SunFounder electronic-kit

www.sunfounder.com

Sep 21, 2022

CONTENTS

CHAPTER

ONE

ABOUT THE ELECTRONIC KIT

This kit is suitable for SunFounder Uno, SunFounder Mega 2560, SunFounder Duemilanove and SunFounder Nano. All the code in this user guide is compatible with these boards. With this kit, we will walk you through the know-how of using the Arduino board in a hands-on way. Starting with the basics of electronics, you'll learn through building several creative projects. Including a selection of the most common and useful electronic components, this kit will help you "control" the physical world.

If you want to learn another projects which we don't have, please feel free to send Email and we will update to our online tutorials as soon as possible, any suggestions are welcomed.

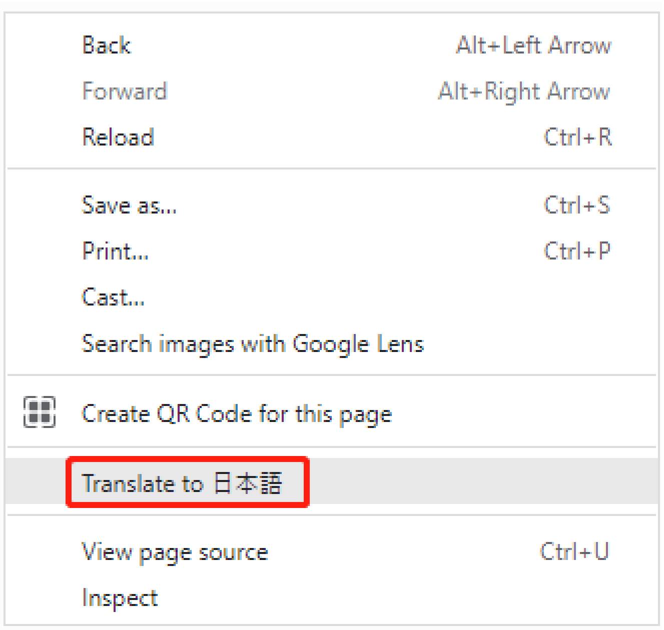
Here is the Email: cs@sunfounder.com.

About the display language

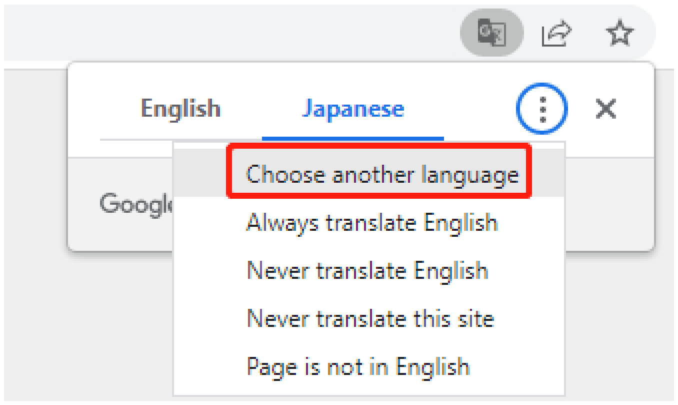
In addition to English, we are working on other languages for this course. Please contact service@sunfounder.com if you are interested in helping, and we will give you a free product in return. In the meantime, we recommend using Google Translate to convert English to the language you want to see.

The steps are as follows.

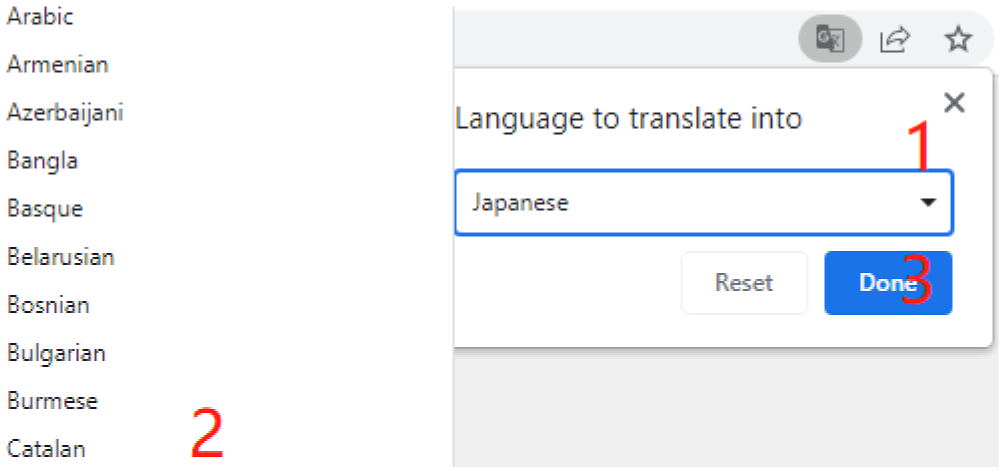
- In this course page, right-click and select **Translate to xx**. If the current language is not what you want, you can change it later.



- There will be a language popup in the upper right corner. Click on the menu button to **choose another language**.



- Select the language from the inverted triangle box, and then click **Done**.



1.1 Component List

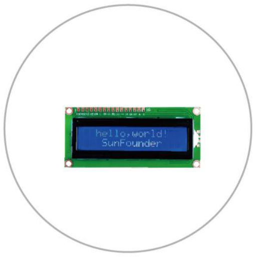
Note: After unpacking, please check that the number of components is correct and that all components are in good condition.

7-Segment

1 PCS

Button

10 PCS

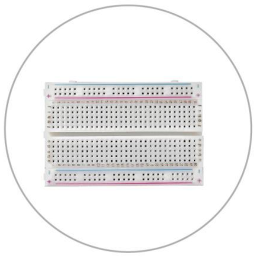


LCD1602

1 PCS

Servo

1 PCS



Breadboard

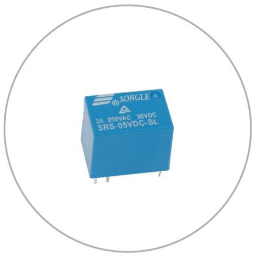
1 PCS

1M USB cable

1 PCS

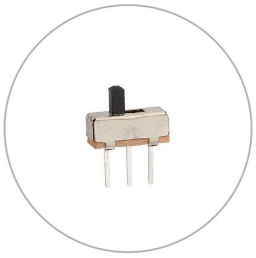
Passive buzzer

1 PCS



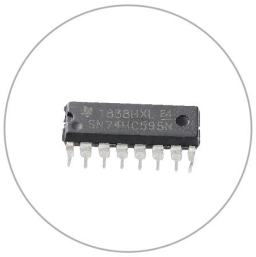
Relay

1 PCS



Slide Switch

5 PCS

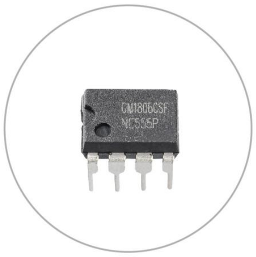


74HC595

1 PCS

4N35

1 PCS

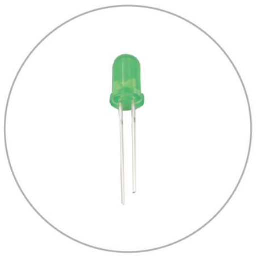


555 Timer

1 PCS

RGB LED
1 PCS

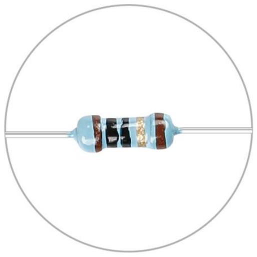
LED (Red)
10 PCS

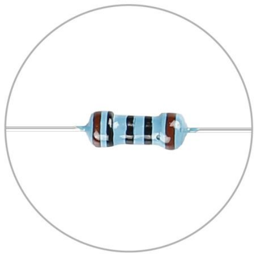

LED (Green)
10 PCS

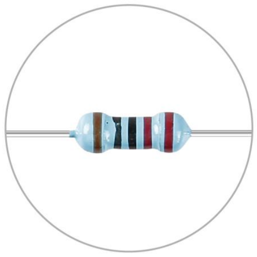
LED (Yellow)
10 PCS

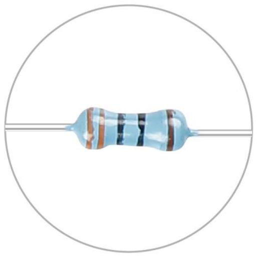
LED (White)
10 PCS

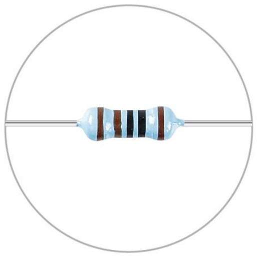
LED(Blue)
10 PCS

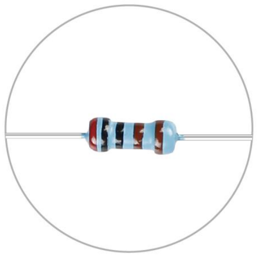

Resistor(10Ω)
10 PCS

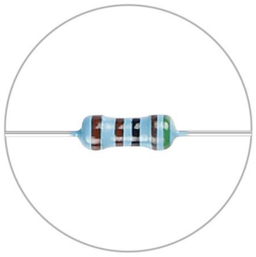

Resistor(100Ω)
10 PCS


Resistor(220Ω)
10 PCS


Resistor(330Ω)
10 PCS


Resistor(1KΩ)
10 PCS


Resistor(2KΩ)
10 PCS

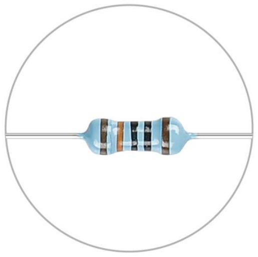


Resistor(5.1KΩ)

10 PCS

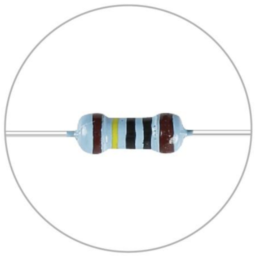
Resistor(10KΩ)

10 PCS



Resistor(100KΩ)

10 PCS



Resistor(1MΩ)

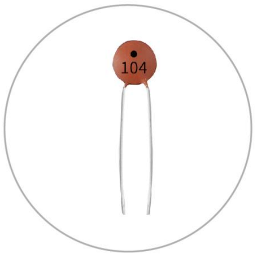
10 PCS

1N4007

5 PCS

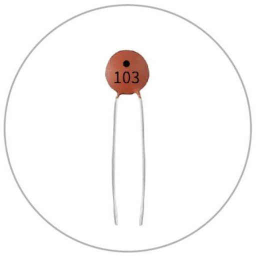
Zener diode

1 PCS



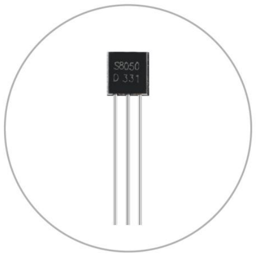
104 Capacitor

10 PCS



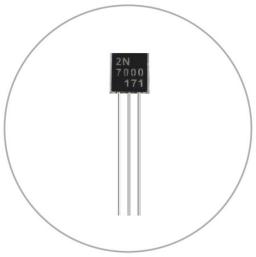
103 Capacitor

10 PCS



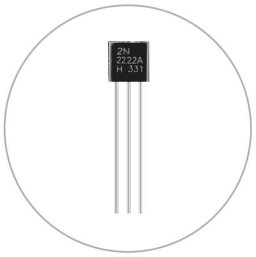
S8050 Transistor

2 PCS



2N7000 Transistor

1 PCS

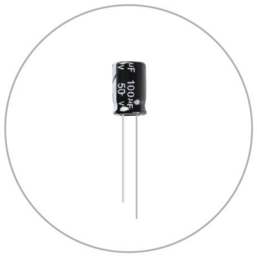


PN222A

5 PCS

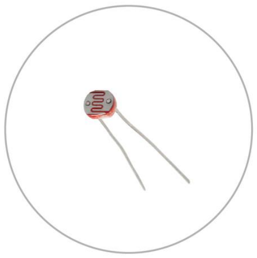
Capacitor (10UF)

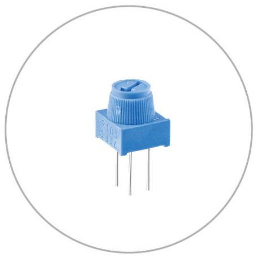
5 PCS


Capacitor (100UF)
5 PCS

Thermistor
1 PCS

Tilt switch
1 PCS


Photoresistor
2 PCS


Potentiometer
2 PCS

Active buzzer
1 PCS

DuPont wires
20 PCS

Battery snap
1 PCS

Jump wires
65 PCS

1.2 For Raspberry Pi User

1.2.1 Preparation

In this chapter, we firstly learn to start up Raspberry Pi. The content includes installing the OS, Raspberry Pi network and how to open terminal.

Note: You can check the complete tutorial on the official website of the Raspberry Pi: raspberry-pi-setting-up.

If your Raspberry Pi is set up, you can skip the part and go into the next chapter.

Installing the OS

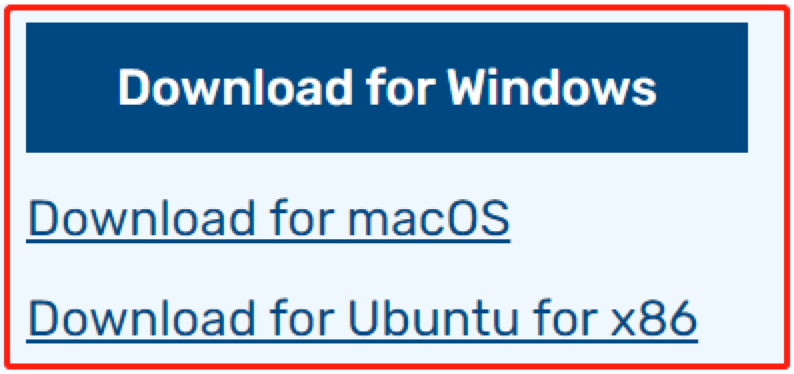
Required Components

Any Raspberry Pi	1 * Personal Computer
1 * Micro SD card	

Step 1

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

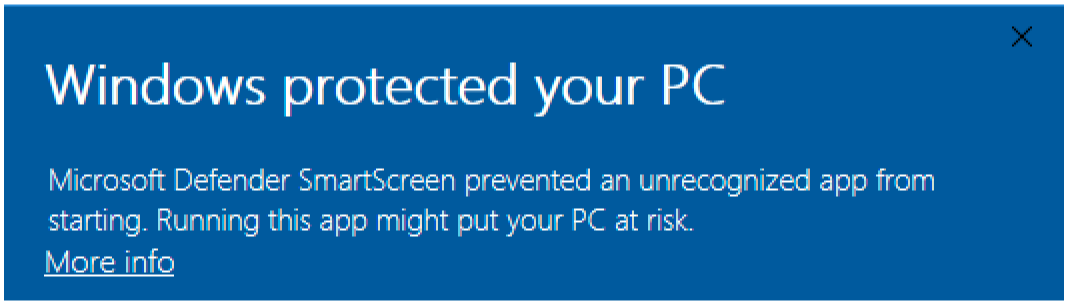
Visit the download page: https://www.raspberrypi.org/software/. Click on the link for the Raspberry Pi Imager that matches your operating system, when the download finishes, click it to launch the installer.



Step 2

When you launch the installer, your operating system may try to block you from running it. For example, on Windows I receive the following message:

If this pops up, click on **More info** and then **Run anyway**, then follow the instructions to install the Raspberry Pi Imager.



Windows protected your PC

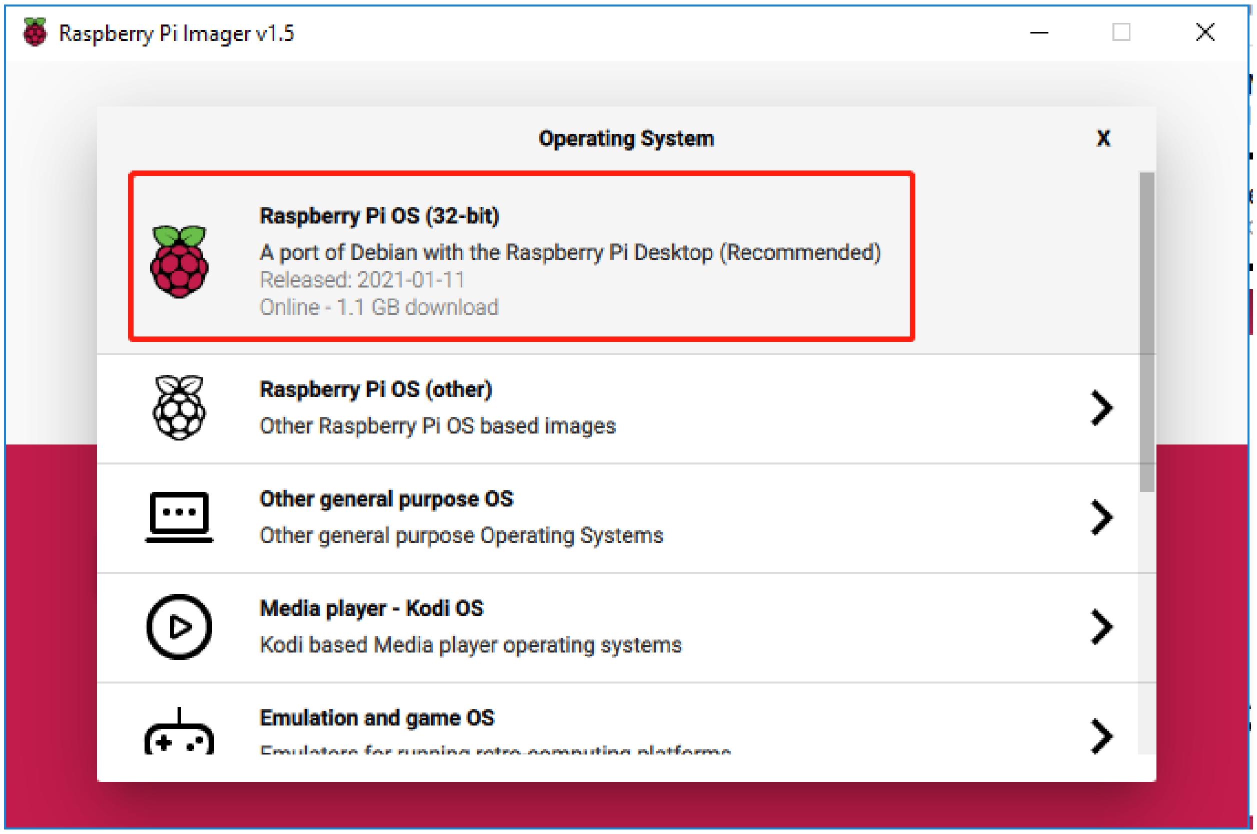
Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.
More info

Step 3

Insert your SD card into the computer or laptop SD card slot.

Step 4

In the Raspberry Pi Imager, select the OS that you want to install and the SD card you would like to install it on.



Raspberry Pi Imager v1.5

Operating System

Raspberry Pi OS (32-bit)
A port of Debian with the Raspberry Pi Desktop (Recommended)
Released: 2021-01-11
Online - 1.1 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

Other general purpose OS
Other general purpose Operating Systems

Media player - Kodi OS
Kodi based Media player operating systems

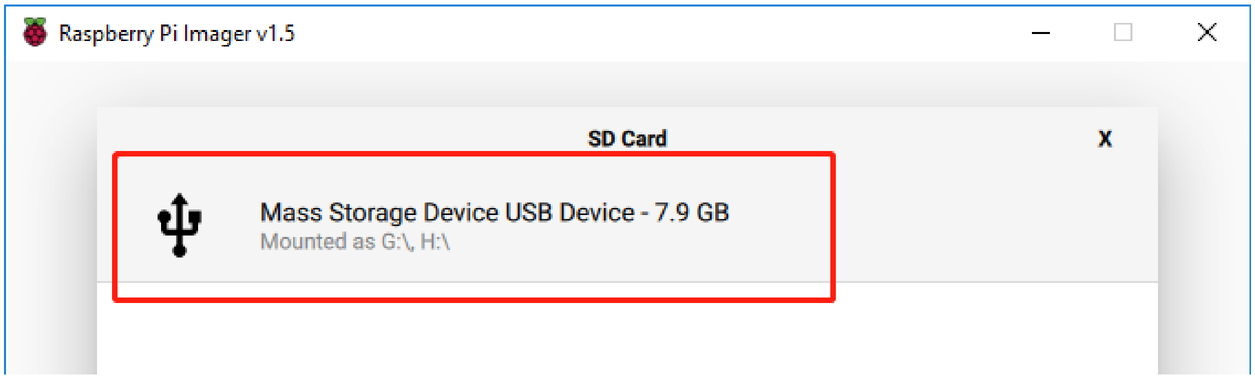
Emulation and game OS
Emulators for running retro-computing platforms

Note:

1) You will need to be connected to the internet the first time.
2) That OS will then be stored for future offline use(lastdownload.cache, C:/Users/yourname/AppData/Local/Raspberry Pi/Imager/cache). So the next time you open the software, it will have the display "Released: date, cached on your computer".

Step 5

Select the SD card you are using.



Mass Storage Device USB Device - 7.9 GB
Mounted as G:\, H:\

Step 6

Press **Ctrl+Shift+X** to open the **Advanced options** page to enable SSH and configure wifi, these 2 items must be set, the others depend on your choice . You can choose to always use this image customization options.

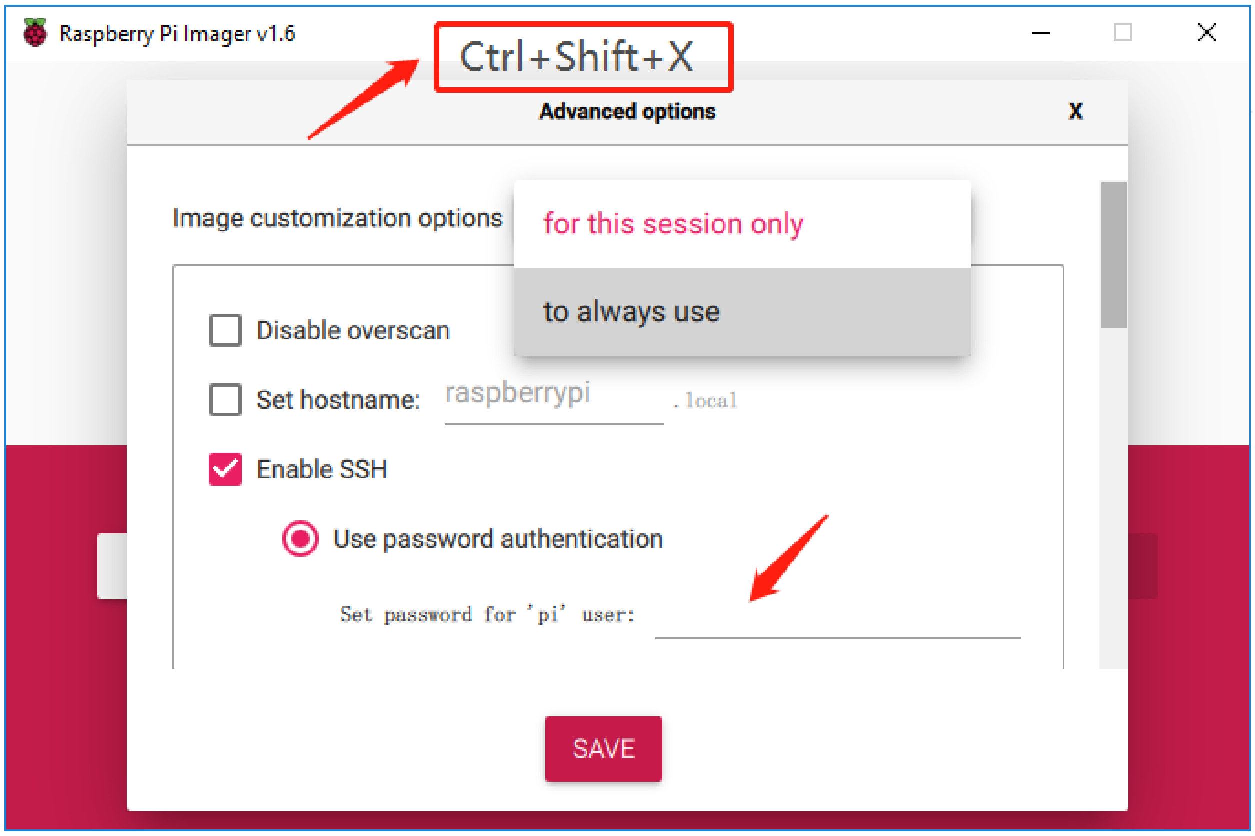


Image customization options: for this session only / to always use

Disable overscan

Set hostname: raspberrypi .local

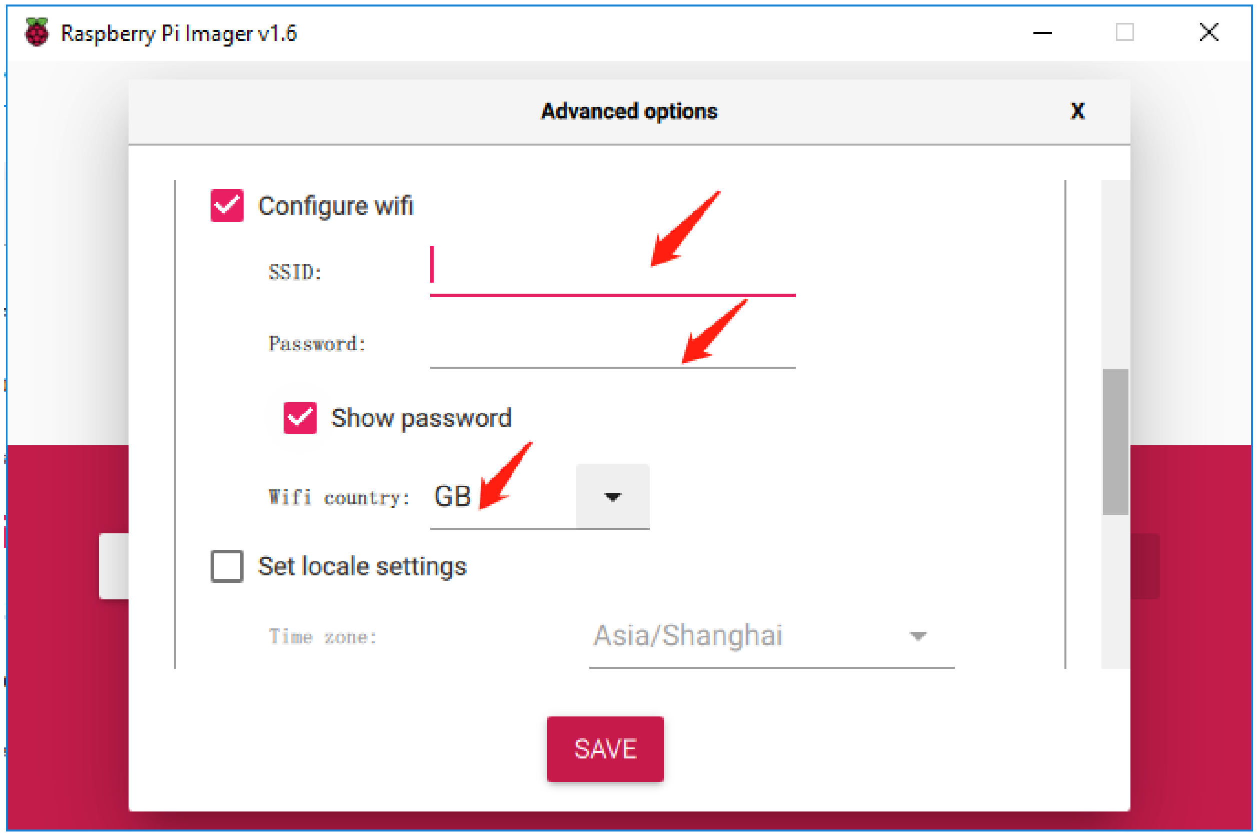
[✓] Enable SSH

● Use password authentication

Set password for 'pi' user:

Then scroll down to complete the wifi configuration and click **SAVE**.

Note: **wifi country** should be set the two-letter ISO/IEC alpha2 code for the country in which you are using your Raspberry Pi, please refer to the following link: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements.



Configure wifi [✓]

SSID:

Password:

Show password [✓]

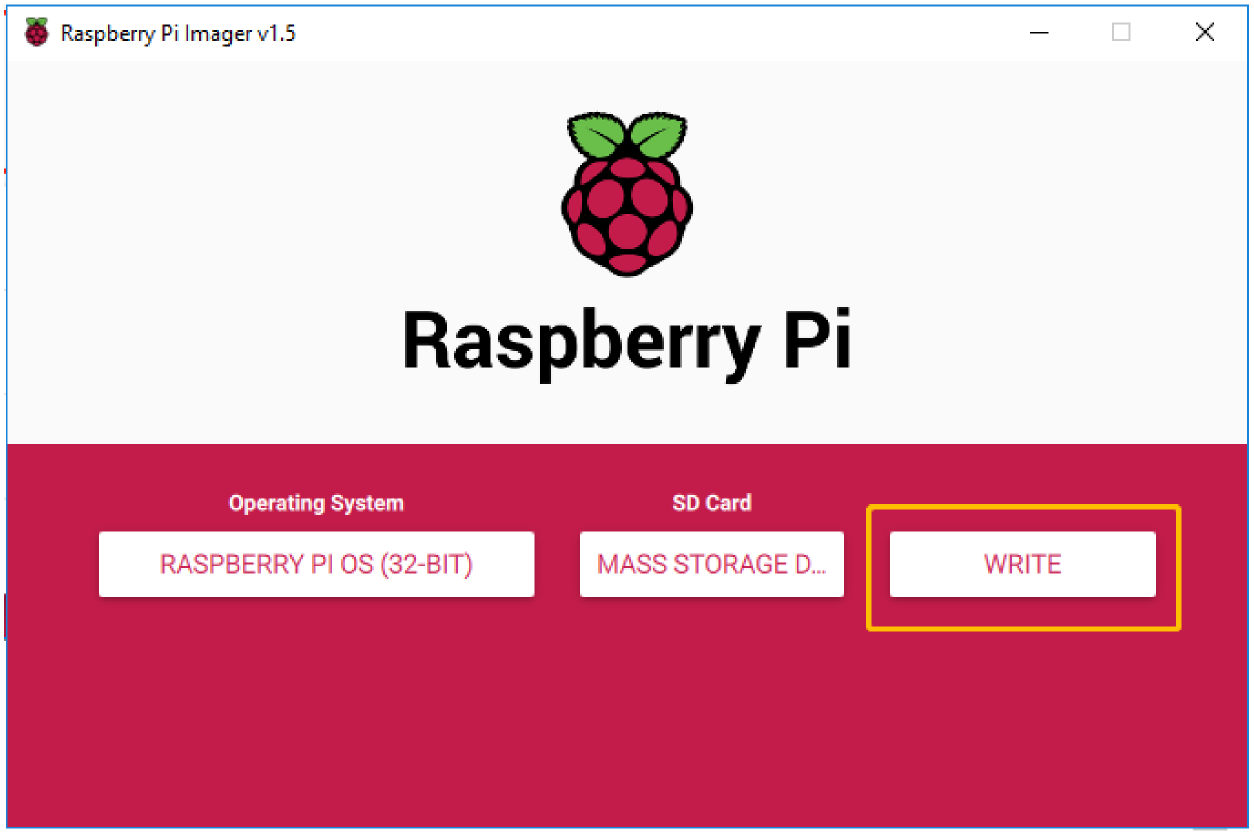
Wifi country: GB

Set locale settings [ ]

Time zone: Asia/Shanghai

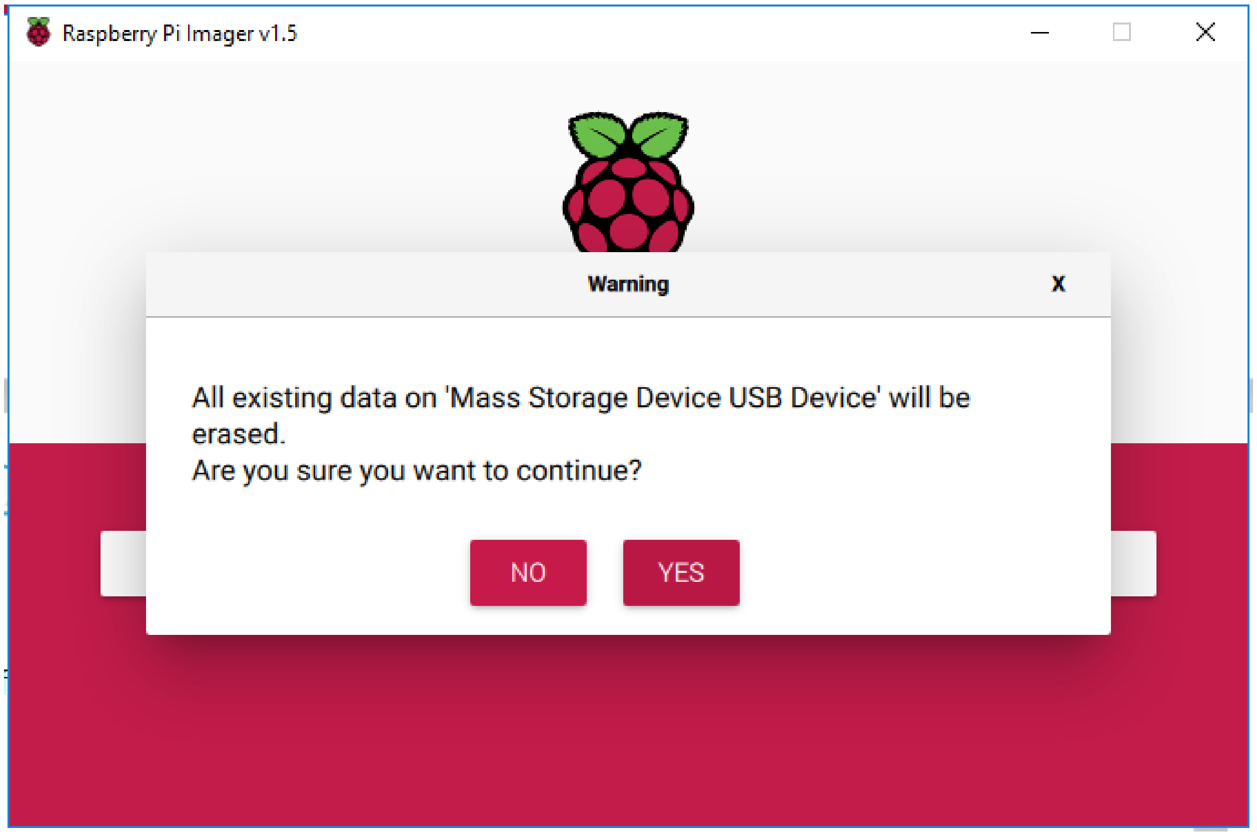
Step 7

Click the **WRITE** button.



