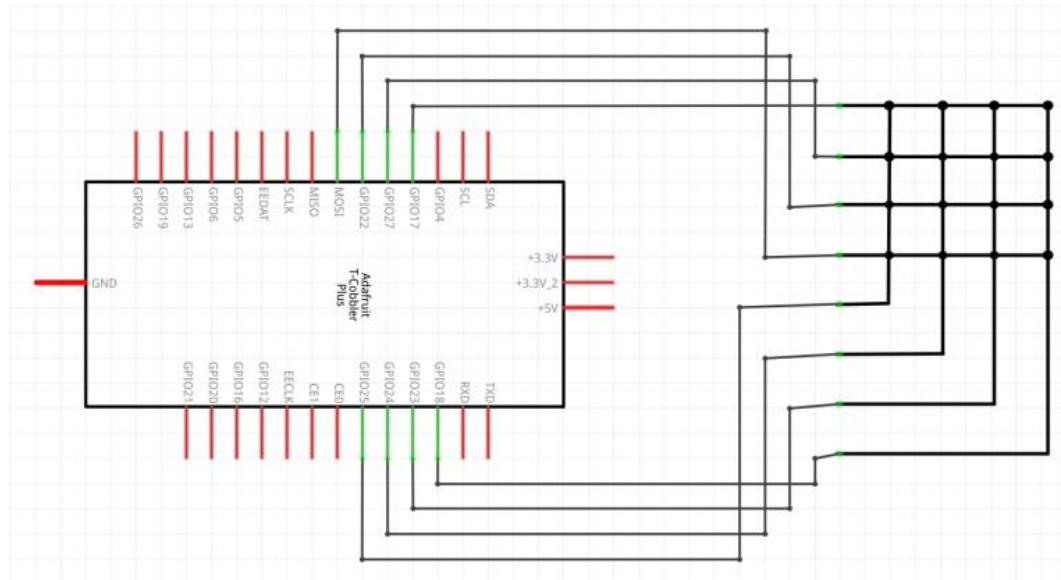
Product Introduction

Matrix Keypad

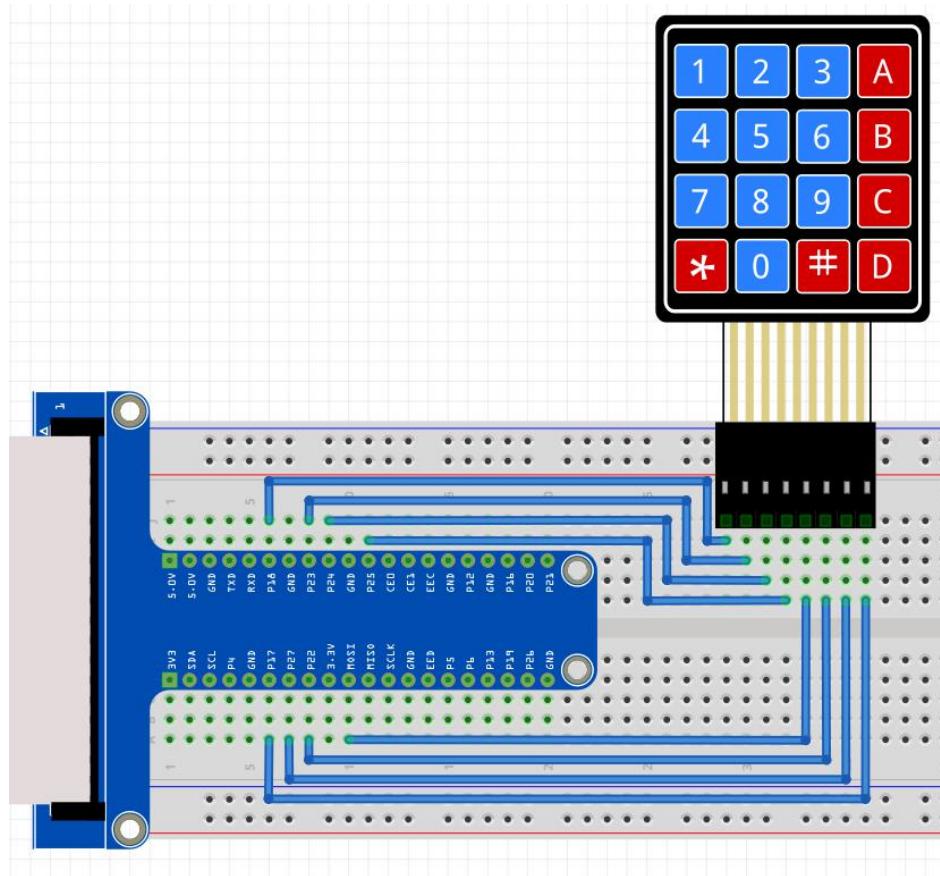
Keypad is a device that integrates a number of keys. Like the integration of LED matrix, in 4x4 Keypad each row of keys is connected in with one pin and it is the same as each column. Such connection can reduce the occupation of processor port.



Connection Diagram

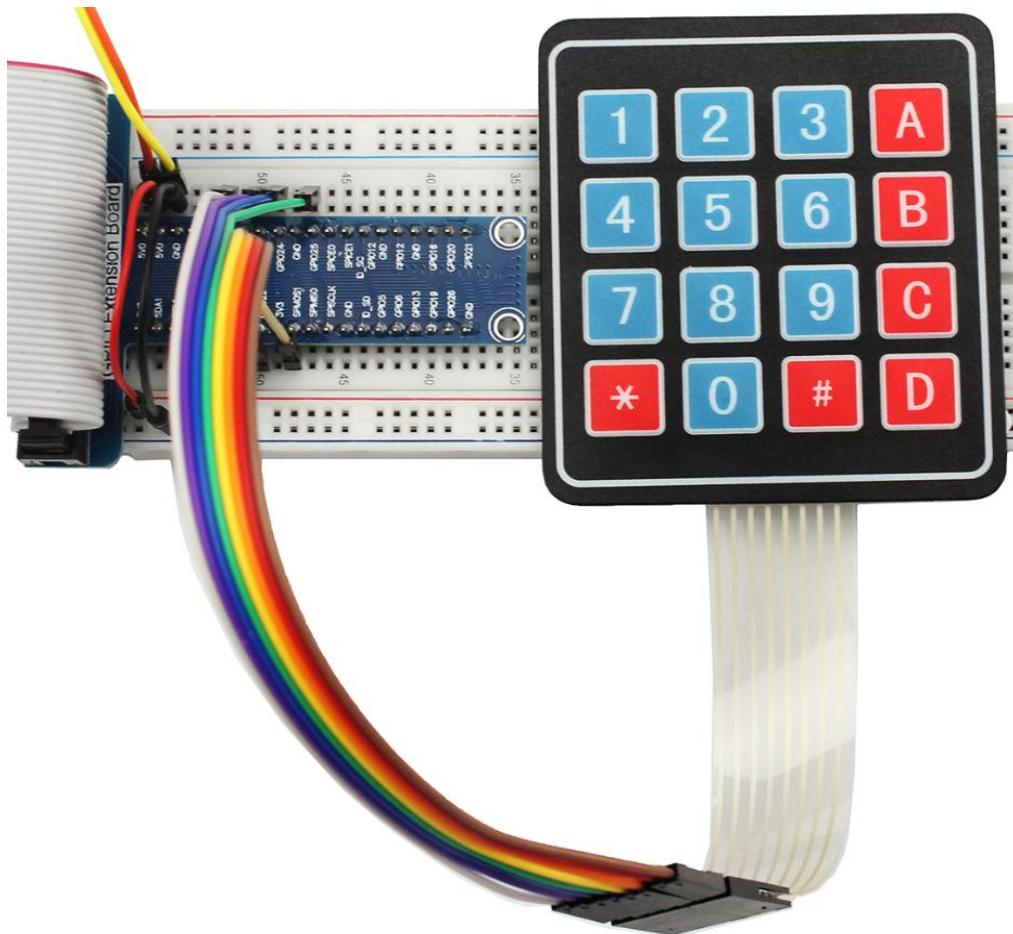


Wiring Diagram





Example Figure



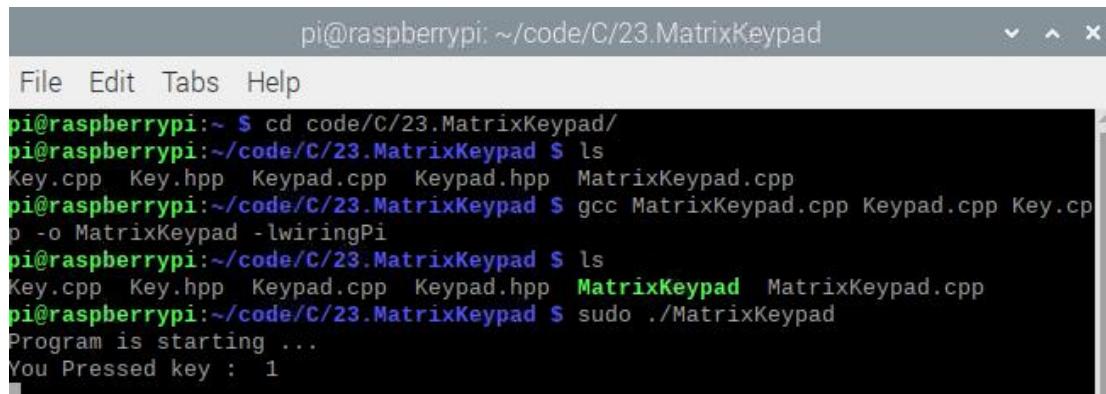
C code

Open the terminal and enter the "cd code / C / 23.MatrixKeypad /" command to enter the "MatrixKeypad" code directory;

Enter the "ls" command to view the files "Key.cpp", "Key.hpp", "Keypad.cpp", "Keypad.hpp", and "MatrixKeypad.cpp" in the directory;

```
pi@raspberrypi:~/code/C/23.MatrixKeypad
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/23.MatrixKeypad/
pi@raspberrypi:~/code/C/23.MatrixKeypad$ ls
Key.cpp  Key.hpp  Keypad.cpp  Keypad.hpp  MatrixKeypad.cpp
pi@raspberrypi:~/code/C/23.MatrixKeypad$
```

Enter "gcc MatrixKeypad.cpp Keypad.cpp Key.cpp -o MatrixKeypad -lwiringPi" command, Generate "Key.cpp", "Key.hpp", "Keypad.cpp", "Keypad.hpp", and "MatrixKeypad.cpp" into executable files "MatrixKeypad", enter "ls" command to view; enter "sudo ./ The MatrixKeypad" command runs the code, and the results are as follows:



```

pi@raspberrypi: ~ cd code/C/23.MatrixKeypad
pi@raspberrypi:~/code/C/23.MatrixKeypad $ ls
Key.cpp Key.hpp Keypad.cpp Keypad.hpp MatrixKeypad.cpp
pi@raspberrypi:~/code/C/23.MatrixKeypad $ gcc MatrixKeypad.cpp Keypad.cpp Key.cpp -o MatrixKeypad -lwiringPi
pi@raspberrypi:~/code/C/23.MatrixKeypad $ ls
Key.cpp Key.hpp Keypad.cpp Keypad.hpp MatrixKeypad MatrixKeypad.cpp
pi@raspberrypi:~/code/C/23.MatrixKeypad $ sudo ./MatrixKeypad
Program is starting ...
You Pressed key : 1

```

This code is used to get all the key codes of the 4x4 matrix keyboard. When one of the keys is pressed, the key code will be printed in the terminal window.

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

```

#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = { //key code
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
//create Keypad object
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

int main(){
    printf("Program is starting ... \n");
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("setup wiringPi failed !");
}

```

```

        return 1;
    }
    char key = 0;
    keypad.setDebounceTime(50);
    while(1){
        key = keypad.getKey(); //get the state of keys
        if (key){           //if a key is pressed, print out its key code
            printf("You Pressed key : %c \n",key);
        }
    }
    return 1;
}

```

Code Interpretation

In this project code, we use two custom library file "Keypad.hpp" and "Key.hpp". They are located in the same directory with program files "MatrixKeypad.cpp", "Keypad.cpp" and "Key.cpp". Library Keypad is transplanted from the Arduino library Keypad. And this library file provides a method to read the keyboard. By using this library, we can easily read the matrix keyboard. First, define the information of the matrix keyboard used in this project: the number of rows and columns, code of each key and GPIO pin connected to each column and each row. It is necessary to include the header file "Keypad.hpp".

```

#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = { //key code
    {'1','2','3','A'},
    {'4','5','6','B'},
    {'7','8','9','C'},
    {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
And then, based on the above information, instantiate a Keypad class object to operate
the matrix keyboard.
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

```

Set the debounce time to 50ms, and this value can be set based on the actual use of the keyboard flexibly, with default time 10ms.

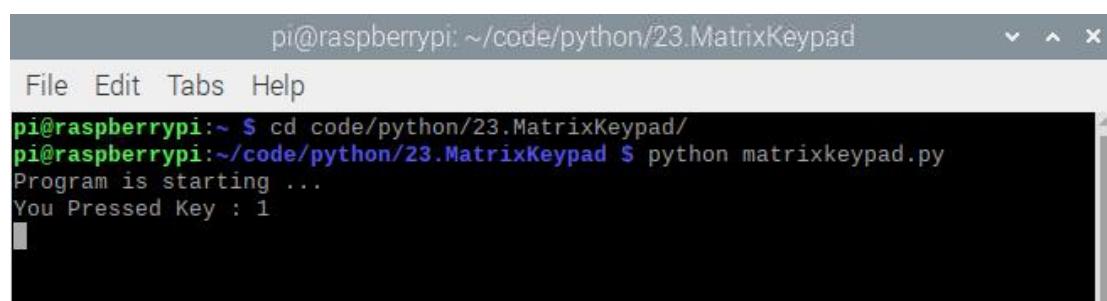
```
keypad.setDebounceTime(50);
```

In the "while" cycle, use the function "key= keypad.getKey ()" to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", then be printed out.

```
while(1){  
    key = keypad.getKey(); //get the state of keys  
    if(key){ //if a key is pressed, print out its key code  
        printf("You Pressed key : %c \n",key);  
    }  
}
```

Python code

1. Use the "cd code / python / 23.MatrixKeypad /" command to enter the directory of matrix keys.
2. Use "python matrixkeypad.py" command to execute "matrixkeypad.py" code.



A terminal window titled "pi@raspberrypi: ~ /code/python/23.MatrixKeypad". The window shows the following text:
File Edit Tabs Help
pi@raspberrypi: ~ \$ cd code/python/23.MatrixKeypad/
pi@raspberrypi: ~/code/python/23.MatrixKeypad \$ python matrixkeypad.py
Program is starting ...
You Pressed Key : 1

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

```
import RPi.GPIO as GPIO
import Keypad
ROWS = 4
COLS = 4
keys = [    '1','2','3','A',
           '4','5','6','B',
           '7','8','9','C',
           '*', '0','#','D'      ]
rowsPins = [12,16,18,22]
colsPins = [19,15,13,11]
def loop():
    keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)
    keypad.setDebounceTime(50)
    while(True):
        key = keypad.getKey()
        if(key != keypad.NULL):
            print ("You Pressed Key : %c "%(key))
    if __name__ == '__main__':
        print ("Program is starting ... ")
        try:
            loop()
        except KeyboardInterrupt:
            GPIO.cleanup()
```

Code Interpretation

```
import RPi.GPIO as GPIO
import Keypad
ROWS = 4
COLS = 4
keys = [    '1','2','3','A',
           '4','5','6','B',
           '7','8','9','C',
           '*', '0','#','D'      ]
rowsPins = [12,16,18,22]

colsPins = [19,15,13,11]
```

And then, based on the above information, instantiate a Keypad class object to operate the matrix keyboard.

```
def loop():
    keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)
    keypad.setDebounceTime(50)
```

Set the debounce time to 50ms, and this value can be set based on the actual use of the keyboard flexibly, with default time 10ms.

```
while(True):
    key = keypad.getKey()
    if(key != keypad.NULL):
        print ("You Pressed Key : %c "%(key))
```

In the "while" cycle, use the function “key= keypad.getKey ()” to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", and then be printed out.

Lesson 24 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

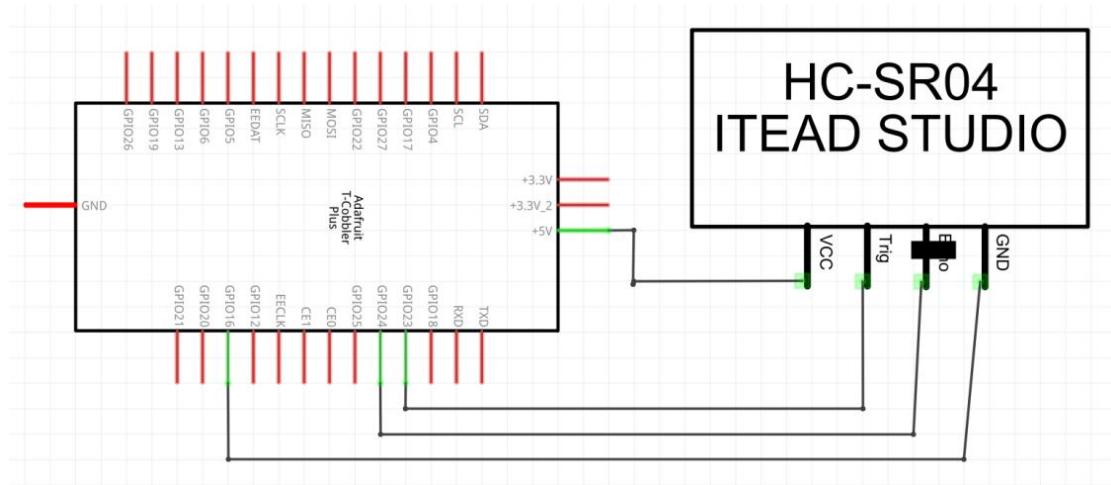
Product Introduction

Ultrasonic ranging

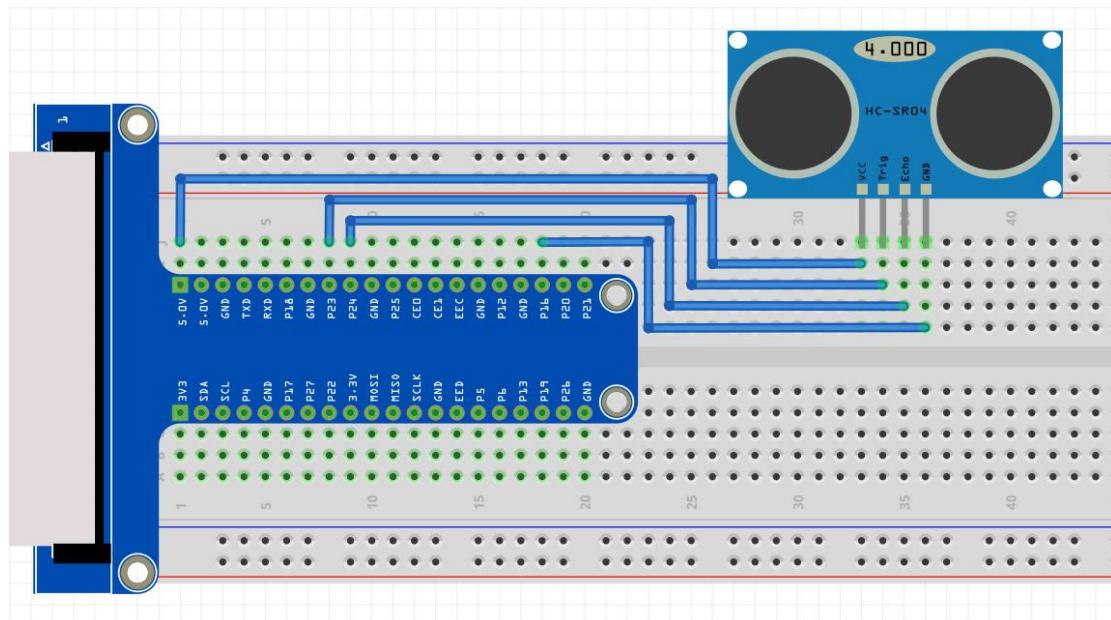
Ultrasonic ranging module use the principle that ultrasonic will reflect when it encounters obstacles. Start counting the time when ultrasonic is transmitted. And when ultrasonic encounters an obstacle, it will reflect back. The counting will end after ultrasonic is received, and the time difference 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.



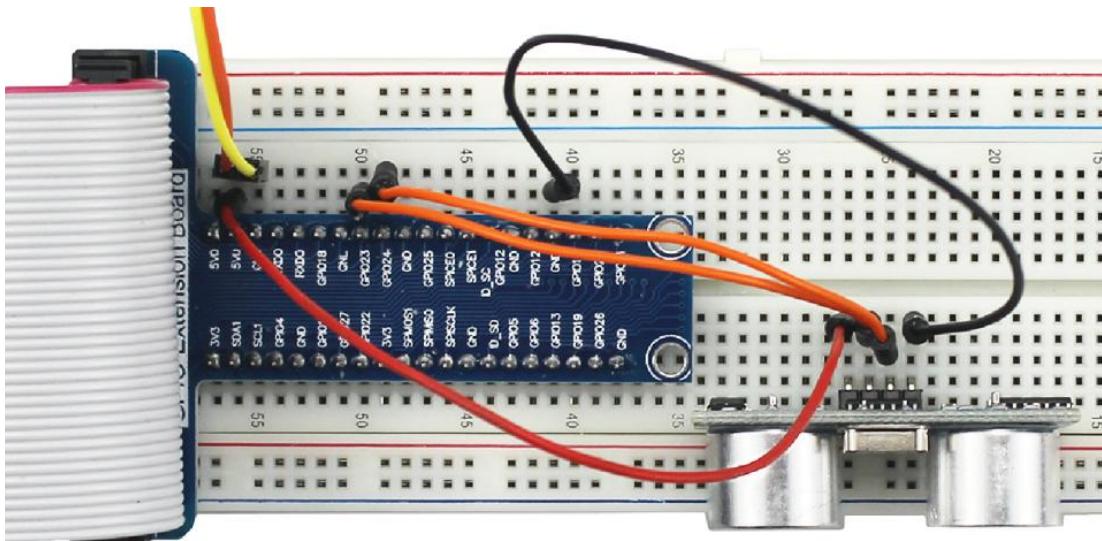
Connection Diagram



Wiring Diagram



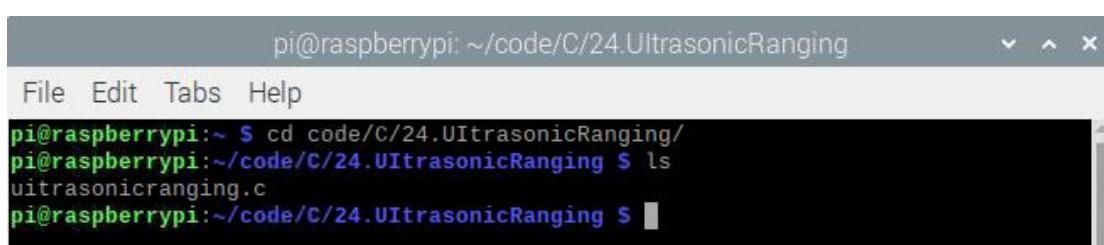
Example Figure



C code

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

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



```
pi@raspberrypi: ~ /code/C/24.UltrasonicRanging
File Edit Tabs Help
pi@raspberrypi: ~ $ cd code/C/24.UltrasonicRanging/
pi@raspberrypi: ~/code/C/24.UltrasonicRanging $ ls
ultrasonicranging.c
pi@raspberrypi: ~/code/C/24.UltrasonicRanging $
```

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

The results are shown below :

```
pi@raspberrypi:~/code/C/24.UltrasonicRanging
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/24.UltrasonicRanging/
pi@raspberrypi:~/code/C/24.UltrasonicRanging$ ls
ultrasonicranging.c
pi@raspberrypi:~/code/C/24.UltrasonicRanging$ gcc ultrasonicranging.c -o ultras
onicranging -lwiringPi
pi@raspberrypi:~/code/C/24.UltrasonicRanging$ ls
ultrasonicranging ultrasonicranging.c
pi@raspberrypi:~/code/C/24.UltrasonicRanging$ sudo ./ultrasonicranging
starting ...
The distance is : 13.35 cm
The distance is : 40.12 cm
The distance is : 19.01 cm
The distance is : 15.98 cm
The distance is : 15.10 cm
The distance is : 15.93 cm
```

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 message to 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 += (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. $\text{timeOut} = 2 * \text{MAX_DISTANCE} / 100 / 340 * 1000000$. The constant part behind is approximately equal to 58.8.