Step 8

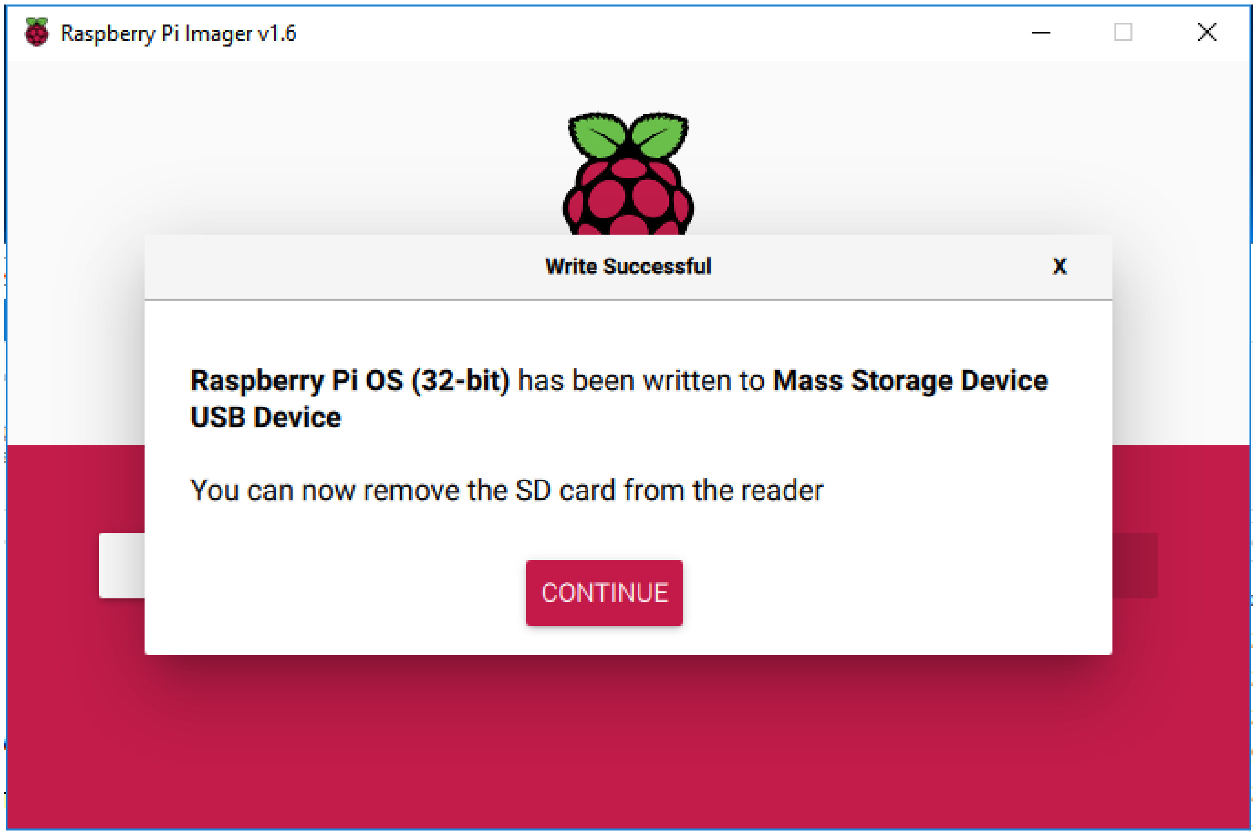
If your SD card currently has any files on it, you may wish to back up these files first to prevent you from permanently losing them. If there is no file to be backed up, click **Yes**.



All existing data on 'Mass Storage Device USB Device' will be erased.
Are you sure you want to continue?

Step 9

After waiting for a period of time, the following window will appear to represent the completion of writing.



Raspberry Pi OS (32-bit) has been written to **Mass Storage Device USB Device**

You can now remove the SD card from the reader

Set up Your Raspberry Pi

If You Have a Screen

If you have a screen, it will be easy for you to operate on the Raspberry Pi.

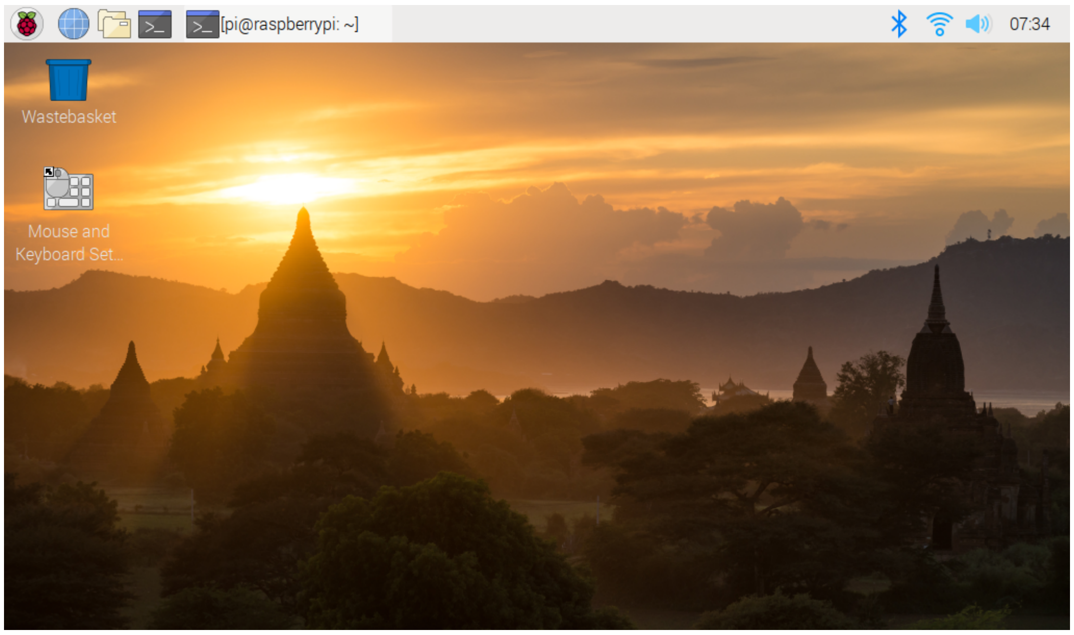
Required Components

Any Raspberry Pi	1 * Power Adapter
1 * Micro SD card	1 * Screen Power Adapter
1 * HDMI cable	1 * Screen
1 * Mouse	1 * Keyboard

1) Insert the SD card you've set up with Raspberry Pi OS into the micro SD card slot on the underside of your Raspberry Pi.
2) Plug in the Mouse and Keyboard.
3) Connect the screen to Raspberry Pi's HDMI port and make sure your screen is plugged into a wall socket and switched on.

Note: If you use a Raspberry Pi 4, you need to connect the screen to the HDMI0 (nearest the power in port).

4) Use the power adapter to power the Raspberry Pi. After a few seconds, the Raspberry Pi OS desktop will be displayed.



If You Have No Screen

If you don't have a display, you can log in to the Raspberry Pi remotely, but before that, you need to get the IP of the Raspberry Pi.

Get the IP Address

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

1. Checking via the router

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

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

2. Network Segment Scanning

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

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspberry Pi OS is **raspberrypi**, if you haven't modified it.

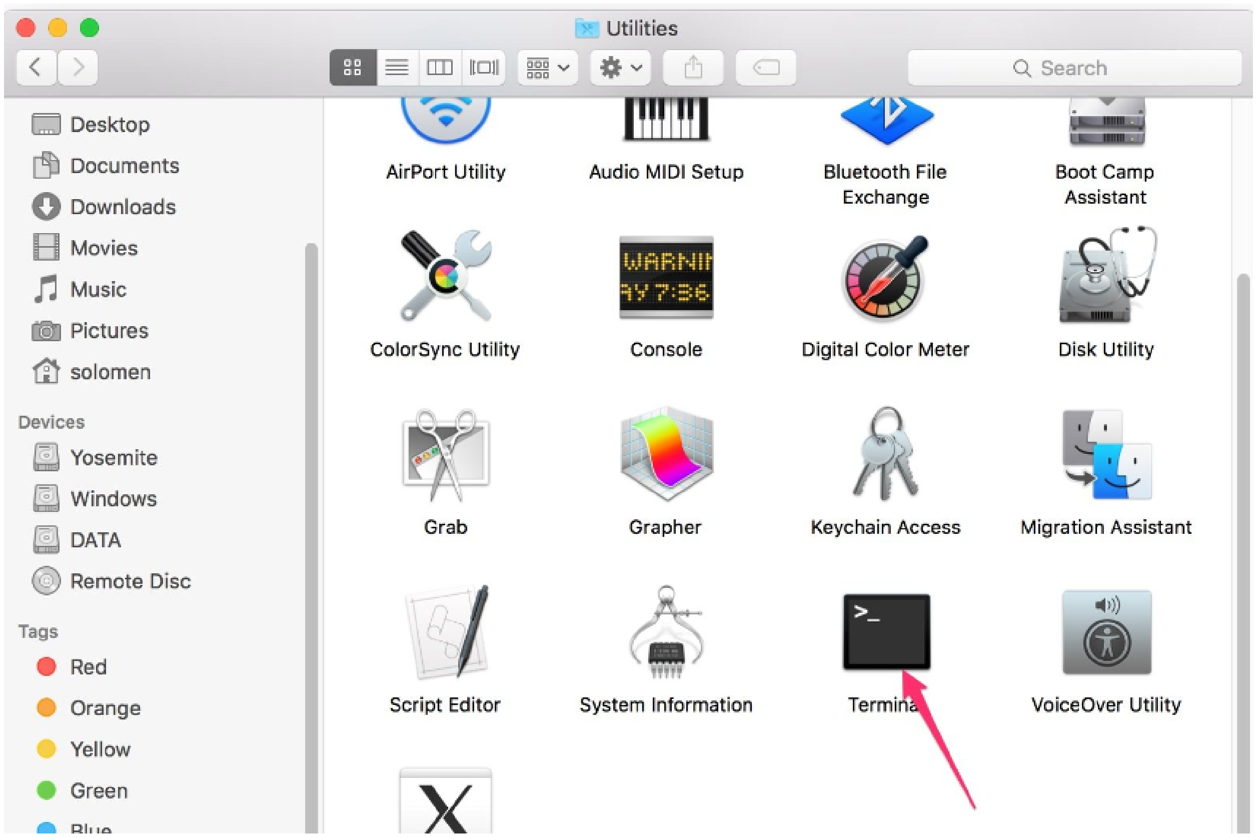
Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

For Linux or/Mac OS X Users

Step 1

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



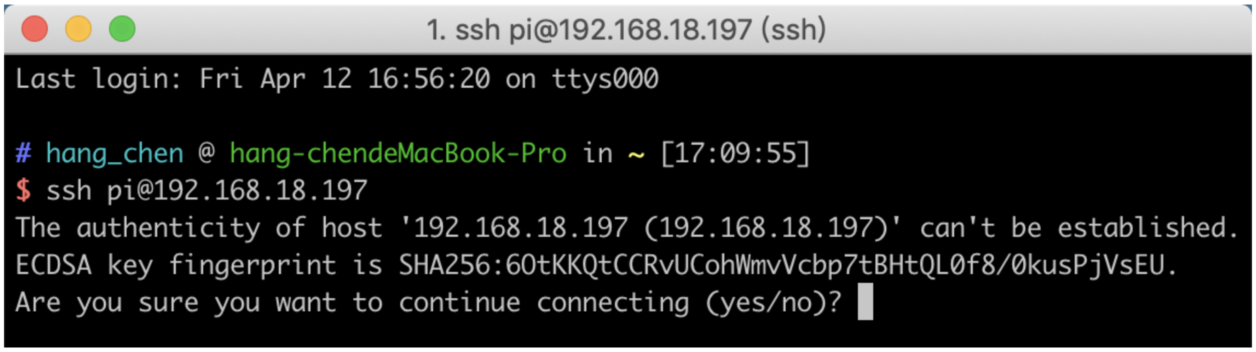
Step 2

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

```
ssh pi@192.168.18.197
```

Step 3

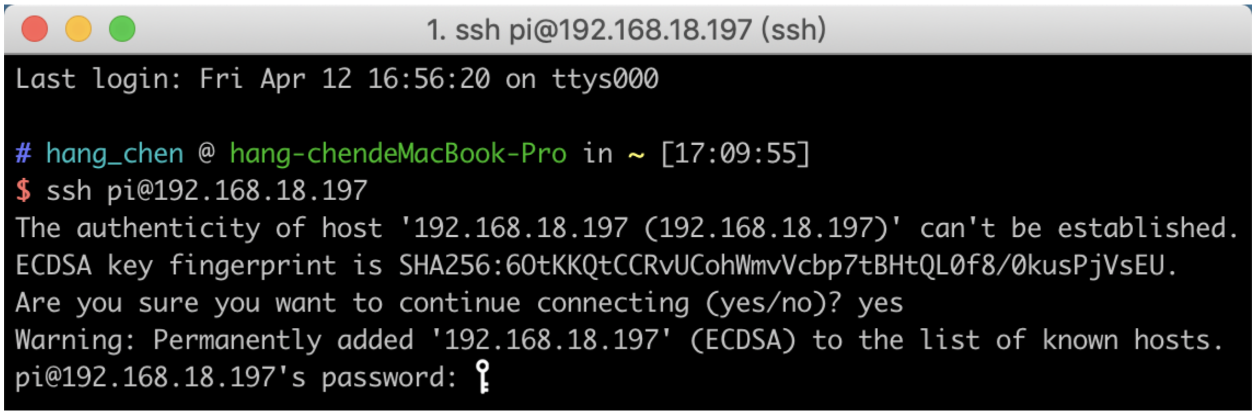
Input "yes".



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

Step 4

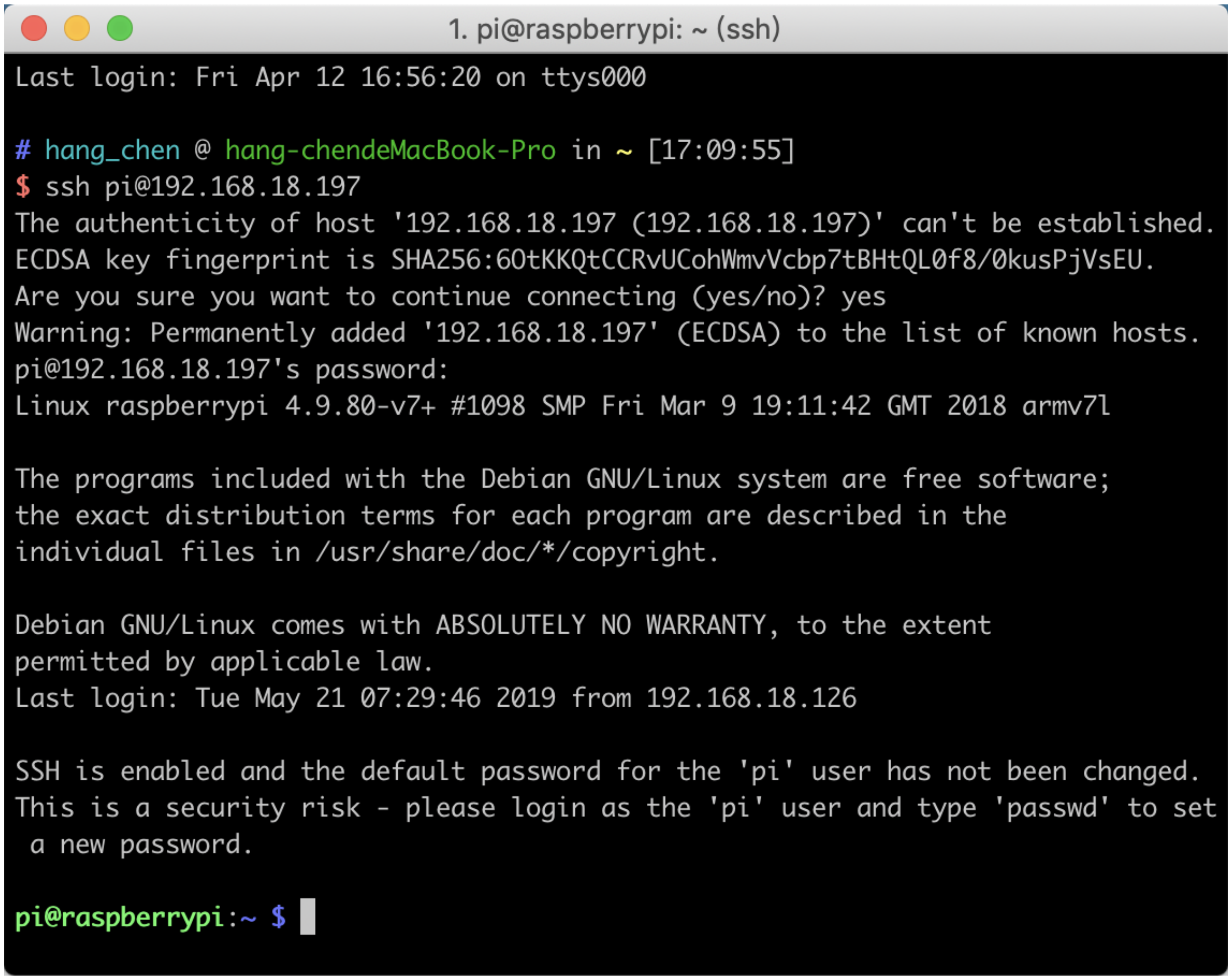
Input the passcode and the default password is **raspberry**.



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
```

Step 5

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



```
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

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

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

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

pi@raspberrypi:~ $
```

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

For Windows Users

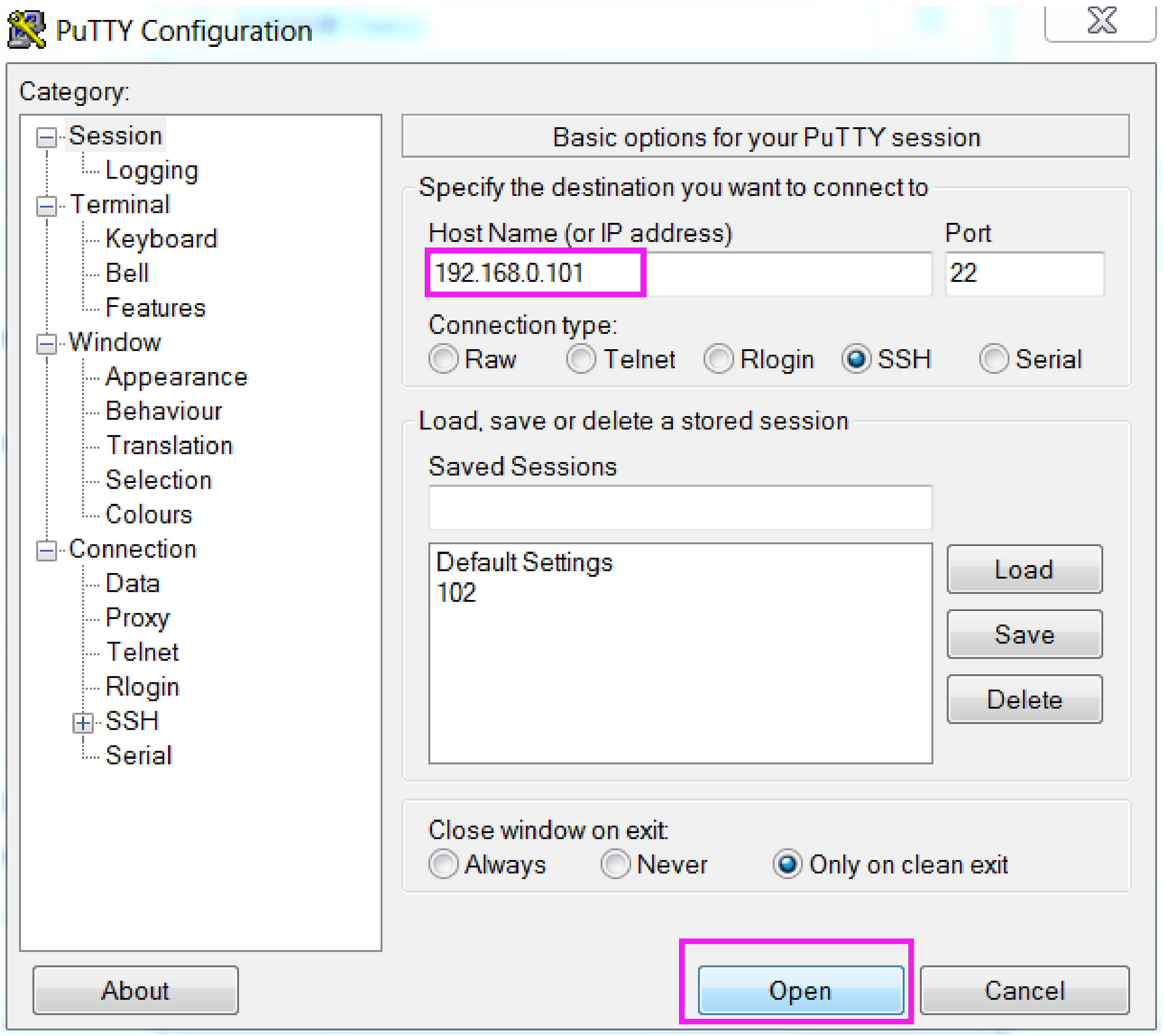
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

Step 1

Download PuTTY.

Step 2

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



Host Name (or IP address): 192.168.0.101

Port: 22

Connection type: ( ) Raw ( ) Telnet ( ) Rlogin [✓ SSH] ( ) Serial

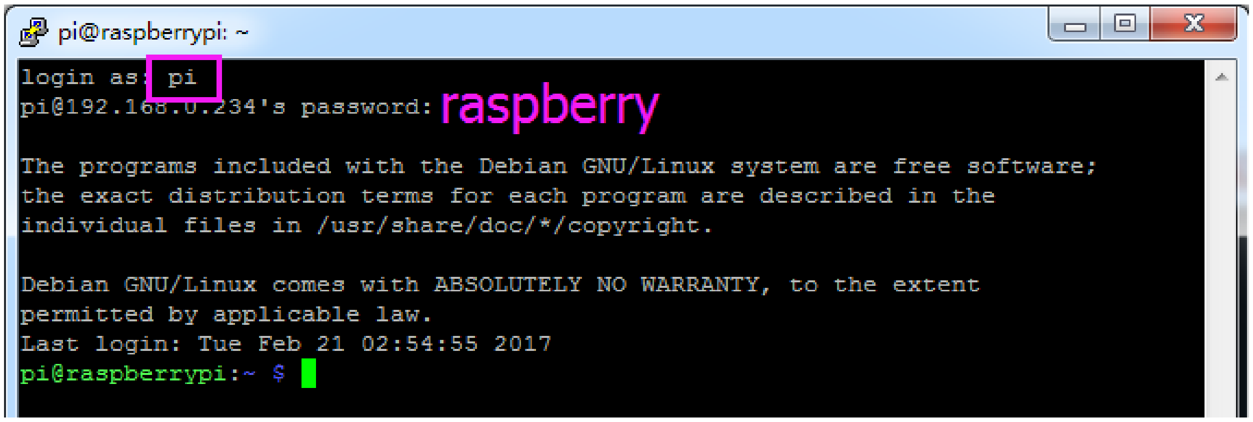
Close window on exit: ( ) Always ( ) Never [✓ Only on clean exit]

Step 3

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

Step 4

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


pi@raspberrypi: ~

```
login as: pi
pi@192.168.0.234's password: raspberry

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 21 02:54:55 2017
pi@raspberrypi:~ $
```

Step 5

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

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

If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily.

For details on how to do this, please refer to *Remote Desktop*.

1.2.2 Libraries

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

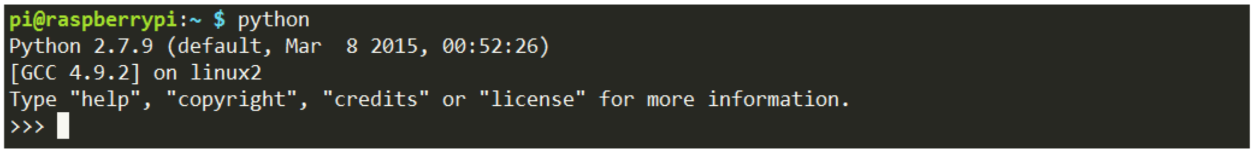
RPi.GPIO

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

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

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

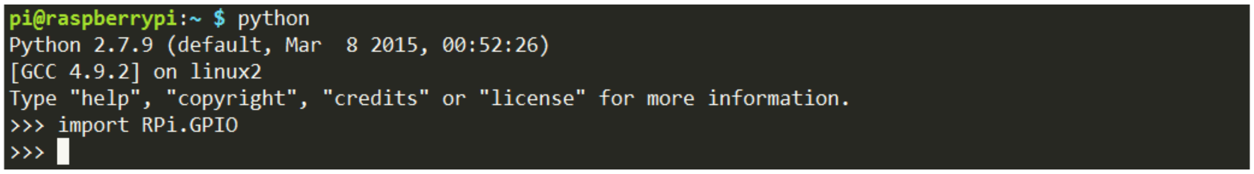
```
python
```



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

In Python CLI, input "**import RPi.GPIO**", if no error prompts, it means RPi.GPIO is installed.

```
import RPi.GPIO
```



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

If you want to quit python CLI, type in:

```
exit()
```

1.2.3 Install and Check the WiringPi

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

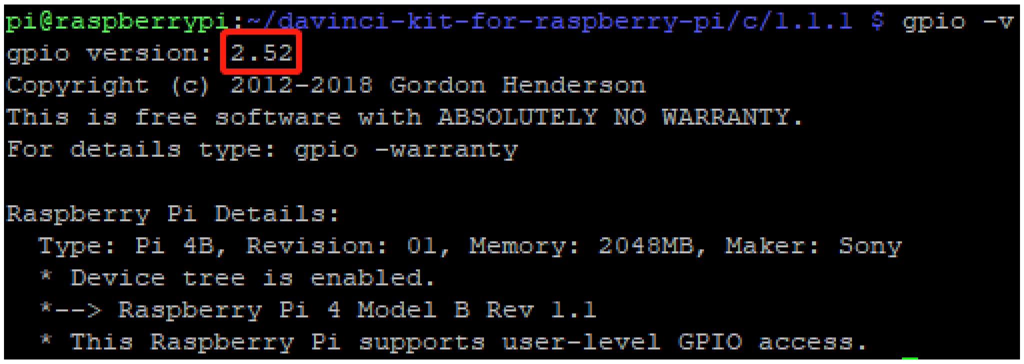
`wiringPi` includes lots of GPIO commands which enable you to control all kinds of interfaces on Raspberry Pi.

Please run the following command to install `wiringPi` library.

```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```

You can test whether the wiringPi library is installed successfully or not by the following instruction.

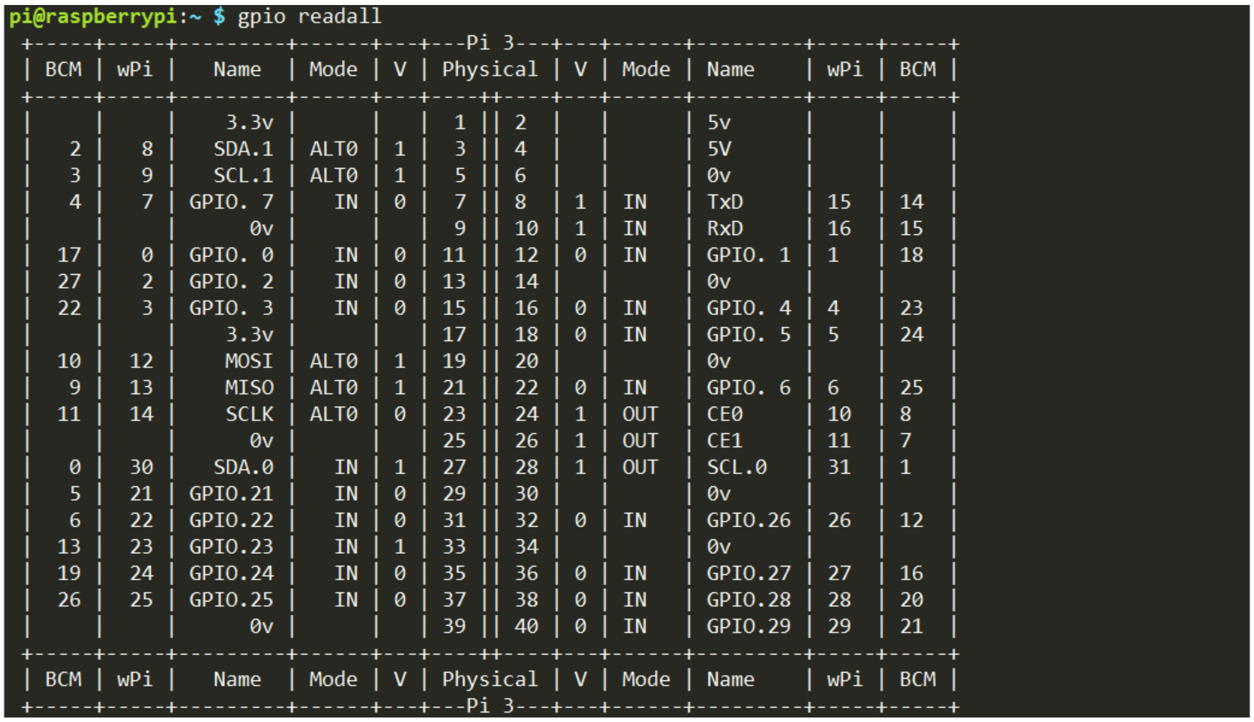
```
gpio -v
```



```
pi@raspberrypi:~/davinci-kit-for-raspberry-pi/c/1.1.1 $ 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: 2048MB, Maker: Sony
  * Device tree is enabled.
  *--> Raspberry Pi 4 Model B Rev 1.1
  * This Raspberry Pi supports user-level GPIO access.
```

Check the GPIO with the following command:

```
gpio readall
```



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

For more details about wiringPi, you can refer to WiringPi.

1.2.4 Download the Code

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

Method 1: Use Git Clone (Recommended)

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

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

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

Clone the repository from GitHub.

```
git clone https://github.com/sunfounder/electronic-kit.git
```

Method 2: Download the Code.

Download the source code from GitHub: https://github.com/sunfounder/electronic-kit.git.