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

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.

```
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.

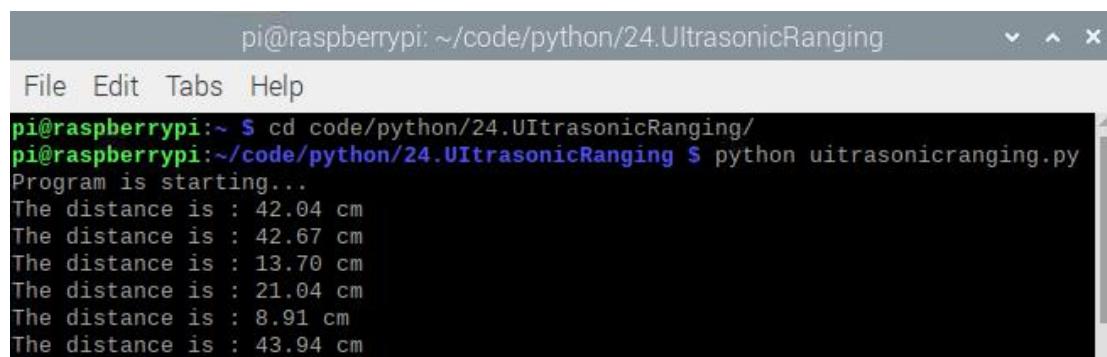
```
while(1){  
    distance = getSonar();  
    printf("The distance is : %.2f cm\n",distance);  
    delay(1000);  
}
```

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.UltrasonicRanging /" command to enter the "UltrasonicRanging" directory.
2. Use "python ultrasonicranging.py" command to execute "ultrasonicranging.py" code.



A terminal window titled "pi@raspberrypi: ~/code/python/24.UltrasonicRanging". The window shows the following text:

```
pi@raspberrypi:~ $ cd code/python/24.UltrasonicRanging/  
pi@raspberrypi:~/code/python/24.UltrasonicRanging $ python ultrasonicranging.py  
Program is starting...  
The distance is : 42.04 cm  
The distance is : 42.67 cm  
The distance is : 13.70 cm  
The distance is : 21.04 cm  
The distance is : 8.91 cm  
The distance is : 43.94 cm
```

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= $2 * \text{MAX_DISTANCE} / 100 / 340 * 1000000$ ”. The constant 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.

Lesson 25 Infrared Sensor

Overview

In this lesson you will learn how to use infrared sensor. The working mode is: output high level when it senses the human body, and output low level when it does not sense the human body.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

1 x 220Ω Resistor

1 x LED

1 x HC SR501

Product Introduction

Infrared Sensor(HC SR501)

The human body has a constant body temperature, usually at 37 degrees, so it emits infrared rays with a specific wavelength of about 10UM. The infrared sensor works by detecting the infrared rays emitted by the human body. About 10UM of infrared light emitted by the human body is enhanced by a Fresnel lens and focused on an infrared sensing source.

Infrared induction sources usually use pyroelectric components. Such components lose their charge balance when they receive changes in the temperature of the human body's infrared radiation, releasing their charges outward, and subsequent circuits can generate alarm signals after detection and processing.

Module parameters:

Operating voltage: DC 5V to 20V

Static power consumption: 65 microamps

Level output: 3.3V high, 0V low

Delay time: adjustable (0.3 seconds to 18 seconds)

Blocking time: 0.2 seconds

Trigger mode: L cannot be repeated, H can be repeated, the default value is H (jump cap selection)

Induction range: less than 120 degrees cone angle, within 7 meters

Working temperature: -15 ~ + 70 degrees

L: Non-repeatable trigger mode. After sensing the human body, the module outputs a high level, and after the delay time expires, the module outputs a low level. During high time, the sensor no longer senses the human body.

H: Repeatable trigger mode. The difference from L is that it can sense the human body all the time, the module outputs high level after sensing the human body, until the human body leaves, and then it starts timing and outputs low level after delaying T time.

Induction blocking time: the time after the induction will remain in the blocking state without causing the external signal to output a high or low level (less than the delay time)

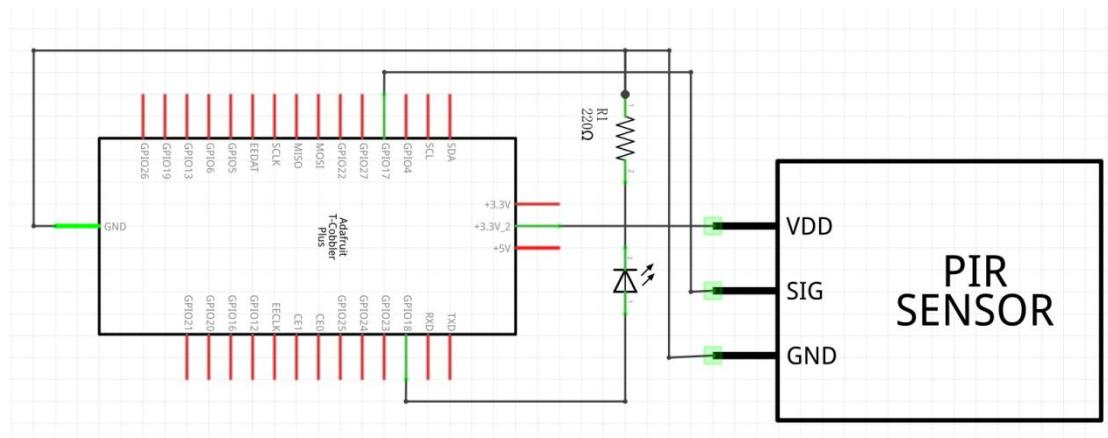
Initialization time: It takes about 1 minute for the module to initialize after power-on. During this time, it will alternately output high or low.

Considering the characteristics of this sensor, the sensor will work highly sensitively when the human body is near or away from the edge. When the human body approaches or moves away from the vertical direction, the sensor cannot work.

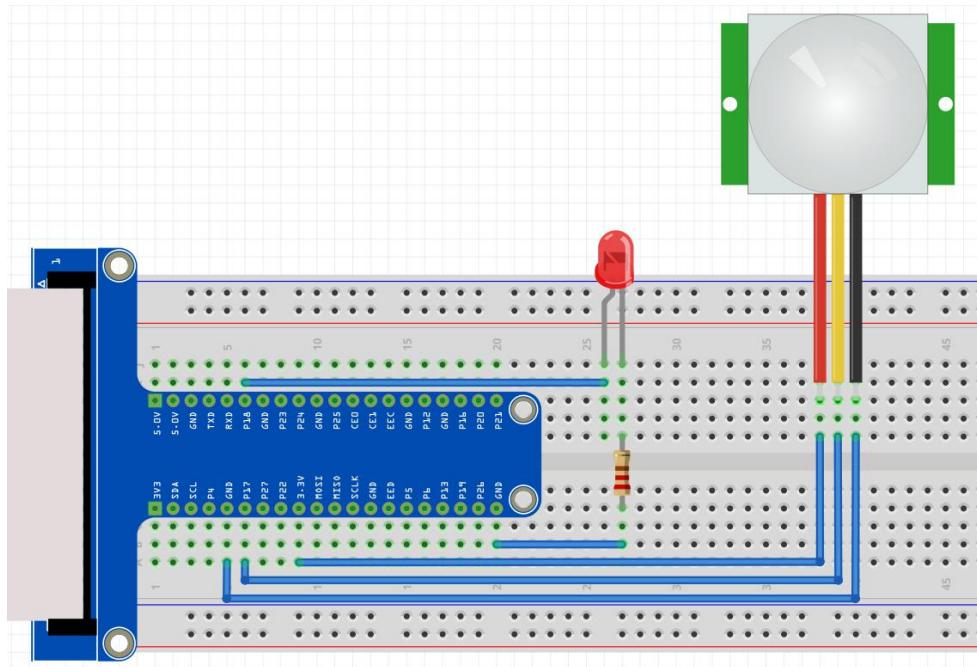
We can think of this sensor as a simple inductive switch when in use.



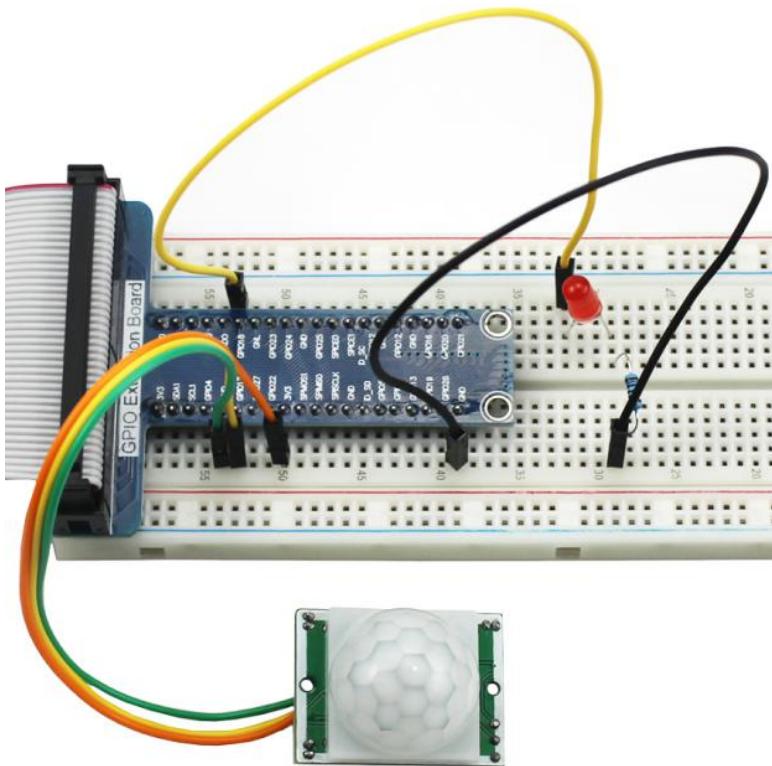
Connection Diagram



Wiring Diagram



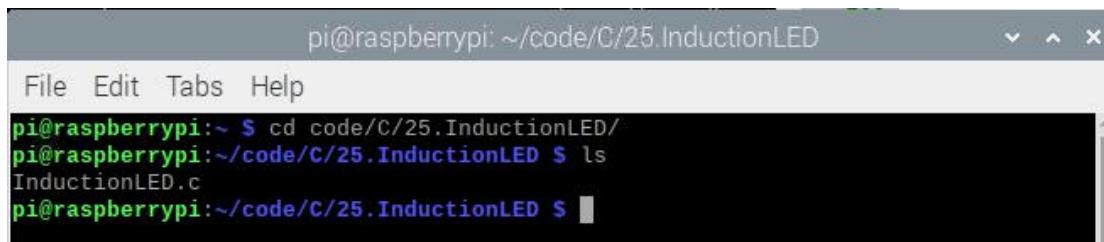
Example Figure



C code

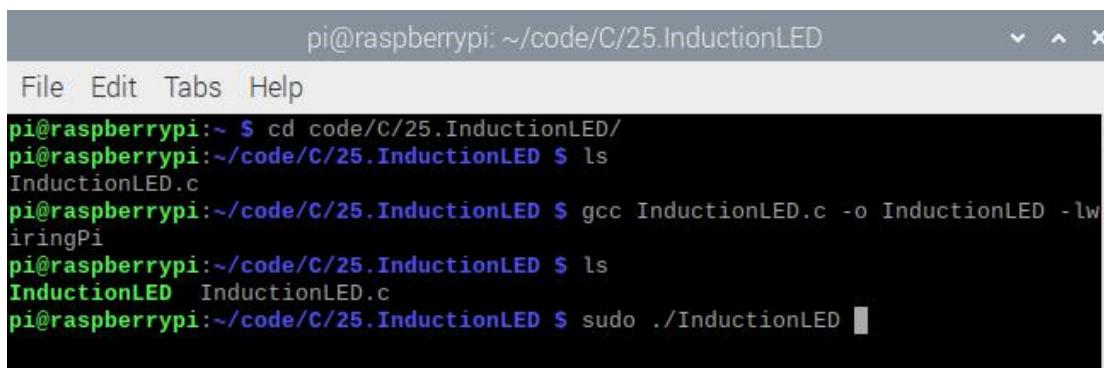
Open the terminal and enter the "cd code / C / 25.InductionLED /" command to enter the "InductionLED" code directory;

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



```
pi@raspberrypi:~/code/C/25.InductionLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/25.InductionLED/
pi@raspberrypi:~/code/C/25.InductionLED $ ls
InductionLED.c
pi@raspberrypi:~/code/C/25.InductionLED $
```

Enter "gcc InductionLED.c -o InductionLED -lwiringPi" command to generate "InductionLED .c" executable file "InductionLED", enter "ls" command to view; enter "sudo ./InductionLED" command to run the code, the results are shown below :



```
pi@raspberrypi:~/code/C/25.InductionLED
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/25.InductionLED/
pi@raspberrypi:~/code/C/25.InductionLED $ ls
InductionLED.c
pi@raspberrypi:~/code/C/25.InductionLED $ gcc InductionLED.c -o InductionLED -lwiringPi
pi@raspberrypi:~/code/C/25.InductionLED $ ls
InductionLED InductionLED.c
pi@raspberrypi:~/code/C/25.InductionLED $ sudo ./InductionLED
```

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

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

#define ledPin      1      //define the ledPin
#define sensorPin 0       //define the sensorPin

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

```
    return 1;
}

pinMode(ledPin, OUTPUT);
pinMode(sensorPin, INPUT);

while(1){

    if(digitalRead(sensorPin) == HIGH){ //if read sensor for high level
        digitalWrite(ledPin, HIGH);    //led on
        printf("led on...\n");
    }
    else {
        digitalWrite(ledPin, LOW);    //led off
        printf("...led off\n");
    }
}

return 0;
}
```

Code Interpretation

In the main function, first use "if (digitalRead (sensorPin) == HIGH)" to detect whether there is a human body and return to high level if a human body is detected. "digitalWrite (ledPin, HIGH)" is used to turn on the LED light when a human body is detected.

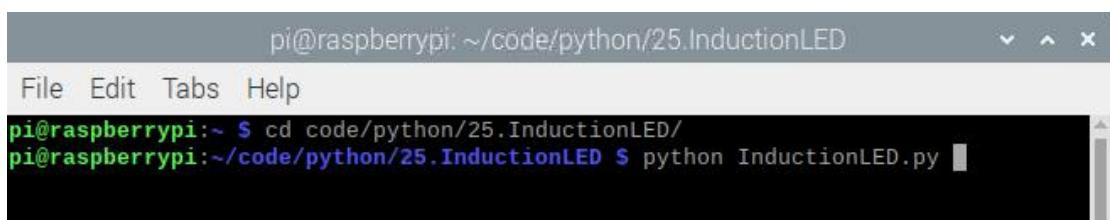
```
while(1){

    if(digitalRead(sensorPin) == HIGH){ //if read sensor for high level
        digitalWrite(ledPin, HIGH);    //led on
        printf("led on...\n");
    }
    else {
        digitalWrite(ledPin, LOW);    //led off
        printf("...led off\n");
    }
}
```

The code is the same as the key control LED course.

Python code

1. Use the "cd code / python / 25.InductionLED /" command to enter the directory of the infrared sensor.
2. Use "python InductionLED.py" command to execute "InductionLED.py" code.



```
pi@raspberrypi: ~/code/python/25.InductionLED
pi@raspberrypi:~/code/python/25.InductionLED $ python InductionLED.py
```

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

```
import RPi.GPIO as GPIO

ledPin = 12      # define the ledPin
sensorPin = 11    # define the sensorPin

def setup():
    print ('starting...')
    GPIO.setmode(GPIO.BOARD)          # Numbers GPIOs by physical location
    GPIO.setup(ledPin, GPIO.OUT)       # Set ledPin's mode is output
    GPIO.setup(sensorPin, GPIO.IN)     # Set sensorPin's mode is input

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

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

Code Interpretation

In the main function "loop()", first use "if GPIO.input(sensorPin)==GPIO.HIGH" to detect whether there is a human body, and return a high level when a human body is detected. When a human body is detected, use the function "GPIO.output(ledPin,GPIO.HIGH)" to turn on the LED light.

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

The code is the same as the “button” course code.

Lesson 26 MPU6050

Overview

In this lesson you will learn an attitude sensor MPU6050 that integrates an accelerometer and a gyroscope.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

1 x Breadboard

1 x MPU6050

Product Introduction

MPU6050

MPU6050 is an integrated 3-axis accelerometer, 3-axis angular accelerometer (called a gyroscope) and a digital attitude processor (DMP). In the mpu6050, we use a gyroscope sensor to measure the angle and an acceleration sensor to measure the acceleration. .

Gyroscope: The principle of a gyroscope is that the direction of the rotating axis of a rotating object will not change when it is not affected by external forces. For this reason, use it to maintain direction. Then use various methods to read the direction indicated by the axis and automatically transmit the data signal to the control system.

Acceleration sensor: An acceleration sensor is a sensor that can measure acceleration. It usually consists of mass, damper, elastic element, sensitive element, and adaptive circuit. During acceleration, the sensor uses the Newton's second law to obtain the acceleration value by measuring the inertial force on the mass. Depending on the sensor's sensitive components, common acceleration sensors include capacitive, inductive, strain, piezoresistive, and piezoelectric.

6-axis gyroscope sensor: The DMP receives and processes data from the gyroscope, accelerometer, and external sensors. The processing results can be read from DMP

registers or buffered through FIFO. The DMP is authorized to generate an interrupt using an external pin of the MPU.

The port description of the MPU6050 module is as follows:

Pin name Pin number Description

VCC 1-----Positive power supply, 5V

GND 2-----Negative power supply

SCL3 I2C-----communication clock pin

SDA 4 I2C-----communication data pin

XDA 5 I2C-----host data pin, can be connected to other devices.

XCL 6 I2C-----master clock pin, can be connected to other devices.

AD0 7 I2C-----address bit control pin.

-----Low: device address is 0x68

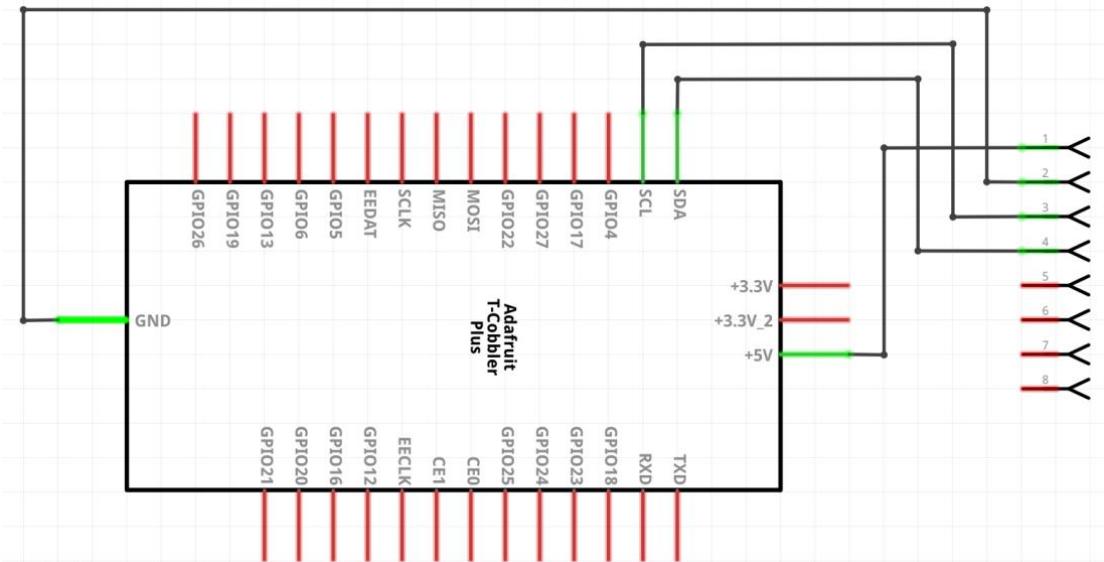
-----High: device address is 0x69

INT 8-----output interrupt pin

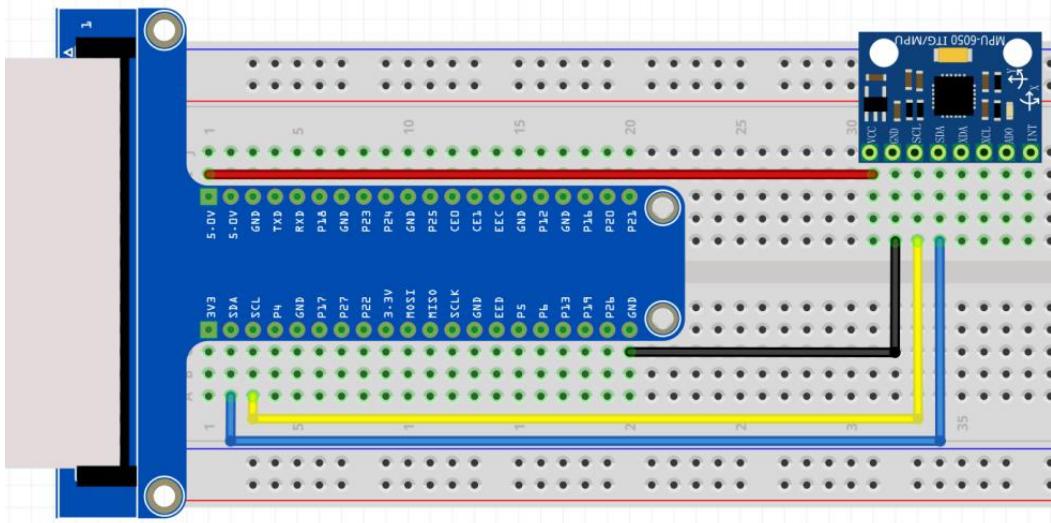




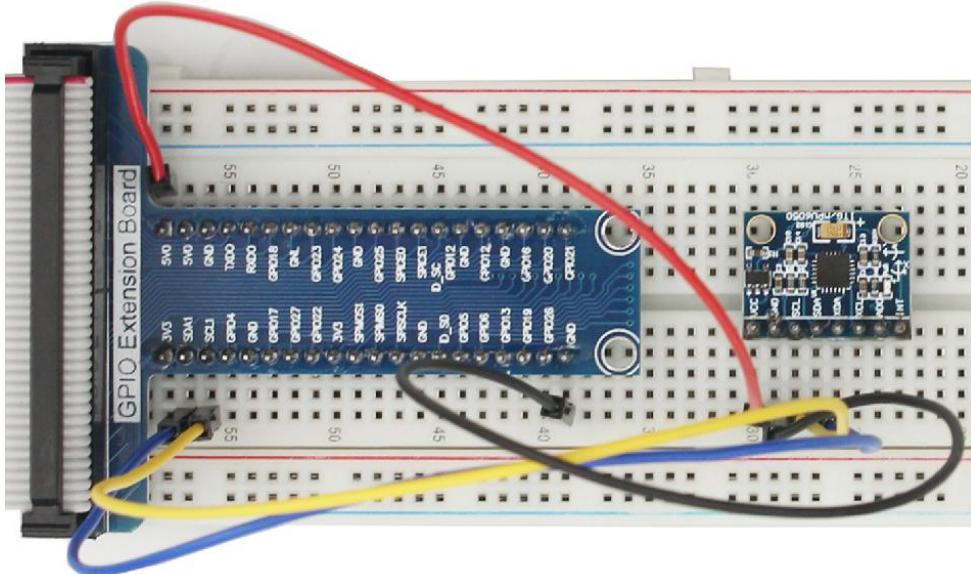
Connection Diagram



Wiring Diagram



Example Figure



C code

Open the terminal and enter the "cd code / C / 26.MPU6050 /" command to enter the "MPU6050" code directory;

Enter the "ls" command to view the files "I2Cdev.cpp", "I2Cdev.h", "MPU6050.cpp", "MPU6050.h", "MPU6050RAW.cpp" in the directory;

```
pi@raspberrypi:~/code/C/26.MPU6050
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/26.MPU6050/
pi@raspberrypi:~/code/C/26.MPU6050$ ls
I2Cdev.cpp I2Cdev.h MPU6050.cpp MPU6050.h MPU6050RAW.cpp
pi@raspberrypi:~/code/C/26.MPU6050$
```

Enter "gcc I2Cdev.cpp MPU6050.cpp MPU6050RAW.cpp -o MPU6050RAW -lwiringPi" to generate the executable file "MPU6050RAW", enter "ls" command to view; enter "sudo ./MPU6050RAW" command to run the code.

The results are as follows:

```
pi@raspberrypi:~/code/C/26.MPU6050
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/26.MPU6050/
pi@raspberrypi:~/code/C/26.MPU6050$ ls
I2Cdev.cpp I2Cdev.h MPU6050.cpp MPU6050.h MPU6050RAW.cpp
pi@raspberrypi:~/code/C/26.MPU6050$ gcc I2Cdev.cpp MPU6050.cpp MPU6050RAW.cpp -o MPU6050RAW -lwiringPi
pi@raspberrypi:~/code/C/26.MPU6050$ ls
I2Cdev.cpp I2Cdev.h MPU6050.cpp MPU6050.h MPU6050RAW MPU6050RAW.cpp
pi@raspberrypi:~/code/C/26.MPU6050$ sudo ./MPU6050RAW
Initializing I2C devices...
Testing device connections...
MPU6050 connection failed
a/g:    128     108   19758      -700     -166     123
a/g:  0.01 g  0.01 g  1.21 g   -5.34 d/s  -1.27 d/s  0.94 d/s
a/g:    120      28   19208      -666     -125     130
a/g:  0.01 g  0.00 g  1.17 g   -5.08 d/s  -0.95 d/s  0.99 d/s
a/g:    120      66   18776      -674     -123     133
a/g:  0.01 g  0.00 g  1.15 g   -5.15 d/s  -0.94 d/s  1.02 d/s
a/g:    140      56   18584      -694     -159     131
a/g:  0.01 g  0.00 g  1.13 g   -5.30 d/s  -1.21 d/s  1.00 d/s
a/g:    180      20   19170      -722     -207     123
```

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

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 accelgyro;          //instantiate a MPU6050 class object

int16_t ax, ay, az;         //store acceleration data
int16_t gx, gy, gz;         //store gyroscope data

void setup() {
    // initialize device
    printf("Initializing I2C devices...\n");
    accelgyro.initialize();    //initialize MPU6050

    // verify connection
    printf("Testing device connections...\n");
    printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" :
```

```
"MPU6050 connection failed\n");
}

void loop() {
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    // display accel/gyro x/y/z values
    printf("a/g: %6hd %6hd %6hd    %6hd %6hd %6hd\n",ax,ay,az,gx,gy,gz);
    printf("a/g: %.2f g %.2f g %.2f g    %.2f d/s %.2f d/s %.2f d/s
\n",(float)ax/16384,(float)ay/16384,(float)az/16384,
    (float)gx/131,(float)gy/131,(float)gz/131);
}

int main()
{
    setup();
    while(1){
        loop();
    }
    return 0;
}
```

Code Interpretation

In the code ,two library files "MPU6050.h" and "I2Cdev.h" are used. They will be compiled like the others. The MPU6050 class is used to operate the MPU6050. When it is used, an object is instantiated first.

MPU6050 accelgyro;

In the function setup "setup ()" function, the MPU6050 will be initialized and the result will be judged.

```
void setup() {
    // initialize device
    printf("Initializing I2C devices...\n");
    accelgyro.initialize();      //initialize MPU6050

    // verify connection
    printf("Testing device connections...\n");
```

```
    printf(accelgyro.testConnection() ? "MPU6050 connection successful\n" :  
        "MPU6050 connection failed\n");  
}
```

In the "loop ()" function, read the raw data of the MPU6050 and print it out, then convert the raw data into the corresponding acceleration and angular velocity, and then print out the converted data.

```
void loop() {  
    // read raw accel/gyro measurements from device  
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);  
    // display accel/gyro x/y/z values  
    printf("a/g: %6hd %6hd %6hd %6hd %6hd %6hd\n", ax, ay, az, gx, gy, gz);  
    printf("a/g: %.2f g %.2f g %.2f g %.2f d/s %.2f d/s %.2f d/s  
    \n", (float)ax/16384, (float)ay/16384, (float)az/16384,  
          (float)gx/131, (float)gy/131, (float)gz/131);  
}
```

In the main function, call the function setup "setup ()" and the data read, print function "loop ()".

```
int main()  
{  
    setup();  
    while(1){  
        loop();  
    }  
    return 0;  
}
```

Python code

1. Use the "`cd code / python / 26.MPU6050 /`" command to enter the directory of the MPU6050.
2. Use "`python MPU6050.py`" command to execute "MPU6050.py" code.

After the program is executed, the terminal will display the raw data read by the MPU6050, as well as the transformed acceleration and angular velocity.

As shown in the following figure:

```
pi@raspberrypi: ~/code/python/26.MPU6050
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/26.MPU6050/
pi@raspberrypi:~/code/python/26.MPU6050 $ python MPU6050.py
Starting ...
a/g:184 34      19196   -86     -21     16
a/g:0.01 g      0.00 g   1.17 g   -0.66 d/s    -0.16 d/s    0.12 d/s
a/g:220 44      19172   -85     -20     15
a/g:0.01 g      0.00 g   1.17 g   -0.65 d/s    -0.15 d/s    0.11 d/s
a/g:188 50      19212   -86     -22     15
a/g:0.01 g      0.00 g   1.17 g   -0.66 d/s    -0.17 d/s    0.11 d/s
a/g:178 48      19174   -87     -22     15
a/g:0.01 g      0.00 g   1.17 g   -0.66 d/s    -0.17 d/s    0.11 d/s
a/g:188 64      19172   -86     -21     15
a/g:0.01 g      0.00 g   1.17 g   -0.66 d/s    -0.16 d/s    0.11 d/s
```

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

```
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the program will exit.  
    pass
```

Code Interpretation

In the code ,the "MPU6050RAW.py" module is used. This module includes classes for operating the MPU6050.

‘MPU6050RAW.py’: This is a library for operating the MPU6050. You can directly read and set the MPU6050.

Constructor 'def dmp_initialize (self)': It is an initialization function used to wake up the MPU6050. The range of the accelerometer is $\pm 2g$ and the range of the gyroscope is ± 250 degrees / second.

‘def get_acceleration (self): & def get_rotation (self)’: Acquires the raw data of accelerometer and gyroscope.

Instantiate an object at the beginning of the code:

```
mpu = MPU6050RAW.MPU6050()
```

In the setup function 'setup()', initialize MPU6050.

```
def setup():  
    mpu.dmp_initialize()
```

In the main function 'loop ()', first read the raw data of MPU6050 and print it, then convert the raw data into the corresponding acceleration and angular velocity, and finally print out the converted acceleration and angular velocity data.

```
def loop():  
    while(True):  
        accel = mpu.get_acceleration()      #get accelerometer data  
        gyro = mpu.get_rotation()           #get gyroscope data  
        print("a/g:%d\%d\%d\%d\%d\%d\%d\%d  
              "%(accel[0],accel[1],accel[2],gyro[0],gyro[1],gyro[2]))  
        print("a/g:%.2f g\%t%.2f g\%t%.2f g\%t%.2f d/s\%t%.2f d/s\%t%.2f  
              d/s"%(accel[0]/16384.0,accel[1]/16384.0,accel[2]/16384.0,gyro[0]/131.0,gyro[1]/131.  
              0,gyro[2]/131.0))  
        time.sleep(0.1)
```

Lesson 27 Sound Sensor

Overview

In this lesson you will learn about sound sensors, and how to use control LED lights by sound.

Parts Required

1 x Raspberry Pi 4

1 x GPIO T-type Expansion Board and Wires

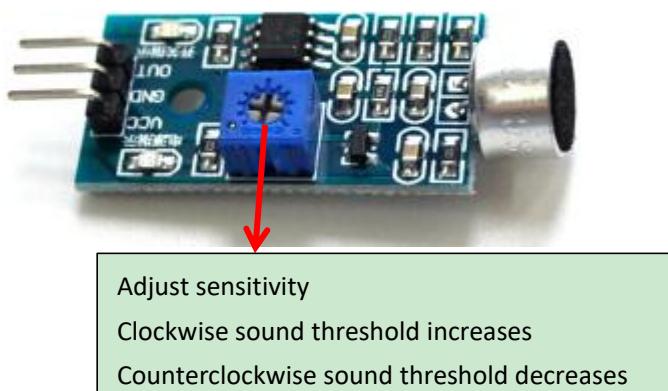
1 x Breadboard

1 x Sound Sensor

Product Introduction

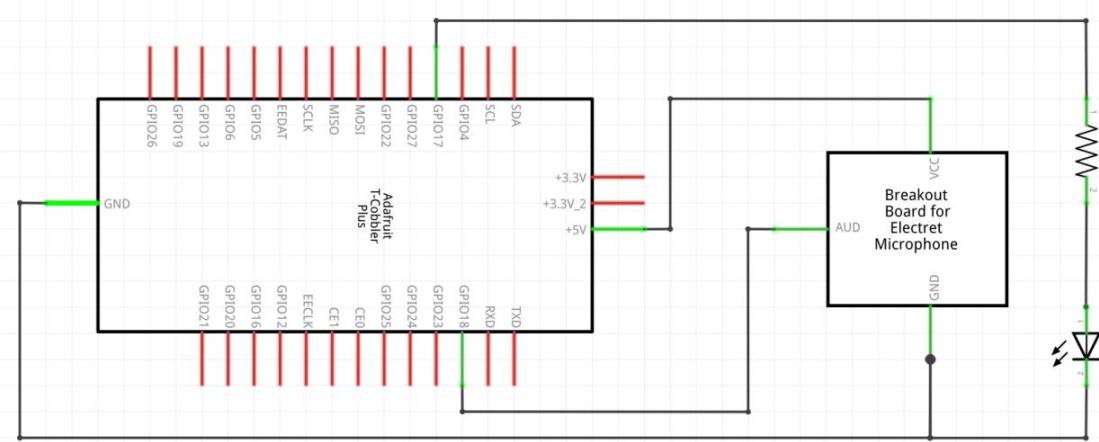
Sound Sensor

The sound sensor is also called a sound switch. When the ambient sound intensity does not reach the set threshold, the module outputs a high level. When the external ambient sound intensity exceeds the set threshold, the module outputs a low level. .

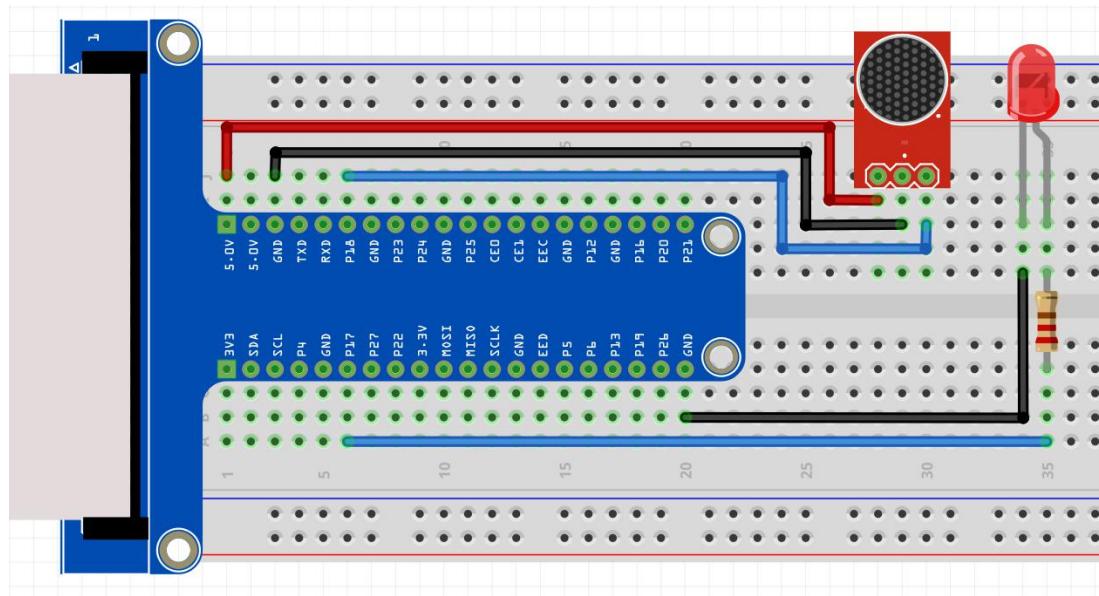




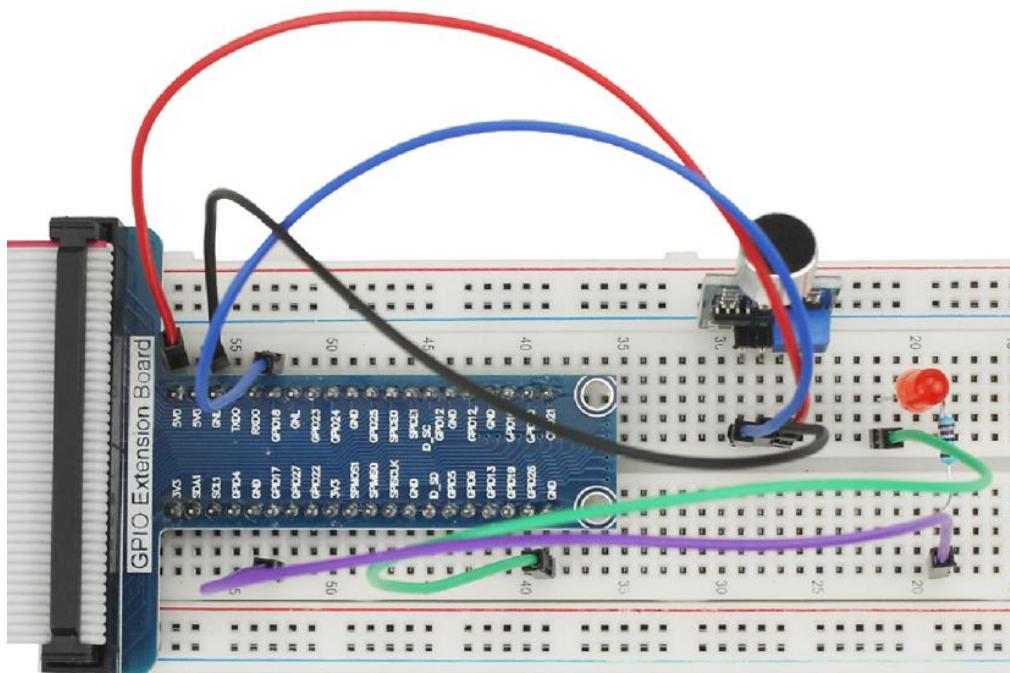
Connection Diagram



Wiring Diagram



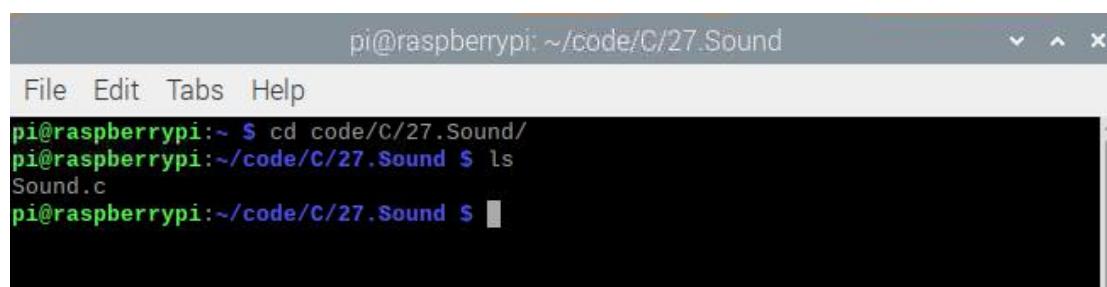
Example Figure



C code

Open the terminal and enter the "cd code / C / 27.Sound /" command to enter the "Sound" code directory;

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



```
pi@raspberrypi: ~code/C/27.Sound
File Edit Tabs Help
pi@raspberrypi:~$ cd code/C/27.Sound/
pi@raspberrypi:~/code/C/27.Sound $ ls
Sound.c
pi@raspberrypi:~/code/C/27.Sound $
```

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

The result is shown in the below:



A terminal window titled "pi@raspberrypi: ~/code/C/27.Sound". The window shows the following command-line session:

```
pi@raspberrypi:~ $ cd code/C/27.Sound/
pi@raspberrypi:~/code/C/27.Sound $ ls
Sound.c
pi@raspberrypi:~/code/C/27.Sound $ gcc Sound.c -o Sound -lwiringPi
pi@raspberrypi:~/code/C/27.Sound $ ls
Sound Sound.c
pi@raspberrypi:~/code/C/27.Sound $ sudo ./Sound
shock:
LED...LOW
shock:
LED...LOW
```

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

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

#define ledPin      0      //define the ledPin
#define soundPin    1      //define the sensorPin

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

    pinMode(ledPin, OUTPUT);
    pinMode(soundPin, INPUT);

    while(1){
        printf("shock:\n");
        if(digitalRead(soundPin) == HIGH){ //if read sensor for high level
            digitalWrite(ledPin, HIGH);    //led on
            printf("LED...HIGH\n");
        }
        else {
            digitalWrite(ledPin, LOW);   //led off
        }
    }
}
```

```
        printf("LED...LOW\n");
    }
    delay(500);
}

return 0;
}
```

Code Interpretation

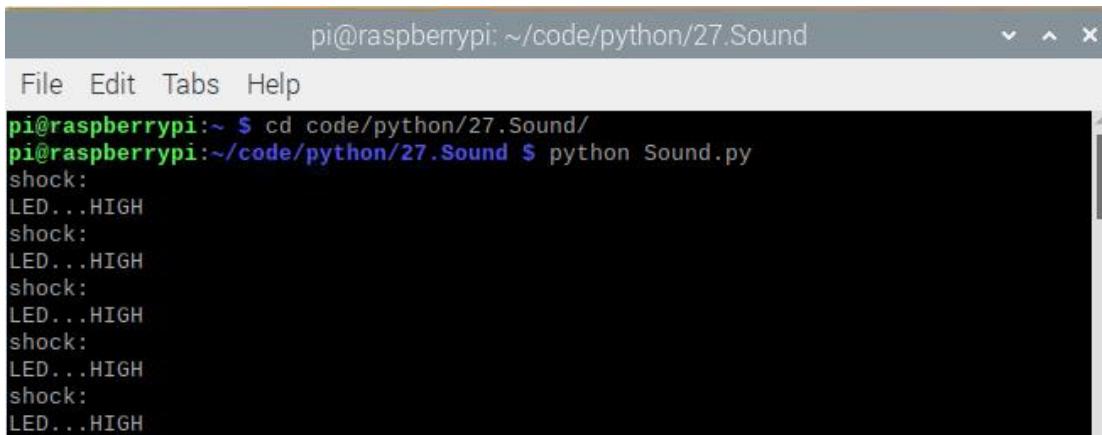
In the main function, use the "if(digitalRead(soundPin) == HIGH)" statement to detect whether there is sound, and return high level if there is sound. After detecting the sound, turn on the LED with "digitalWrite(ledPin, HIGH)".

```
while(1){
    printf("shock:\n");
    if(digitalRead(soundPin) == HIGH){ //if read sensor for high level
        digitalWrite(ledPin, HIGH);    //led on
        printf("LED...HIGH\n");
    }
    else {
        digitalWrite(ledPin, LOW);    //led off
        printf("LED...LOW\n");
    }
    delay(500);
}
```

The code is the same as the code for the key control LED course.

Python code

1. Use the "cd code / python / 27.Sound /" command to enter the directory of the sound sensor.
2. Use "python Sound.py" command to execute "Sound.py" code.