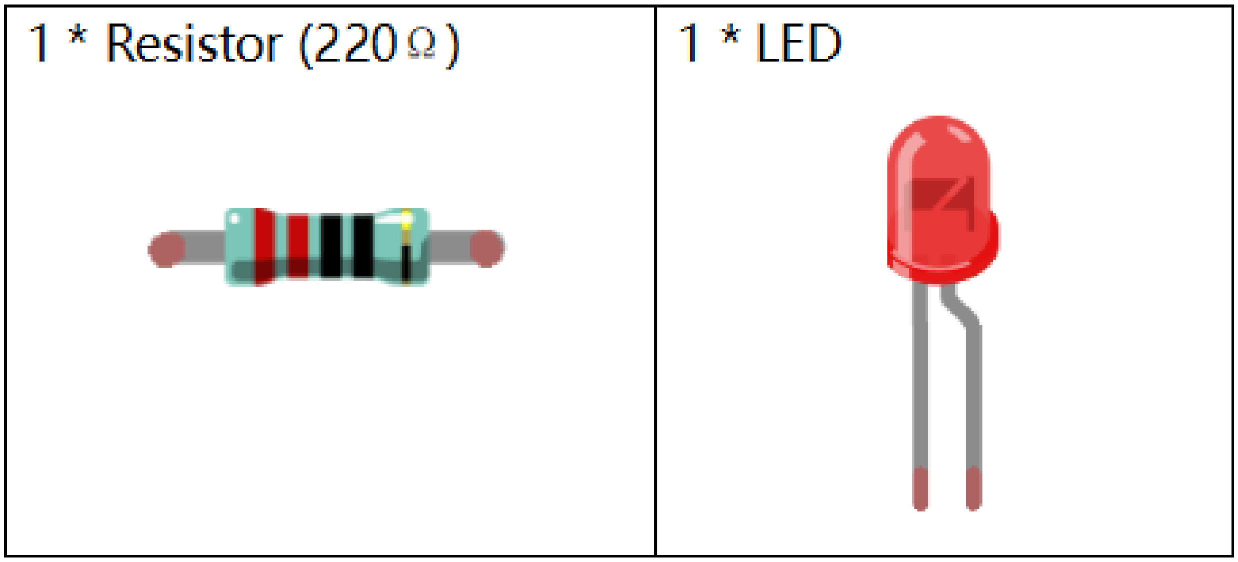
1.2.5 Lessons

Lesson 1 Blinking LED

Introduction

In this lesson, with Raspberry Pi, we will learn how to make a blinking LED by programming. By the way, you can get many interesting phenomena by applying LED. Now get to start and you will enjoy the fun of DIY at once!

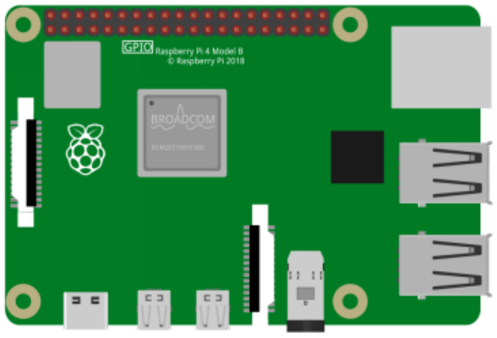
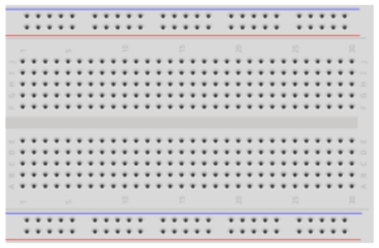
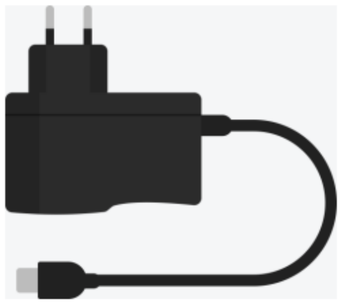
Newly Added Components



Components

Note: This table gives the necessary product components of all lessons.

In the following lessons, if there is no newly added component, the table will not appear again; instead, the list of newly added components will present for you.

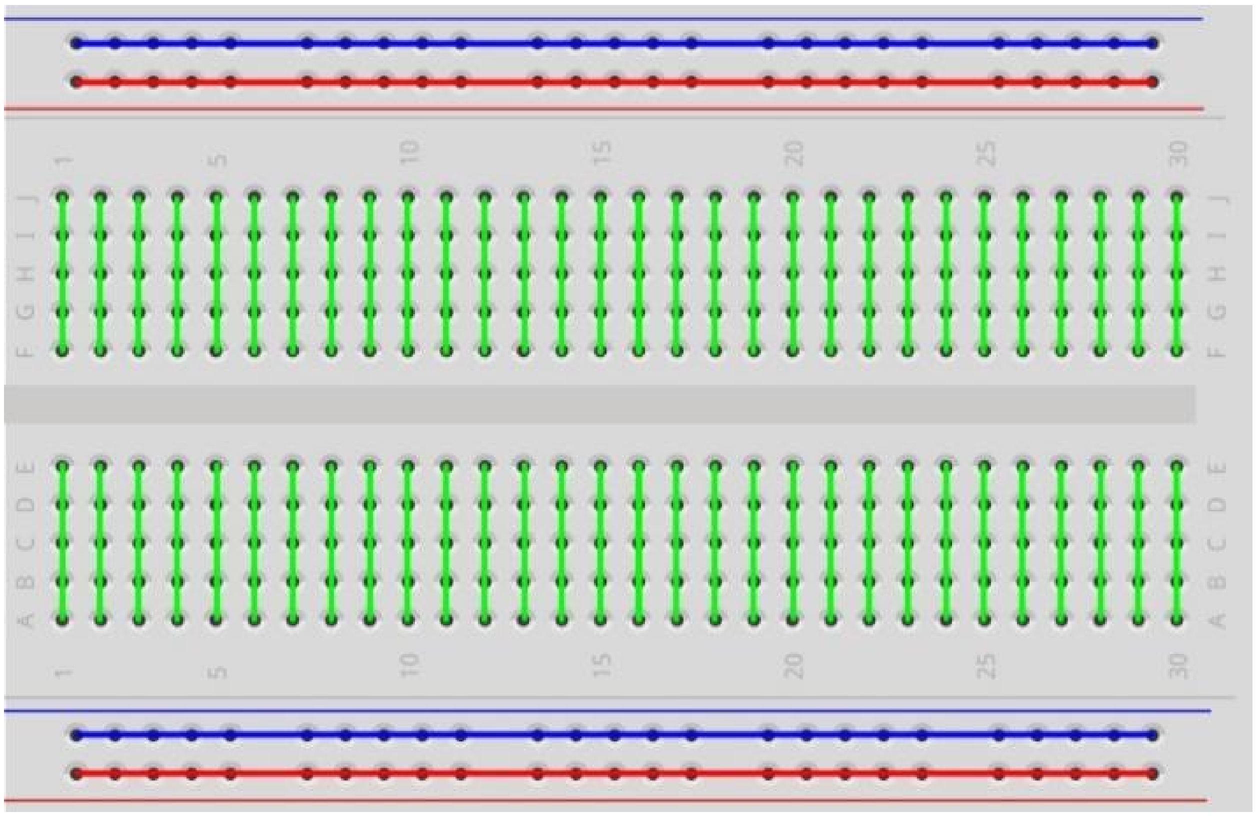
1 * Raspberry Pi 	1 * Breadboard 
1 * Power Adapter 	Several Jumper Wires
	1 * TF card

Principle

Breadboard

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

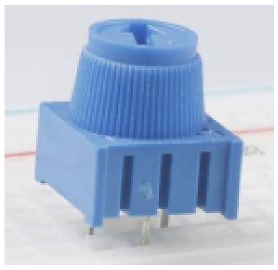
The picture shows the internal structure of a half+ breadboard. Although these holes on the breadboard appear to be independent of each other, they are actually connected to each other through metal strips internally.



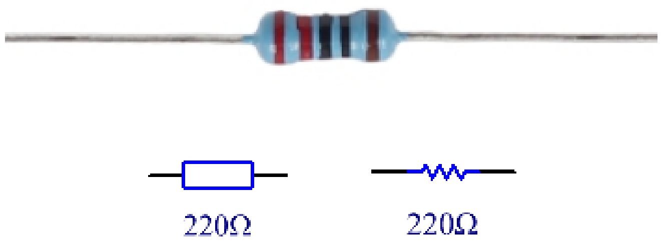
Resistor

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

Fixed resistor is applied in this kit. In the circuit, it is essential to protect the connected components. The following pictures show a real object, 220 resistor and two generally used circuit symbols of resistor. is the unit of resistance and the larger units include K, M, etc. Their relationship can be shown as follows: 1 M=1000 K, 1 K = 1000 , which means 1 M = 1000,000 = 10^6 .



Potentiometer

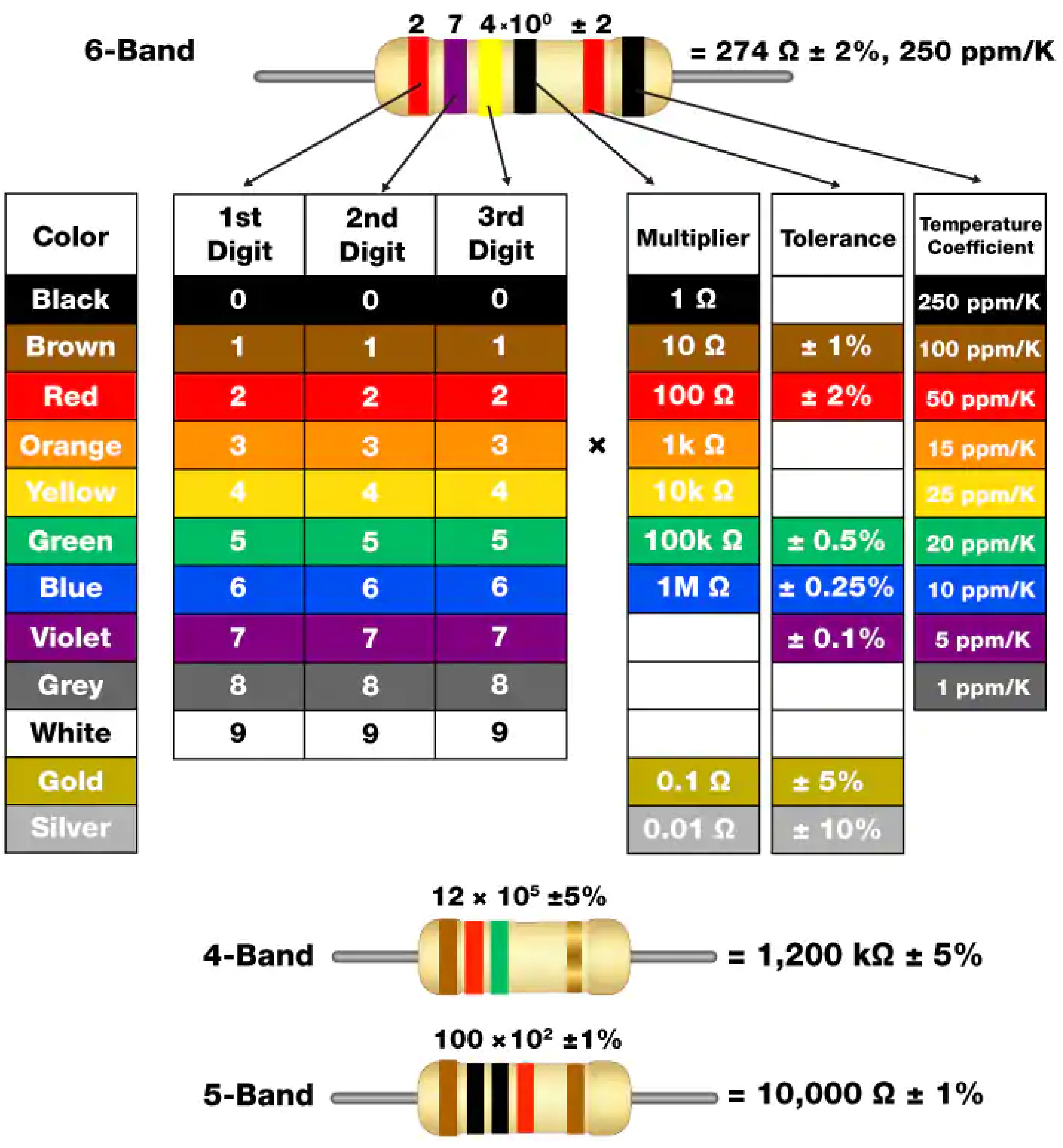


Fixed Resistor

Normally, the resistance can be marked directly, in color code, and by character. The resistors offered in this kit are marked by different colors. Namely, the bands on the resistor indicate the resistance.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster.

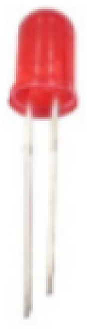
As shown in the card, each color stands for a number.

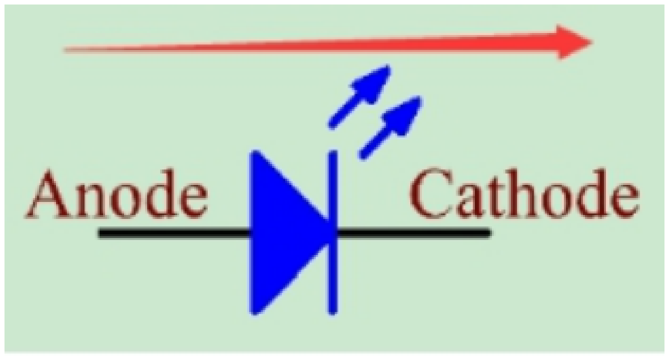


Color	1st Digit	2nd Digit	3rd Digit	Multiplier	Tolerance	Temperature Coefficient
Black	0	0	0	1 Ω		250 ppm/K
Brown	1	1	1	10 Ω	± 1%	100 ppm/K
Red	2	2	2	100 Ω	± 2%	50 ppm/K
Orange	3	3	3	1k Ω		15 ppm/K
Yellow	4	4	4	10k Ω		25 ppm/K
Green	5	5	5	100k Ω	± 0.5%	20 ppm/K
Blue	6	6	6	1M Ω	± 0.25%	10 ppm/K
Violet	7	7	7		± 0.1%	5 ppm/K
Grey	8	8	8			1 ppm/K
White	9	9	9			
Gold				0.1 Ω	± 5%	
Silver				0.01 Ω	± 10%	

LED

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





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

An LED has two pins. The longer one is anode, and the shorter one, cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$R = (V_{supply} – V_D)/I$

R stands for the resistance value of the current limiting resistor, Vsupply for voltage supply, VD for voltage drop and I for the working current of the LED.

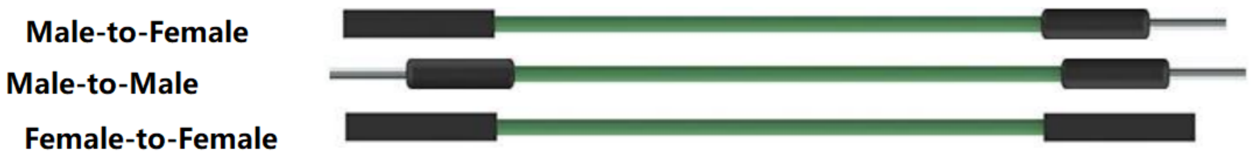
If we provide 5 Volt for the red LED, the minimum resistance of the current limiting resistor should be: (5V-1.8V)/20mA = 160. Therefore, you need a 160or larger resistor to protect the LED. You are recommended to use the 220 resistor offered in the kit.

Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Especially, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jump wires are fitted by inserting their “end connectors” into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The “end connectors” are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female.



More than one type of them may be used in a project. The color of the jump wires is different but it doesn’t mean their function is different accordingly; it’s just designed as this way to better identify the connection between each circuit.

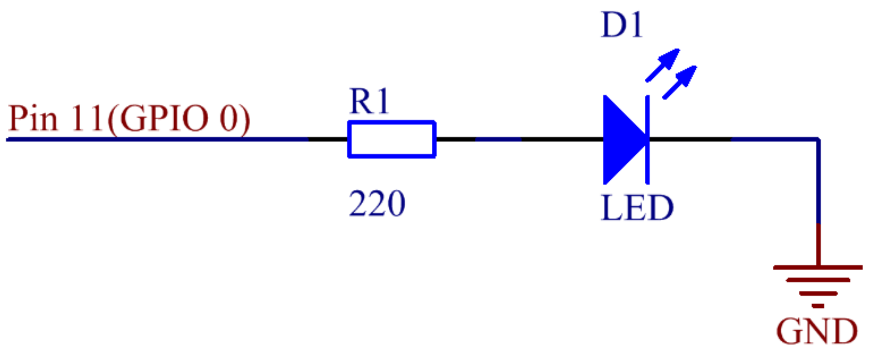
Schematic Diagram

In this experiment, connect a 220 resistor to the anode (the long pin of the LED), then the resistor to Pin11 of Raspberry Pi, and connect the cathode (the short pin) of the LED to GND. **Therefore**, to turn on a LED, we need to make pin11 high level. We can get this phenomenon by programming.

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

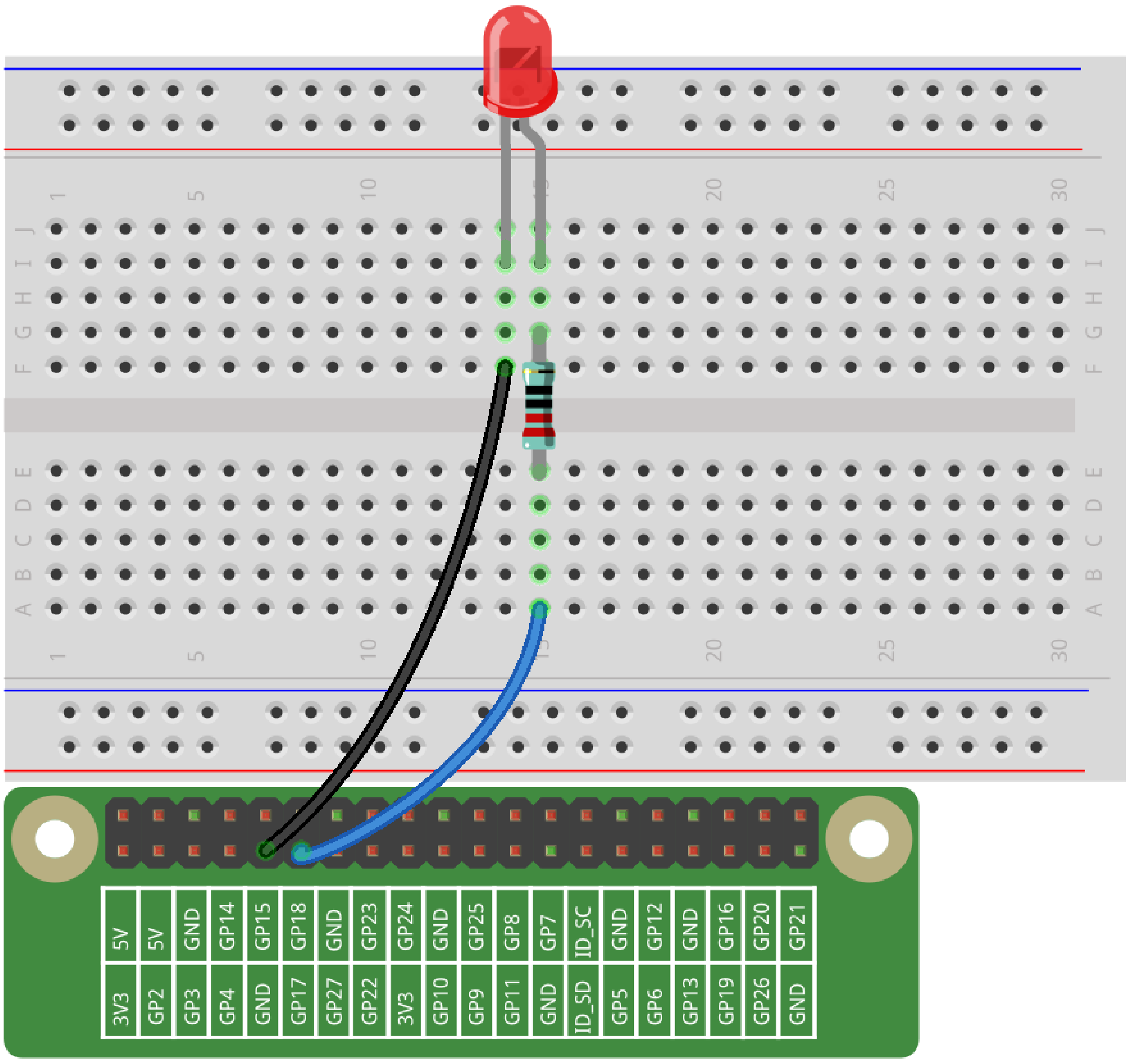
In the C language related content, we make GPIO 0 equivalent to 0 in the wiringPi. Among the Python language related content, BCM 17 is 17 in the BCM column of the following table. At the same time, they are the same as the 11th pin on the Raspberry Pi Physical, Pin 11.

wiringPi	Physical	BCM
0	Pin11	17



Build the Circuit

Note: the pin with a curve is the anode of the LED.



For C Language Users:

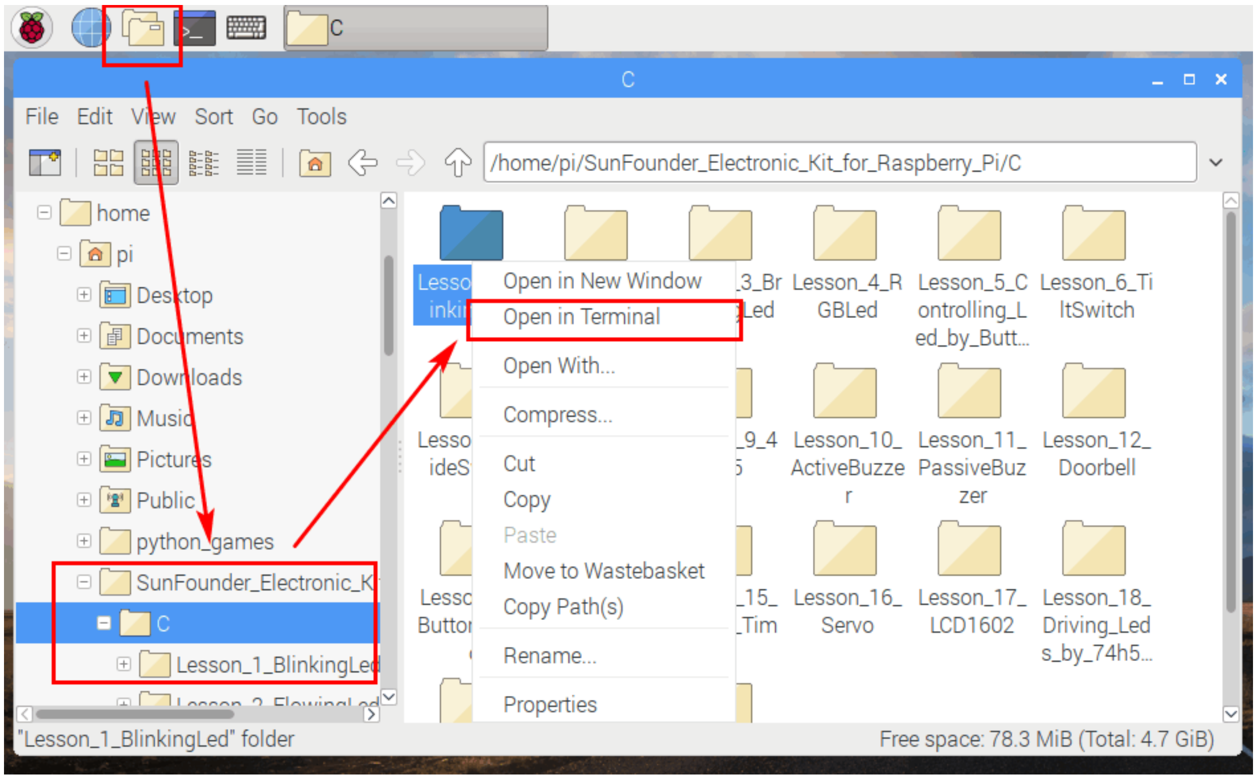
Command

1. Go to the folder of the code.

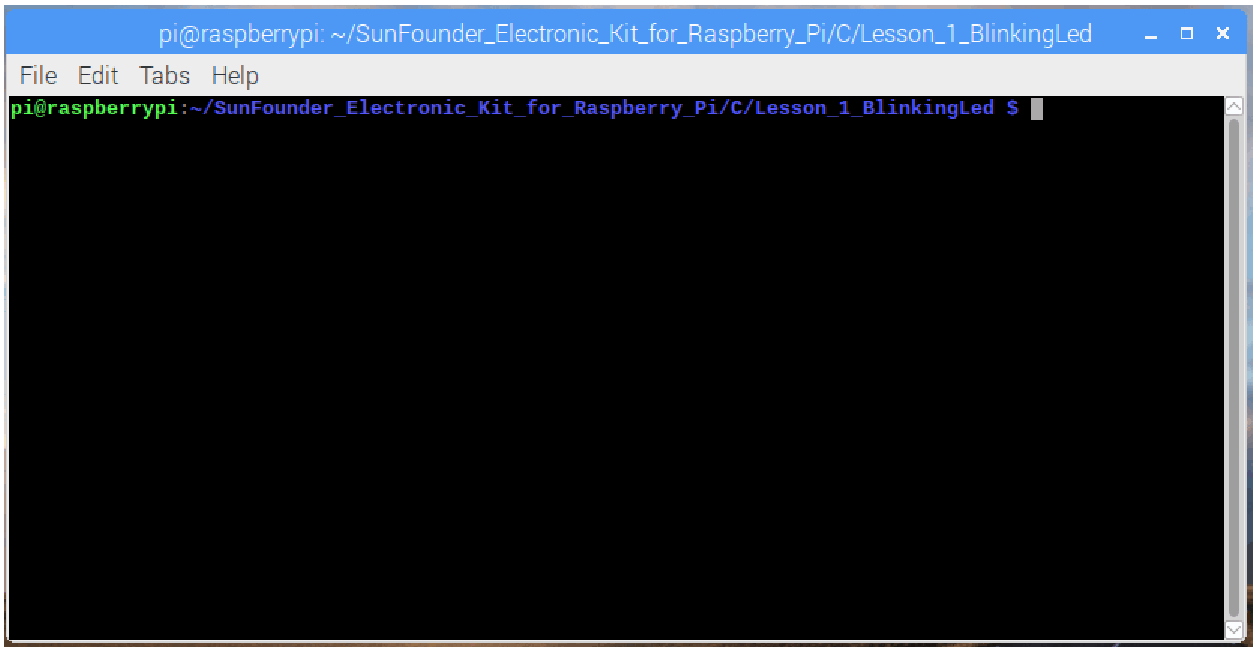
If you use a monitor, you're recommended to take the following steps.

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

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



Then a window will pop up as shown below. So now you've entered the path of the code **1_BlinkingLed.c**


pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingLed
File Edit Tabs Help

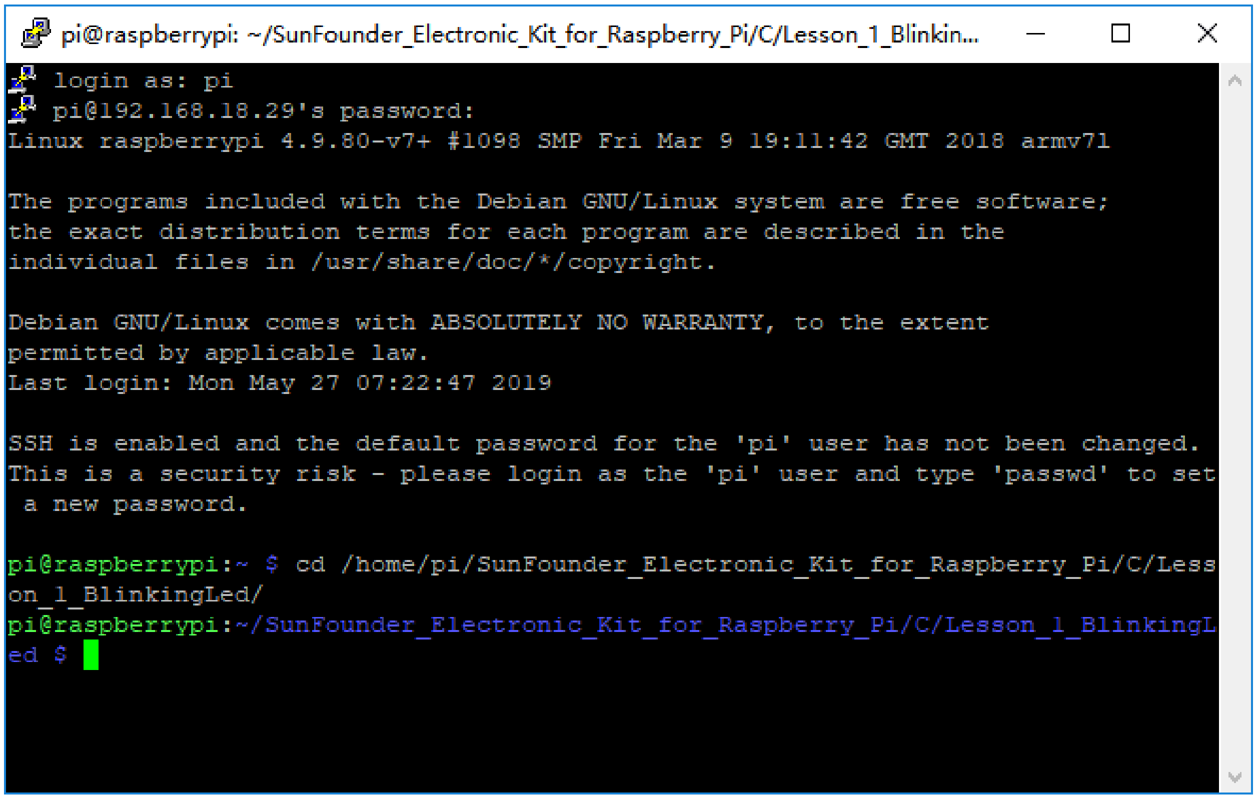
```
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingLed $
```

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

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

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_1_BlinkingLed
```

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



pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_Blinkin...

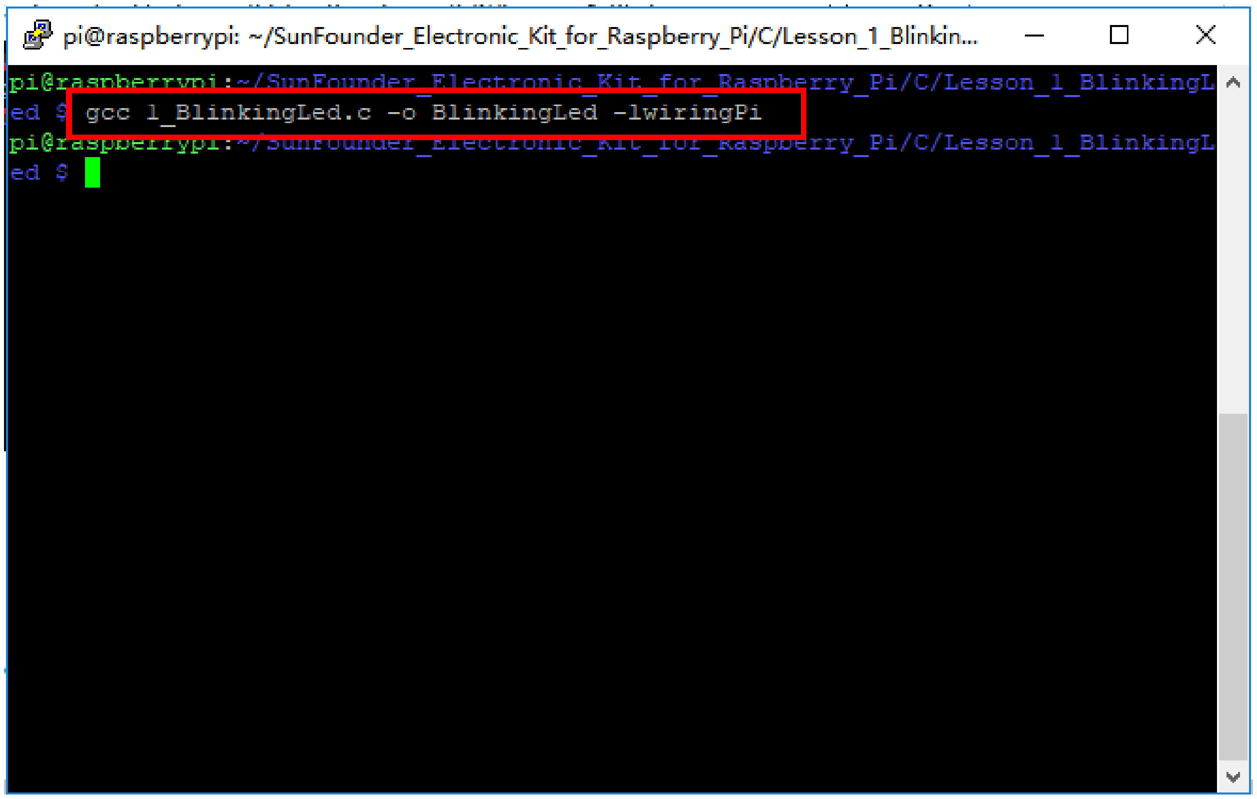
```
login as: pi
pi@192.168.18.29's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 27 07:22:47 2019

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

pi@raspberrypi:~ $ cd /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Less
on_1_BlinkingLed/
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL
ed $
```

In either way, now you are in the folder *Lesson_1_BlinkingLed*. The subsequent procedures based on these two methods are the same. Let's move on.

2. Compile the code.