A terminal window titled "pi@raspberrypi: ~/code/python/27.Sound". The window shows the command "cd code/python/27.Sound/" followed by "python Sound.py". The output of the program is displayed below, showing repeated messages "shock:" and "LED...HIGH".

```
pi@raspberrypi:~ $ cd code/python/27.Sound/
pi@raspberrypi:~/code/python/27.Sound $ python Sound.py
shock:
LED...HIGH
shock:
LED...HIGH
shock:
LED...HIGH
shock:
LED...HIGH
shock:
LED...HIGH
shock:
LED...HIGH
```

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

```
import RPi.GPIO as GPIO
import time
D0=12
LED=11
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(D0,GPIO.IN)
    GPIO.setup(LED,GPIO.OUT)
    GPIO.output(LED,GPIO.LOW)

def loop():
    while True:
        print('shock:')
        PIN=GPIO.input(D0)
        if PIN == GPIO.HIGH :
            GPIO.output(LED,GPIO.HIGH)
            print("LED...HIGH")
        else :
            GPIO.output(LED,GPIO.LOW)
            print("LED...LOW")
        time.sleep(0.5)
```

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

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

Code Interpretation

First, in the function 'setup ()', define the sound sensor as the input mode and the LED as the output mode.

```
def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(D0,GPIO.IN)
    GPIO.setup(LED,GPIO.OUT)
    GPIO.output(LED,GPIO.LOW)
```

In the 'while' loop of the main function 'loop ()', the value returned by the sound sensor is continuously obtained. If the sound sensor returns a high level, the LED light is turned on, otherwise the LED light is turned off.

```
def loop():
    while True:
        print('shock:')
        PIN=GPIO.input(D0)
        if PIN == GPIO.HIGH :
            GPIO.output(LED,GPIO.HIGH)
            print("LED...HIGH")
        else  :
            GPIO.output(LED,GPIO.LOW)
            print("LED...LOW")
        time.sleep(0.5)
```

Lesson 28 RFID RC522

Overview

In this lesson you will learn how to use RFID, and how to use RC522 RFID card reader to read and write the M1-S50 card.

Parts Required

1 x Raspberry pi

1 x GPIO Extension Board & Wire

1 x BreadBoard

7 x Jumper M/F

1 x RC522 module

1 x Mifare1 S50 Non-standard card

1 x Mifare1 S50 Standard card

Product Introduction

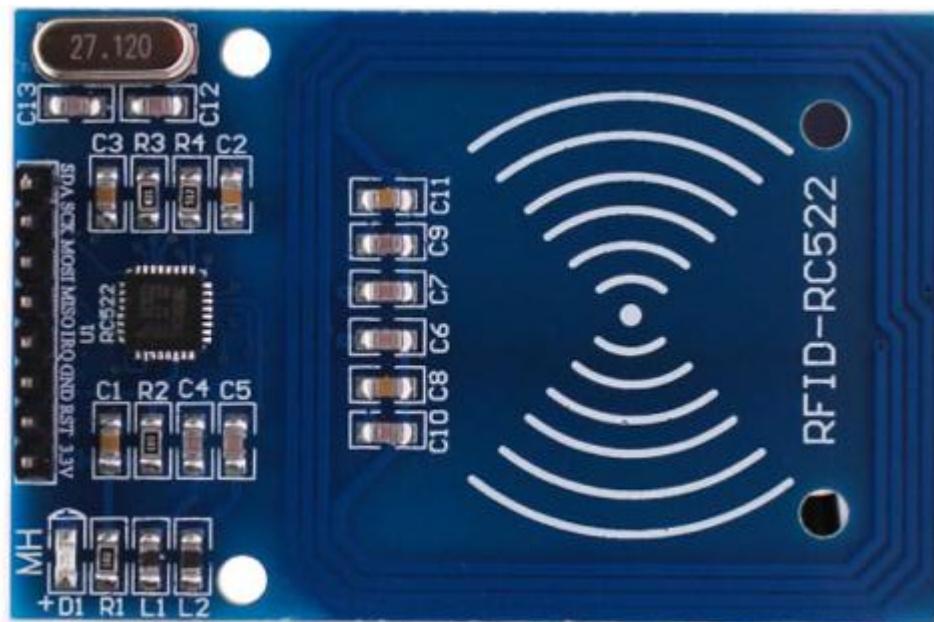
MFRC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality. This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13–26mA (DC) \3. 3V
Idle current	10–13mA (DC) \3. 3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13. 56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm
Operation temperature	20–80 degrees (Celsius)
Storage temperature	40–85 degrees (Celsius)
Operation humidity	5%–95% (Relative humidity)



Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years. The ordinary Mifare1 S50 Card and non-standard Mifare1 S50 Card equipped for RFID Kit are shown below. The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 datablock (Block0-Block3). 64 blocks of 16 sectors will be numbered according absolute address,

from 0 to 63). And each block contains 16 bytes (Byte0-Byte15), $64 \times 16 = 1024$. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
.....
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control which are put in the last block of each sector, and the block is also known as sector trailer, that is Block 3 in each sector. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the vendor code, which has been solidified and can't be changed, and the card serial number is stored here. In addition to the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

- (1) used as general data storage and can be operated for reading and writing.
- (2) used as data value, and can be operated for initializing the value, adding value, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, 4-byte accesscontrol and 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5 password A	FF 07 80 69 access control	B0 B1 B2 B3 B4 B5 password B
---------------------------------	-------------------------------	---------------------------------

The default password of a brand new card is generally 0A1A2A3A4A5 for password A, B0B1B2B3B4B5 for password B, or both the password A and password B are 6

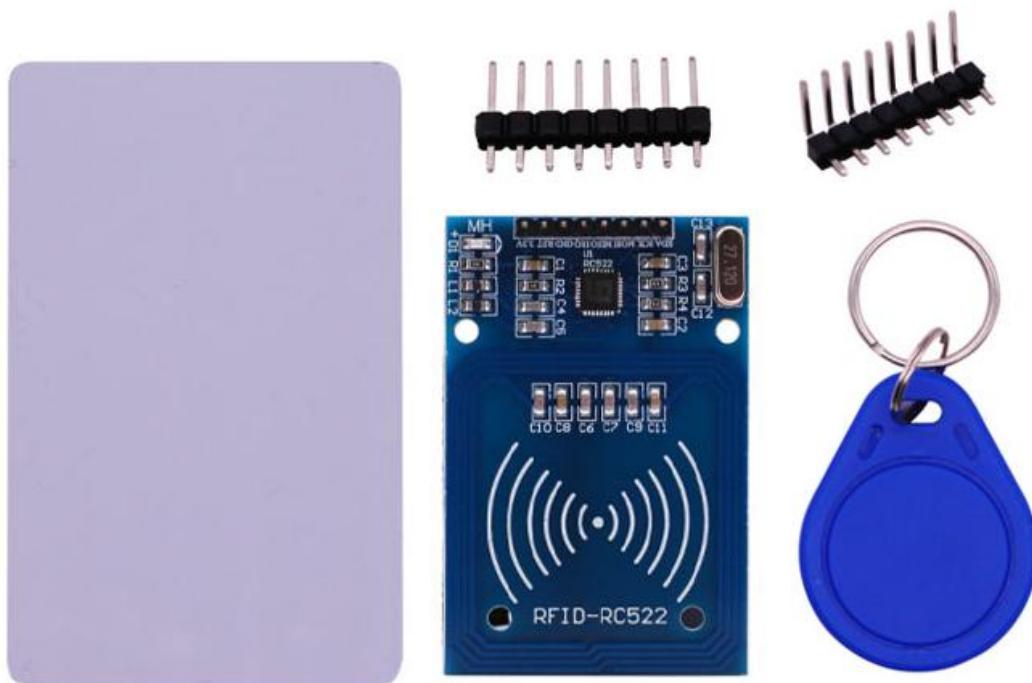
FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

For more details about how to set data blocks and control blocks, please refer to Datasheet.

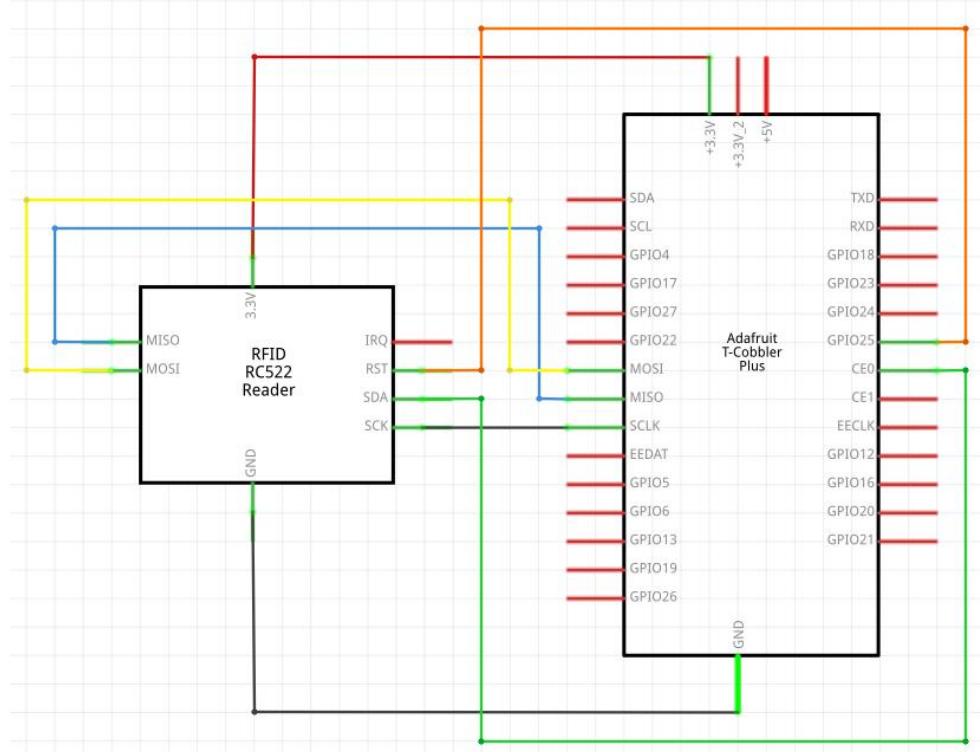
By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read. If you choose to verify password A and then you forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

For Mifare1 S50 cards equipped with RFID kits, the default passwords A and B are FFFFFFFFFFFF.

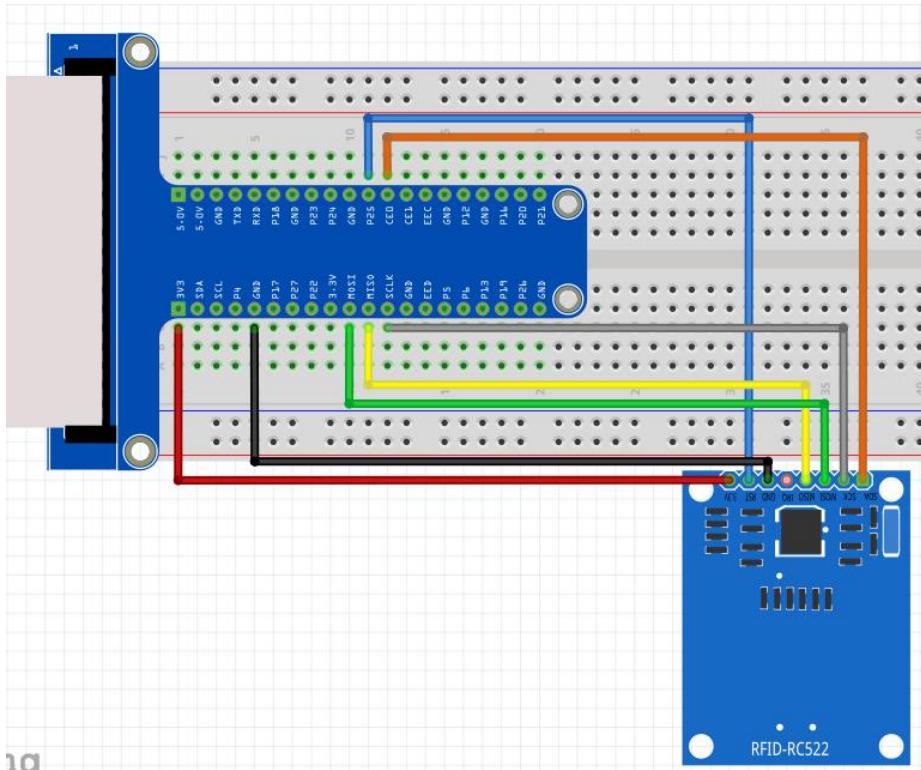




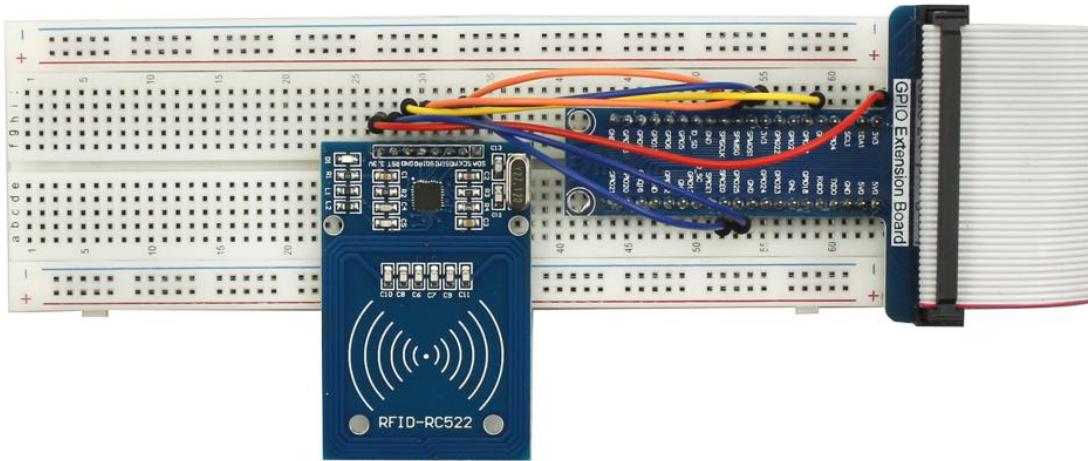
Connection Diagram



Wiring Diagram



Example Figure

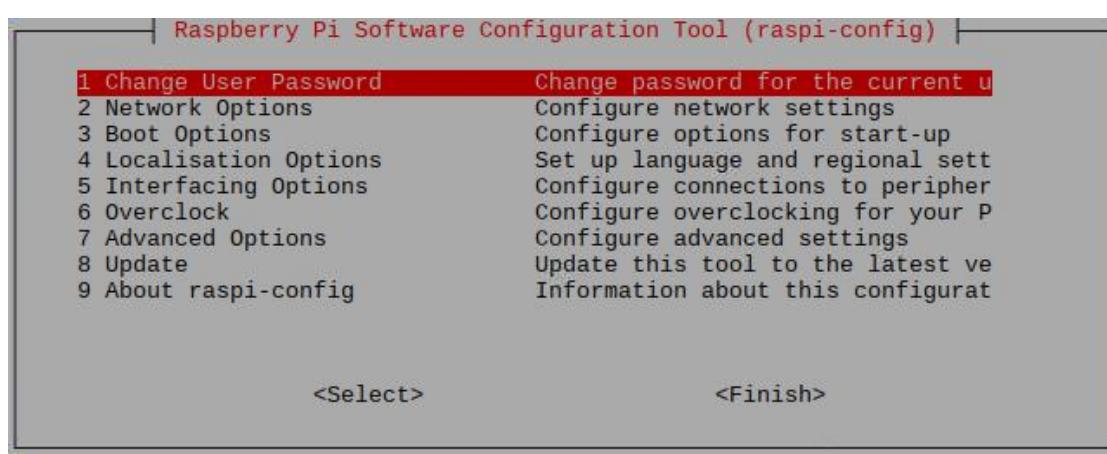


Configure SPI

Enable SPI

The SPI interface raspberry pie is closed in default. You need to open it manually. You can enable the SPIinterface in the following way. Type command in the terminal:[sudo raspi-config](#)

Then open the following dialog box:



Choose “5 Interfacing Options” “P4 SPI” “Yes” “Finish” in order and restart your RPi later. Then the SPImodule is started.

Install SPI-Py, PY-spidev module

Enter sudo apt-get install python-spidev python3-spidev in the terminal to install the "Py-spidev" module;

Enter the following command to install the SPI-Py module:

```
git clone https://github.com/WayinTop/SPI-Py
```

```
cd SPI-Py
```

```
sudo python setup.py install
```

C Code

The project code use human-computer interaction command line mode to read and write the M1-S50 card.

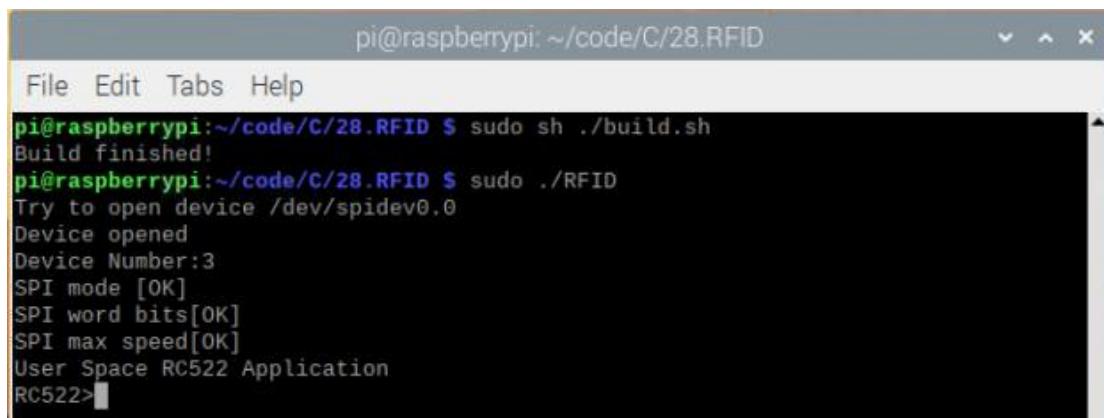
First observe the running result, and then analyze the code.

Enter the command [cd code / C / 28.RFID](#) / to switch to the RFID code directory as follows:



```
pi@raspberrypi:~/code/C/28.RFID
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/28.RFID/
pi@raspberrypi:~/code/C/28.RFID $ ls
build.sh  include  main.c  README.md  src
pi@raspberrypi:~/code/C/28.RFID $
```

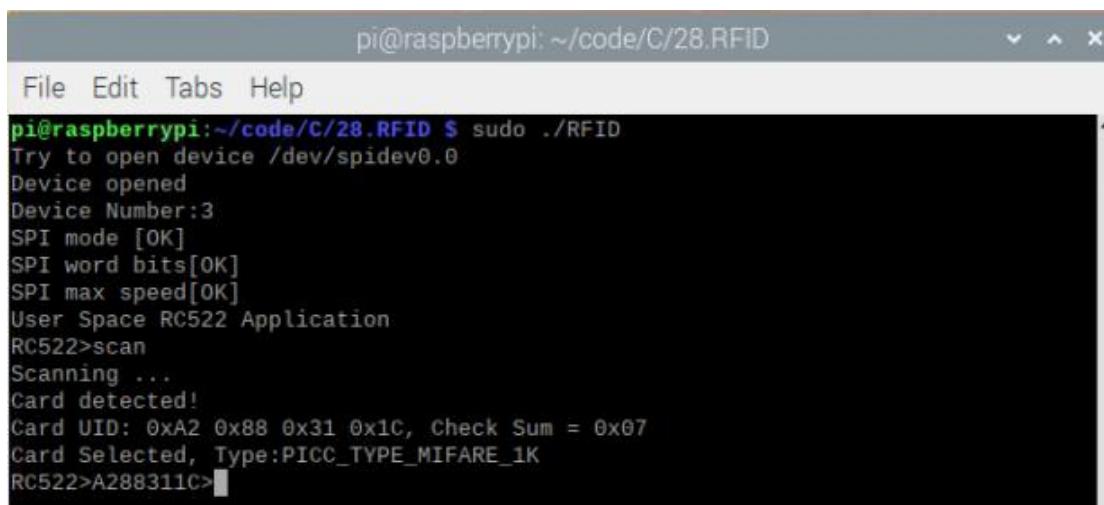
Use the command [sudo sh ./build.sh](#) to generate the RFID executable file, and then run the file using [sudo ./RFID](#). The result is as follows:



```
pi@raspberrypi:~/code/C/28.RFID
File Edit Tabs Help
pi@raspberrypi:~/code/C/28.RFID $ sudo sh ./build.sh
Build finished!
pi@raspberrypi:~/code/C/28.RFID $ sudo ./RFID
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>
```

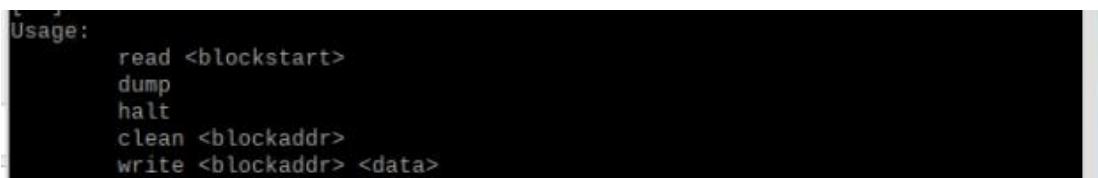
Here, type the command “**quit**” to exit the program.

Type command "**scan**", then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place an M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is A288311C (HEX) .



```
pi@raspberrypi:~/code/C/28.RFID$ sudo ./RFID
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>scan
Scanning ...
Card detected!
Card UID: 0xA2 0x88 0x31 0x1C, Check Sum = 0x07
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>A288311C>
```

When the Card is placed in the sensing area, you can read and write the card through the following command.



```
Usage:
    read <blockstart>
    dump
    halt
    clean <blockaddr>
    write <blockaddr> <data>
```

In the command **read<blockstart>**, the parameter **blockstart** is the address of the data block, and the range is 0-63. This command is used to display all the data from **blockstart** address to the end of the sector. For example, sector 0 contains data block 0,1,2,3. Using the command **read 0** can display all contents of data block 0,1,2,3. Using the command **read 1** can display all contents of data block 1,2,3.

As is shown below:

```
RC522>A288311C>read 1
read
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  32: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
RC522>A288311C>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: a2 88 31 1c 07 08 04 00 62 63 64 65 66 67 68 69 : ..1.....bcdefghi
  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
RC522>A288311C>
```

Command dump is used to display the content of all data blocks in all sectors.

Command <address><data> is used to write "data" to data block with address "address". Where the address range is 0-63 and the data length is 0-16. For example, if you want to write the string to the data block "hello" with address "1", you can type the following command: "write 1 hello".

```
RC522>A288311C>write 1 hello
write
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to write block 1 with 5 byte data...OK
RC522>A288311C>
```

Enter the "read 0" command to read the contents of the sector and check the data just written.

The following results show that the string "hello" has been successfully written into the data block 1.

```
RC522>A288311C>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
  0: a2 88 31 1c 07 08 04 00 62 63 64 65 66 67 68 69 : ..1.....bcdefghi
  16: 68 65 6c 6c 6f 00 00 00 00 00 00 00 00 00 00 00 : hello.....
  32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : .....
  48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .....i.....
RC522>A288311C>
```

Command `clean <address>` is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command: "clean 1"

```
RC522>A288311C>clean 1
clean
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to clean block 1...OK
RC522>A288311C>
```

Read the contents of data blocks in this sector again to test whether the data is erased. The following results indicate that the contents of data block 1 have been erased.

```
RC522>A288311C>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
    0: a2 88 31 1c 07 08 04 00 62 63 64 65 66 67 68 69 : ..1.....bcdefghi
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff :
RC522>A288311C>
```

Command `halt` is used to quit the selection state of the card.

```
RC522>A288311C>halt
halt
Halt...
Try to open device /dev/spidev0.0
Device opened
Device Number:6
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
RC522>
```

The following is the program code:

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include <stdlib.h>
#include "mfrc522.h"
#define DISP_COMMANDLINE() printf("RC522>")
```

```
int scan_loop(uint8_t *CardID);
int tag_select(uint8_t *CardID);

int main(int argc, char **argv) {
    MFRC522_Status_t ret;
    //Recognized card ID
    uint8_t CardID[5] = { 0x00, };
    uint8_t tagType[16] = {0x00,};
    static char command_buffer[1024];

    ret = MFRC522_Init('B');
    if (ret < 0) {
        printf("Failed to initialize.\r\nProgram exit.\r\n");
        exit(-1);
    }

    printf("User Space RC522 Application\r\n");

    while (1) {
        /*Main Loop Start*/
        DISP_COMMANDLINE();

        scanf("%os", command_buffer);
        if (strcmp(command_buffer, "scan") == 0) {
            puts("Scanning ... ");
            while (1) {
                ret = MFRC522_Request(PICC_REQIDL, tagType);
                if (ret == MI_OK) {
                    printf("Card detected!\r\n");
                    ret = MFRC522_Anticoll(CardID);
                    if(ret == MI_OK){
                        ret = tag_select(CardID);
                        if (ret == MI_OK) {
                            ret = scan_loop(CardID);
                            if (ret < 0) {
                                printf("Card error... \r\n");
                                break;
                            } else if (ret == 1) {
                                puts("Halt... \r\n");
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
    }
}
else{
    printf("Get Card ID failed!\r\n");
}
}
MFRC522_Halt();
}
MFRC522_Halt();
MFRC522_Init('B');
} else if(strcmp(command_buffer, "quit") == 0
    || strcmp(command_buffer, "exit") == 0) {
    return 0;
} else {
    puts("Unknown command");
    puts("scan:scan card and dump");
    puts("quit:exit program");
}
/*Main Loop End*/
}
}
int scan_loop(uint8_t *CardID) {

while (1) {

    char input[32];
    int block_start;
    DISP_COMMANDLINE();
    printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2],
CardID[3]);
    scanf("%s", input);
    puts((char*)input);
    if (strcmp(input, "halt") == 0) {
        MFRC522_Halt();
        return 1;
    } else if (strcmp(input, "dump") == 0) {
        if (MFRC522_Debug_CardDump(CardID) < 0)
            return -1;
    } else if (strcmp(input, "read") == 0) {
```

```
scanf("%d", &block_start);
if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
    return -1;
}
} else if(strcmp(input, "clean") == 0){
    char c;
    scanf("%d", &block_start);
    while ((c = getchar()) != '\n' && c != EOF)
        ;
    if (MFRC522_Debug_Clean(CardID, block_start)) {
        return -1;
    }

} else if (strcmp(input, "write") == 0) {
    char write_buffer[256];
    size_t len = 0;
    scanf("%d", &block_start);
    scanf("%s", write_buffer);
    if (len >= 0) {
        if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
            strlen(write_buffer)) < 0) {
            return -1;
        }
    }
} else {
    printf(
        "Usage:\r\n""\tread <blockstart>\r\n""\tdump\r\n""\thalt\r\n"
        "\tclean <blockaddr>\r\n""\twrite <blockaddr><data>\r\n");
    //return 0;
}
return 0;
}

int tag_select(uint8_t *CardID) {
    int ret_int;
    printf(
        "Card UID: 0x%02X 0x%02X 0x%02X 0x%02X, Check Sum =
0x%02X\r\n",

```

```

CardID[0], CardID[1], CardID[2], CardID[3], CardID[4]);
ret_int = MFRC522_SelectTag(CardID);
if (ret_int == 0) {
    printf("Card Select Failed\r\n");
    return -1;
} else {
    printf("Card Selected, Type:%s\r\n",
        MFRC522_TypeToString(MFRC522_ParseType(ret_int)));
}
ret_int = 0;
return ret_int;
}

```

Code Interpretation

```

ret = MFRC522_Init('B');
if (ret < 0) {
    printf("Failed to initialize.\r\nProgram exit.\r\n");
    exit(-1);
}

```

In the code, first initialize the MFRC522. If the initialization fails, the program will exit.

```

if (strcmp(command_buffer, "scan") == 0) {
    puts("Scanning ... ");
    while (1) {
        ret = MFRC522_Request(PICC_REQIDL, tagType);
        if (ret == MI_OK) {
            printf("Card detected!\r\n");
            ret = MFRC522_Anticoll(CardID);
            if (ret == MI_OK) {
                ret = tag_select(CardID);
                if (ret == MI_OK) {
                    ret = scan_loop(CardID);
                    if (ret < 0) {
                        printf("Card error...\r\n");
                        break;
                    } else if (ret == 1) {
                        puts("Halt...\r\n");
                    }
                }
            }
        }
    }
}

```

```
                break;
            }
        }
    }
} else{
    printf("Get Card ID failed!\r\n");
}
}
MFRC522_Halt();
}
MFRC522_Halt();
MFRC522_Init('B');
} else if (strcmp(command_buffer, "quit") == 0
           || strcmp(command_buffer, "exit") == 0) {
    return 0;
} else {
    puts("Unknown command");
    puts("scan:scan card and dump");
    puts("quit:exit program");
}
/*Main Loop End*/
}
```

In the main function, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function “scan_loop ()”. If command "quit" or "exit" is received, the program will exit.

```
int scan_loop(uint8_t *CardID) {

    while (1) {

        char input[32];
        int block_start;
        DISP_COMMANDLINE();
        printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2],
CardID[3]);
        scanf("%s", input);
        puts((char*)input);
```

```
if (strcmp(input, "halt") == 0) {
    MFRC522_Halt();
    return 1;
} else if (strcmp(input, "dump") == 0) {
    if (MFRC522_Debug_CardDump(CardID) < 0)
        return -1;
} else if (strcmp(input, "read") == 0) {
    scanf("%d", &block_start);
    if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
        return -1;
    }
} else if(strcmp(input, "clean") == 0){
    char c;
    scanf("%d", &block_start);
    while ((c = getchar()) != '\n' && c != EOF)
        ;
    if (MFRC522_Debug_Clean(CardID, block_start)) {
        return -1;
    }
}

} else if (strcmp(input, "write") == 0) {
    char write_buffer[256];
    size_t len = 0;
    scanf("%d", &block_start);
    scanf("%s", write_buffer);
    if (len >= 0) {
        if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
            strlen(write_buffer)) < 0) {
            return -1;
        }
    }
} else {

    printf(
        "Usage:\r\n""\tread <blockstart>\r\n""\tdump\r\n""\thalt\r\n"
        "\tclean <blockaddr>\r\n""\twrite <blockaddr><data>\r\n");
    //return 0;
}
return 0;
```

{}

The function “scan_loop()” will detect command read, “write”, “clean”, ‘halt’, “dump” and do the corresponding processing to each command. The function of each command and the method have been introduced before.

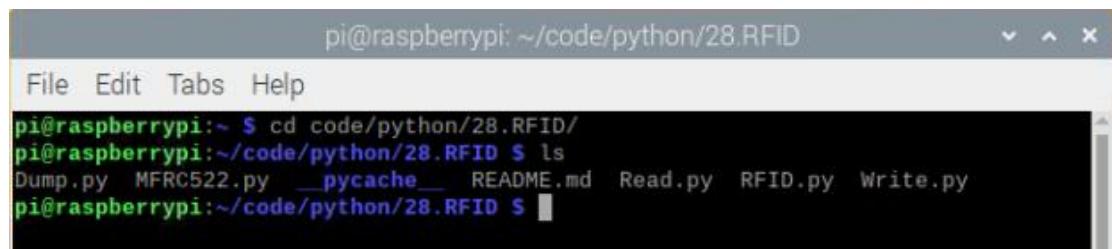
The header file "mfrc522.h" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Python code

The project code use human-computer interaction command line mode to read and write the M1-S50 card.

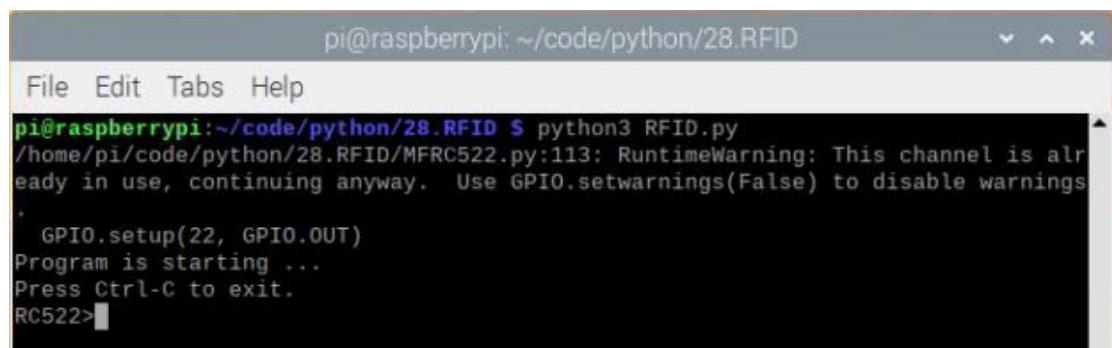
First observe the running result, and then analyze the code.

Enter the [cd code / python / 28.RFID /](#) command to switch to the RFID.py code directory, as shown in the following figure:



```
pi@raspberrypi: ~ /code/python/28.RFID
File Edit Tabs Help
pi@raspberrypi: ~ $ cd code/python/28.RFID/
pi@raspberrypi: ~/code/python/28.RFID $ ls
Dump.py MFRC522.py __pycache__ README.md Read.py RFID.py Write.py
pi@raspberrypi: ~/code/python/28.RFID $
```

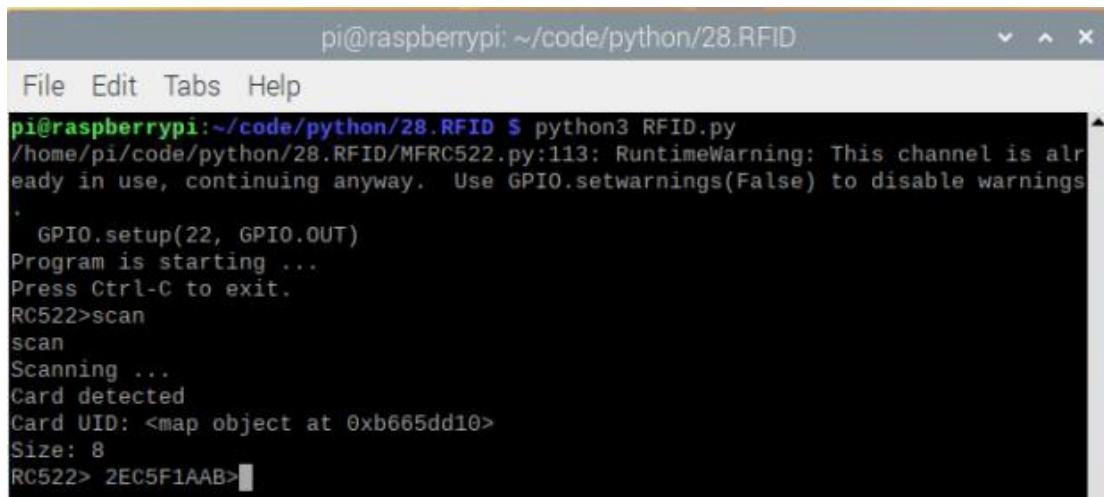
Enter the [python3 RFID.py](#) command and run the RFID.py code. The result is shown below:



```
pi@raspberrypi: ~ /code/python/28.RFID
File Edit Tabs Help
pi@raspberrypi: ~/code/python/28.RFID $ python3 RFID.py
/home/pi/code/python/28.RFID/MFRC522.py:113: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings
  GPIO.setup(22, GPIO.OUT)
Program is starting ...
Press Ctrl-C to exit.
RC522>
```

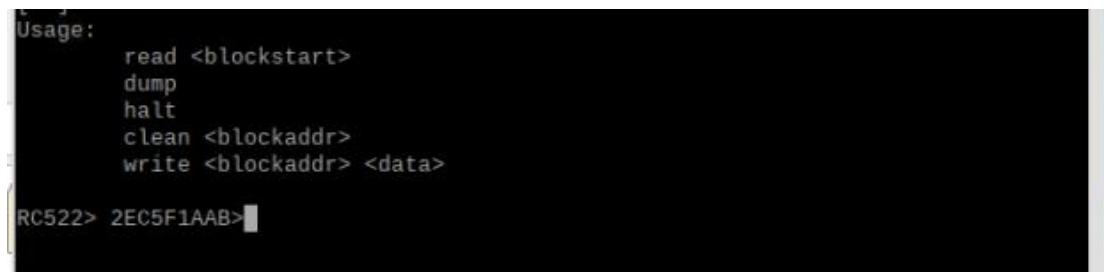
Type command ["scan"](#), then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place an M1-S50 card in the sensing area.

The following results indicate that the M1-S50 card has been detected (If there is no response when you put the card in the induction area, please disconnect and reconnect the power of MFRC522 before running the code).



```
pi@raspberrypi:~/code/python/28.RFID$ python3 RFID.py
/home/pi/code/python/28.RFID/MFRC522.py:113: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings
  GPIO.setup(22, GPIO.OUT)
Program is starting ...
Press Ctrl-C to exit.
RC522>scan
scan
Scanning ...
Card detected
Card UID: <map object at 0xb665dd10>
Size: 8
RC522> 2EC5F1AAB>
```

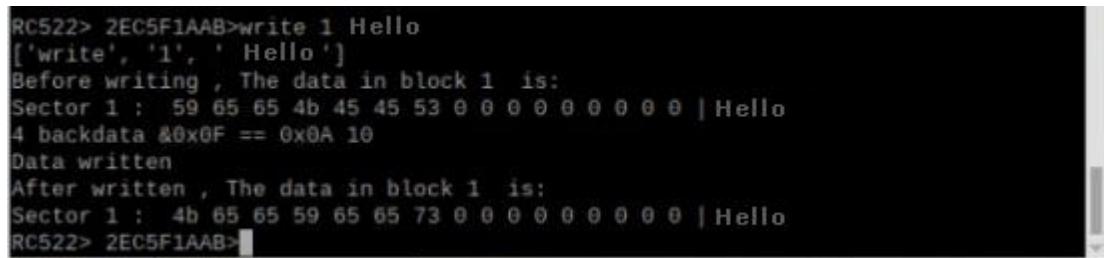
When the Card is placed in the sensing area, you can read and write the card through the following command.



```
Usage:
      read <blockstart>
      dump
      halt
      clean <blockaddr>
      write <blockaddr> <data>

RC522> 2EC5F1AAB>
```

Command <address><data> is used to write "data" to data block with address "address". Where the address range is 0-63 and the data length is 0-16. In the process of writing data to the data block, both the contents of data block before written and after written will be displayed. For example, if you want to write the string "Hello" to the data block with address "1", you can type the following command **write 1 Hello**.



```
RC522> 2EC5F1AAB>write 1 Hello
['write', '1', 'Hello']
Before writing , The data in block 1  is:
Sector 1 :  59 65 65 4b 45 45 53 0 0 0 0 0 0 0 0 | Hello
4 backdata &0x0F == 0xA 10
Data written
After written , The data in block 1  is:
Sector 1 :  4b 65 65 59 65 65 73 0 0 0 0 0 0 0 0 | Hello
RC522> 2EC5F1AAB>
```

In the command `read<blockstart>`, the parameter `blockstart` is the address of the data block, and the range is 0-63. This command is used to read the data of data block with address “`blockstart`”. For example, using command `read 0` can display the content of data block 0. Using the command `read 1` can display the content of data block 1. As is shown below:

```
RC522> 2EC5F1AAB>read 1
['read', '1']
Sector 1 : 4b 65 65 59 65 65 73 0 0 0 0 0 0 0 0 | Hello
RC522> 2EC5F1AAB>read 0
['read', '0']
Sector 0 : 2 ec 5f 1a ab 8 4 0 62 63 64 65 66 67 68 69 | i_#bcdefghi
RC522> 2EC5F1AAB>■
```

Command `clean <address>` is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command: “`clean 1`”

```
RC522> 2EC5F1AAB>clean 1
['clean', '1']
Before cleaning , The data in block 1 is:
Sector 1 : 4b 65 65 59 65 65 73 0 0 0 0 0 0 0 0 | Hello
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1 is:
Sector 1 : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | ■
RC522> 2EC5F1AAB>■
```

Command `halt` is used to quit the selection state of the card.

```
RC522> 2EC5F1AAB>halt
['halt']
RC522>■
```

The following is the program code:

```
import RPi.GPIO as GPIO
import MFRC522
import sys
import os

# Create an object of the class MFRC522
mfrc = MFRC522.MFRC522()
```

```
def dis_CommandLine():
    print ("RC522>",end="")
def dis_CardID(cardID):
    print
    ("%2X%2X%2X%2X>%%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]),
    end=""")
def setup():
    print ("Program is starting ... ")
    print ("Press Ctrl-C to exit.")
    pass

def loop():
    global mfrc
    while(True):
        dis_CommandLine()
        inCmd = input()
        print (inCmd)
        if (inCmd == "scan"):
            print ("Scanning ... ")
            mfrc = MFRC522.MFRC522()
            isScan = True
            while isScan:
                # Scan for cards
                (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
                # If a card is found
                if status == mfrc.MI_OK:
                    print ("Card detected")
                    # Get the UID of the card
                    (status,uid) = mfrc.MFRC522_Anticoll()
                    # If we have the UID, continue
                    if status == mfrc.MI_OK:
                        print ("Card UID: "+ str(map(hex,uid)))
                        # Select the scanned tag
                        if mfrc.MFRC522_SelectTag(uid) == 0:
                            print ("MFRC522_SelectTag Failed!")
                        if cmdloop(uid) < 1 :
                            isScan = False

        elif inCmd == "quit":
            destroy()
```

```
        exit(0)
    else :
        print ("\tUnknown command\n"+ "\tscan:scan card and
dump\n"+ "\tquit:exit program\n")

def cmdloop(cardID):
    pass
    while(True):
        dis_CommandLine()
        dis_CardID(cardID)
        inCmd = input()
        cmd = inCmd.split("")
        print (cmd)
        if(cmd[0] == "read"):
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print ("Invalid Address!")
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr,
key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print ("Authentication error")
                return 0

        elif cmd[0] == "dump":
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            mfrc.MFRC522_Dump_Str(key,cardID)

        elif cmd[0] == "write":
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print ("Invalid Address!")
            data = [0]*16
            if(len(cmd)<2):
```

```
        data = [0]*16
    else:
        data = cmd[2][0:17]
        data = map(ord,data)
        data = list(data)
        lenData = len(list(data))
        if lenData<16:
            data+=[0]*(16-lenData)
    # This is the default key for authentication
    key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
    # Authenticate
    status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr,
key, cardID)
    # Check if authenticated
    if status == mfrc.MI_OK:
        print ("Before writing , The data in block %d  is: "%(blockAddr))
        mfrc.MFRC522_Readstr(blockAddr)
        mfrc.MFRC522_Write(blockAddr, data)
        print ("After written , The data in block %d  is: "%(blockAddr))
        mfrc.MFRC522_Readstr(blockAddr)
    else:
        print ("Authentication error")
        return 0

elif cmd[0] == "clean":
    blockAddr = int(cmd[1])
    if((blockAddr<0) or (blockAddr>63)):
        print ("Invalid Address!")
    data = [0]*16
    # This is the default key for authentication
    key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
    # Authenticate
    status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr,
key, cardID)
    # Check if authenticated
    if status == mfrc.MI_OK:
        print ("Before cleaning , The data in block %d  is: "%(blockAddr))
        mfrc.MFRC522_Readstr(blockAddr)
        mfrc.MFRC522_Write(blockAddr, data)
        print ("After cleaned , The data in block %d  is: "%(blockAddr))
```

```

        mfrc.MFRC522_Readstr(blockAddr)
    else:
        print ("Authentication error")
        return 0
    elif cmd[0] == "halt":
        return 0
    else :
        print ("Usage:\r\n""\tread
<blockstart>\r\n""\tdump\r\n""\thalt\r\n""\tclean <blockaddr>\r\n""\twrite
<blockaddr><data>\r\n")
def destroy():
    GPIO.cleanup()
if __name__ == "__main__":
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Ctrl+C captured, exit
        destroy()

```

Code Interpretation

Zu Beginn des Programms müssen wir ein MFRC522 Objekt mfrc erstellen.

mfrc = MFRC522.MFRC522()

In the function loop, waiting for command input. If the command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If the card is detected, the card will be selected and card UID will be acquired and then enter the function camloop(). If the command "quit" is received, the program will exit.

```

if (inCmd == "scan"):
    print ("Scanning ... ")
    mfrc = MFRC522.MFRC522()
    isScan = True
    while isScan:
        # Scan for cards
        (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
        # If a card is found
        if status == mfrc.MI_OK:
            print ("Card detected")
        # Get the UID of the card

```

```
(status,uid) = mfrc.MFRC522_Anticoll()
# If we have the UID, continue
if status == mfrc.MI_OK:
    print ("Card UID: "+ str(map(hex,uid)))
    # Select the scanned tag
    if mfrc.MFRC522_SelectTag(uid) == 0:
        print ("MFRC522_SelectTag Failed!")
    if cmdloop(uid) < 1 :
        isScan = False

elif inCmd == "quit":
    destroy()
    exit(0)
else :
    print ("\tUnknown command\n"+ "\tscan:scan card and
dump\n"+ "\tquit:exit program\n")
```

The function cmdloop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command.