```
gcc 1_BlinkingLed.c -o BlinkingLed -lwiringPi
```

Note: gcc is GNU Compiler Collection. Here, its functions like compiling the C language file *1_BlinkingLed.c* and outputting an executable file. In the command, -o means outputting (the character immediatelyfollowing -o is the filename output after compilation, and an executable named **BlinkingLed** will generate here) and -lwiringPi is to load the library wiringPi (l is the abbreviation of library).

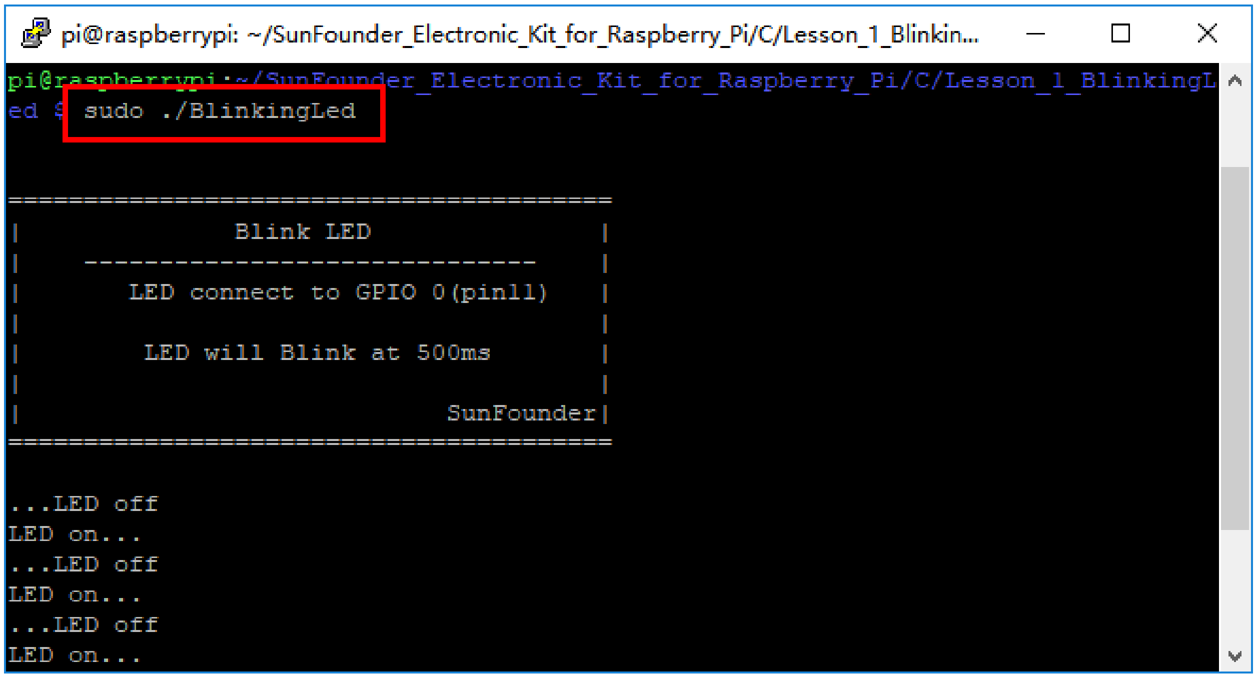


```
pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_Blinkin...
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL
ed $ gcc 1_BlinkingLed.c -o BlinkingLed -lwiringPi
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL
ed $
```

3. Run the executable file output in the previous step:

```
sudo ./BlinkingLed
```

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



```
pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_Blinkin...
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/C/Lesson_1_BlinkingL
ed $ sudo ./BlinkingLed


========================================
|              Blink LED               |
|    ------------------------------    |
|        LED connect to GPIO 0(pin11)  |
|                                      |
|         LED will Blink at 500ms      |
|                                      |
|                            SunFounder|
========================================

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

As the code runs, you will see the LED blinking. You can press **Ctrl + C** to stop running the current code.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

4. If you want to edit the code file *1_BlinkingLed.c*, type the following command to open *1_BlinkingLed.c* .

```
nano 1_BlinkingLed.c
```

Note: nano is a text editor tool. The command is used to open the code file **1_BlinkingLed.c** by this tool.

Code

The program code is shown as follows:

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin      0

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

    pinMode(LedPin, OUTPUT);
```

(continues on next page)

(continued from previous page)

```
    while(1){
        // LED off
        digitalWrite(LedPin, LOW);
        printf("...LED off\n");
        delay(500);
        // LED on
        digitalWrite(LedPin, HIGH);
        printf("LED on...\n");
        delay(500);
    }

    return 0;
}
```

Code Explanation

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

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

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

Standard I/O library. The printf function used for printing the data displayed on the screen is realized by this library. There are many other performance functions for you to explore.

```
#define LedPin      0
```

Assign GPIO 0 to LedPin that represents GPIO 0 in the code later.

```
    if (wiringPiSetup() == -1){

        printf("setup wiringPi failed !");

        return 1;
```

This initializes wiringPi library and assumes that the calling program is going to be using the wiringPi pin numbering scheme. This function needs to be called with root privileges. When initialize wiring failed, print message to screen.

```
    pinMode(LedPin, OUTPUT);
```

This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports hardware PWM output, you can also set other pins to PWM output using the softPWM library. Only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes. Here we set LedPin as OUTPUT mode to write value to it.

```
        digitalWrite(LedPin, LOW);
```

Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as OUTPUT. On Raspberry Pi, when the output voltage is less than 0.4V, by default, it is low level, LOW, and when the voltage is greater than 2.4V, it is high level, HIGH. Since the anode of LED is connected to GPIO 0, thus the LED will light up if GPIO 0 is set high. On the contrary, set GPIO 0 as low level, digitalWrite (LedPin, LOW), LED will go out.

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

The **printf** function is a standard library function and its function prototype is in the header file "stdio.h". The general form of the call is: printf("format control string", output table columns). The format control string is used to specify the output format, which is divided into format string and non-format string. The format string starts with "%" followed by format characters such as "%d"for decimal integer output. Non-format strings are printed as prototypes. What is used here is a non-format string, followed by "n" that is a newline character, representing automatic line wrapping after printing a string.

```
        delay(500);
```

This is a function that suspends the program for a period of time. And the speed of the program is determined by our hardware. Here we turn on or off the LED. If there is no **delay** function, the program will run the whole program very fast and continuously loop and we can hardly observe the phenomenon. So we need the **delay** function to help us write and debug the program. **delay (500)** keeps the current HIGH or LOW state for 500ms(0.5s).

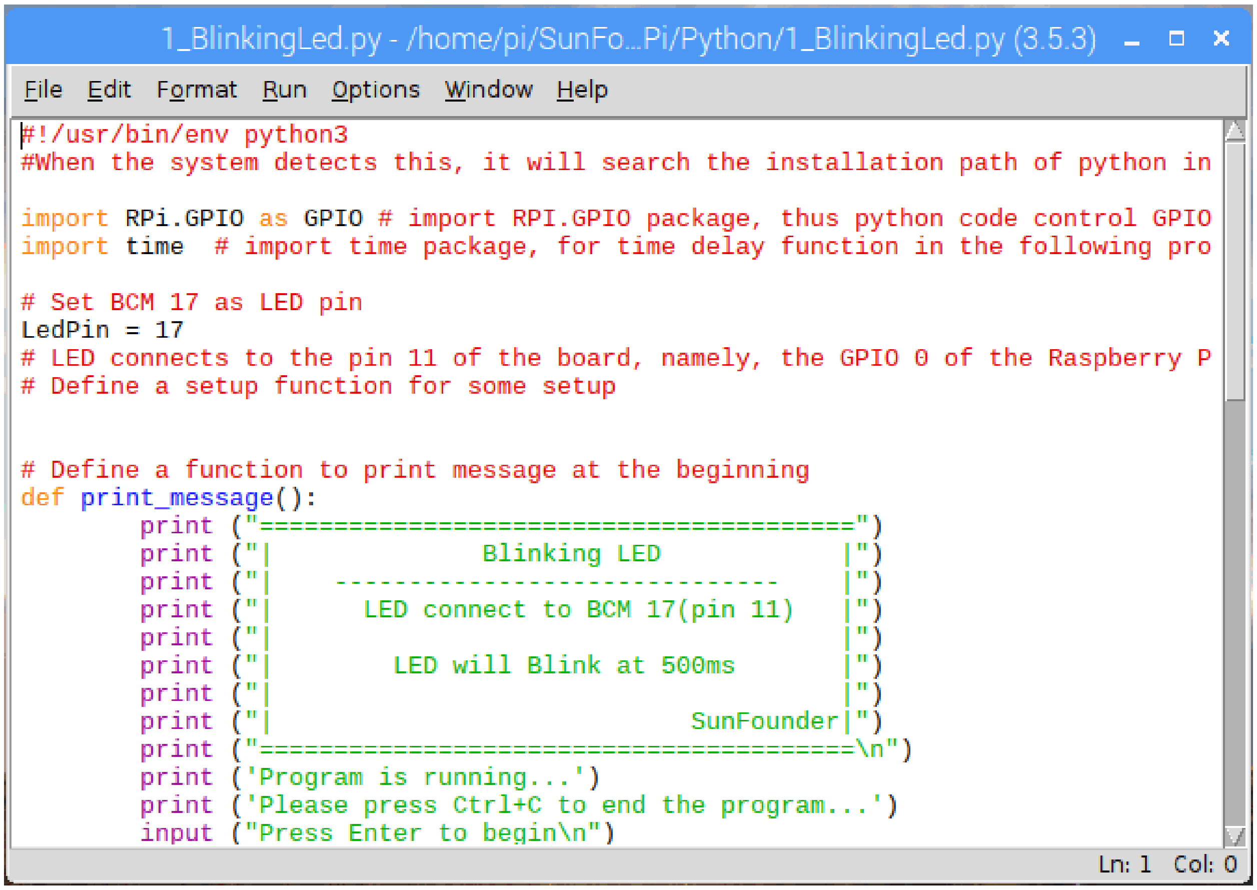
```
    return 0;
```

Usually, it is placed in the last position of the **main** function, indicating that the function returns 0 after executing the function.

For Python Language Users

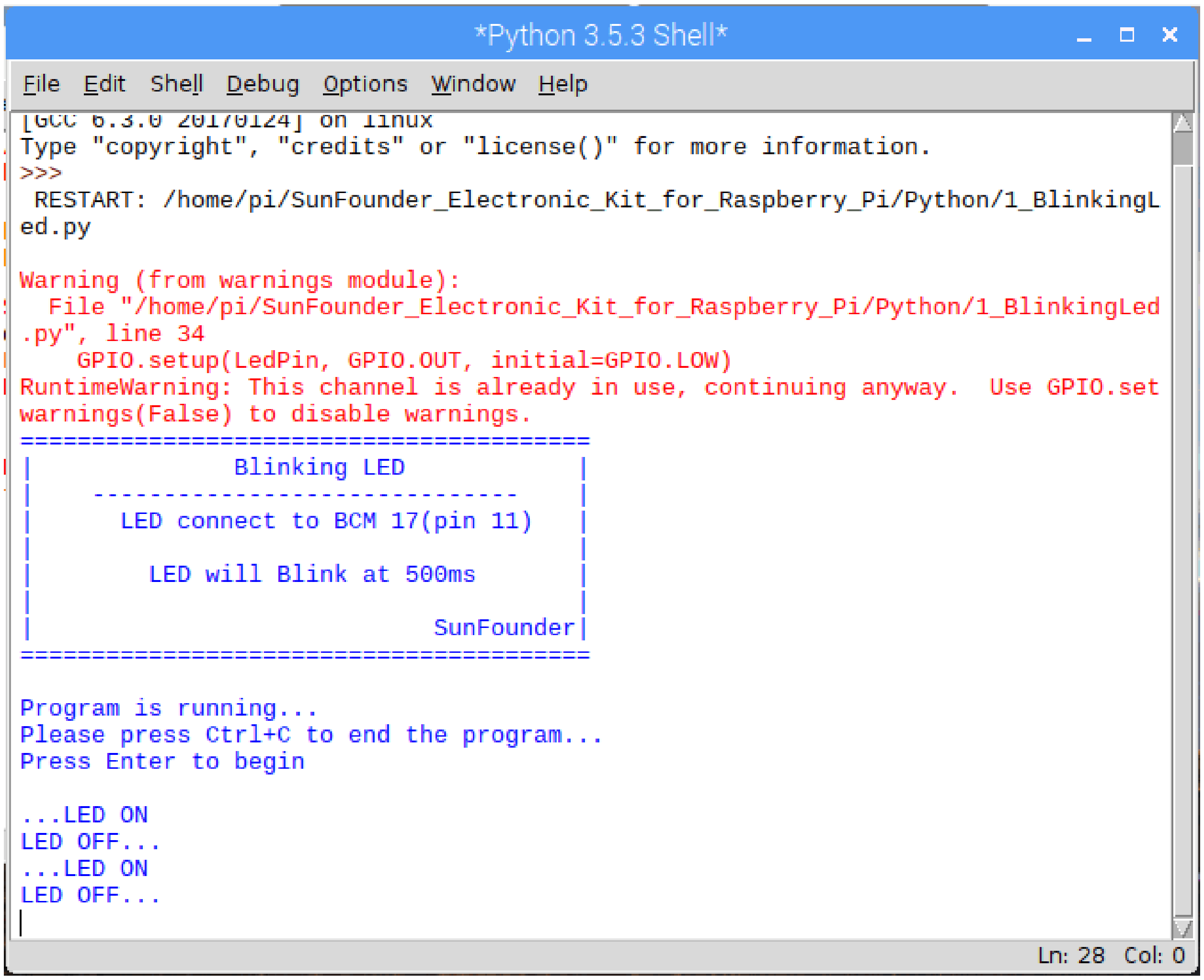
If you use a monitor, you're recommended to take the following steps.

Find **1_BlinkingLed.py** and double click it to open the file.


1_BlinkingLed.py - /home/pi/SunFo...Pi/Python/1_BlinkingLed.py (3.5.3)

```
#!/usr/bin/env python3
#When the system detects this, it will search the installation path of python in

import RPi.GPIO as GPIO # import RPI.GPIO package, thus python code control GPIO
import time  # import time package, for time delay function in the following pro

# Set BCM 17 as LED pin
LedPin = 17
# LED connects to the pin 11 of the board, namely, the GPIO 0 of the Raspberry P
# Define a setup function for some setup


# Define a function to print message at the beginning
def print_message():
        print ("========================================")
        print ("|              Blinking LED            |")
        print ("|    ------------------------------    |")
        print ("|      LED connect to BCM 17(pin 11)   |")
        print ("|                                      |")
        print ("|        LED will Blink at 500ms       |")
        print ("|                                      |")
        print ("|                            SunFounder|")
        print ("========================================\n")
        print ('Program is running...')
        print ('Please press Ctrl+C to end the program...')
        input ("Press Enter to begin\n")
```

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



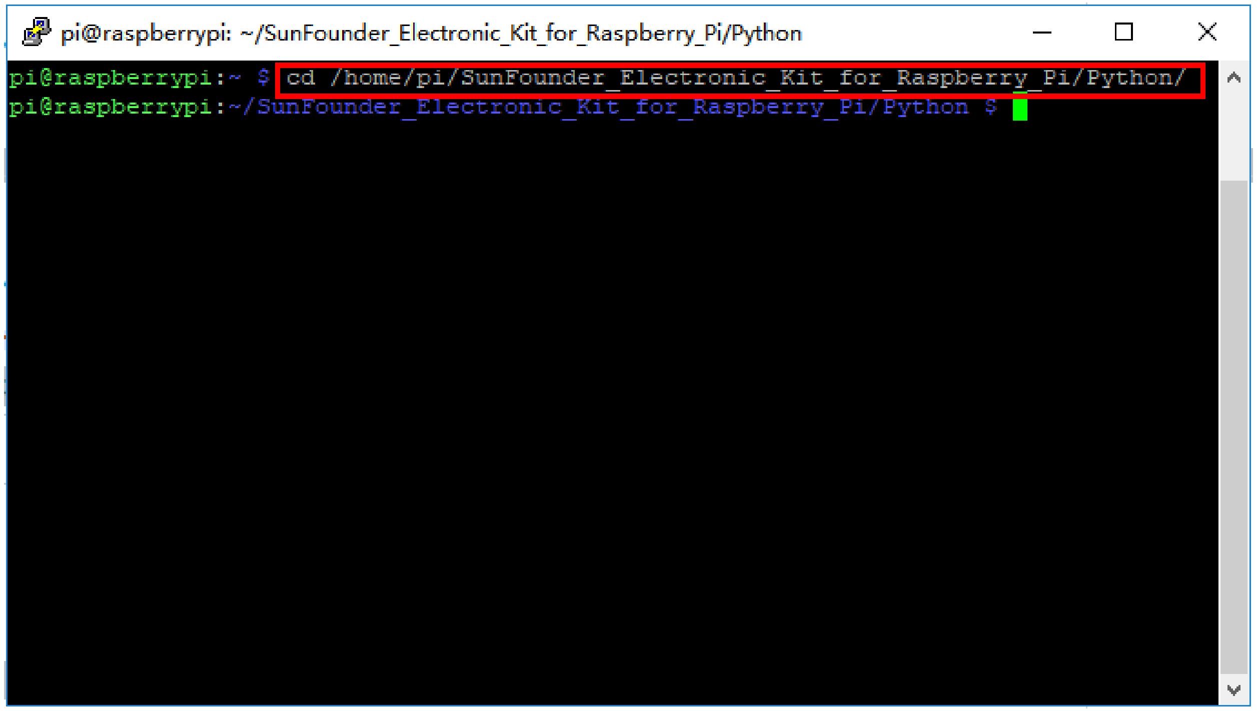
```
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/1_BlinkingL
ed.py

Warning (from warnings module):
  File "/home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/1_BlinkingLed
.py", line 34
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
RuntimeWarning: This channel is already in use, continuing anyway.  Use GPIO.set
warnings(False) to disable warnings.
========================================
|              Blinking LED            |
|     ------------------------------   |
|       LED connect to BCM 17(pin 11)  |
|                                      |
|         LED will Blink at 500ms      |
|                                      |
|                            SunFounder|
========================================

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

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

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

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

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

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

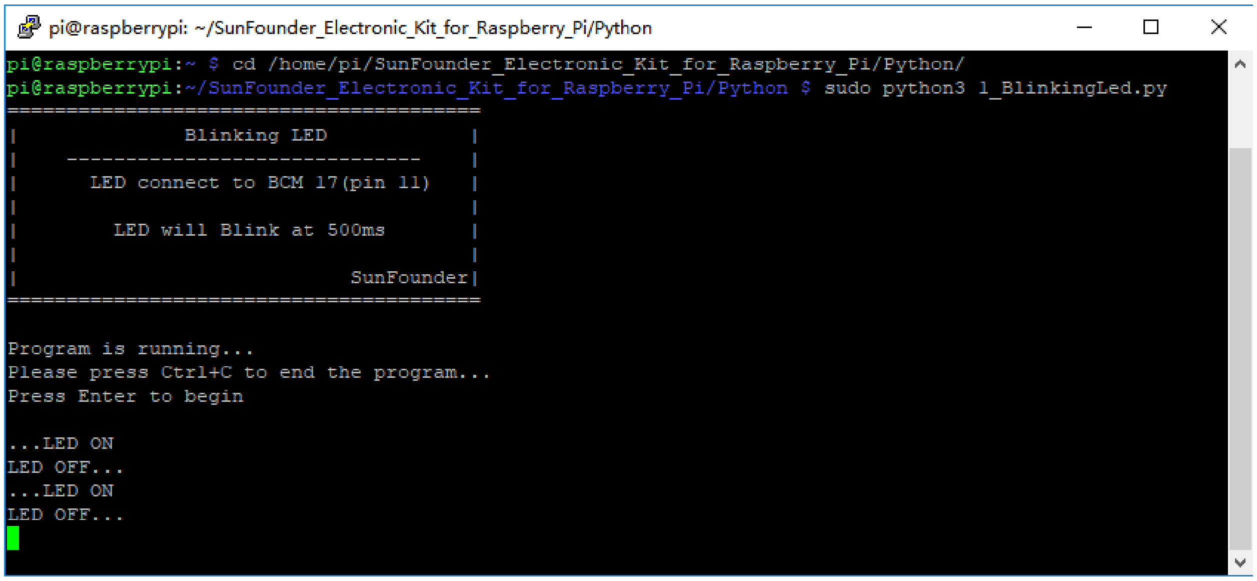

pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python

```
pi@raspberrypi:~ $ cd /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python $
```

2. Run the code.

```
sudo python3 1_BlinkingLed.py
```

Note: Here, sudo means superuser do, and the command python3 means to run the file by the programming language, Python 3.0.


pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python

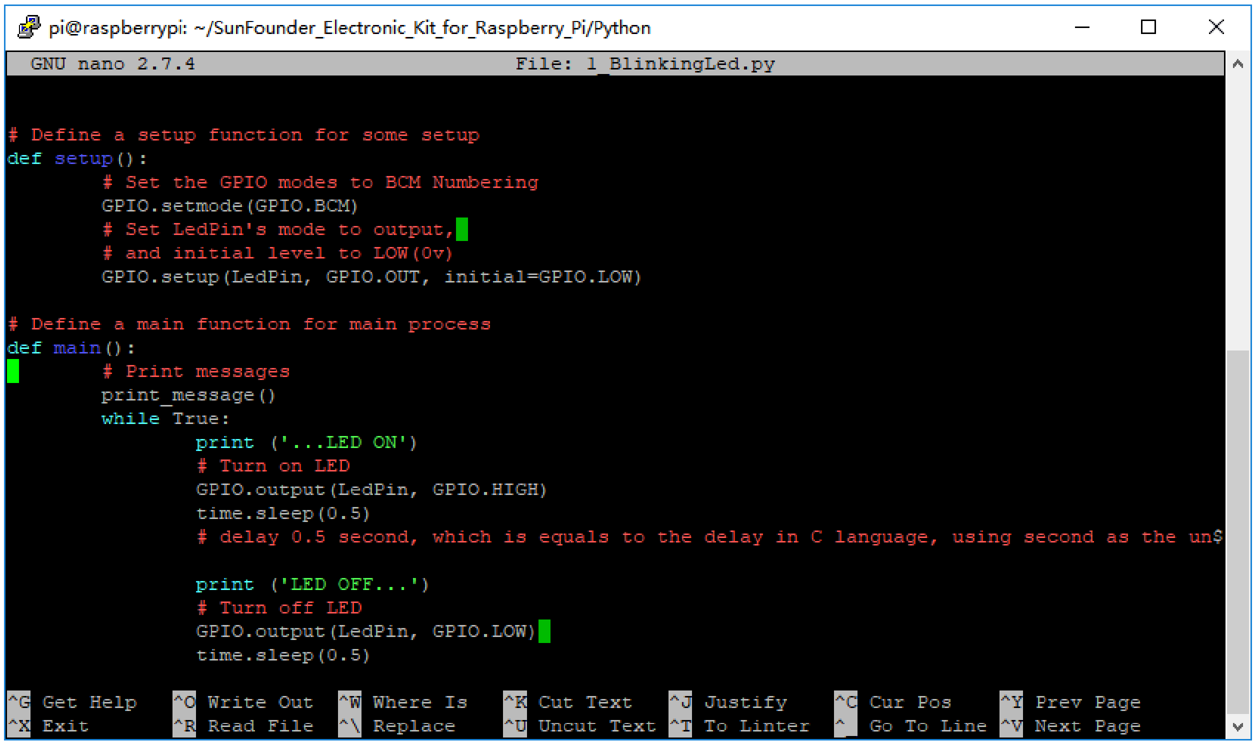
```
pi@raspberrypi:~ $ cd /home/pi/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python/
pi@raspberrypi:~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python $ sudo python3 1_BlinkingLed.py
========================================
|              Blinking LED            |
|    ------------------------------    |
|      LED connect to BCM 17(pin 11)   |
|                                      |
|        LED will Blink at 500ms       |
|                                      |
|                            SunFounder|
========================================

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

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

As the code runs, you will see the LED blinking. You can press **Ctrl+C** to stop running the current code.

3. If you want to edit the code file **1_BlinkingLed.c**, type the following command to open **1_BlinkingLed.c**

```
nano 1_BlinkingLed.py
```

Note: nano is a text editor tool. The command is used to open thecode file **1_BlinkingLed.c** by this tool.


pi@raspberrypi: ~/SunFounder_Electronic_Kit_for_Raspberry_Pi/Python

```
  GNU nano 2.7.4                           File: 1_BlinkingLed.py

# Define a setup function for some setup
def setup():
        # Set the GPIO modes to BCM Numbering
        GPIO.setmode(GPIO.BCM)
        # Set LedPin's mode to output,
        # and initial level to LOW(0v)
        GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
        # Print messages
        print_message()
        while True:
                print ('...LED ON')
                # Turn on LED
                GPIO.output(LedPin, GPIO.HIGH)
                time.sleep(0.5)
                # delay 0.5 second, which is equals to the delay in C language, using second as the un$

                print ('LED OFF...')
                # Turn off LED
                GPIO.output(LedPin, GPIO.LOW)
                time.sleep(0.5)

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    ^Y Prev Page
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Linter  ^_ Go To Line ^V Next Page
```

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

# Set BCM 17 as LED pin
LedPin = 17

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
    while True:
```

(continues on next page)

(continued from previous page)

```
        print ('...LED ON')
        # Turn on LED
        GPIO.output(LedPin, GPIO.HIGH)
        time.sleep(0.5)
        print ('LED OFF...')
        # Turn off LED
        GPIO.output(LedPin, GPIO.LOW)
        time.sleep(0.5)

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

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

Code Explanation

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

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

```
2.import time
```

Import time library to help use delay function in the following program.

```
5.LedPin = 17
```

LED connects to the pin 11 of the board, namely, the BCM 17 of the Raspberry Pi.

```
9.    GPIO.setmode(GPIO.BCM)
```

There are two ways of numbering the I/O pins on a Raspberry Pi within RPi.GPIO: BOARD numbers and BCM numbers. In our lessons, what we use is BCM numbering method.

```
10.    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
```

You need to set up every channel you use as input mode or output mode. Here we set the mode of LedPin to GPIO.OUT, and initial level to LOW(0v).

```
17.        GPIO.output(LedPin, GPIO.HIGH)
```

Set LedPin to output high level to light up LED.

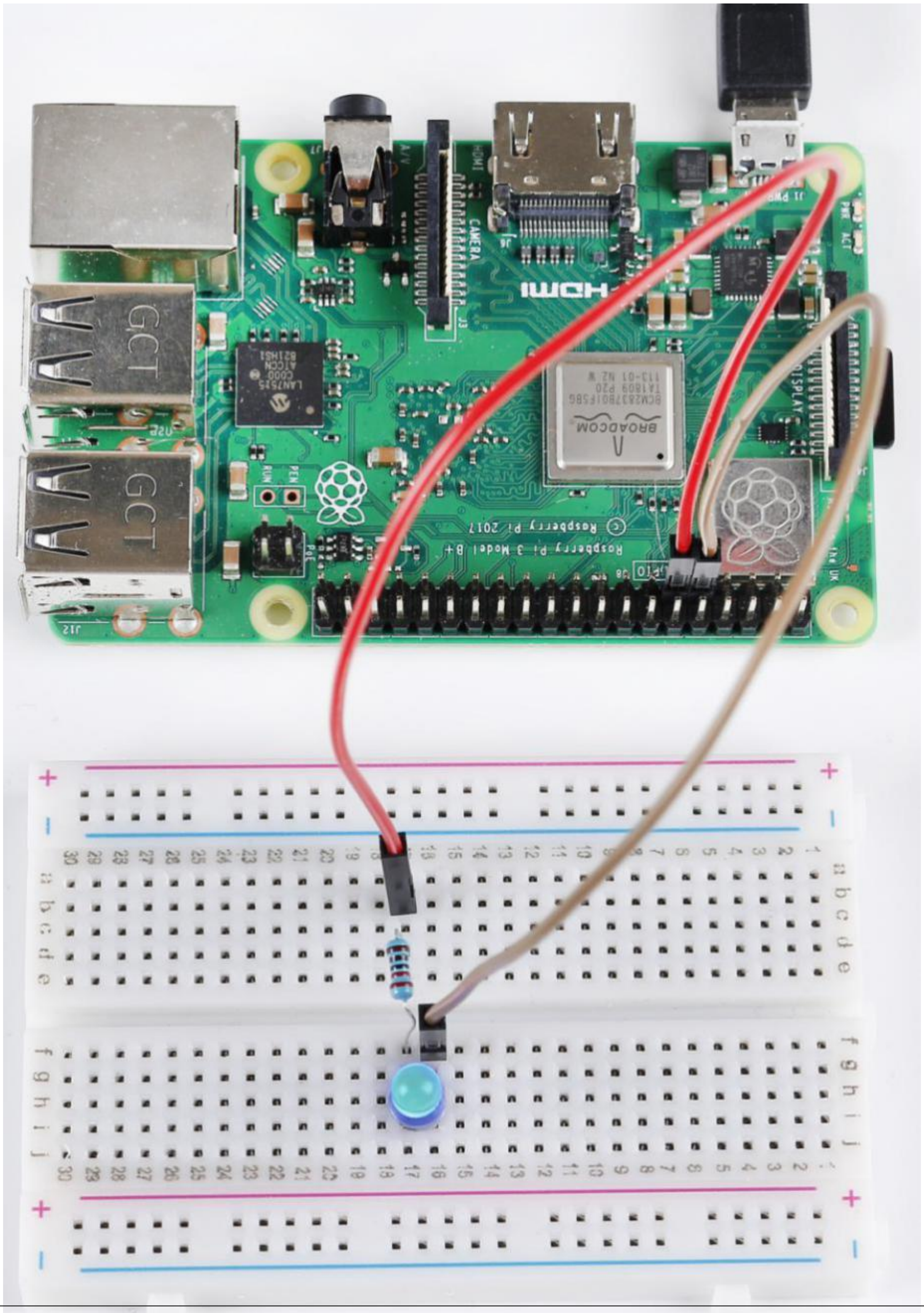
```
18.        time.sleep(0.5)
```

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

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

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

Phenomenon Picture

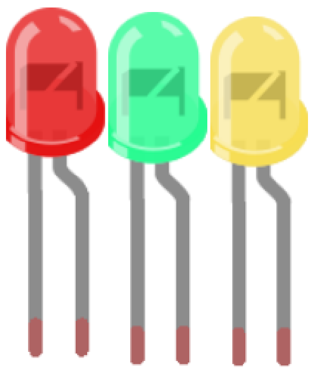
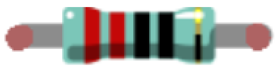


Lesson 2 Flowing LED Lights

Introduction

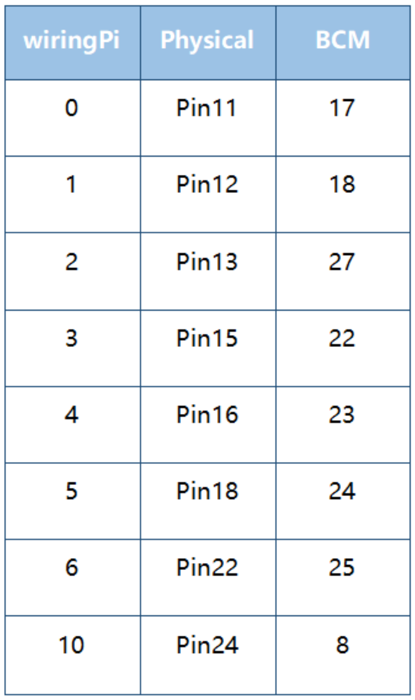
In this lesson, we will learn how to make eight LEDs blink as flowing water based on Raspberry Pi.

Newly Added Components

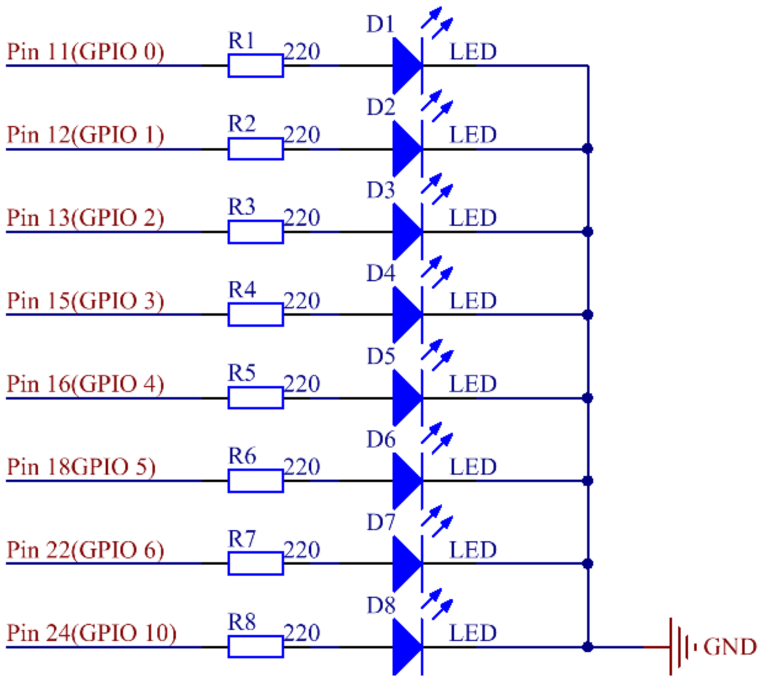
8 * LED	8 * Resistor (220Ω)
	

Schematic Diagram

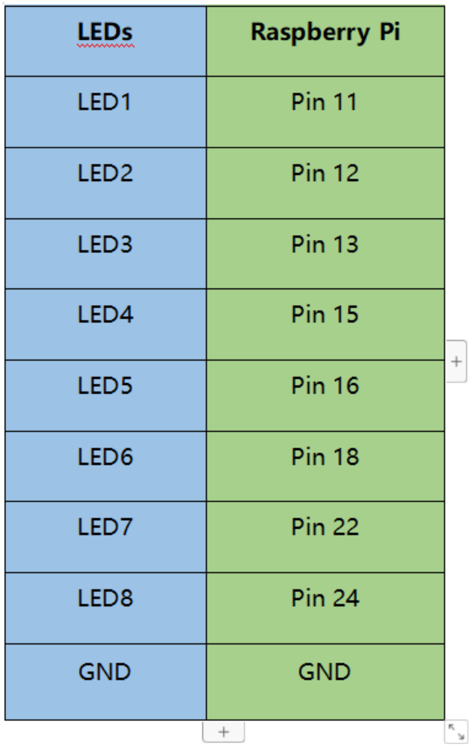
In this experiment, connect **220**resistors to the anode (the longer pin of the LED) respectively, then the resistors to Pin **11, 12, 13, 15, 16, 18, 22** and **24** of Raspberry Pi, and connect the cathode (the short pin) of the LEDs to **GND**. We can see from the schematic diagram that the anode of LED connect to a current-limiting resistor and then to Raspberry Pi. Therefore, to turn on an LED, we need to make pins high level. This process can be realized by programming.



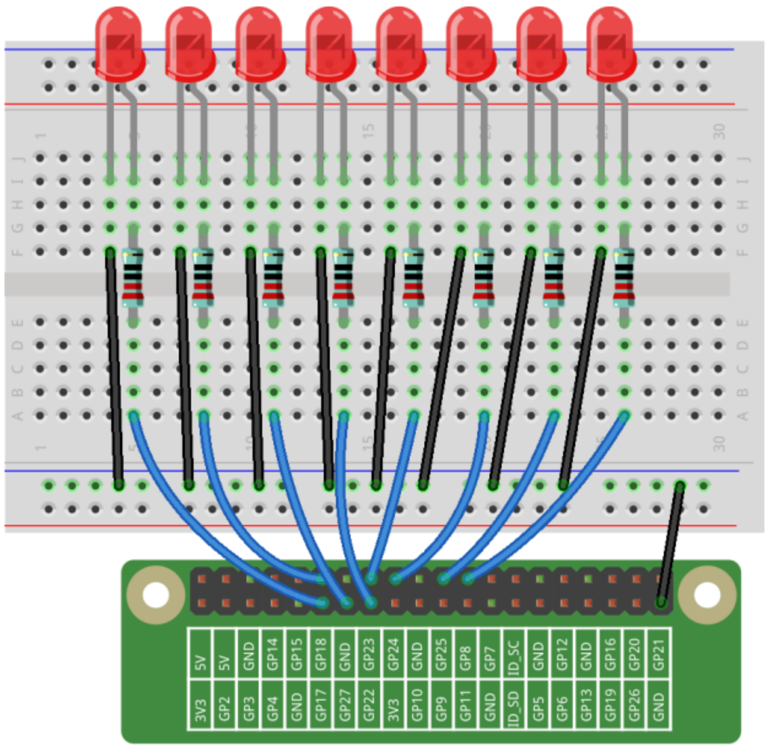
wiringPi	Physical	BCM
0	Pin11	17
1	Pin12	18
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24
6	Pin22	25
10	Pin24	8



Build the Circuit



LEDs	Raspberry Pi
LED1	Pin 11
LED2	Pin 12
LED3	Pin 13
LED4	Pin 15
LED5	Pin 16
LED6	Pin 18
LED7	Pin 22
LED8	Pin 24
GND	GND



For C Language Users:

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_2_FlowingLedLights
```

2. Compile the code.

```
gcc 2_FlowingLedLights.c -lwiringPi
```

Note: When using the gcc command, if you do not use -o, it will automatically output as a.out.

3. Run the executable file.

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

Now, you will see these 8 LEDs are lit one by one from left to right, and then one by one from right to left.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

const int LedPin[]={0,1,2,3,4,5,6,10};   //Define 8 LED pin

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

    for(int j=0;j<8;j++)
    {
        pinMode(LedPin[j], OUTPUT);// Set LedPin as output to write value to it.
        digitalWrite(LedPin[j], LOW);
    }

    while(1){
        for(int i=0;i<8;i++)
        {
            // LED on
            digitalWrite(LedPin[i], HIGH);
            delay(100);
        }
```

(continues on next page)

(continued from previous page)

```
        for(int i=7;i>-1;i--)
        {
            // LED off
            digitalWrite(LedPin[i], LOW);
            delay(100);
        }
    }

    return 0;
}
```

Code Explanation

```
const int LedPin[]={0,1,2,3,4,5,6,10};
```

Create an array, **LedPin** to define the eight LEDs then connect them to **GPIO0 ~ GPIO6**, **GPIO10** respectively.

```
    for(int j=0;j<8;j++)
    {
        pinMode(LedPin[j], OUTPUT);
        digitalWrite(LedPin[j], LOW);
    }
```

Use a **for** loop to set all 8 pins connected to LEDs to **OUTPUT** mode and **LOW** level.

```
        for(int i=0;i<8;i++)
        {
            // LED on
            digitalWrite(LedPin[i], HIGH);
            delay(100);
        }
```

Light up the LEDs in GPIO0~6 and GPIO10 successively. i increases progressively from **0** to **7**, LED0 to LED7 changes accordingly, making it like a flowing LED light from left to right.

```
        for(int i=7;i>-1;i--)
        {
            // LED off
            digitalWrite(LedPin[i], LOW);
            delay(100);
        }
```

Close the LEDs in GPIO0~6 and GPIO10 successively. i increases progressively from **7** to **0**, LED0 to LED7 changes accordingly, making it like a flowing LED light from right to left.

For Python Language Users

Command

1. Go to the folder of the code

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

2. Run the code.

```
sudo python3 2_FlowingLed.py
```

Now, you will see these 8 LEDs are lit one by one from left to right, and then one by one from right to left.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

pins = [17,18,27,22,23,24,25,8]

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    for i in range(0, 8, 1):
        GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
    while True:
        # print ('...LED ON')
        # Turn on LED
        for i in range(0, 8, 1):
            GPIO.output(pins[i], GPIO.HIGH)
            time.sleep(0.1)

        # print ('LED OFF...')
        # Turn off LED
        for i in range(7, -1, -1):
            GPIO.output(pins[i], GPIO.LOW)
            time.sleep(0.1)

# Define a destroy function for clean up everything after the script finished
def destroy():
    # Turn off LED
    for i in range(0, 8, 1):
            GPIO.output(pins[i], GPIO.LOW)
    # Release resource
    GPIO.cleanup()
```

(continues on next page)

(continued from previous page)

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

Code Explanation

```python
for i in range(0, 8, 1):
    GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)
```

Use a **for** loop to set all 8 pins connected to LEDs to output mode and LOW level.

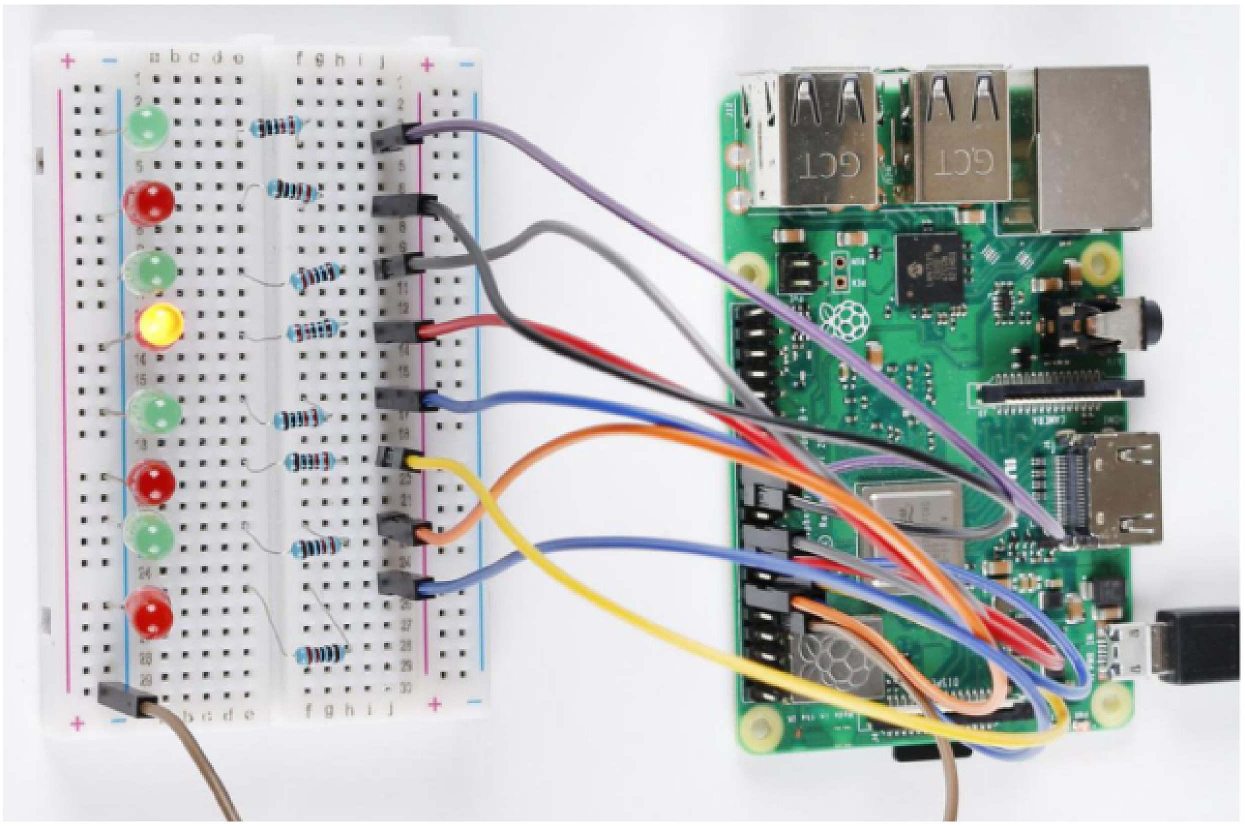
```python
for i in range(0, 8, 1):
    GPIO.output(pins[i], GPIO.HIGH)
                 time.sleep(0.1)
```

Variable **i** increases progressively from **0** to **8**, increasing by 1 every time. Accordingly, set the pins in the array **pins[i]** to **HIGH** respectively to light up the LEDs and the lighting time is **0.1**s. Then, you will see 8 LEDs light up one by one.

```python
for i in range(7, -1, -1):
    GPIO.output(pins[i], GPIO.LOW)
    time.sleep(0.1)
```

Variable **i** decreases progressively from **7** to **-1**, decreasing by 1 every time. Then LED0~LED7 change accordingly, making it like a flowing LED light from right to left.

Phenomenon Picture

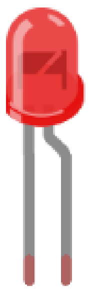


Lesson 3 Breathing LED

Introduction

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

Newly Added Components

1 * Resistor (220 Ω)	1 * LED
	

Principle

PWM

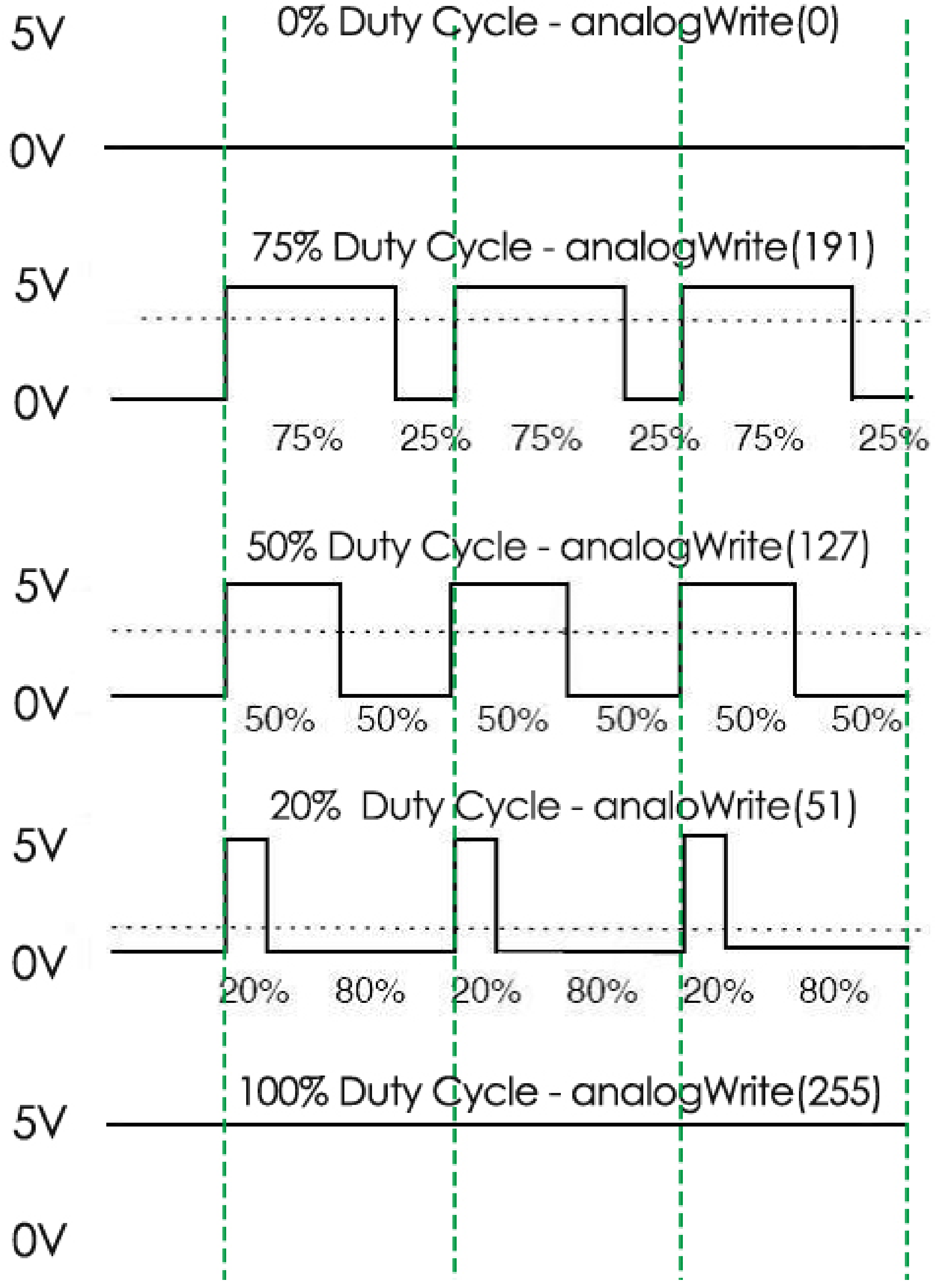
Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you can change or modulate this width. If you repeat this on-off pattern fast enough with some device, an LED for example, the result would be like this: the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

Duty Cycle

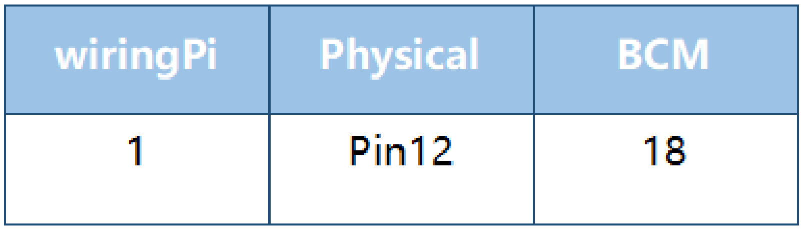
A duty cycle is the percentage of one period in which a signal is active. A period is the time it takes for a signal to complete an on-and-off cycle. As a formula, a duty cycle may be expressed as:

D=T/Px100%

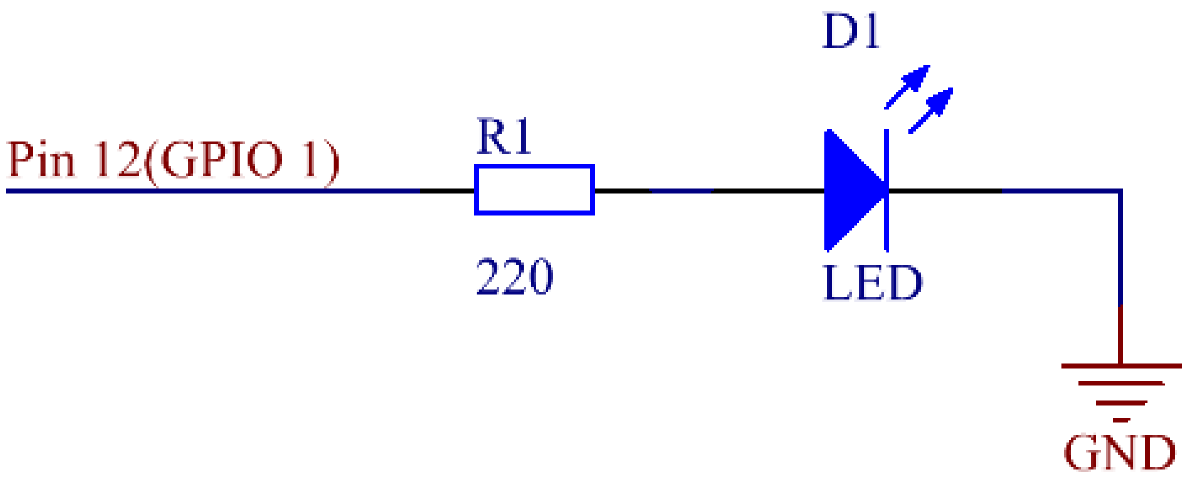
Where is the duty cycle, is the time the signal is active, and is the total period of the signal. Thus, a 60% duty cycle means the signal is on 60% of the time but off 40% of the time. The "on time" for a 60% duty cycle could be a fraction of a second, a day, or even a week, depending on the length of the period.


0% Duty Cycle - analogWrite(0)
5V
0V
75% Duty Cycle - analogWrite(191)
5V
0V
75% 25% 75% 25% 75% 25%
50% Duty Cycle - analogWrite(127)
5V
0V
50% 50% 50% 50% 50% 50%
20% Duty Cycle - analoWrite(51)
5V
0V
20% 80% 20% 80% 20% 80%
100% Duty Cycle - analogWrite(255)
5V
0V

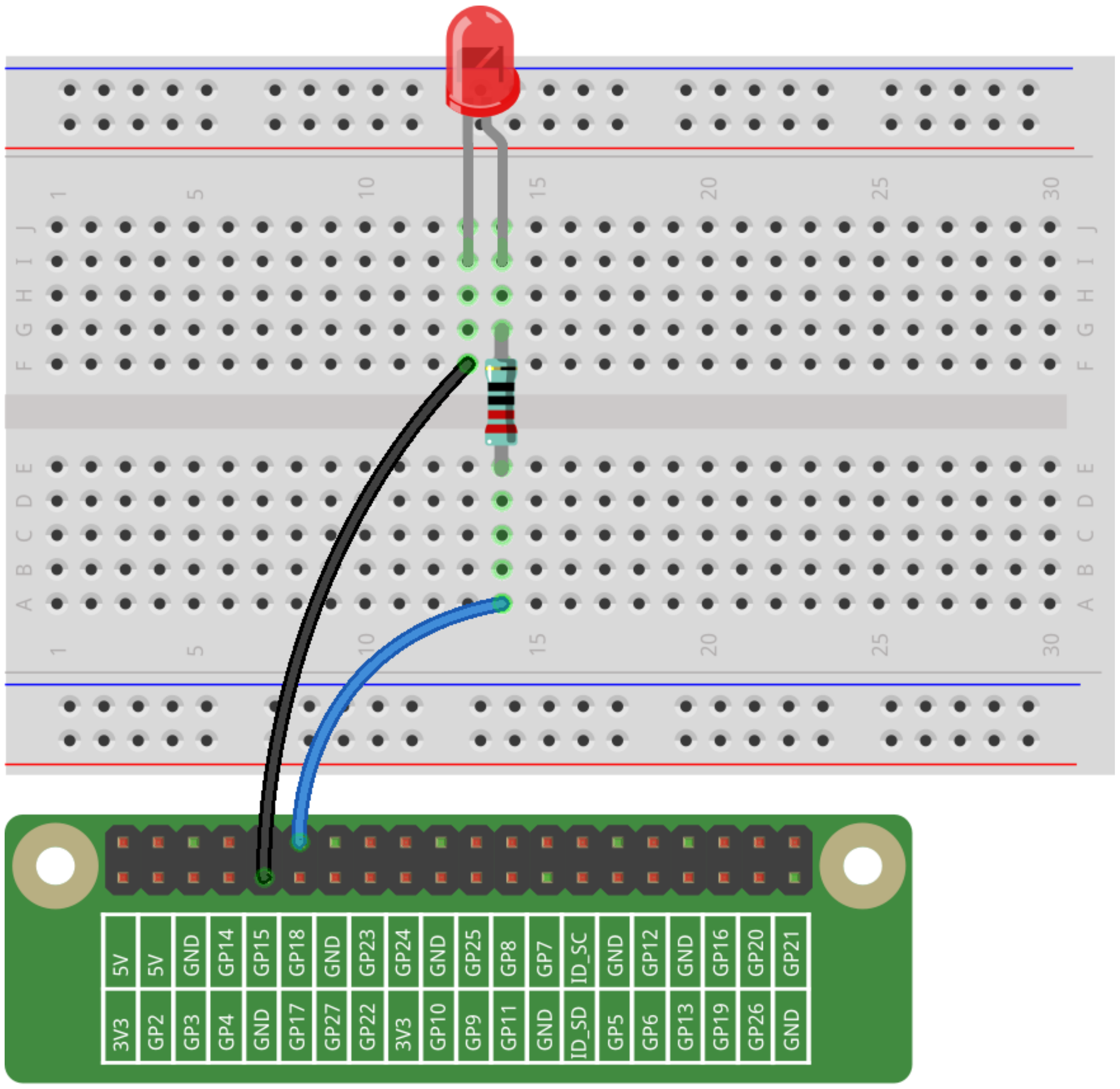
Schematic Diagram



wiringPi	Physical	BCM
1	Pin12	18



Build the Circuit



For C Language Users:

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_3_BreathingLed
```

2. Compile the code.

```
gcc 3_BreathingLed.c -lwiringPi
```

3. Run the executable file.

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

As the code runs, you can see that the brightness of the LED becomes stronger or weaker.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define LedPin 1

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

    softPwmCreate(LedPin,  0, 100);

    int i;

    while(1) // loop forever
    {
        for(i=0;i<100;i++){  // i,as the value of pwm, increases progressively during␣
↪0-1024.
            softPwmWrite(LedPin, i);
            delay(10);
        }

        for(i=100;i>=0;i--){
            softPwmWrite(LedPin, i);
            delay(10);
        }
    }
    return 0 ;
}
```

Code Explanation

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

WiringPi includes a software-driven **PWM** library of outputting a PWM signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is 100S. That combined with the default suggested range of 100 gives a PWM frequency of 100Hz. Within these limitations, control of a light/LED or a motor is very achievable.

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

The function is to use software library to create a PWM pin, set its period between 0x100us-100x100us. The prototype of the function softPwmCreate(LedPinRed, 0, 100) is as follows

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

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

initialValue: The initial pulse width is that initialValue times100us.

pwmRange: The period of PWM is that pwmRange times100us.

```
            softPwmWrite(LedPin, i);
```

The function is used to write the PWM value **i** to the **LedPin**.

The prototype of the function softPwmWrite(LedPinBlue, b_val) is as follows

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

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

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

```
            delay(10);
```

Wait for 10ms, interval time between the changes indicates the speed of breathing.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 3_BreathingLed.py
```

As the code runs, you can see that the brightness of the LED becomes stronger or weaker.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

LedPin = 18

def setup():
    global pLed
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
    pLed = GPIO.PWM(LedPin, 1000)
    pLed.start(0)

def main():
    # Set increase/decrease step
    step =2
    delay = 0.05
    while True:
        # Increase duty cycle from 0 to 100
        for dc in range(0, 101, step):
            pLed.ChangeDutyCycle(dc)
            print (' ++ Duty cycle: %s'%dc)
            time.sleep(delay)
        time.sleep(1)

        # decrease duty cycle from 100 to 0
        for dc in range(100, -1, -step):
            # Change duty cycle to dc
            pLed.ChangeDutyCycle(dc)
            print ('  -- Duty cycle: %s'%dc)
            time.sleep(delay)
        time.sleep(1)

def destroy():
    # Stop pLed
    pLed.stop()
    # Turn off LED
    GPIO.output(LedPin, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

```
   pLed = GPIO.PWM(LedPin, 1000)
```

To create a PWM instance. Set **pLed** as pwm output and frequence to **1K** Hz.

```
   pLed.start(0)
```

Set pLed begin with value **0**.

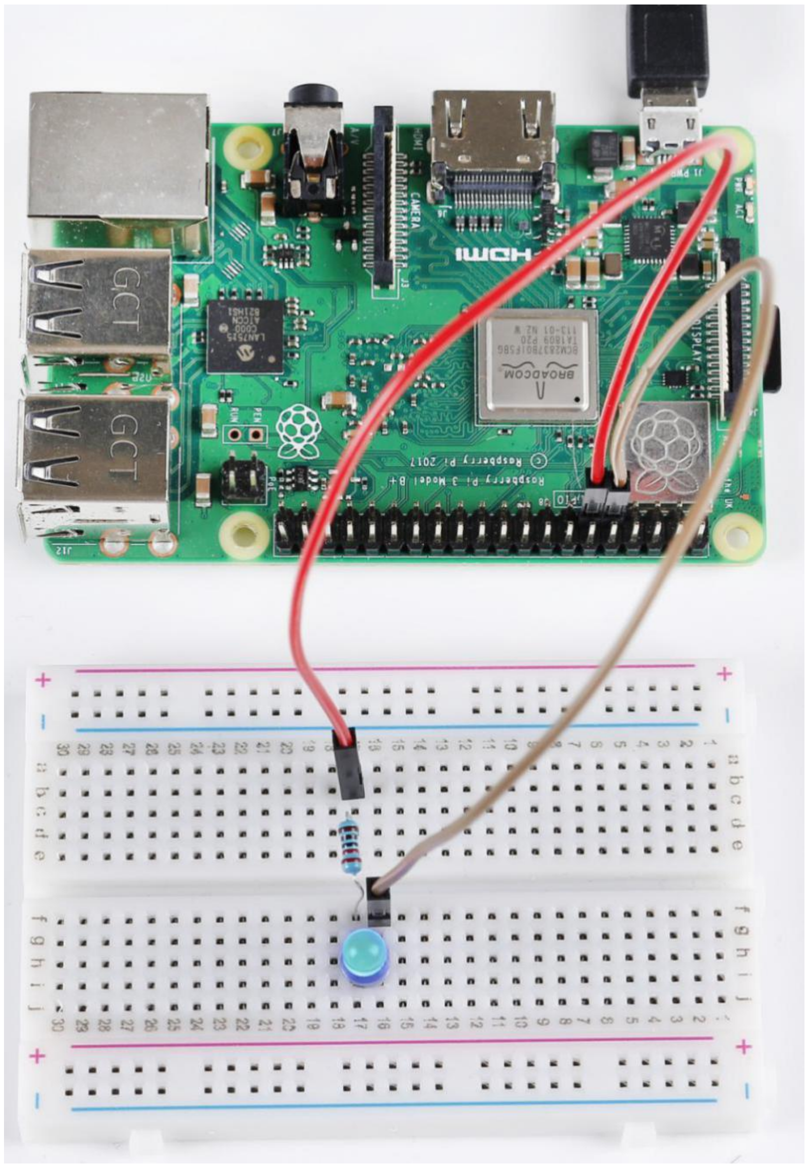
```
       for dc in range(0, 101, step):
           # Change duty cycle to dc
           pLed.ChangeDutyCycle(dc)
           print (' ++ Duty cycle: %s'%dc)
           time.sleep(delay)
```

Increase the duty cycle by 2 at a time, from **0** to **101**, and you'll see the LED getting brighter and brighter.