```
def cmdloop( cardID ):
pass
while( ( True ):
dis_CommandLine ()
dis_CardID( cardID )
inCmd == raw_input ()
cmd == inCmd. .split( "" ) )
print cmd
if( (cmd[ [0] ] == "read"):
.....
elif cmd[ [0] ] == "dump": :
.....
elif cmd[ [0] ] == "write": :
.....
elif cmd[ [0] ] == "clean": :
.....
elif cmd[ [0] ] == "halt": :
return 0
else :::
print "Usage:\r\n""\tread <blockstart>\r\n""\tdump\r\n""\thalt\r\n"
```

```
"\tclear <blockaddr>\r\n""\twrite <blockaddr><data>\r\n"
```

The file "MFRC522.py" contains the related operation method for the MFRC522.
You can open the file to view all the definitions and functions.

Lesson 29 4N35

Overview

In this lesson you will learn how to use 4N35. 4N35 is a photocoupler for general applications. It consists of a GaAs infrared LED and a silicon NPN phototransistor. When an input signal is applied to the LED in the input terminal, the LED lights up. After receiving the optical signal, the optical receiver then converts it into an electrical signal and outputs the signal directly or after amplifying the signal to a standard digital level. The transition and transmission of electricity and electricity are completed. Since light is a transmission medium, it means that the input and output terminals are electrically isolated, so this process is also called electrical isolation.

Parts Required

1 x Raspberry pi

1 x 4N35

1 x LED

1 x 220 ohm Resistor

1 x 1k ohm Resistor

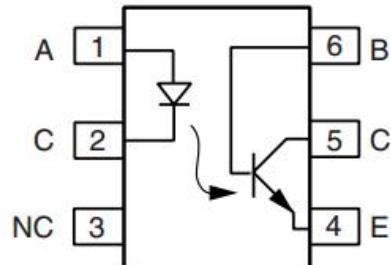
Some Jumper Wires

Product Introduction

4N35

The role of the photocoupler is to break the connection between the signal source and the signal receiver in order to stop electrical interference. In other words, it is used to prevent interference from external electrical signals. 4N35 can be used for AV conversion audio circuit.

It is widely used for the electrical insulation of ordinary optocouplers.

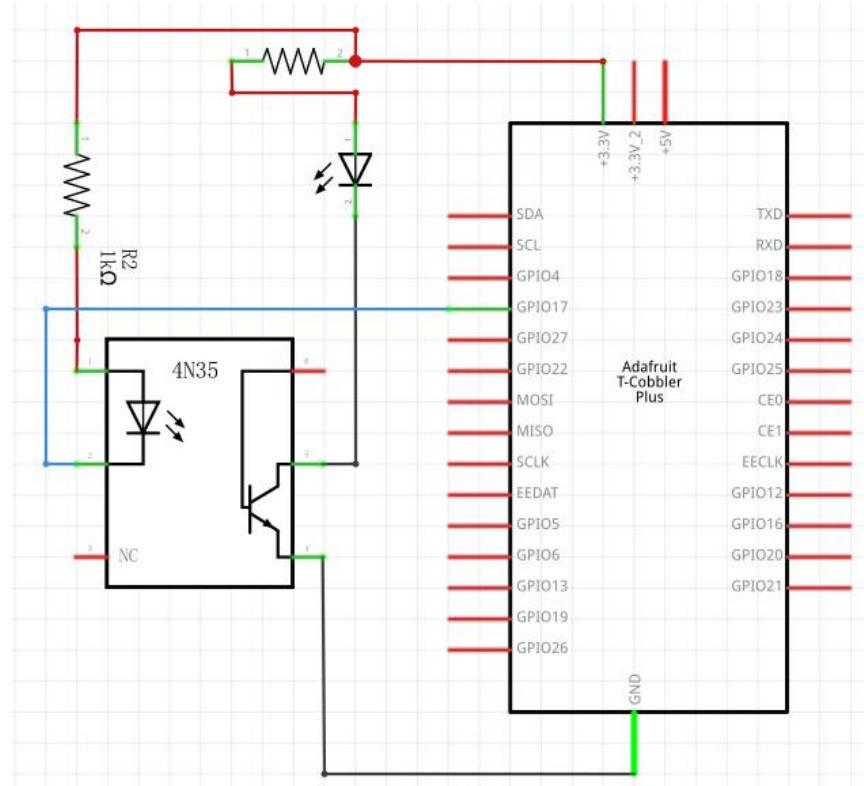


The above is the internal structure of 4N35. Pins 1 and 2 are connected to the infrared LED. When the LED is powered, it emits infrared light. To prevent LED burnout, a resistor (about 1K) is usually connected to pin 1. The NPN phototransistor then turns on the power when receiving light. This can be used to control the load connected to the phototransistor. Even if a load short circuit occurs, it will not affect the control board, thus achieving good electrical isolation.



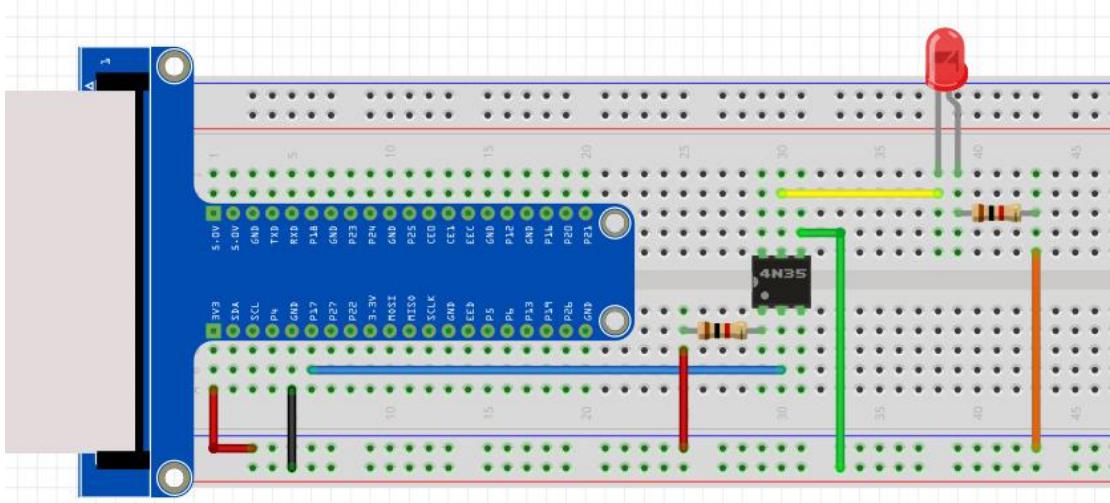
In this lesson, an LED is used as the load connected to the NPN phototransistor. Connect pin 2 of 4N35 to pin P17, connect pin 1 to a 1K current-limiting resistor, and then connect to 3.3V. Connect pin 4 to GND and pin 5 to the cathode of the LED. Then connect one end of the 220 ohm resistor to the anode of the LED and the other end to 3.3V. In the program, the P17 pin is low voltage, and the infrared LED emits infrared rays. The phototransistor then receives infrared light and is energized, and the LED cathode is at a low voltage to turn on the LED. You can also control the LED via the circuit only-connect pin 2 to ground and it will light up.

Connection Diagram

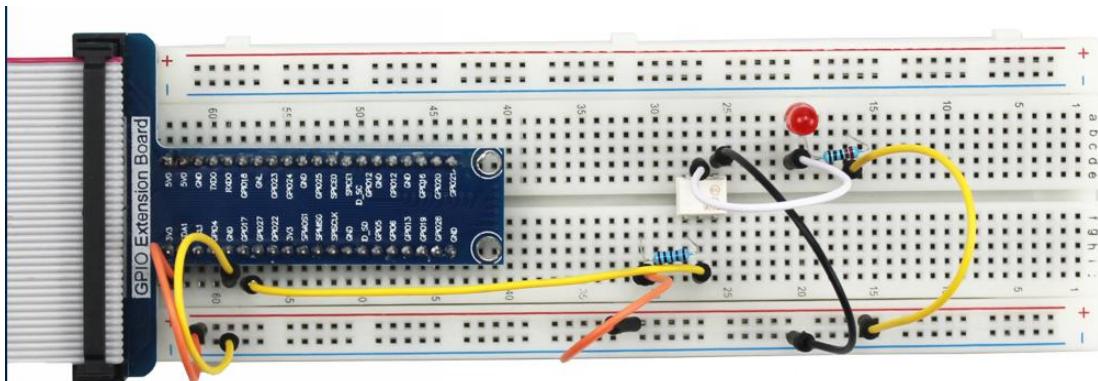




Wiring Diagram



Example Figure



C code

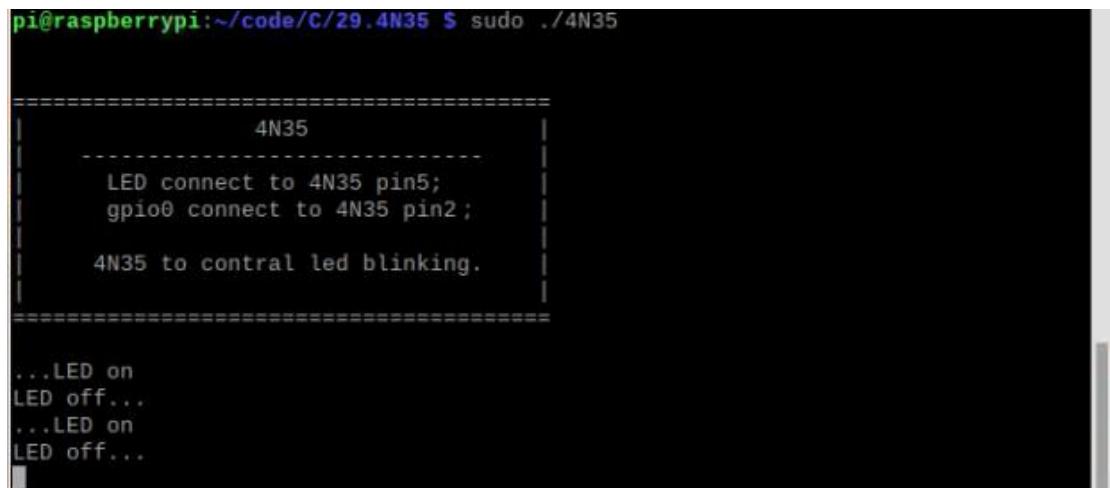
Open the Raspberry Pi terminal and enter the `cd code / C / 29.4N35 /` command to switch to the 4N35 directory;

Enter `gcc 4N35.c -o 4N35 -lwiringPi` command to generate 4N35 executable file; as shown below;



```
pi@raspberrypi:~/code/C/29.4N35$ gcc 4N35.c -o 4N35 -lwiringPi
pi@raspberrypi:~/code/C/29.4N35$ ls
4N35  4N35.c
pi@raspberrypi:~/code/C/29.4N35$
```

Enter `sudo ./4N35` to execute the code, and the execution result is shown in the following figure:



```
pi@raspberrypi:~/code/C/29.4N35$ sudo ./4N35

=====
|          4N35
|          -----
|          LED connect to 4N35 pin5;
|          gpio0 connect to 4N35 pin2;
|
|          4N35 to control led blinking.
|          -----
...
...LED on
LED off...
...LED on
LED off...
|
```

The following is the program code:

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

#define _4N35Pin      0

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

```
pinMode(_4N35Pin, OUTPUT);

printf("\n");
printf("\n");
printf("=====\\n");
printf("|          4N35          |\\n");
printf("|-----|\\n");
printf("|      LED connect to 4N35 pin5;      |\\n");
printf("|      gpio0 connect to 4N35 pin2;      |\\n");
printf("|          |\\n");
printf("|      4N35 to contral led blinking.    |\\n");
printf("|          |\\n");
printf("=====");
printf("\n");
printf("\n");

while(1){
    // LED on
    digitalWrite(_4N35Pin, LOW);
    printf("...LED on\\n");
    delay(1000);
    // LED off
    digitalWrite(_4N35Pin, HIGH);
    printf("LED off...\\n");
    delay(1000);
}

return 0;
}
```

Code Interpretation

```
while(1){
    // LED on
    digitalWrite(_4N35Pin, LOW);
    printf("...LED on\\n");
    delay(1000);
    // LED off
    digitalWrite(_4N35Pin, HIGH);
    printf("LED off...\\n");
```

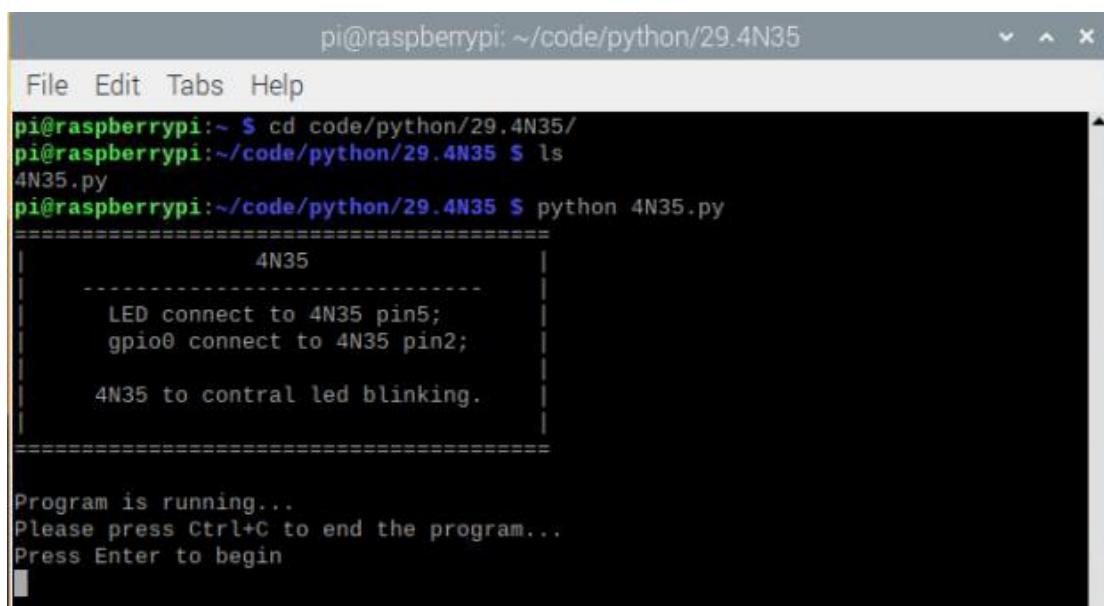
```
    delay(1000);  
}
```

The program is very simple. We have explained all the sentences in the previous lessons. The main thing is to use the 'digitalWrite' statement to output high and low levels to realize the function of LED on and off in the 'while' loop.

Python code

Open the Raspberry Pi terminal and enter the [cd code / python / 29.4N35 /](#) command to switch to the 4N35 directory;

Enter the [python 4N35.py](#) command to run the code; as shown below:



```
pi@raspberrypi:~/code/python/29.4N35  
File Edit Tabs Help  
pi@raspberrypi:~ $ cd code/python/29.4N35/  
pi@raspberrypi:~/code/python/29.4N35 $ ls  
4N35.py  
pi@raspberrypi:~/code/python/29.4N35 $ python 4N35.py  
=====  
| 4N35 |  
|-----|  
| LED connect to 4N35 pin5; |  
| gpio0 connect to 4N35 pin2; |  
| 4N35 to control led blinking. |  
=====  
  
Program is running...  
Please press Ctrl+C to end the program...  
Press Enter to begin
```

Press the 'Enter' key to see the LED blinking (press 'Ctrl + c' to end the run); as shown in the figure below:



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

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

The following is the program code:

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

# Set #17 as 4N35 pin
Pin_4N35 = 17
# Define a function to print message at the beginning
def print_message():
    print ("====")
    print ("|          4N35          |")
    print ("| ----- |")
    print ("|      LED connect to 4N35 pin5;      |")
    print ("|      gpio0 connect to 4N35 pin2;      |")
    print ("|          |")
    print ("|      4N35 to contral led blinking.    |")
    print ("|          |")
    print ("=====\n")
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'
    raw_input ("Press Enter to begin\n")
```

```
# Define a setup function for some setup
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set Pin_4N35's mode to output,
    # and initial level to High(3.3v)
    GPIO.setup(Pin_4N35, GPIO.OUT, initial=GPIO.HIGH)

# Define a loop function for loop process
def loop():
    # Print messages
    print_message()
    while True:
        print '...LED ON'
        # Turn on LED
        GPIO.output(Pin_4N35, GPIO.LOW)
        time.sleep(0.5)
        print 'LED OFF...'
        # Turn off LED
        GPIO.output(Pin_4N35, GPIO.HIGH)
        time.sleep(0.5)

# Define a destroy function for clean up everything after
# the script finished
def destroy():
    # Turn off LED
    GPIO.output(Pin_4N35, GPIO.HIGH)
    # Release resource
    GPIO.cleanup()

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

Code Interpretation

No new sentence is used in this course, and there are detailed statement comments in the program, so this course does not explain the program. If you have any questions, please refer to previous tutorials and translation program comments

4N35 is also used to drive relays and motor circuits. Because there is no direct connection between the input and output, the control board will not be burned even if a short circuit occurs at the output. If you are interested in it, you can experiment.

Lesson 30 NE555

Overview

In this lesson you will learn how to use the NE555 timer.

Parts Required

1 x Raspberry Pi

1 x NE555 Chip

1 x 220 ohm Resistor

1 x 1K Resistor

2 x 10K Resistor

2 x Capacitors (100nF)

20 x Double Male Jumper Wires

1 x Breadboard

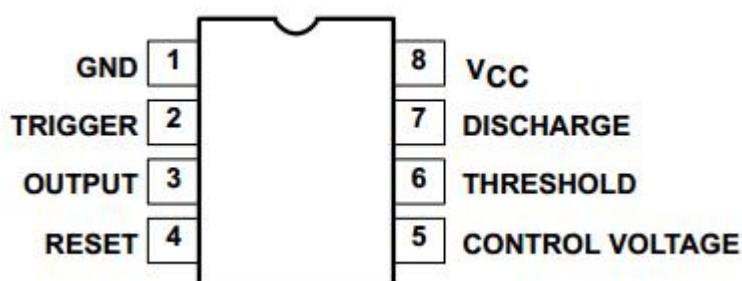
1 x LED

Product Introduction

The NE555 timer is a hybrid circuit composed of analog and digital circuits. It integrates analog and logic functions into independent IC, thereby greatly expanding the application of analog integrated circuits. It is widely used in various timers, pulse generators and oscillators.

The 555 timer is a medium-sized IC device that combines analog and digital functions. It has low cost and reliable performance. It only needs to connect external resistors and capacitors to implement multivibrator, monostable flip-flop, Schmitt trigger, and other circuits that can generate and convert pulses. It is also often used as a timer and is widely used in instruments, household appliances, electronic measurement, automatic control and other fields.

Pins and functions:



As shown, the pins are set up in a dual row with the 8-pin package.

Pin 1 (GND): ground

Pin 2 (TRIGGER): input of low-side comparator

Pin 3 (OUTPUT): has two states 0 and 1, determined by the input level

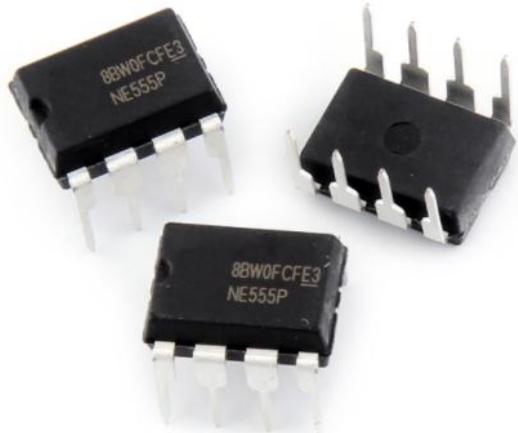
Pin 4 (RESET): output low level when low level is provided

Pin 5 (control voltage): change the trigger value

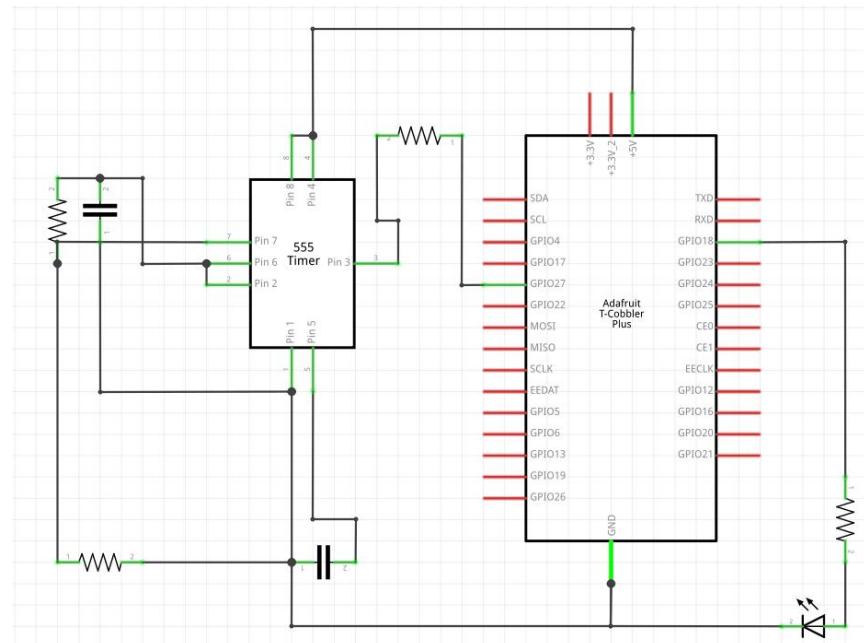
Pin 6 (THRESHOLD): Input to the upper comparator

Pin 7 (DISCHARGE): The two states of suspension and ground are also determined by the input and output of the internal discharge tube

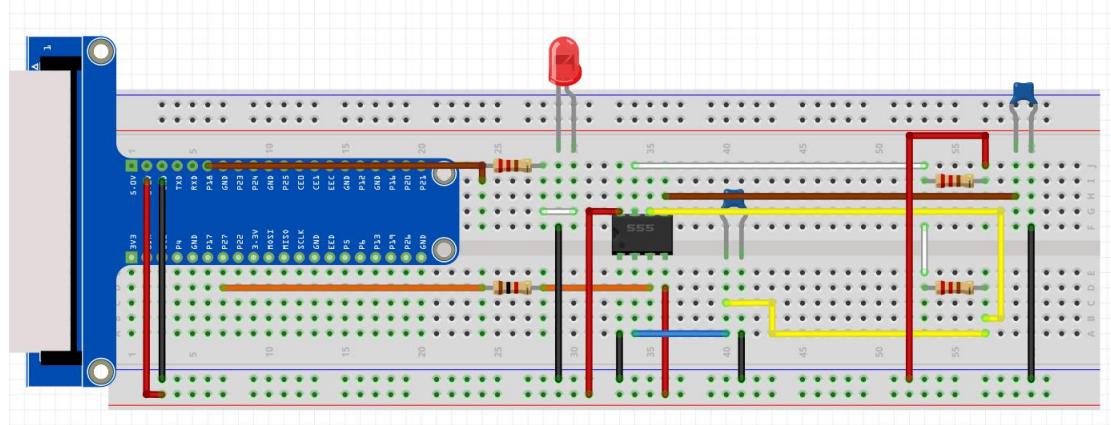
Pin 8 (VCC): Power



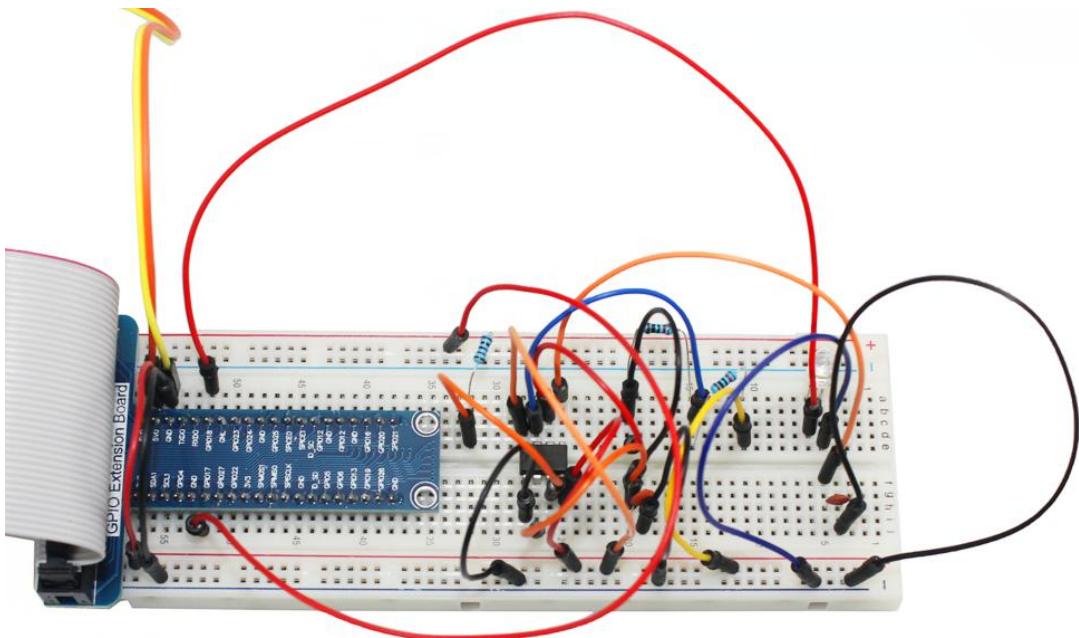
Connection Diagram



Wiring Diagram



Example Figure



C code

Open terminal and enter `cd code / C / 30.NE555 /` command to enter ne555.c code directory;

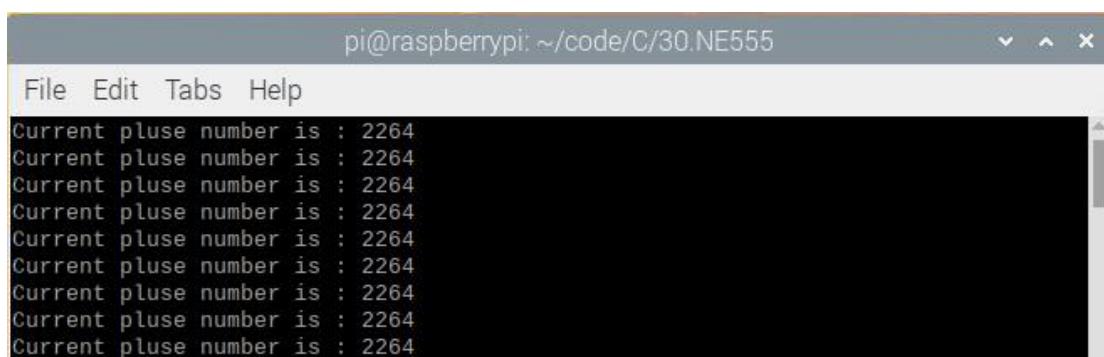
Enter the `ls` command to view the file ne555.c in the directory;