```
for dc in range(100, -1, -step):
           pLed.ChangeDutyCycle(dc)
           print ('  -- Duty cycle: %s'%dc)
           time.sleep(delay)
```

Similarly, when the duty cycle is reduced by 2 from **100** to **-1**, the LED brightness will be dimmer and dimmer.

Phenomenon Picture

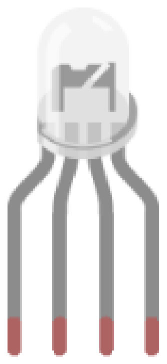


Lesson 4 RGB LED

Introduction

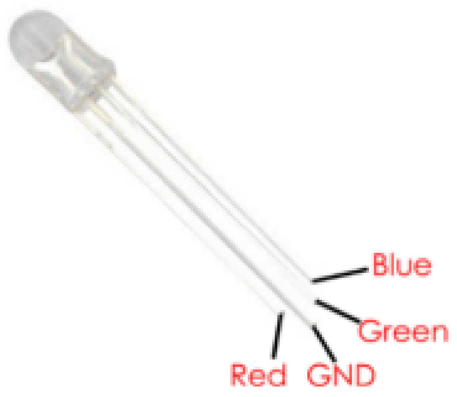
Previously we've used the PWM technology to control an LED's brightness. In this lesson, we will use it to control an RGB LED to flash various kinds of colors.

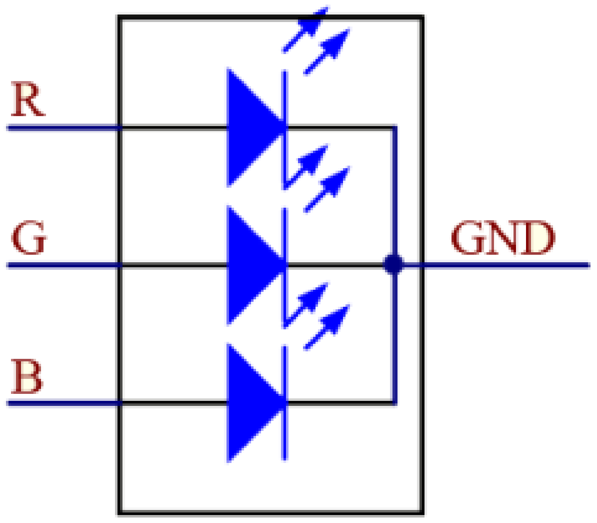
Newly Added Components

1 * RGB LED	3 * Resistor (220 Ω)
	

Principle

RGB LED

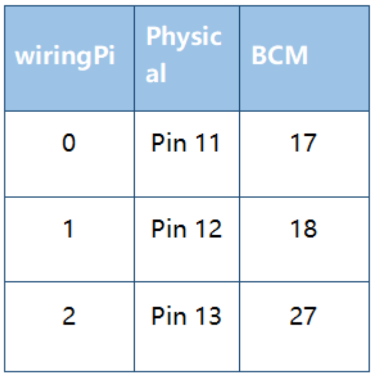




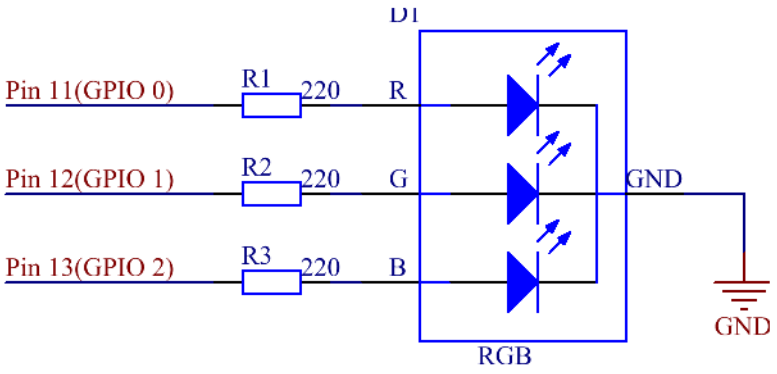
The three primary colors of the RGB LED can be mixed into various colors by brightness. The brightness of LED can be adjusted with PWM. Raspberry Pi has only one channel for hardware PWM output, but it needs three channels to control the RGB LED, which means it is difficult to control the RGB LED with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

Schematic Diagram

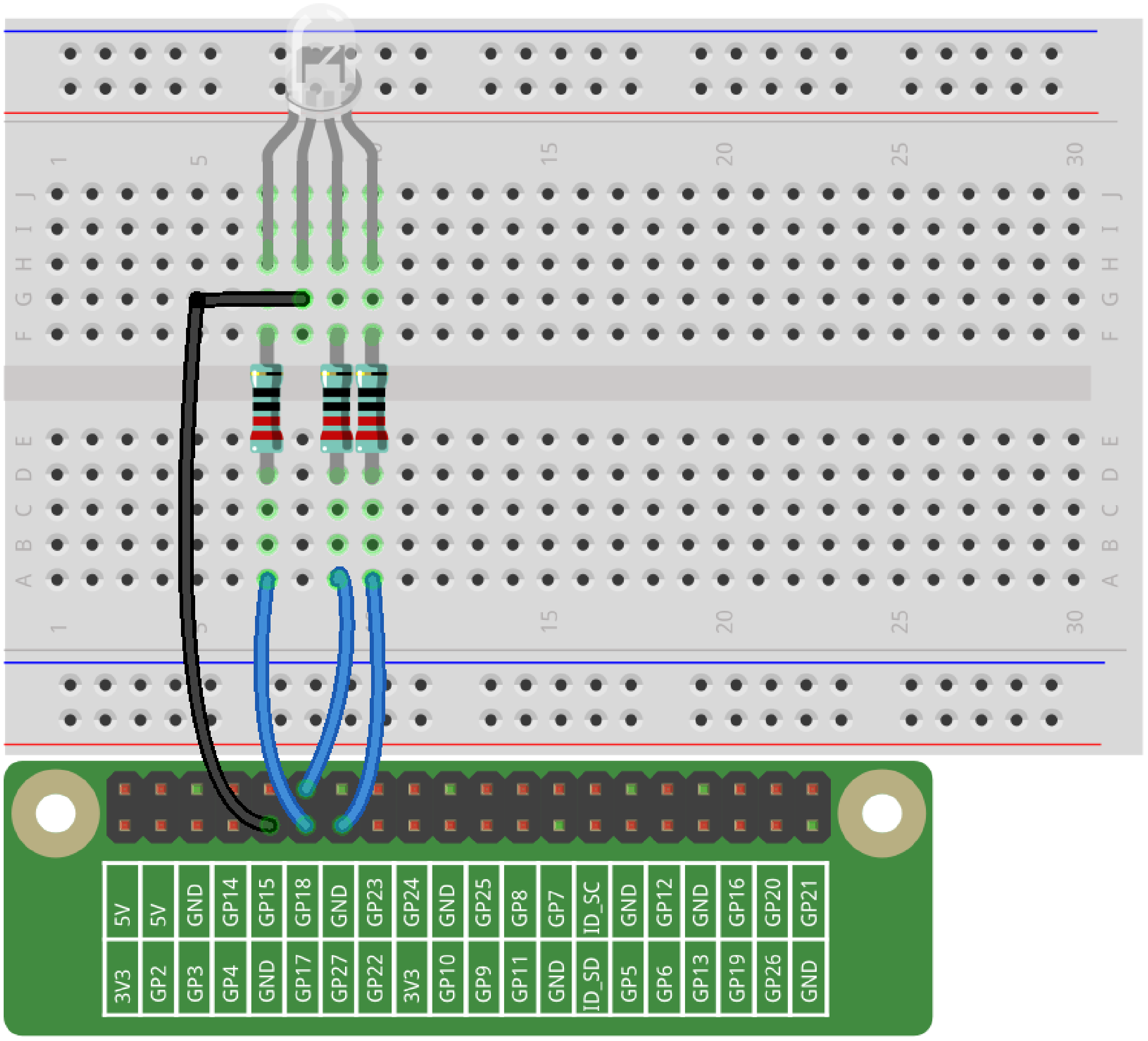
After connecting the pins of R, G, and B to a current limiting resistor, connect them to the pin 11, pin 12, and pin 13 respectively. The longest pin (GND) of the LED connects to the GND of the Raspberry Pi. When the three pins are given different PWM values, the RGB LED will display different colors.



wiringPi	Physic al	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27



Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_4_RGBLed
```

2. Compile the code.

```
gcc 4_rgbLed.c -lwiringPi
```

3. Run the executable file.

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

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

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define uchar unsigned char

#define LedPinRed    0
#define LedPinGreen  1
#define LedPinBlue   2

// define function used for initializing I/O port to output for pwm.
void ledInit(void){
    softPwmCreate(LedPinRed,  0, 100);
    softPwmCreate(LedPinGreen,0, 100);
    softPwmCreate(LedPinBlue, 0, 100);
}

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

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

    ledInit();

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

(continues on next page)

(continued from previous page)

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

    return 0;
}
```

Code Explanation

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

Create a function to set the **LedPinRedLedPinGreen** and **LedPinBlue** as PWM pins, then set their period between 0x100us-100x100us.

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

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

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

initialValue: The initial pulse width is that initialValue times100us.

pwmRange: the period of PWM is that pwmRange times100us.

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

This function is to set the colors of the LED. Using RGB, the formal parameter **r_val** represents the luminance of the red one, **g_val** of the green one, **b_val** of the blue one.

The prototype of the function softPwmWrite(LedPinBlue, b_val) is as follows

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

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

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

```
    ledInit();
```

Call the **ledInit()** function in the **main** function to initialize the LED.

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

Call the function defined before. Write **0xff** into LedPinRed and **0x00** into LedPinGreen and LedPinBlue.

Only the Red LED lights up after running this code. If you want to light up LEDs in other colors, just modify the parameters.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 4_rgbLed.py
```

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

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

COLOR = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]
pins = {'Red':17, 'Green':18, 'Blue':27}

def setup():
    global p_R, p_G, p_B
    GPIO.setmode(GPIO.BCM)
    for i in pins:
        GPIO.setup(pins[i], GPIO.OUT, initial=GPIO.LOW)

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

    # Set all begin with value 0
    p_R.start(0)
    p_G.start(0)
    p_B.start(0)

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

def setColor(color):
    # Devide colors from 'color' veriable
```

(continues on next page)

(continued from previous page)

```
    R_val = (color & 0xFF0000) >> 16
    G_val = (color & 0x00FF00) >> 8
    B_val = (color & 0x0000FF) >> 0

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

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

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

def main():
    while True:
        for color in COLOR:
            setColor(color)
            time.sleep(0.5)

def destroy():
    # Stop all pwm channel
    p_R.stop()
    p_G.stop()
    p_B.stop()
    # Turn off all LEDs
    GPIO.output(pins, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

```
14.    p_R = GPIO.PWM(pins['Red'], 2000)
```

This statement is used to set the pin to a specific PWM frequency, in this case **2000** Hz.

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

Define a **map** function for mapping values. For instance, x=50, in_min=0, in_max=255, out_min=0, out_max=100. After the map function mapping, it returns (50-0) * (100-0)/(255-0) +0=19.6, meaning that 50 in 0-255 equals 19.6 in 0-100.

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

Create a **setColor()** function to assign different value to the three variables: R_val, G_val, B_val. Input color should be hexadecimal with red value, blue value, green value. Assign the first two values of the hexadecimal to R_val, the middle two to G_val, assign the last two values to B_val, please refer to the shift operation of the hexadecimal for details. For example, color=0xFF00FF, then R_val=(0xFF00FF & 0xFF0000)>>16 = 0xFF, G_val = 0x00, B_val=0xFF.

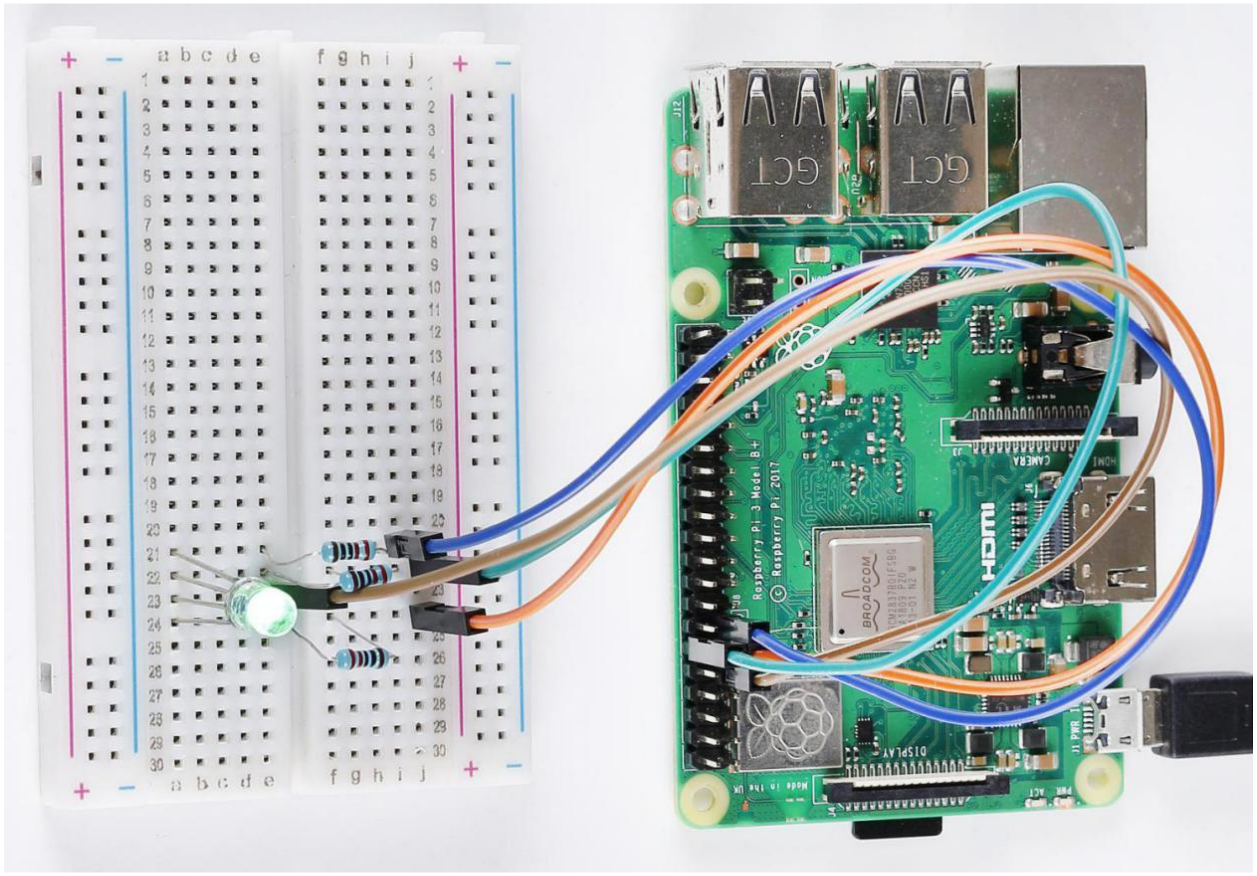
```
# Map color value from 0~255 to 0~100
    R_val = MAP(R_val, 0, 255, 0, 100)
    G_val = MAP(G_val, 0, 255, 0, 100)
    B_val = MAP(B_val, 0, 255, 0, 100)

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

Map the RGB value from **0** - **255** to **0** - **100**. After that, get a value. Then set it to be the duty cycle of R_val, G_val and B_val, and the RGB LED displays corresponding colors.

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

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

Phenomenon Picture



Lesson 5 Controlling LED by Button

Introduction

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

Newly Added Components

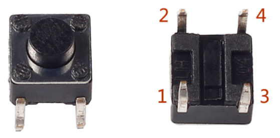
1 * Resistor (10kΩ)	1 * Button	1 * Resistor (220Ω)	1 * LED

Principle

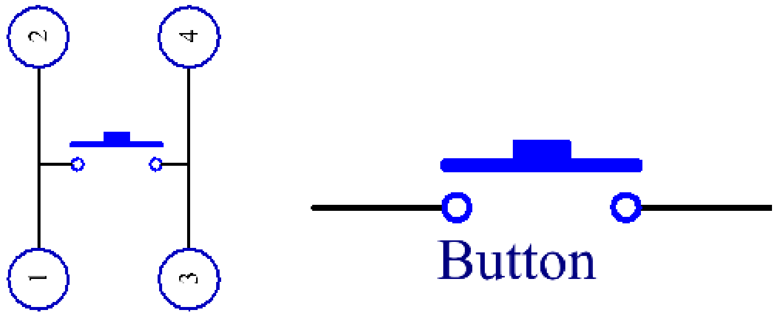
Button

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

Two pins on the same side are connected, which is shown below:



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

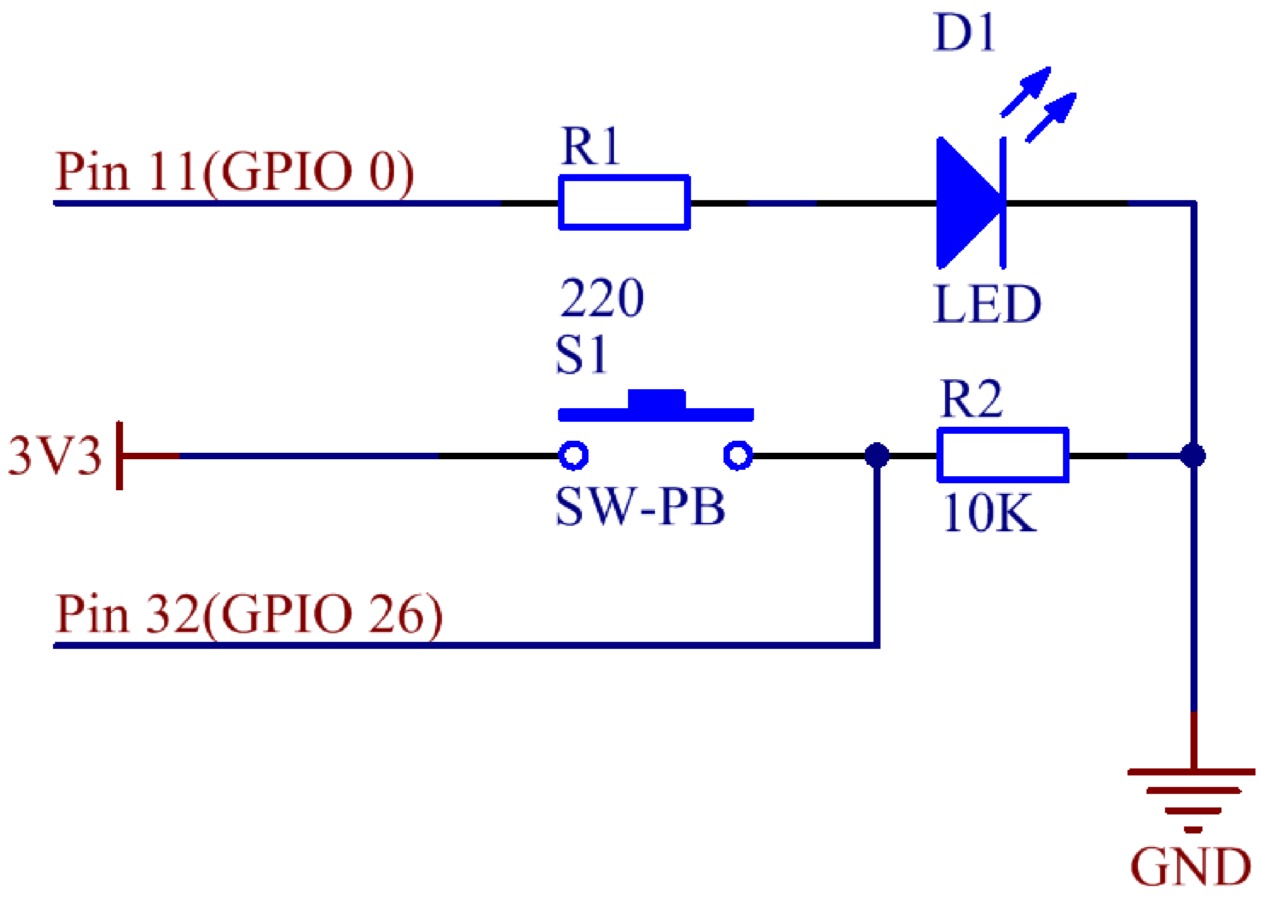


When the button is pressed, the 4 pins are connected, thus closing the circuit.

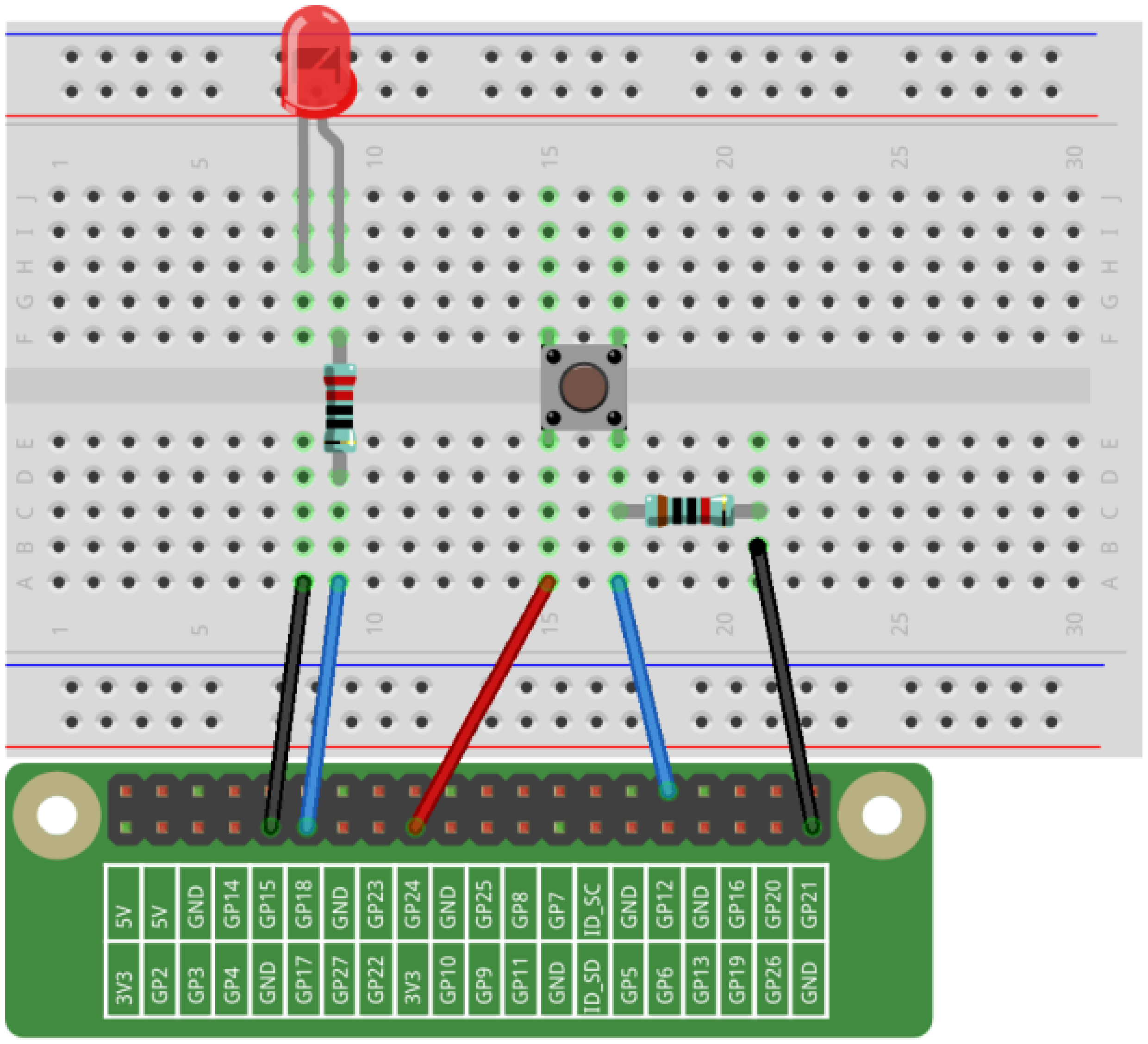
Schematic Diagram

When the button is pressed once, pin 32 is 3.3V (HIGH). Set the pin 11(integrated with an LED) as high level by programming at the same time. Then press the button again and set pin 11 to Low. So we will see the LED light on and off alternately as the button is pressed many times.

wiringPi	Physical	BCM
0	Pin 11	17
26	Pin 32	12


D1
R1
Pin 11(GPIO 0)
220
LED
S1
R2
3V3
SW-PB
10K
Pin 32(GPIO 26)
GND

Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_5_Controlling_Led_by_Button
```

2. Compile the code.

```
gcc 5_Button.c -lwiringPi
```

3. Run the executable file.

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

When you press the button for the first time, the LED lights up. When the button is pressed again, the LED lights off.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define LedPin      0
#define ButtonPin   26
int state = 0;

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

    pinMode(LedPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    pullUpDnControl(ButtonPin, PUD_DOWN);

    while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 1)
        {
            delay(10);
            if(digitalRead(ButtonPin) == 1)
            {
                state ++;
                if(state%2 == 1)
                {
                    digitalWrite(LedPin,HIGH);
                    delay(100);
                }
                if(state%2 == 0)
                {
                    digitalWrite(LedPin,LOW);
                    delay(100);
                }
            }
        }
    }
    return 0;
}
```

Code Explanation

```
int state = 0;
```

Define a variable **state** to record the number of times it is pressed and the initial number of times is **0**.

```
 pinMode(LedPin, OUTPUT);
 pinMode(ButtonPin, INPUT);
```

Set the LedPin to **OUTPUT** mode, ButtonPin to **INPUT** mode.

```
   pullUpDnControl(ButtonPin, PUD_DOWN);
```

When the button is not pressed, ButtonPin is in suspension at which time the read value is changing. To enable ButtonPin to output a stable low level, **PUD_DOWN** is added to the code, keeping ButtonPin at the forced pull-down state till the button is pressed.

```
        if(digitalRead(ButtonPin) == 1)
        {
            delay(10);
            if(digitalRead(ButtonPin) == 1)
            {
```

Usually the buttons we use are mechanical buttons, so in the process of pressing down and releasing, there will be no direct change from 0 to 1, but will be more than 10ms of level jitter. In order to ensure that the program only responds to the button once when it is closed or broken, the jitter elimination of the button must be carried out. An **if** function is used to detect whether the button is pressed. When the signal of the button is pressed is detected, a delay of 10ms is used to eliminate the possibility of false judgment, and another **if** function is used to detect again. If both **if** conditions are met, confirm that it is a button press, and then execute the program in the **if**.

```
                state ++;
```

If the button is pressed, the number of times it is pressed is increased by one. (state ++ is the same as state = state+1).

```
                if(state%2 == 1)
                {
                    digitalWrite(LedPin,HIGH);
                    delay(100);
                }
```

% is a modulo operator in C language; state%2 is that state is divided by 2 to return the remainder. If state=17, then state%2 =1. Here, determine whether state%2 is equal to 1. If it is, it means that the number of times of pressing the button is a singular number, and then turn on the LED.

```
                if(state%2 == 0)
                {
                    digitalWrite(LedPin,LOW);
                    delay(100);
                }
```

Here, judge whether state%2 is equal to 0. If so, it means that the number of times the button is pressed is an even number, and then turn off the LED.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 5_Button.py
```

When you press the button for the first time, the LED lights up. When the button is pressed again, the LED lights off.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

LedPin = 17
BtnPin = 12
Led_status = False

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN)
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)

# Define a callback function for button callback
def swLed(ev=None):
    global Led_status
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)

# Define a main function for main process
def main():
    while True:
        # Don't do anything.
        time.sleep(1)

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

(continues on next page)

(continued from previous page)

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

Code Explanation

```
Led_status = False
```

Set a variable **Led_status** to record the current status of the LED; when Led_status is **True**, it indicates that the current lamp is in bright state; when Led_status is **False**, it means that the light is off.

```
    GPIO.setup(BtnPin, GPIO.IN)
```

Set **BtnPin** as input mode to read the state of the button to determine whether to execute the corresponding program. Note that when **GPIO.setup** sets the pin to input mode, then there is no need to set the initial value.

```
    GPIO.setup(LedPin, GPIO.OUT, initial=GPIO.LOW)
```

Specify an initial value for your output channel. Here the LED is the output component, so we set **LedPin** to **GPIO.OUT** mode. Then initialize the state of LED to **GPIO.LOW** which means that the light is off.

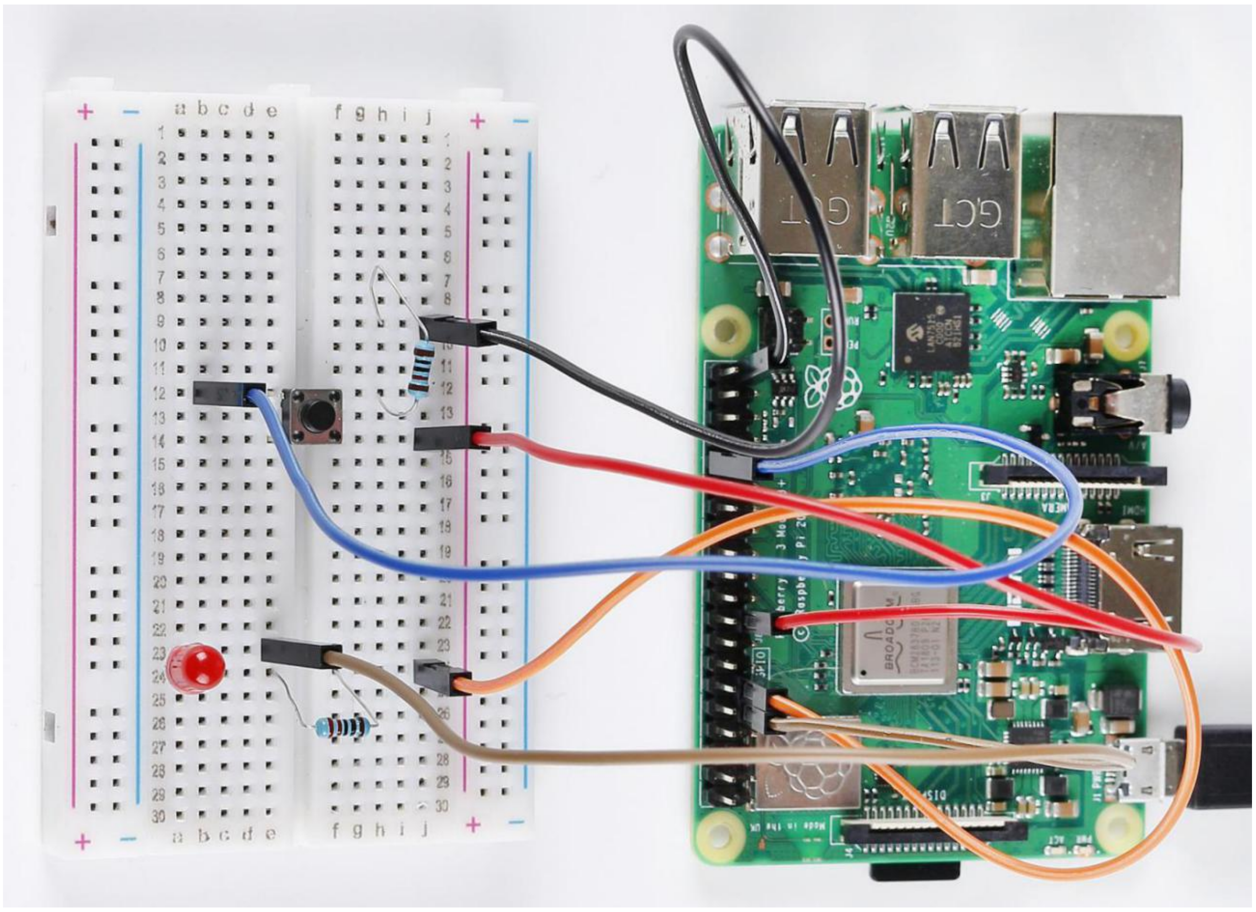
```
    GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed)
```

The **event_detected()** function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy working on other things. Set up a falling detect on BtnPin, when the BtnPin pin is detected to change from high level to low level, **swLed** function is called.

```
def swLed(ev=None):
    global Led_status
    Led_status = not Led_status
    GPIO.output(LedPin, Led_status)
```

RPi.GPIO runs a second thread for callback functions. This means that callback functions can be run at the same time as your main program, in immediate response to an edge. Define a callback function for button callback, execute the function after the callback of the interrupt. When this function is executed, the state of the LED is firstly reversed(If **True**, make it **False**, and vice versa). Then input the function to LedPin. And "**ev = None**" means that if no parameter is passed when calling **swLed**, take **None** as the default value of **ev**.

Phenomenon Picture

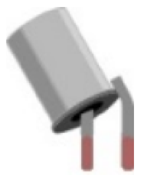
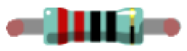
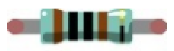
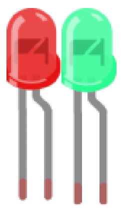


Lesson 6 Tilt Switch

Introduction

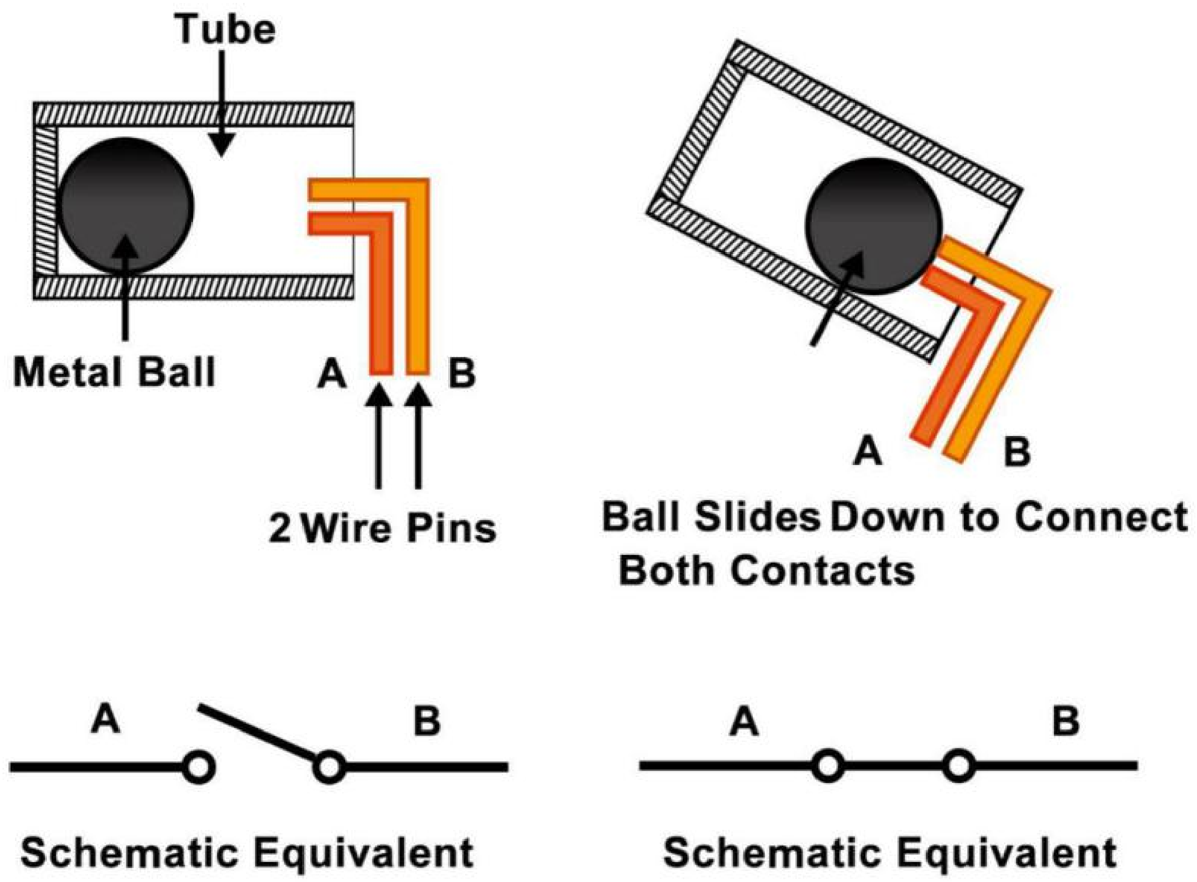
In this lesson, we'll learn a new switch component, tilt switch. Here we apply two LEDs to indicate the current state of tilt switch. You can also use this kind of switch to make a sense light with the clamshell box.

Newly Added Components

1 * Tilt Switch	2 * Resistor (220Ω)	1 * Resistor(1kΩ)	2 * LED
			

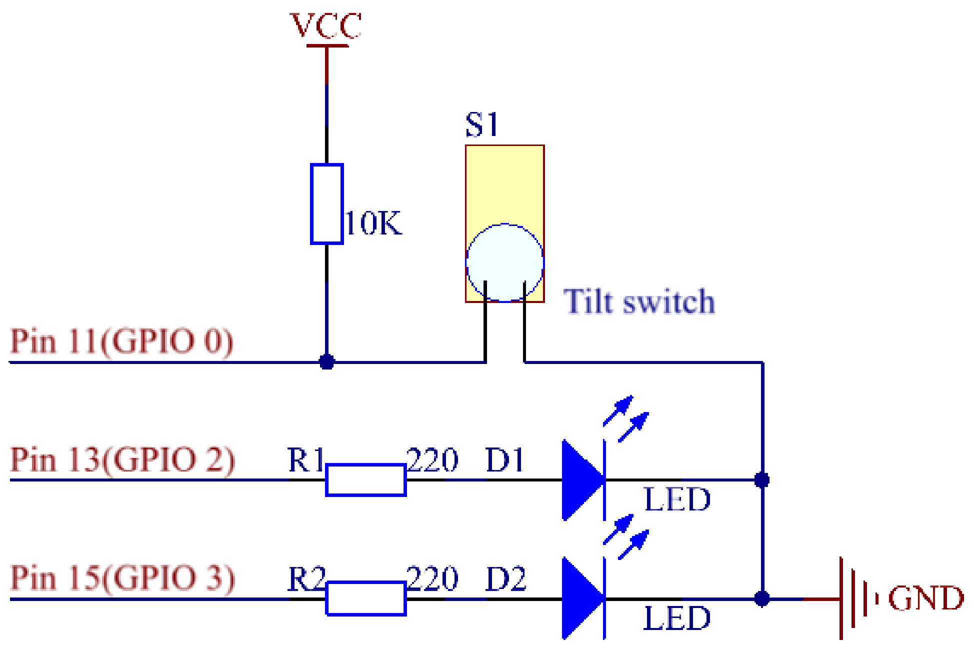
Principle

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.

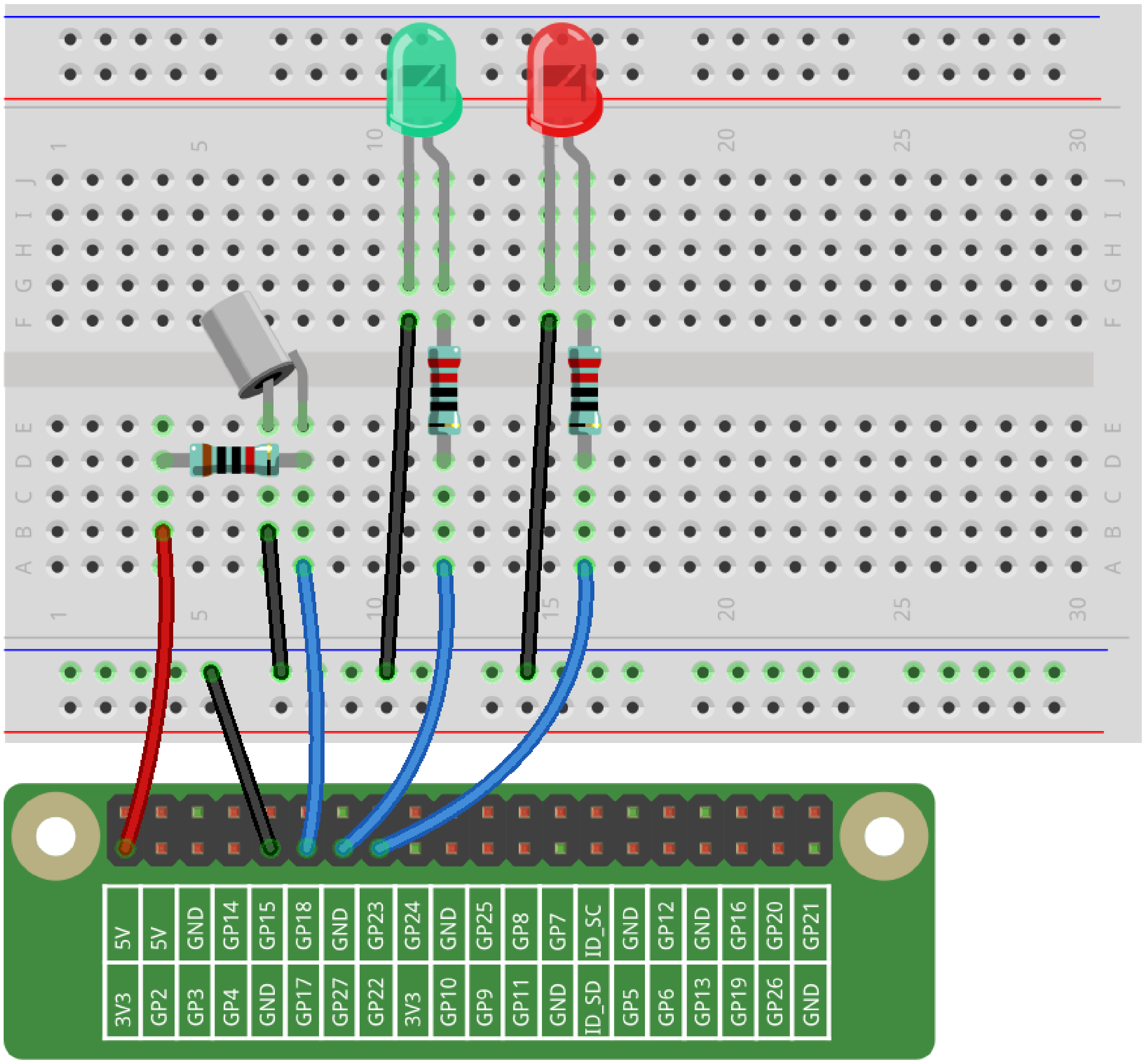


Schematic Diagram

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22



Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_6_TiltSwitch
```

2. Compile the code.

```
gcc 6_Tilt.c -lwiringPi
```

3. Run the executable file.

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

When the tilt switch is level, the green LED turns on. If you tilt the switch, the red LED will turn on.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define TiltPin     0
#define Gpin        2
#define Rpin        3

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

    pinMode(TiltPin, INPUT);
    pinMode(Gpin, OUTPUT);
    pinMode(Rpin, OUTPUT);

    while(1){

        if(1 == digitalRead(TiltPin)){
            delay(10);
            if(1 == digitalRead(TiltPin)){
                digitalWrite(Rpin, HIGH);
                digitalWrite(Gpin, LOW);
                printf("RED\n");
            }
        }
        else if(0 == digitalRead(TiltPin)){
            delay(10);
            if(0 == digitalRead(TiltPin)){
                while(!digitalRead(TiltPin));
                digitalWrite(Rpin, LOW);
                digitalWrite(Gpin, HIGH);
                printf("GREEN\n");
            }
        }
    }
    return 0;
}
```

Code Explanation

```
pinMode(TiltPin, INPUT);
pinMode(Gpin, OUTPUT);
pinMode(Rpin, OUTPUT);
```

Initialize pins, then set the pin of tilt switch to **INPUT** mode, and LEDs to **OUTPUT** mode.

```
    if(1 == digitalRead(TiltPin)){
```

It is used to judge whether the tilt switch is tilted or not. The value of **TiltPin** is firstly read, if it is equal to **1**, the codes inside the **if()** statement run; otherwise, the codes of **if** are skipped.

```
if(1 == digitalRead(TiltPin)){
        delay(10);
        if(1 == digitalRead(TiltPin)){
            digitalWrite(Rpin, HIGH);
            digitalWrite(Gpin, LOW);
            printf("RED\n");
        }
    }
```

When the tilt is tilted, the tilt switch is on; the Raspberry Pi reads a high level at the tilt pin, so the red LED is on and green LED off.

```
    else if(0 == digitalRead(TiltPin)){
        delay(10);
        if(0 == digitalRead(TiltPin)){
            while(!digitalRead(TiltPin));
            digitalWrite(Rpin, LOW);
            digitalWrite(Gpin, HIGH);
            printf("GREEN\n");
        }
    }
```

When the tilt is level, the tilt switch is off; the Raspberry Pi reads a low level at the tilt pin, so the red LED is off and green LED on.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 6_Tilt.py
```

When the tilt switch is level, the green LED turns on. If you tilt the switch, the red LED will turn on.

code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```python
import RPi.GPIO as GPIO

TiltPin = 17
Gpin   = 27
Rpin   = 22

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Gpin, GPIO.OUT,initial=GPIO.HIGH)
    GPIO.setup(Rpin, GPIO.OUT,initial=GPIO.LOW)
    GPIO.setup(TiltPin, GPIO.IN)
    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)

def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)

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

def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))

def loop():
    while True:
        pass

def destroy():
    GPIO.output(Gpin, GPIO.LOW)       # Green LED off
    GPIO.output(Rpin, GPIO.LOW)       # Red LED off
    GPIO.cleanup()                     # Release resource

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

Code Explanation

```
    GPIO.add_event_detect(TiltPin, GPIO.BOTH, callback=detect, bouncetime=200)
```

Set up a falling detect on **TiltPin**, and callback function to detect. Here **bouncetime** is to add rise threshold detection on the channel and ignore edge operations less than 200ms caused by switch jitter.

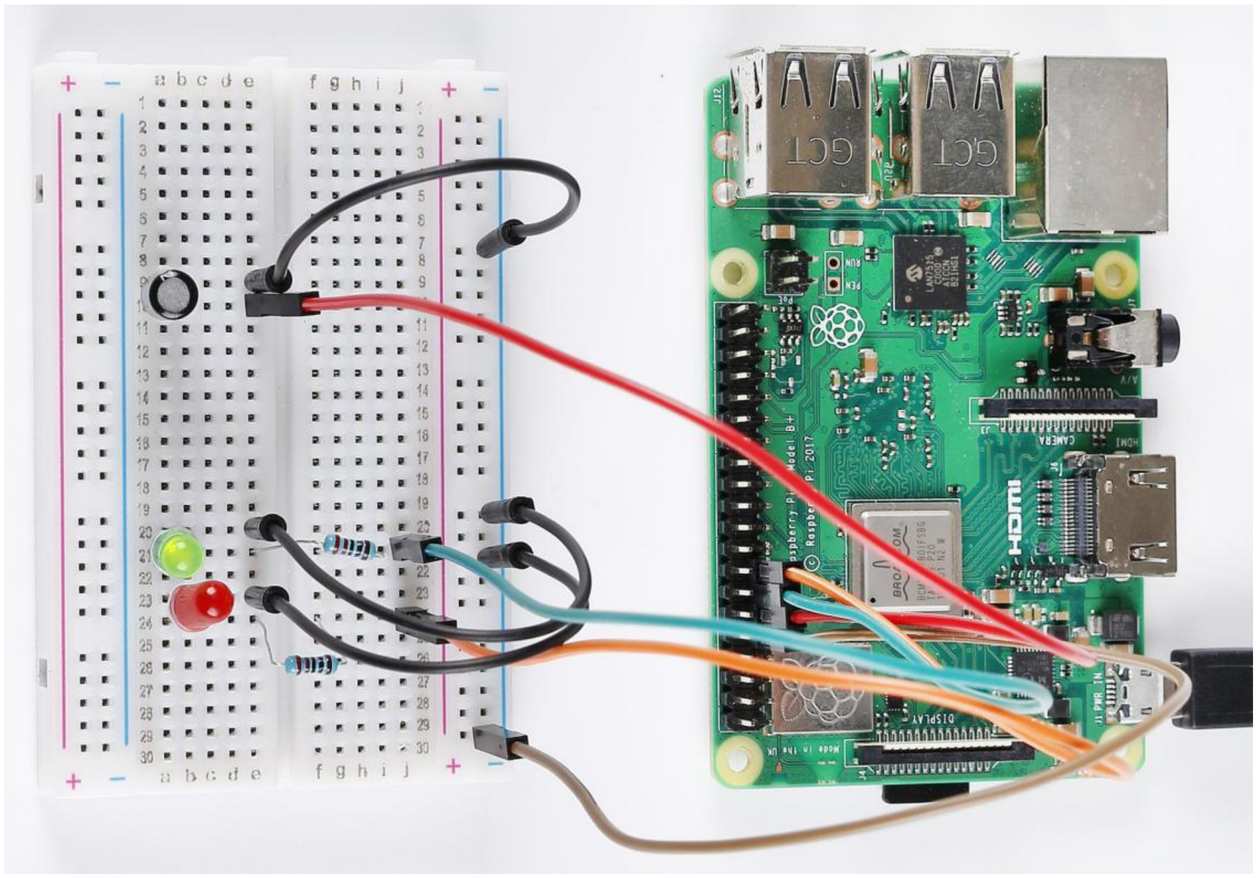
```
def Led(x):
    if x == 0:
        GPIO.output(Rpin, 1)
        GPIO.output(Gpin, 0)
    if x == 1:
        GPIO.output(Rpin, 0)
        GPIO.output(Gpin, 1)
```

Define a **Led()** function to set the mode of the two LEDs. When x = 0, the red LED goes on and the green light goes off; when x = 1, the red LED goes off, the green LED goes on. When the function is called, the mode of the LED can be set directly with the statement **Led(1)** or **Led(0)**.

```
def detect(chn):
    Led(GPIO.input(TiltPin))
    Print(GPIO.input(TiltPin))
```

This is a callback function that executes when a trigger is detected. Assign the current **TiltPin** state (0 or 1) to the Led function, that is, pass parameters to the **Led** function. The **Led** function then performs the corresponding operation on the LEDs.

Phenomenon Picture

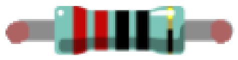
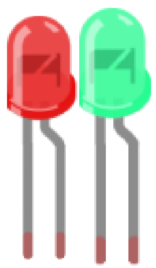
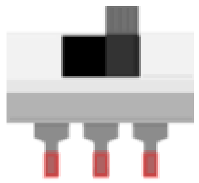


Lesson 7 Slide Switch

Introduction

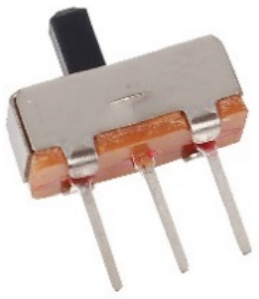
In this lesson, we are going to use a slide switch to turn on the 2 LEDs. The slide switch is a device to connect or disconnect the circuit by sliding its handle. They are quite common in our surroundings. Now let's see how it works.

Newly Added Components

2 * Resistor (220Ω)	2 * LED	1 * Slide Switch
		

Principle

Slide Switch

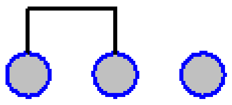


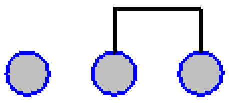
Just as its name suggests, slide switch is to connect or disconnect the circuit by sliding its switch handle so as to switch the circuit. The common types of slide switch include single pole double throw, single pole triple throw, double pole double throw, and double pole triple throw and so on. Generally, it is used in circuits with a low voltage and features flexibility and stabilization. Slide switches are commonly used in all kinds of instruments/meters equipment, electronic toys and other fields related.

How it works: The middle pin is fixed. When the handle is pushed to the left, the left two pins are connected; when push it to the right, the two pins on the right connect, thus switching circuits.

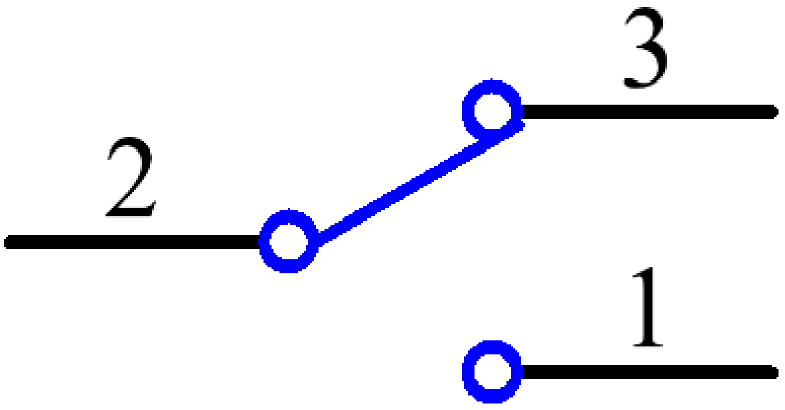
Just as its name suggests, slide switch is to connect or disconnect the circuit by sliding its switch handle so as to switch the circuit. The common types of slide switch include single pole double throw, single pole triple throw, double pole double throw, and double pole triple throw and so on. Generally, it is used in circuits with a low voltage and features flexibility and stabilization. Slide switches are commonly used in all kinds of instruments/meters equipment, electronic toys and other fields related.

How it works: The middle pin is fixed. When the handle is pushed to the left, the left two pins are connected; when push it to the right, the two pins on the right connect, thus switching circuits.





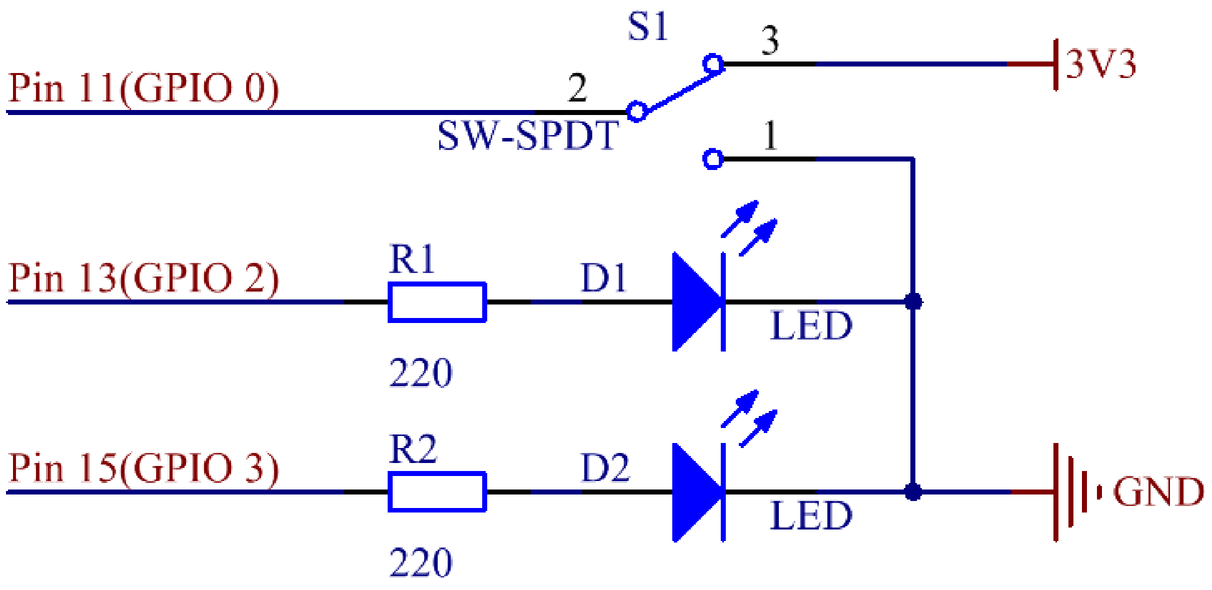
See the circuit symbol of slide switch and 2 is the middle pin.



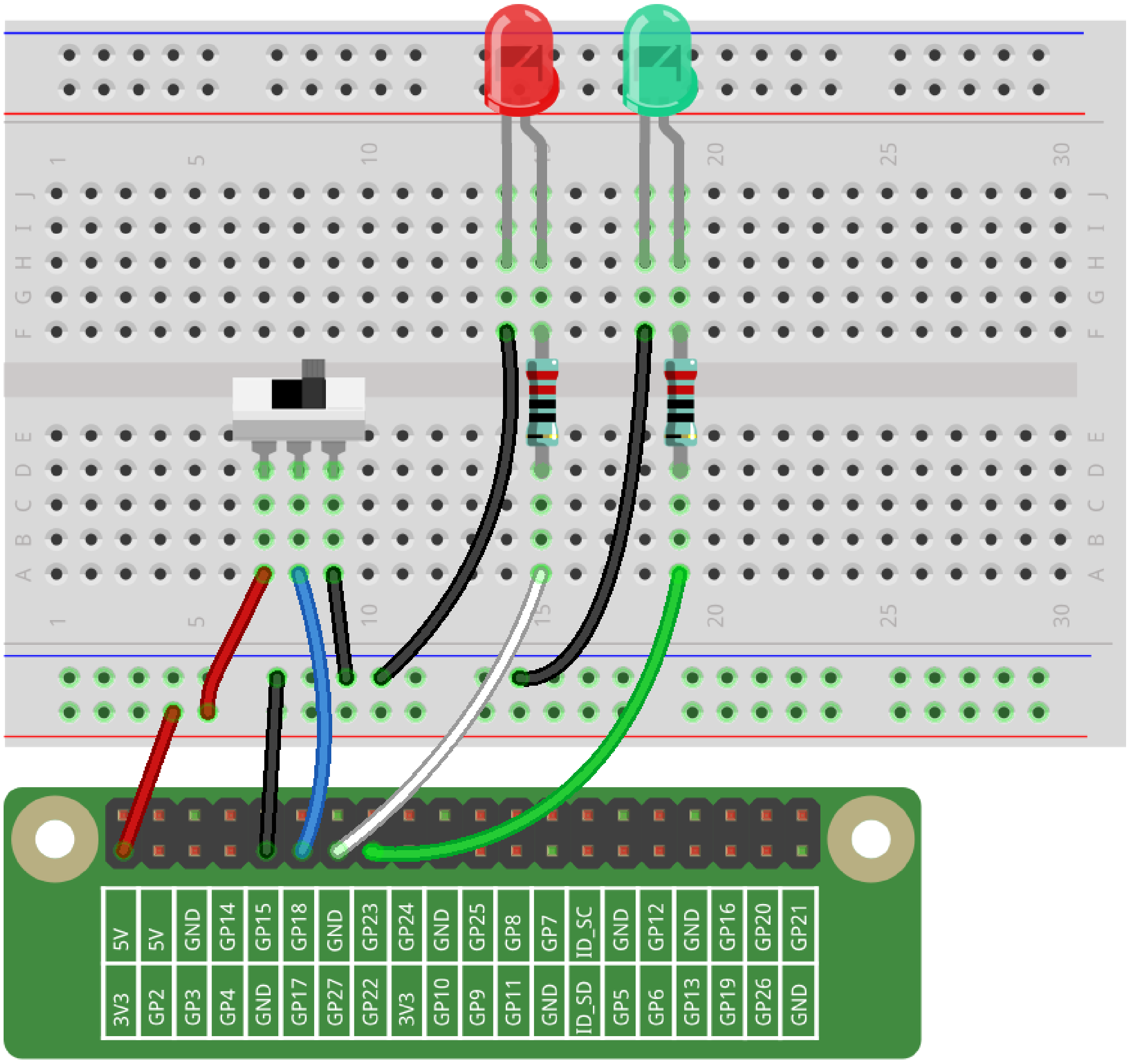
Schematic Diagram

Here we use a slide switch to turn the LED on/off, which is simple. Connect the middle pin of the switch to pin 11. Connect the left pin of the switch to **GND**, the right to **3.3V**. Attach the anode pins (the longer pins) of the two LEDs to pin **13** and pin **15** respectively after getting them connected with two **220** resistors. In addition, insert the cathodes of the two LEDs into **GND**. Get the slide switch connected to the left, the signal read on pin 11 is 0 (a low level), so the LED 1 lights up; to the right, the signal read on pin 11 is 1 (a high level), then the LED 2 turns on.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22



Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_7_SlideSwitch
```

2. Compile the code.

```
gcc 7_Slider.c -lwiringPi
```

3. Run the executable file.

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

When the slide is pulled to the right, the LED2 is on and LED1 off. If the slide is pulled to the left, the LED1 is on and LED2 off.

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

#define slidePin        0
#define led1            2
#define led2            3

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

    pinMode(slidePin, INPUT);
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);

    while(1){
        // slide switch high, led1 on
        if(digitalRead(slidePin) == 1){
            digitalWrite(led1, HIGH);
            digitalWrite(led2, LOW);
            printf("LED1 on\n");
            delay(100);
        }
        // slide switch low, led2 on
        if(digitalRead(slidePin) == 0){
            digitalWrite(led2, HIGH);
            digitalWrite(led1, LOW);
            printf(".....LED2 on\n");
            delay(100);
        }
    }
    return 0;
}
```

Code Explanation

```
    pinMode(slidePin, INPUT);
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
```

Initialize the pins connected to slide switch to the **INPUT** mode, and initialize the LED lights to the **OUTPUT** mode.

```
        if(digitalRead(slidePin) == 1){
            digitalWrite(led1, HIGH);
            digitalWrite(led2, LOW);
            printf("LED1 on\n");
            delay(100);
        }
```

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off.

```
        if(digitalRead(slidePin) == 0){
            digitalWrite(led2, HIGH);
            digitalWrite(led1, LOW);
            printf(".....LED2 on\n");
            delay(100);
        }
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off.

For Python Language Users

Command

1. Go to the folder of the code

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

2. Run the code

```
sudo python3 7_Slider.py
```

When the slide is pulled to the right, the LED2 is on and LED1 off. If the slide is pulled to the left, the LED1 is on and LED2 off.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

slidePin = 17
led1Pin = 27
led2Pin = 22

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(slidePin, GPIO.IN)
    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.LOW)

def main():
    while True:
        # slide switch high, led1 on
        if GPIO.input(slidePin) == 1:
            print ('LED1 ON ')
            GPIO.output(led2Pin, GPIO.LOW)
            GPIO.output(led1Pin, GPIO.HIGH)

        # slide switch low, led2 on
        if GPIO.input(slidePin) == 0:
            print ('    LED2 ON ')
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)
        time.sleep(0.5)

def destroy():
    # Turn off LED
    GPIO.output(led1Pin, GPIO.LOW)
    GPIO.output(led2Pin, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

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

Code Explanation

```
11.    GPIO.setup(slidePin, GPIO.IN)
12.    GPIO.setup(led1Pin, GPIO.OUT, initial=GPIO.LOW)
13.    GPIO.setup(led2Pin, GPIO.OUT, initial=GPIO.LOW)
```

Initialize the pin, then set the pin connected to slide switch to the input mode and LEDs to the output mode.

```
18.    if GPIO.input(slidePin) == 1:
```

(continues on next page)

(continued from previous page)

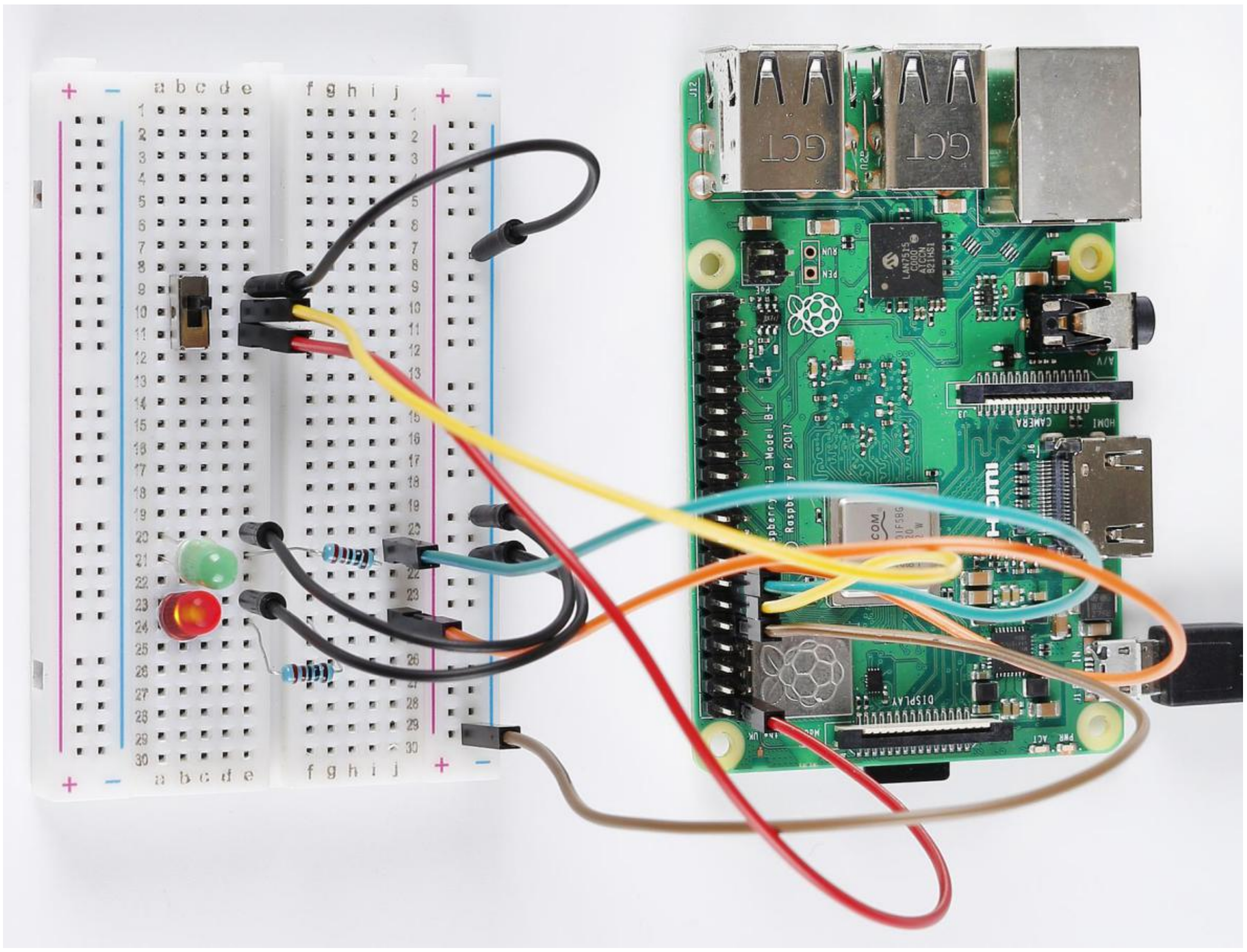
```
            GPIO.output(led2Pin, GPIO.LOW)
            GPIO.output(led1Pin, GPIO.HIGH)
```

When the slide is pulled to the left, the middle pin and left one are connected; the Raspberry Pi reads a high level at the middle pin, so the LED1 is on and LED2 off.

```
        if GPIO.input(slidePin) == 0:
            GPIO.output(led1Pin, GPIO.LOW)
            GPIO.output(led2Pin, GPIO.HIGH)
```

When the slide is pulled to the right, the middle pin and right one are connected; the Raspberry Pi reads a low, so the LED2 is on and LED1 off.

Phenomenon Picture

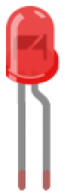
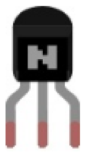


Lesson 8 Relay

Introduction

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

Newly Added Components

1 * Resistor(1kΩ)	1 * Resistor (220Ω)	1 * Diode1N4007 (Rectifier)
1 * LED 	1 * NPN S8050 	1 * Relay

Principle

Relay

As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

There are 5 parts in every relay:

1. **Electromagnet -** It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

2. **Armature -** The movable magnetic strip is known as armature. When current flows through them, the coil is energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

3. **Spring** - When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

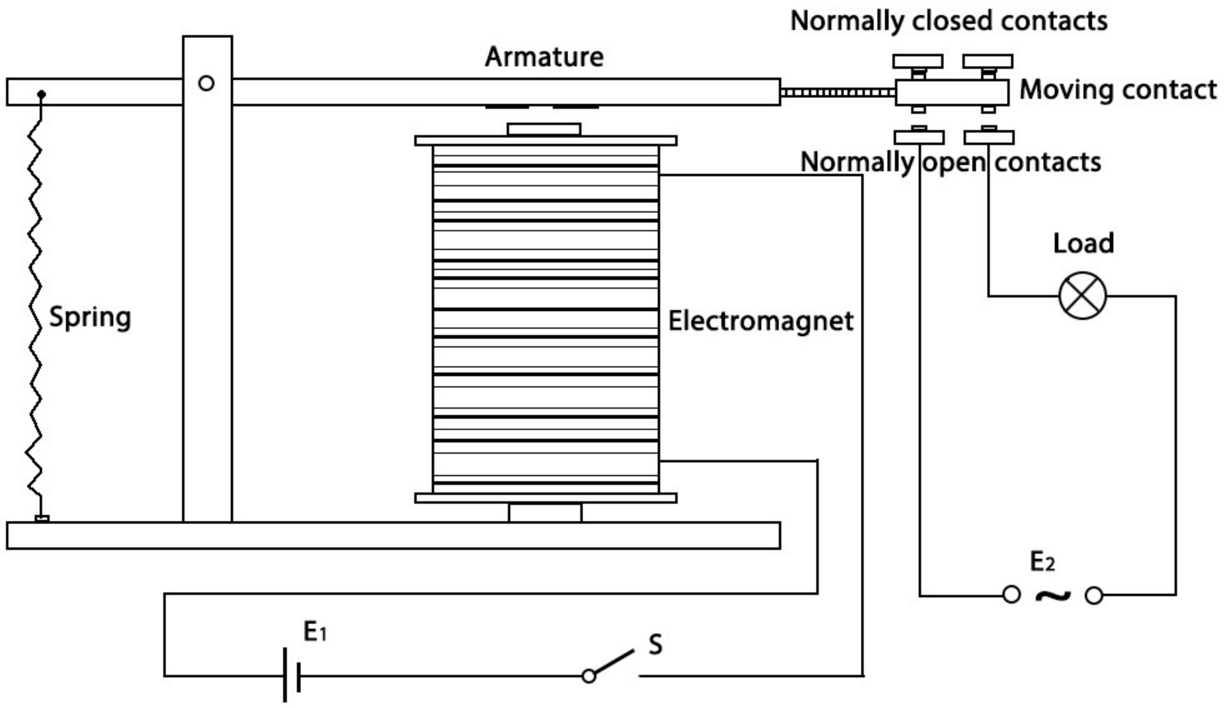
4. Set of electrical **contacts** - There are two contact points:

- **Normally open** - connected when the relay is activated, and disconnected when it is inactive.
- **Normally close** - not connected when the relay is activated, and connected when it is inactive.