```
pi@raspberrypi: ~/code/C/30.NE555
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/30.NE555/
pi@raspberrypi:~/code/C/30.NE555 $ ls
ne555.c
pi@raspberrypi:~/code/C/30.NE555 $
```

Enter `gcc ne555.c -o ne555 -lwiringPi` command to generate ne555.c executable file ne555, enter `ls` command to view;

Enter the `sudo ./ne555` command to run the code. The result is as follows:



```
pi@raspberrypi: ~/code/C/30.NE555
File Edit Tabs Help
Current pluse number is : 2264
```

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

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <wiringPi.h>

#define Pin0 0
#define led 1

static volatile int globalCounter = 0 ;

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

```
int main (void)
{
    if(wiringPiSetup() < 0){
        fprintf(stderr, "Unable to setup wiringPi:%s\n",strerror(errno));
        return 1;
    }

    wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR);
    printf("wiringPi initialize successfully, GPIO %d(wiringPi pin)\n",led);
    pinMode(led, OUTPUT);
    while(1){
        if(globalCounter/100%2==0 )
        {
            digitalWrite(led, HIGH);
        }
        else
        {
            digitalWrite(led, LOW);

        }
        printf("Current pluse number is : %d\n", globalCounter);
    }

    return 0;
}
```

Code Interpretation

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

The interrupt function is executed once and the 'globalCounter' is incremented by one;

```
while(1){
    if(globalCounter/100%2==0 )
    {
```

```
        digitalWrite(led, HIGH);
    }
else
{
    digitalWrite(led, LOW);

}
printf("Current pluse number is : %d\n", globalCounter);
}
```

In the loop, determine whether '`g_count / 100% 2`' is equal to 0 to make the LED blink, and print the value of '`globalCounter`'.

Python code

Open the terminal and use the [cd code / python / 30.NE555 /](#) command to enter the code directory;

Enter the command `ls` to view the file `ne555.py` in the directory.



```
pi@raspberrypi: ~/code/python/30.NE555
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/python/30.NE555/
pi@raspberrypi:~/code/python/30.NE555 $ ls
ne555.py
pi@raspberrypi:~/code/python/30.NE555 $
```

Enter the command [python3 ne555.py](#) to run the code. The result is as follows:



```
pi@raspberrypi: ~/code/python/30.NE555
File Edit Tabs Help
g_count = 9106
```

The following is the program code:

```
import RPi.GPIO as GPIO
SigPin = 11
ledPin = 12
g_count = 0
def count(ev=None):
    global g_count
    g_count += 1

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
        GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count)
        GPIO.setup(ledPin, GPIO.OUT)
    GPIO.output(ledPin, GPIO.LOW)

def loop():
    while True:
        if(g_count/100%2==0):
            GPIO.output(ledPin, GPIO.HIGH)
        else:
            GPIO.output(ledPin, GPIO.LOW)
        print ('g_count = %d' % g_count)

def destroy():
    GPIO.cleanup()

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

Code Interpretation

```
def count(ev=None):
    global g_count
    g_count += 1
```

Interrupt function = execute once to increase 'g_count' by 1;

```
def loop():
    while True:
        if(g_count/100%2==0):
            GPIO.output(ledPin, GPIO.HIGH)
        else:
            GPIO.output(ledPin, GPIO.LOW)
        print ('g_count = %d' % g_count)
```

In the loop, it is judged whether ' $g_count / 100\% 2$ ' is equal to 0 to make the LED blink, and the value of ' g_count ' is printed.

Lesson 31 Infrared Remote Control

Overview

In this lesson you will learn how to use Raspberry Pi to drive an infrared remote control LED lights.

Parts Required

1 x Raspberry Pi

1 x IR Remote

8 x Double Male Jumper Wires

1 x Breadboard

3 x LED

3 x 220 Ohm Resistor

Product Introduction

Infrared Remote Control

The infrared remote control transmitting circuit uses infrared light emitting diodes to emit modulated infrared light waves. The infrared receiving circuit consists of infrared receiving diodes, triodes or silicon photocells. They convert the infrared light emitted by the infrared transmitter into the corresponding electrical signal, and then send it to the rear amplifier.

The transmitter is generally composed of a command key (or joystick), a command coding system, a modulation circuit, a driving circuit, and a transmitting circuit.

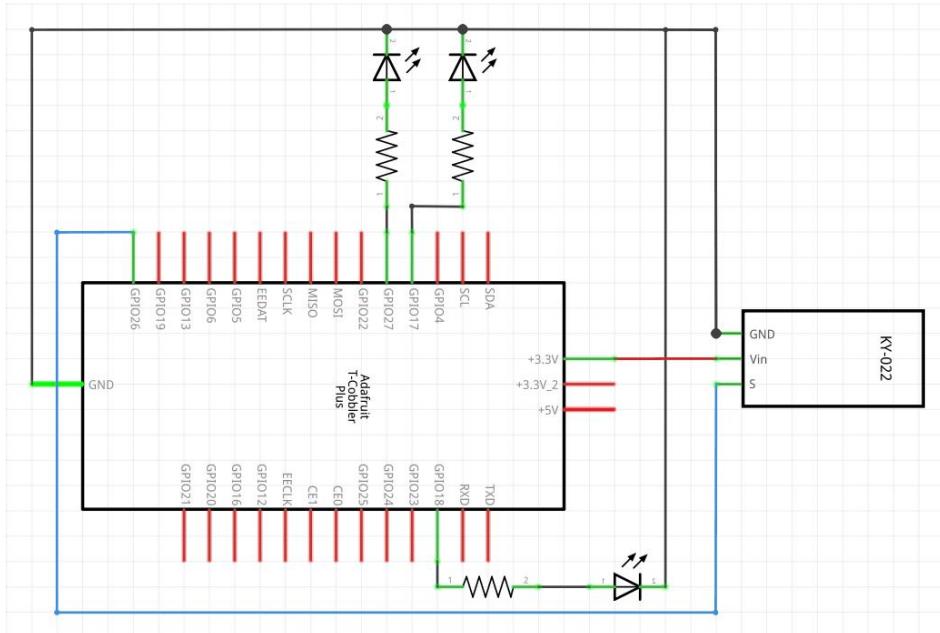
When the instruction key is pressed or the joystick is pushed, the instruction encoding circuit generates the required instruction encoding signal, the instruction encoding signal modulates the carrier wave, and then the driver circuit performs power amplification, and the transmitting circuit launch outward the command code signal.

The receiving circuit is generally composed of a receiving circuit, an amplification circuit, a modulation circuit, an instruction decoding circuit, a driving circuit, and an execution circuit (mechanism). The receiving circuit receives the modulated coded instruction signal sent by the transmitter, amplifies it, and sends it to the demodulation circuit. The demodulation circuit demodulates the modulated instruction coded signal, that is, the coded signal is restored. The instruction decoder decodes the encoded instruction signal, and finally the driving circuit drives the execution circuit to realize the operation control of various instructions.

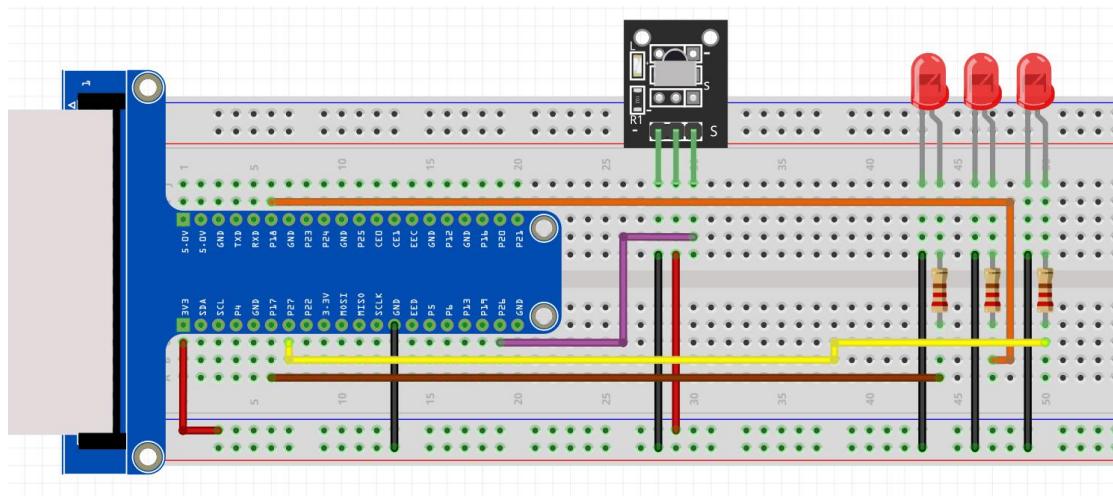




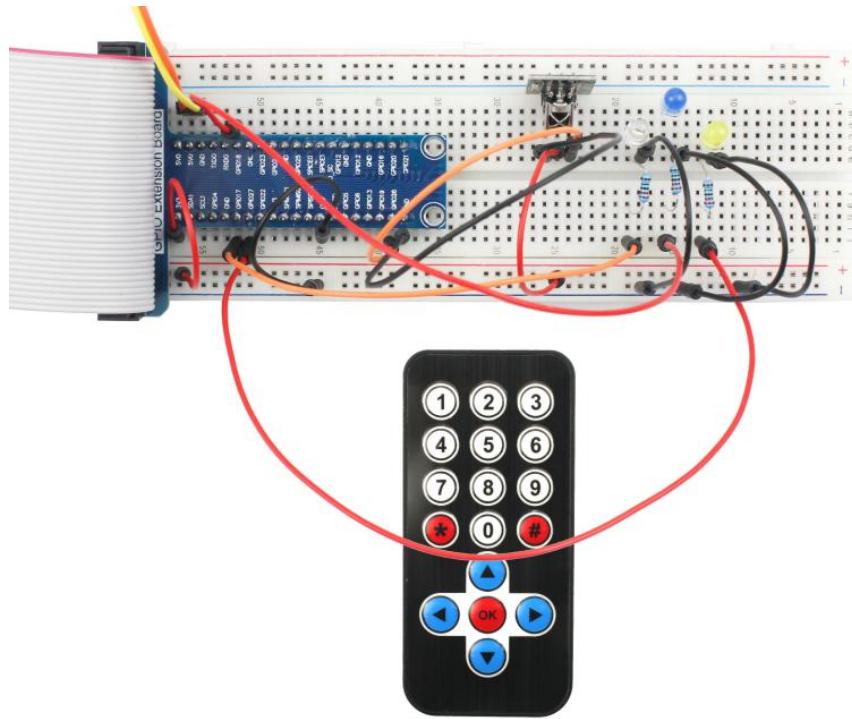
Connection Diagram



Wiring Diagram



Example Figure



C code

Open the terminal and enter the "cd code / C / 31.Infrared /" command to enter the infrared.c code directory;

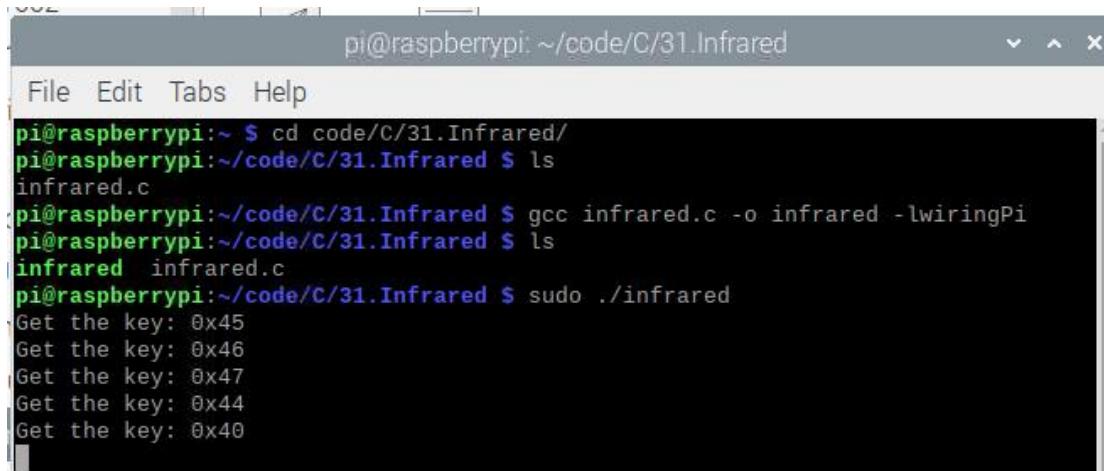
Enter the ls command to view the file "infrared.c" in the directory;



```
pi@raspberrypi:~/code/C/31.Infrared
File Edit Tabs Help
pi@raspberrypi:~ $ cd code/C/31.Infrared/
pi@raspberrypi:~/code/C/31.Infrared $ ls
infrared.c
pi@raspberrypi:~/code/C/31.Infrared $
```

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

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



The screenshot shows a terminal window titled "pi@raspberrypi: ~/code/C/31.Infrared". The terminal displays the following command-line session:

```
pi@raspberrypi:~ $ cd code/C/31.Infrared/
pi@raspberrypi:~/code/C/31.Infrared $ ls
infrared.c
pi@raspberrypi:~/code/C/31.Infrared $ gcc infrared.c -o infrared -lwiringPi
pi@raspberrypi:~/code/C/31.Infrared $ ls
infrared infrared.c
pi@raspberrypi:~/code/C/31.Infrared $ sudo ./infrared
Get the key: 0x45
Get the key: 0x46
Get the key: 0x47
Get the key: 0x44
Get the key: 0x40
```

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

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

#define Pin 25
#define delay_time 120
int LED[3]={0,1,2};
char i,idx,ent;
char count;
char data[4];

static int flag = 0;
int exec_cmd(char key_val)
{
    switch(key_val) {
        case 0x45://1
            digitalWrite(LED[0],HIGH);
            digitalWrite(LED[1],LOW);
            digitalWrite(LED[2],LOW);
            break;
        case 0x46://2
            digitalWrite(LED[0],LOW);
            digitalWrite(LED[1],HIGH);
            digitalWrite(LED[2],LOW);
    }
}
```

```
        break;
    case 0x47://3
        digitalWrite(LED[0],LOW);
        digitalWrite(LED[1],LOW);
        digitalWrite(LED[2],HIGH);
        break;
    default:
        digitalWrite(LED[0],LOW);
        digitalWrite(LED[1],LOW);
        digitalWrite(LED[2],LOW);
        break;
    }

    return 0;
}

void setup()
{
    int i;
    pinMode(Pin, INPUT);
    for(i=0;i<3;i++)
    {
        pinMode(LED[i], OUTPUT);
    }
}

int main(int argc, char const *argv[])
{
    if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
        printf("failed !");
        return 1;
    }
    setup();

    while (1) {
        if(digitalRead(Pin) == LOW) {
            count = 0;
            while (digitalRead(Pin) == LOW && count++ < 200)      //9ms
                delayMicroseconds(delay_time);
            count = 0;
        }
    }
}
```

```

        while (digitalRead(Pin) == HIGH && count++ < 80)      //4.5ms
            delayMicroseconds(delay_time);
        idx = 0;
        cnt = 0;
        data[0]=0;
        data[1]=0;
        data[2]=0;
        data[3]=0;
        for (i =0;i<32;i++) {
            count = 0;
            while (digitalRead(Pin) == LOW && count++ < 15) //0.56ms
                delayMicroseconds(delay_time);

            count = 0;
            while (digitalRead(Pin) == HIGH && count++ < 40) //0: 0.56ms;
1: 1.69ms delayMicroseconds(60); if (count > 25)
            delayMicroseconds(delay_time);
            if (count > 8)
                data[idx] |= 1<<cnt;
            if (cnt== 7) {
                cnt=0;
                idx++;
            } else {
                cnt++;
            }
            if ((data[0]+data[1] == 0xFF) && (data[2]+data[3]==0xFF))
{
    //check
        printf("Get the key: 0x%02x\n",data[2]);
        exec_cmd(data[2]);
    }
}
}
return 0;
}

```

Code Interpretation

The function "setup ()" sets the infrared remote control to the input mode and the LED to the output mode.

```
void setup()
{
    int i;
    pinMode(Pin, INPUT);
    for(i=0;i<3;i++)
    {
        pinMode(LED[i], OUTPUT);
    }
}
```

The function "exec_cmd (char key_val)" is mainly used to turn on the LED light. First of all, it is judged whether the decoded value is equal to the corresponding code value of the button. If it is the code value corresponding to the "1, 2, 3" button, turn on the corresponding LED light, otherwise turn off the LED light.

```
int exec_cmd(char key_val)
{
    switch(key_val) {
        case 0x45://1
            digitalWrite(LED[0],HIGH);
            digitalWrite(LED[1],LOW);
            digitalWrite(LED[2],LOW);
            break;
        case 0x46://2
            digitalWrite(LED[0],LOW);
            digitalWrite(LED[1],HIGH);
            digitalWrite(LED[2],LOW);
            break;
        case 0x47://3
            digitalWrite(LED[0],LOW);
            digitalWrite(LED[1],LOW);
            digitalWrite(LED[2],HIGH);
            break;
        default:
            digitalWrite(LED[0],LOW);
            digitalWrite(LED[1],LOW);
            digitalWrite(LED[2],LOW);
            break;
    }
}
```

```

    return 0;
}

```

In the "for" loop of the main function, the code value of the corresponding key value received is decoded according to the approximate time difference between the high and low levels of the infrared remote control received.

```

for (i =0;i<32;i++) {
    count = 0;
    while (digitalRead(Pin) == LOW && count++ < 15) //0.56ms
        delayMicroseconds(delay_time);

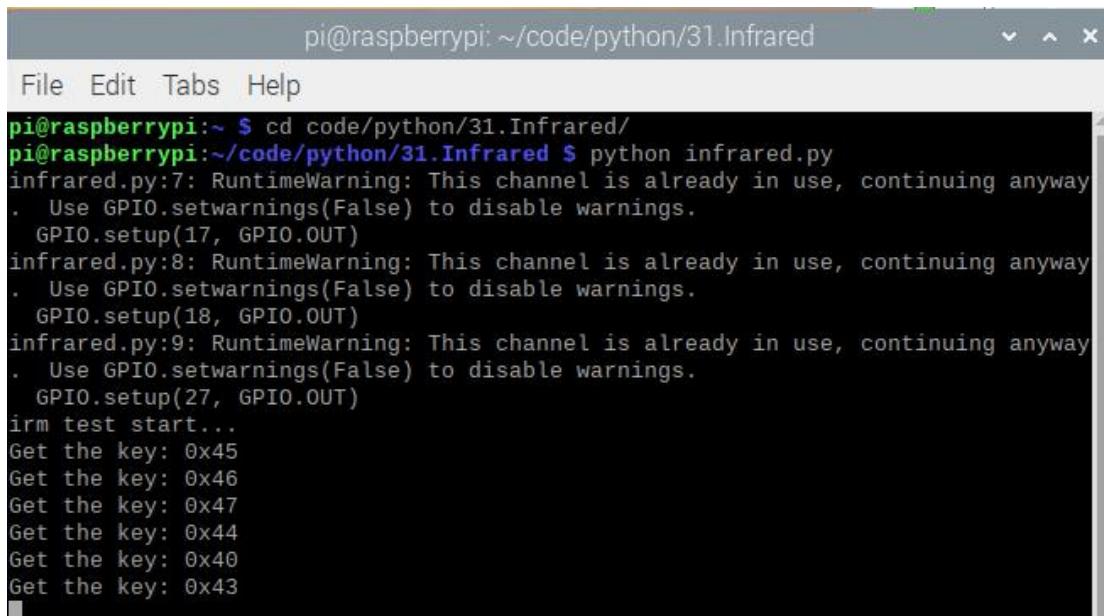
    count = 0;
    while (digitalRead(Pin) == HIGH && count++ < 40) //0: 0.56ms;
1: 1.69ms delayMicroseconds(60); if (count > 25)
    delayMicroseconds(delay_time);
    if (count > 8)
        data[idx] |= 1<<cnt;
    if (cnt== 7) {
        cnt=0;
        idx++;
    } else {
        cnt++;
    }
    if ((data[0]+data[1] == 0xFF) && (data[2]+data[3]==0xFF))
{
    //check
        printf("Get the key: 0x%02x\n",data[2]);
        exec_cmd(data[2]);
    }
}

```

Python code

1. Use the "cd code / python / 31.Infrared /" command to go to the "Infrared" directory.

2. Use the "python infrared.py" command to execute the "infrared.py" code.



```
pi@raspberrypi:~/code/python/31.Infrared
pi@raspberrypi:~/code/python/31.Infrared$ python infrared.py
infrared.py:7: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(17, GPIO.OUT)
infrared.py:8: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(18, GPIO.OUT)
infrared.py:9: RuntimeWarning: This channel is already in use, continuing anyway
. Use GPIO.setwarnings(False) to disable warnings.
    GPIO.setup(27, GPIO.OUT)
irm test start...
Get the key: 0x45
Get the key: 0x46
Get the key: 0x47
Get the key: 0x44
Get the key: 0x40
Get the key: 0x43
```

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

```
import RPi.GPIO as GPIO
import time
PIN = 26;
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(PIN,GPIO.IN,GPIO.PUD_UP)
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(18, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)
    print("irm test start...")

def exec_cmd(key_val):
    if(key_val==0x45):
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(27, GPIO.LOW)
    elif(key_val==0x46):
        GPIO.output(17, GPIO.LOW)
        GPIO.output(18, GPIO.HIGH)
        GPIO.output(27, GPIO.LOW)
    elif(key_val==0x47):
```

```
GPIO.output(17, GPIO.LOW)
GPIO.output(18, GPIO.LOW)
GPIO.output(27, GPIO.HIGH)

else :
    GPIO.output(17, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(27, GPIO.LOW)

def loop():
    while True:
        if GPIO.input(PIN) == 0:
            count = 0
            while GPIO.input(PIN) == 0 and count < 200:
                count += 1
                time.sleep(0.00006)

            count = 0
            while GPIO.input(PIN) == 1 and count < 80:
                count += 1
                time.sleep(0.00006)

        idx = 0
        cnt = 0
        data = [0,0,0,0]
        for i in range(0,32):
            count = 0
            while GPIO.input(PIN) == 0 and count < 15:
                count += 1
                time.sleep(0.00006)

            count = 0
            while GPIO.input(PIN) == 1 and count < 40:
                count += 1
                time.sleep(0.00006)

        if count > 8:
            data[idx] |= 1<<cnt
        if cnt == 7:
            cnt = 0
            idx += 1
```

```

        else:
            cnt += 1
            if data[0]+data[1] == 0xFF and data[2]+data[3] == 0xFF:
                print("Get the key: 0x%02x" %data[2])
                exec_cmd(data[2])

def destroy():
    GPIO.output(PIN, GPIO.LOW)      # led off
    GPIO.cleanup()                  # Release resource

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

```

Code Interpretation

The function "setup ()" sets the infrared remote control to the input mode and the LED to the output mode.

```

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(PIN,GPIO.IN,GPIO.PUD_UP)
    GPIO.setup(17, GPIO.OUT)
    GPIO.setup(18, GPIO.OUT)
    GPIO.setup(27, GPIO.OUT)
    print("irm test start...")

```

The function "exec_cmd (key_val)" is mainly used to turn on the LED light. First of all, it is judged whether the decoded value is equal to the corresponding code value of the key. If it is the code value corresponding to the "1, 2, 3" button, turn on the corresponding LED light, otherwise turn off the LED light.

```

def exec_cmd(key_val):
    if(key_val==0x45):
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(27, GPIO.LOW)
    elif(key_val==0x46):

```

```
GPIO.output(17, GPIO.LOW)
GPIO.output(18, GPIO.HIGH)
GPIO.output(27, GPIO.LOW)
elif(key_val==0x47):
    GPIO.output(17, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(27, GPIO.HIGH)
else :
    GPIO.output(17, GPIO.LOW)
    GPIO.output(18, GPIO.LOW)
    GPIO.output(27, GPIO.LOW)
```

In the "for" loop of the main function, the code value of the corresponding key value received is decoded according to the approximate time difference between the high and low levels of the infrared remote control received.

```
for i in range(0,32):
    count = 0
    while GPIO.input(PIN) == 0 and count < 15:
        count += 1
        time.sleep(0.00006)

    count = 0
    while GPIO.input(PIN) == 1 and count < 40:
        count += 1
        time.sleep(0.00006)

    if count > 8:
        data[idx] |= 1<<cnt
    if cnt == 7:
        cnt = 0
        idx += 1
    else:
        cnt += 1
```

Lesson 32 Buzzer Welding

Overview

We will use PCB boards to solder circuits and components. By the end of this chapter, I hope to help you master how to design your own circuits and inspire ideas for building and printed circuit boards. To complete this chapter, you need to prepare the necessary soldering equipment, including a soldering iron and wire, and other auxiliary materials. We have prepared for you welding materials such as universal board, tin wire, pin header, and female header. Because the temperature of the soldering iron can reach hundreds of degrees or more, please be careful during operation.

In this lesson, we will learn to weld a circuit that controls the buzzer by pressing a button. The buzzer sounds when the button is pressed

This circuit requires no programming and can work at power up. When the button is not pressed, the circuit will not consume power. You can install it on a bicycle, bedroom door or anywhere you need it.

Parts Required

1 x Pin Header

1 x Red LED

1 x 220 Ohm Resistor

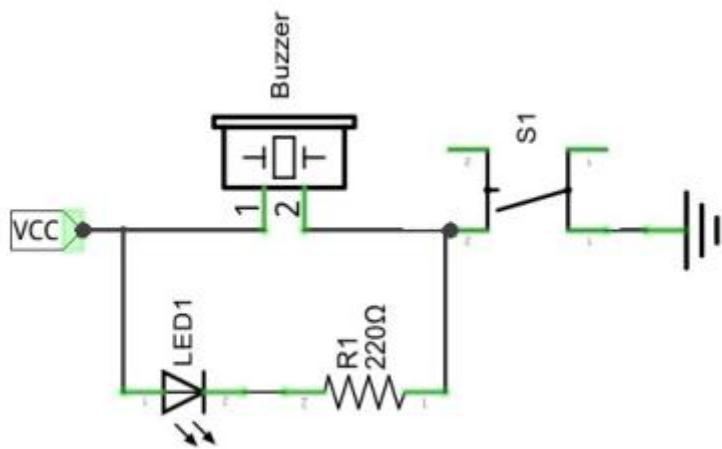
1 x Active Buzzer

1 x Button

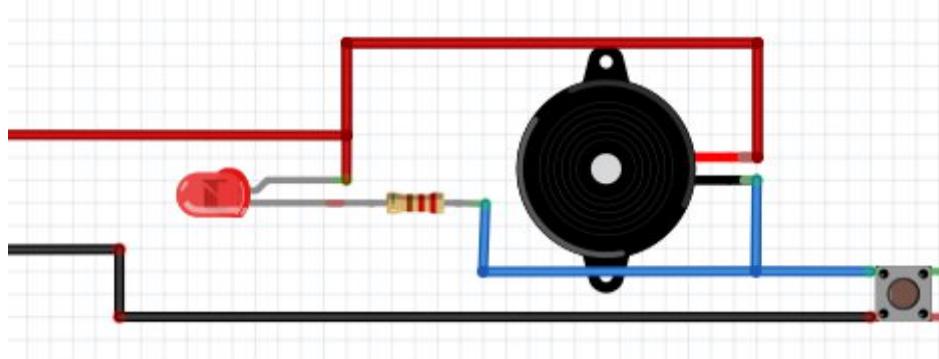
Connection Circuit:

We will use the universal board to solder the following circuits;

Connection Diagram

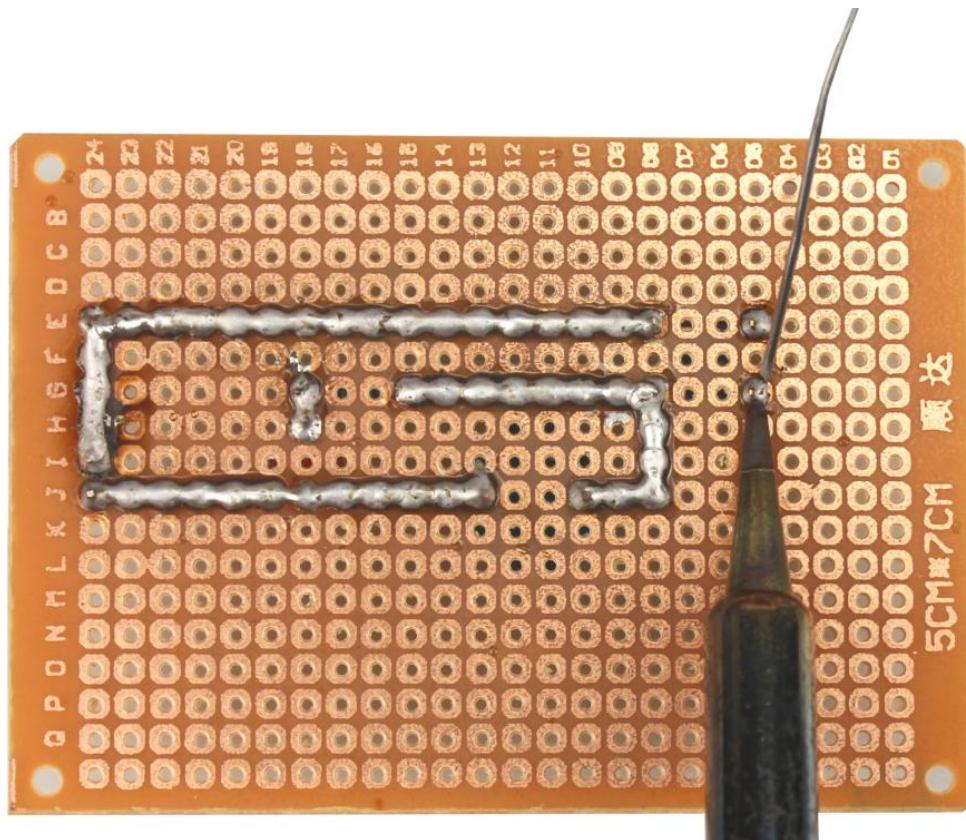


Wiring Diagram

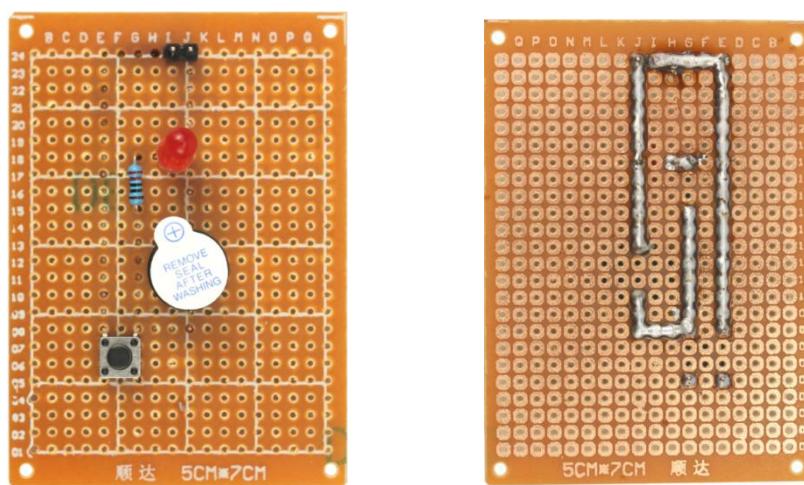


Welding circuit

Insert the component into the universal board, insert the pins from the side without copper clad, and solder the circuit to the side with copper clad. As shown below:

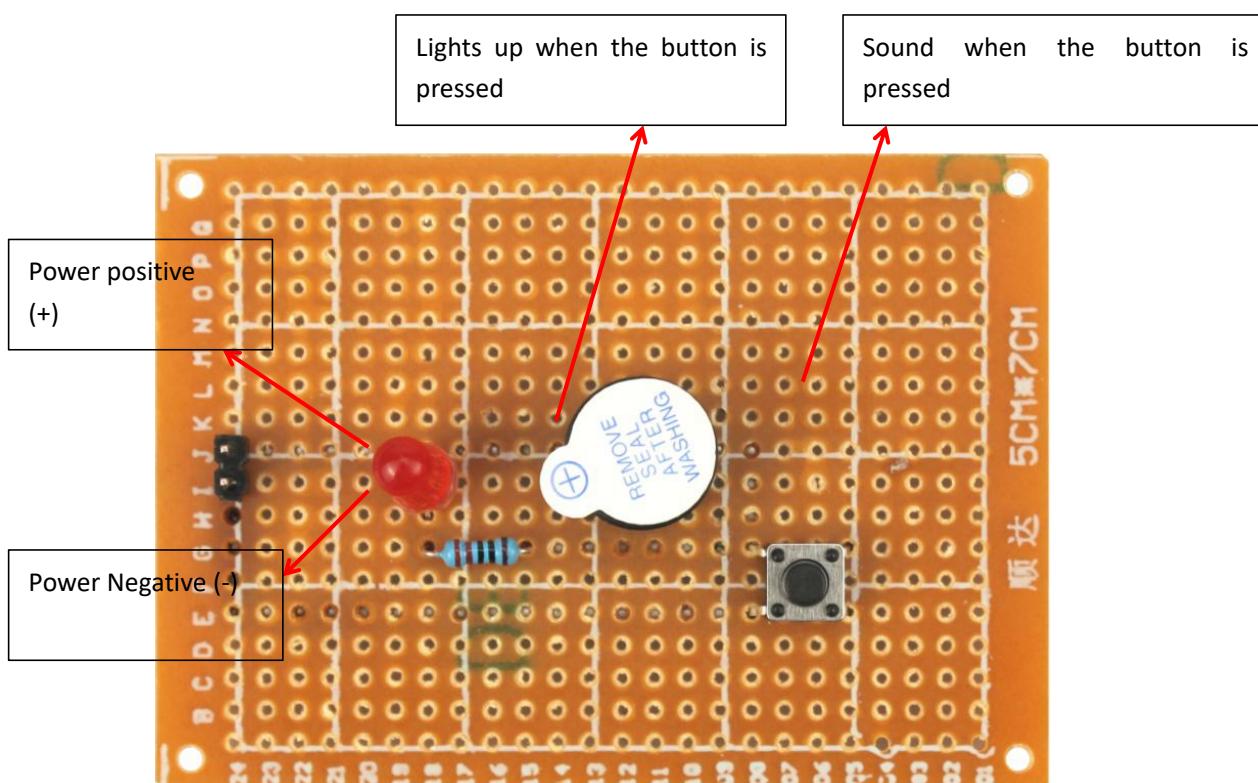


The effect diagram after welding is as follows:



Test if the circuit is soldered successfully:

Connect the two pins of the circuit ($3.3V \sim 5V$) to the power supply. This power supply can be powered by battery, arduino development board, raspberry pie, etc. After connecting the power, press the button, the power indicator LED will light up and the buzzer will sound A sound sounds to prove that the circuit is successfully welded. If the welding equipment does not show these two effects, it proves that there is a problem with the welded circuit and the circuit needs to be rechecked.



Lesson 33 Running Water Welding

Overview

In this lesson, we will learn how to weld the water lamp. Because there are 8 LED lights, we need 8 arduino or raspberry pi I/O ports, but this wastes the pin resources of the development board, so we use 74HC595 chip to expand our I/O port, the specific tutorial is as follows.

Parts Required

7 x Header

8 x 220 Ohm Resistor

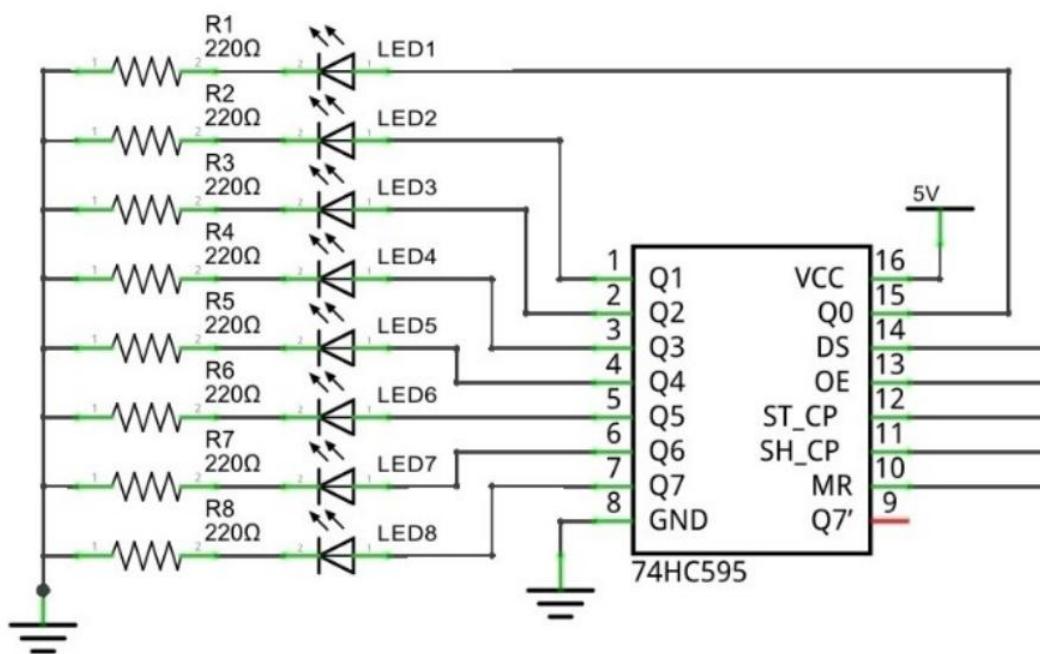
8 x LED

1 x 74HC595

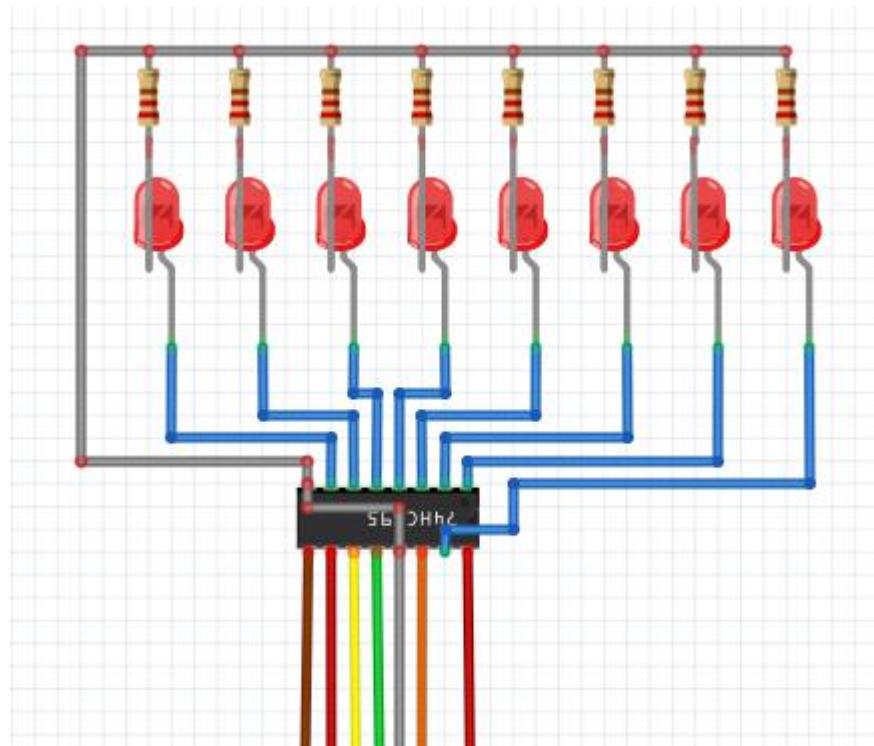
Circuit

Solder the circuit on the universal board

Connection Diagram

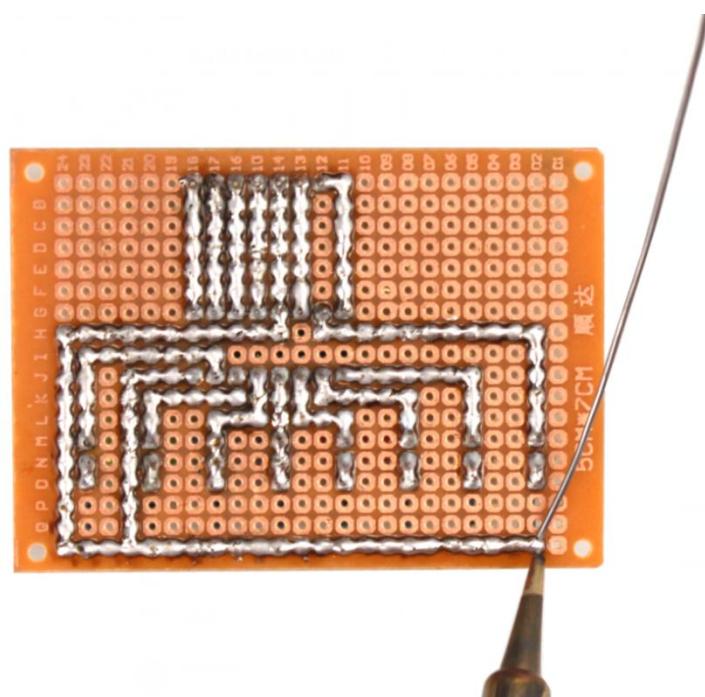


Wiring Diagram

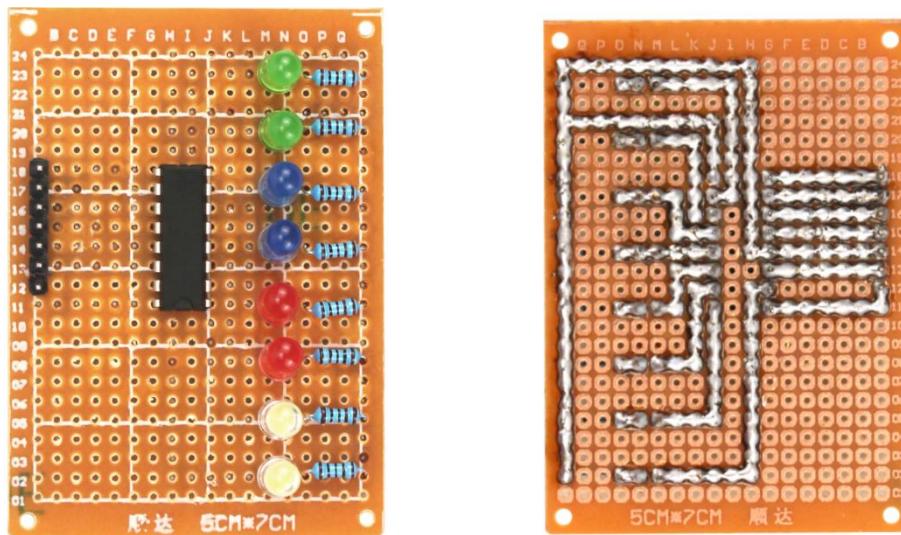


Welding Circuit

Insert the component into the universal board, insert the pins from the side without copper clad, and solder the circuit to the side with copper clad. As shown below.



The effect diagram after welding is as follows:



Test if the circuit is soldered successfully:

Connect the circuit board to the Raspberry Pi. For detailed connection and procedures, please refer to the course ‘74HC595 and LED’;