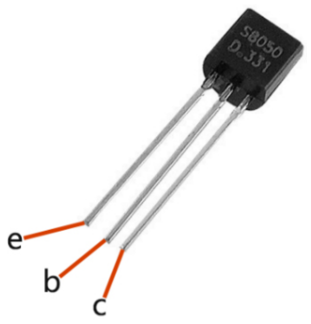
5. Molded frame - Relays are covered with plastic for protection.

Working of Relay

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

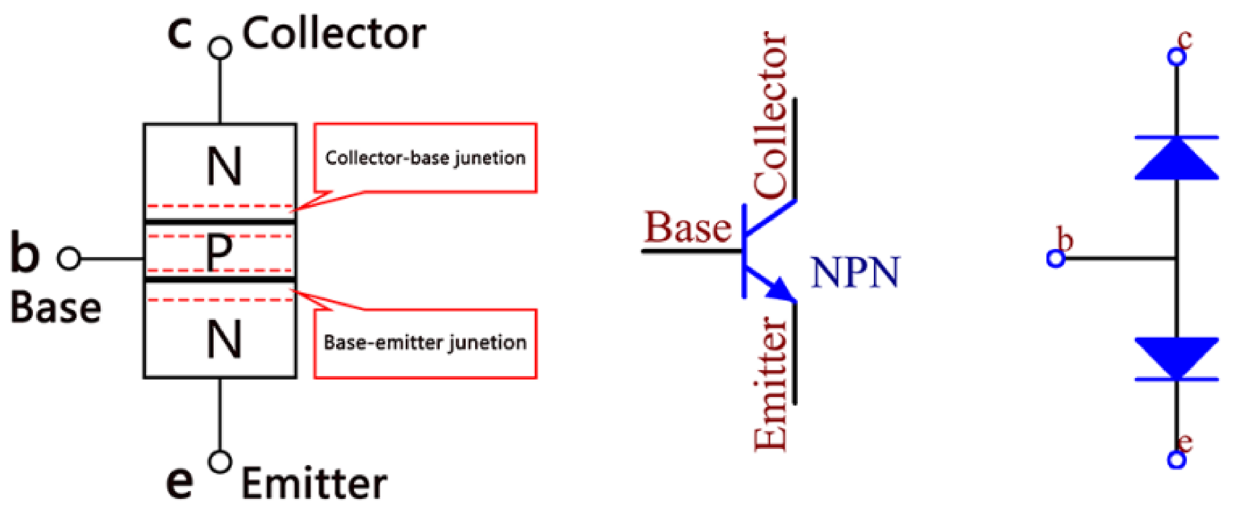


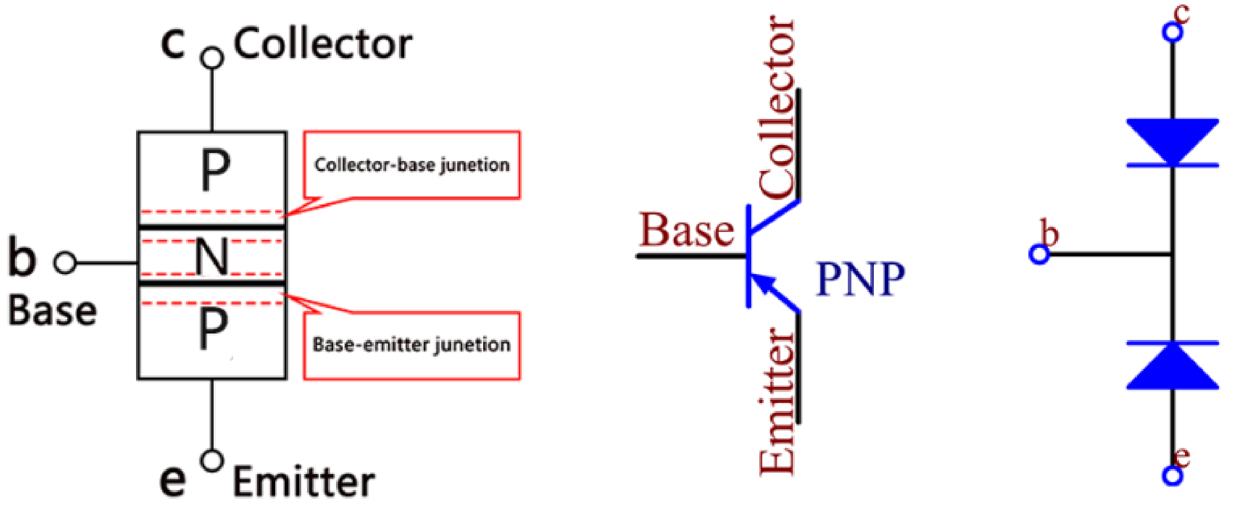
Transistor



Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch. A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, while the other one is the collector region. This composition enables the transistor to be an amplifier.

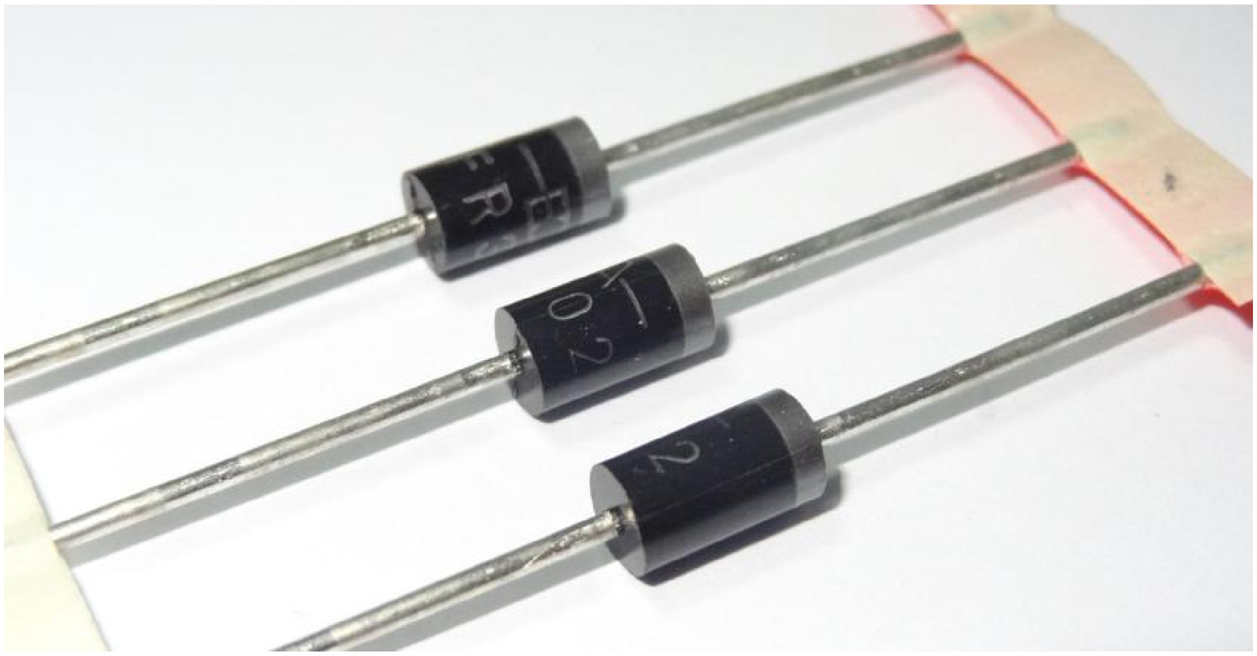
From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction. Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.





When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

Diode1N4007



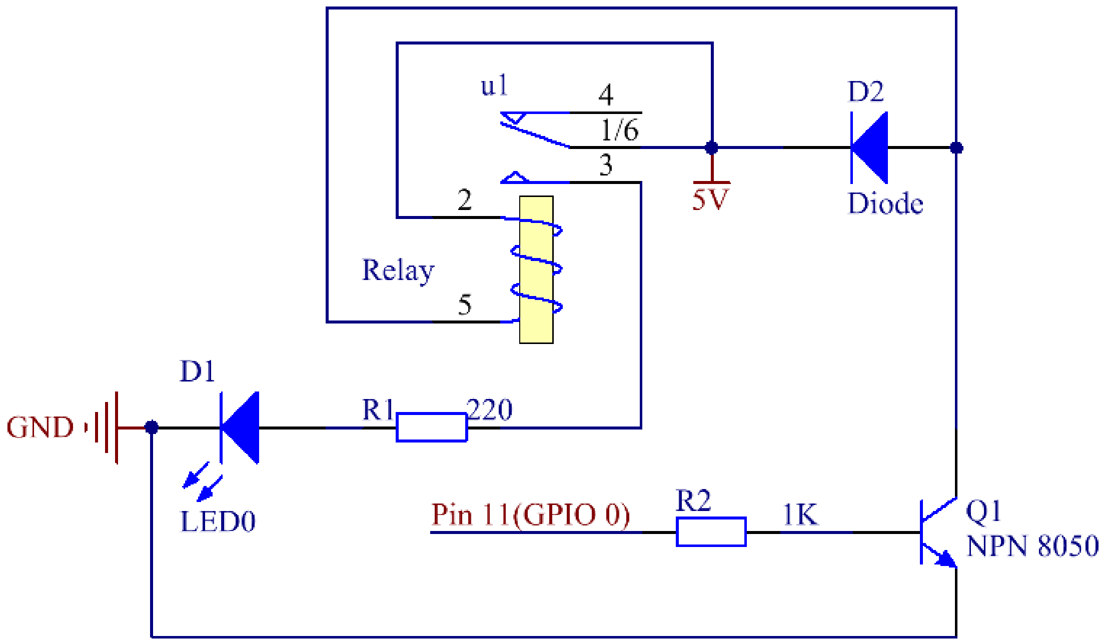
1N4007 is a semiconductor device for converting alternating current into direct current. By using the one-way conductivity of the diode, alternating current with alternating directions can be converted into a single-direction pulse direct current.

With a positive large current, 1N4007 has a low voltage drop (representative value 0.7 V) called as forward conduction state. If the opposite voltage is applied, the potential barrier is increased to withstand a high reverse voltage or to flow through a very small reverse current (called reverse leakage current) called as a reverse blocking state. Thus, the rectifier diode has a significant one-way conductivity. In this lesson, we apply this characteristic of diode.

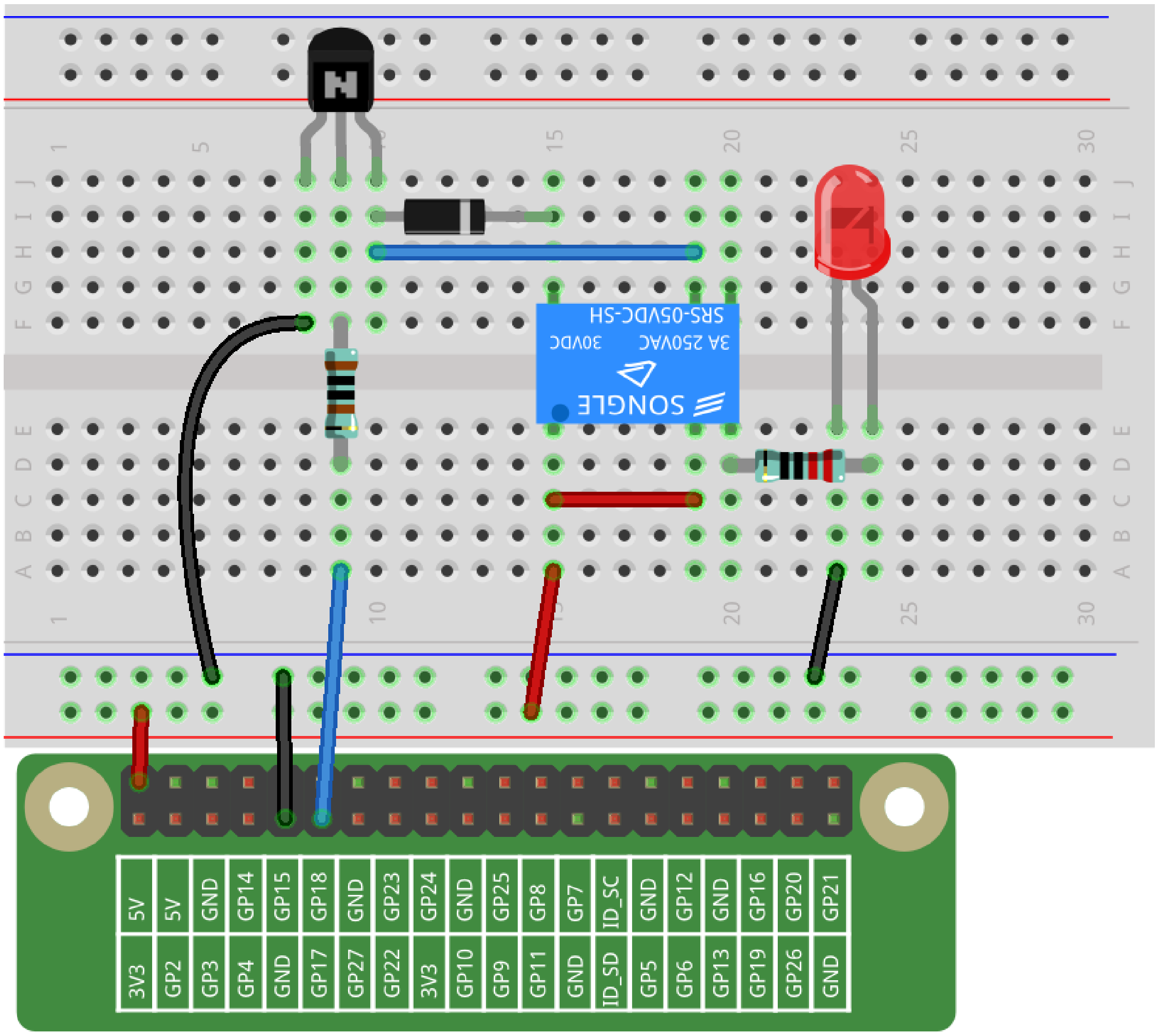
Schematic Diagram

When a high level signal is given to Pin 11, the transistor is energized, thus making the coil of the relay conductive. Then its normally open contact is closed, and the LED will light up. When Pin 11 is given a Low level, the LED will stay dim. In this experiment, we apply Freewheeling Diode that connects to both ends of the relay coil in parallel to prevent relay from breakdown or burnout caused by induced voltage.

wiringPi	Physical	BCM
0	Pin11	17



Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_8_Relay
```

2. Compile the code.

```
gcc 8_Relay.c -lwiringPi
```

3. Run the executable file.

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

Now, the LED will blink, you can hear a tick-tock caused by breaking the normally close contact and closing the normally open one.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define RelayPin 0

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

    pinMode(RelayPin, OUTPUT);

    while(1){
        // Tick
        printf("......Relay Open \n");
        digitalWrite(RelayPin, LOW);
        delay(1000);
        // Tock
        printf("Relay Close......\n");
        digitalWrite(RelayPin, HIGH);
        delay(1000);
    }
    return 0;
}
```

Code Explanation

```
17.         digitalWrite(RelayPin, LOW);
```

Set the I/O port **RelayPin** as **LOW** (0V), so the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens and the LED remains off.

```
21.         digitalWrite(RelayPin, HIGH);
```

Set the I/O port as **HIGH** (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes. Then you can see the LED is lit.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 8_Relay.py
```

Now, the LED is blinking, you can hear a tick-tock caused by breaking the normally closed contact and closing the normally open one.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

relayPin = 17

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
    while True:
        print ('...Relay open')
        # Tick
        GPIO.output(relayPin, GPIO.LOW)
        time.sleep(1)
        print ('Relay close...')
        # Tock
        GPIO.output(relayPin, GPIO.HIGH)
        time.sleep(1)

def destroy():
    # Turn off LED
    GPIO.output(relayPin, GPIO.LOW)
    # Release resource
    GPIO.cleanup()

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

(continues on next page)

(continued from previous page)

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

Code Explanation

```
9.    GPIO.setup(relayPin, GPIO.OUT, initial=GPIO.LOW)
```

Initialize pins. And the output pin of relay is set to output mode and default low level.

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

Wait for 1 second. Change the switching frequency of the relay by changing this parameter. Note: Relay is a kind of metal dome formed in mechanical structure. So its lifespan will be shortened under high-frequency using.

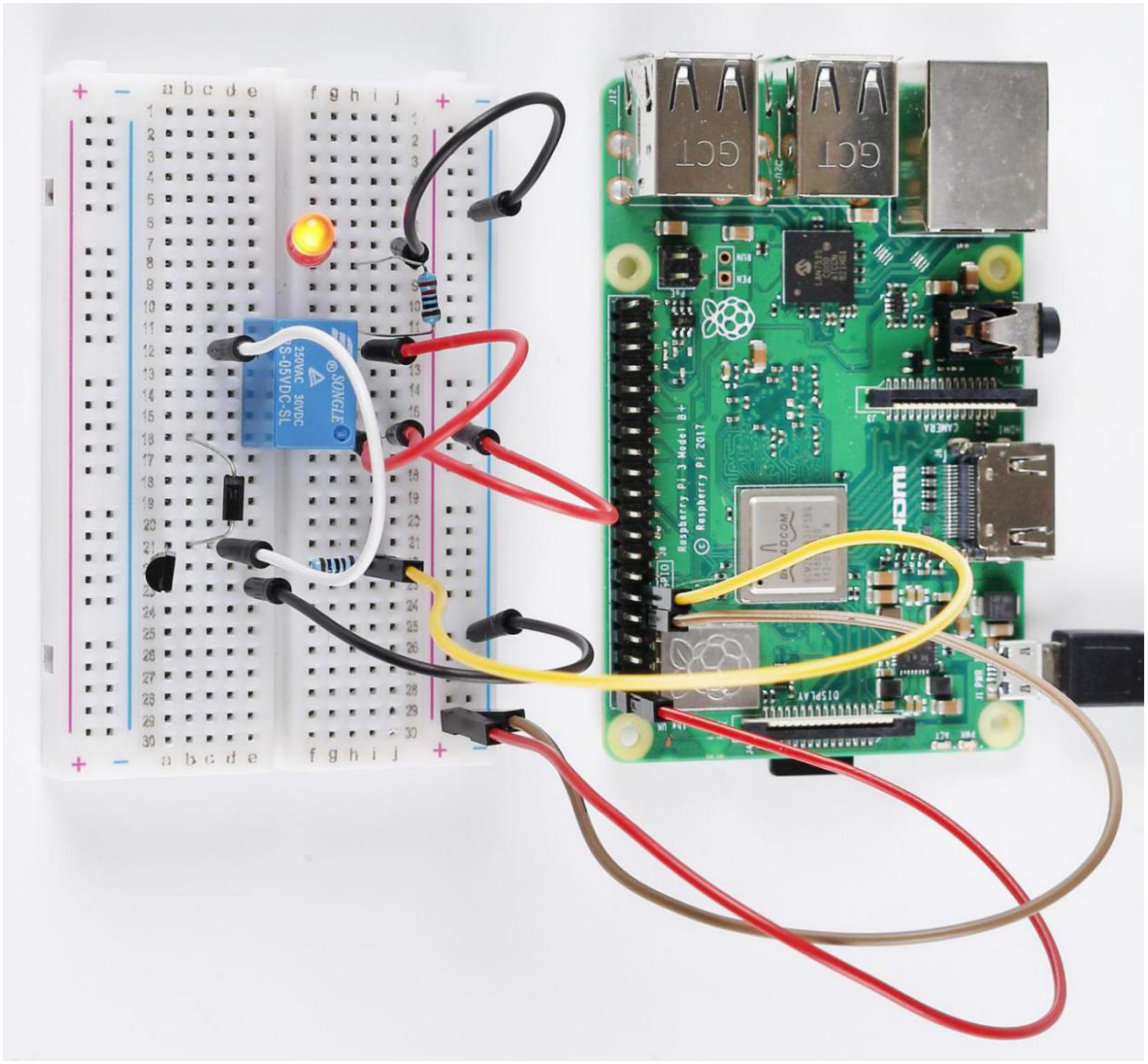
```
16.        GPIO.output(relayPin, GPIO.LOW)
```

Set the I/O port as low level (0V), thus the transistor is not energized and the coil is not powered. There is no electromagnetic force, so the relay opens and the LED remains off.

```
20.        GPIO.output(relayPin, GPIO.HIGH)
```

Set the I/O port as high level (5V) to energize the transistor. The coil of the relay is powered and generate electromagnetic force, and the relay closes. Then you can see the LED is lit.

Phenomenon Picture

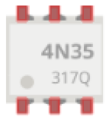
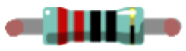
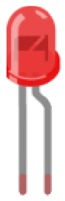


Lesson 9 4N35

Introduction

In this lesson, let's learn the operational principle of 4N35 chip. We analyze the principle in this way: using 4N35 chip to drive a LED, and then explaining the phenomenon of LED and the internal structure of the chip.

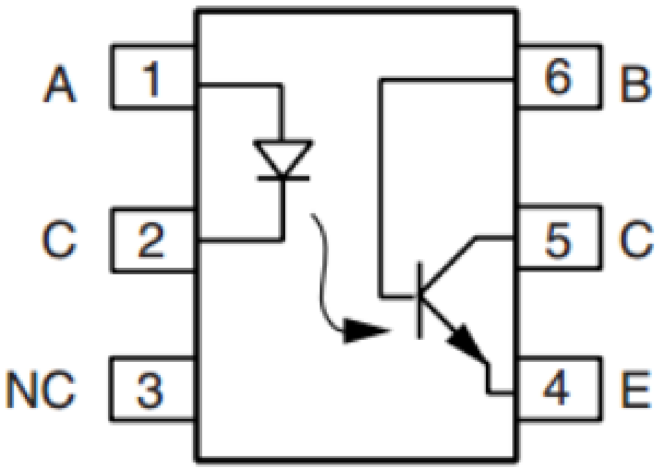
Newly Added Components

1 * 4N35	1 * Resistor (220Ω)	1 * LED	1 * Resistor(1kΩ)
			

Principle

4N35

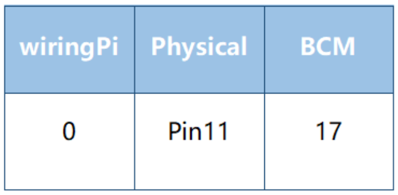
4N35 is a general-purpose optocoupler. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor. What an optocoupler does is to break the connection between signal source and signal receiver, so as to stop electrical interference. 4N35 can be used in AV conversion audio circuits that is widely used in electrical isolation of a general optocoupler.



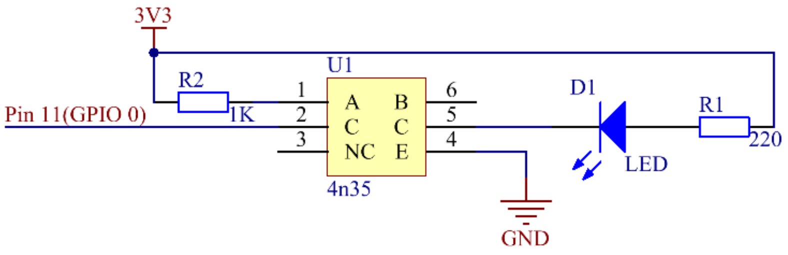
See the internal structure of the 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays, and then it can control the load connected to the phototransistor. Even when the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

Schematic Diagram

In this experiment, use an LED as the load connected to the NPN phototransistor. In program, a LOW level is given to **Pin 11**, then the infrared LED will emit infrared rays. After that, the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus turning on the LED.

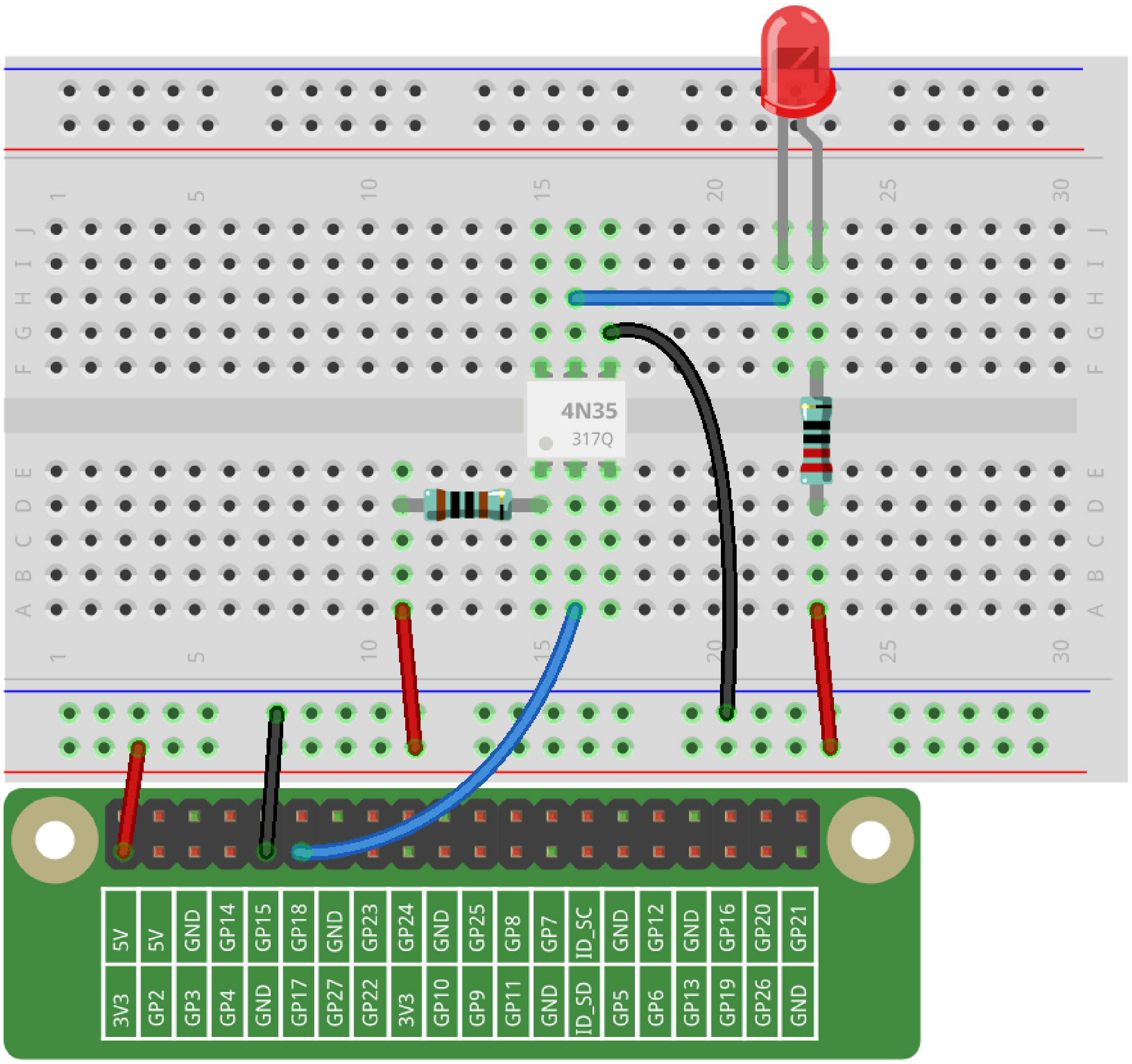


wiringPi	Physical	BCM
0	Pin11	17



Build the Circuit

Note: pay attention to the direction of the chip by the concave on it.



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_9_4N35
```

2. Compile the code.

```
gcc 9_4N35.c -lwiringPi
```

3. Run the executable file.

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

You will see the LED blinking.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define OptoPin  0

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

    pinMode(OptoPin,OUTPUT);

    while(1){
        // Turn LED off
        digitalWrite(OptoPin, HIGH);
        delay(500);
        // Turn LED on
        digitalWrite(OptoPin, LOW);
        delay(500);
    }
    return 0;
}
```

Code Explanation

```c
    pinMode(OptoPin,OUTPUT);
```

Initialize pins. Set the output pin of 4N35, Optopin to **OUTPUT** mode.

```c
        digitalWrite(OptoPin, HIGH);
```

Set **OptoPin** as **LOW** (0V), thus the optocoupler is energized, and the pin connected to LED conduct to low level. Then the LED will light up.

```c
        digitalWrite(OptoPin, LOW);
```

Set **OptoPin** as **HIGH** (3.3V), thus the optocoupler is not energized, and the pin connected to LED cannot conduct to low level. Then the LED goes out.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 9_4N35.py
```

You will see the LED blinking.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

Pin_4N35 = 17

# Define a setup function for some setup
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Pin_4N35, GPIO.OUT, initial=GPIO.LOW)

# Define a main function for main process
def main():
    while True:
        # Turn off LED
        GPIO.output(Pin_4N35, GPIO.HIGH)
        time.sleep(0.5)
        # Turn on LED
        GPIO.output(Pin_4N35, GPIO.LOW)
        time.sleep(0.5)

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

(continues on next page)

(continued from previous page)

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

Code Explanation

```
          GPIO.output(Pin_4N35, GPIO.HIGH)
```

Set **OptoPin** as **high** level (3.3V), thus the optocoupler is not energized, and the pin connected to LED cannot conduct to low level. Then the LED goes out.

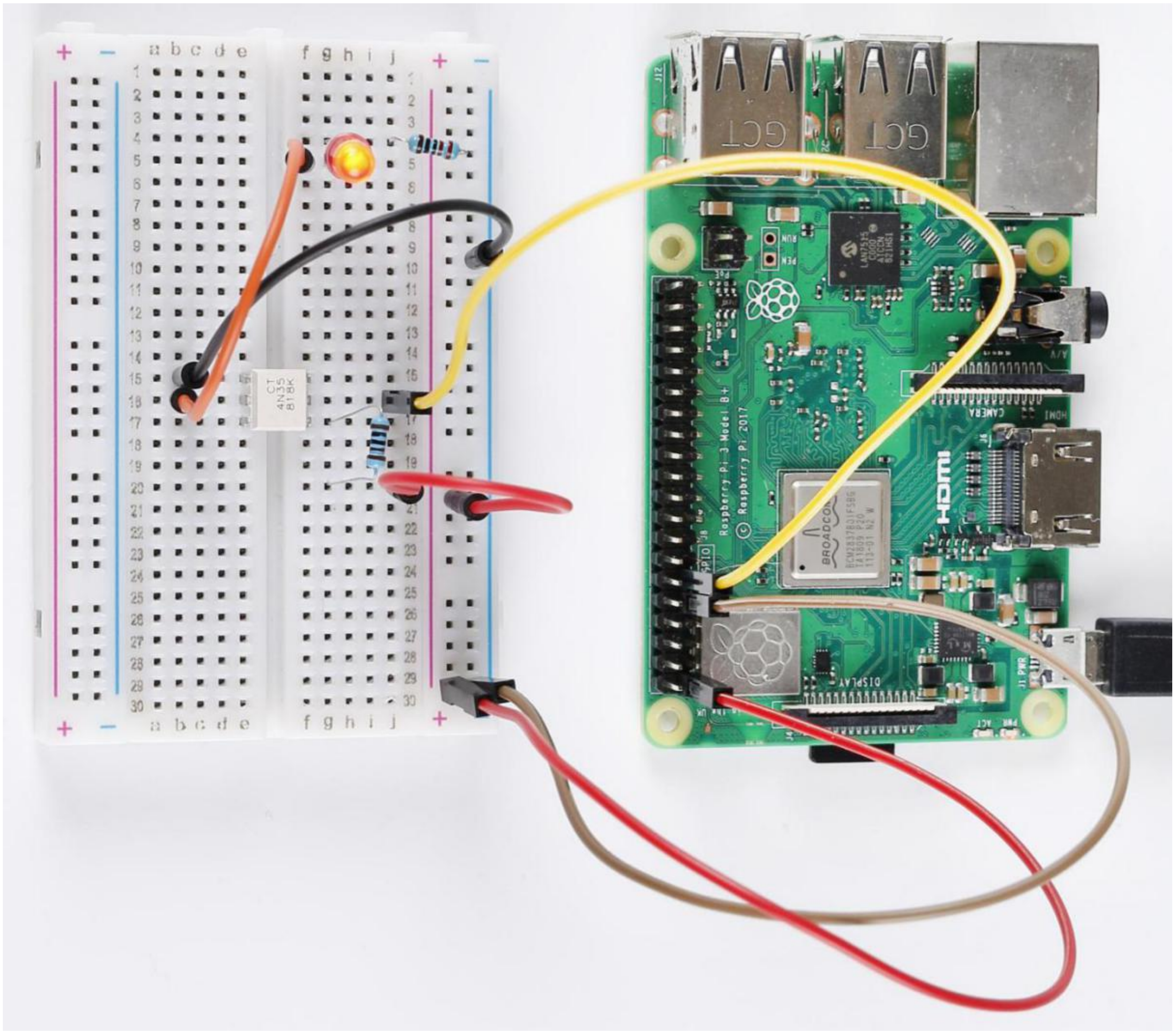
```
          time.sleep(0.5)
```

Wait for **0.5** second. The on-off frequency of the optocoupler can be changed by modifying this parameter.

```
          GPIO.output(Pin_4N35, GPIO.LOW)
```

Set **OptoPin** as low level (0V), thus the optocoupler is energized, and the pin connected to LED conduct to low level. Then the LED will light up.

Phenomenon Picture

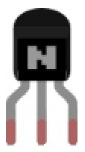
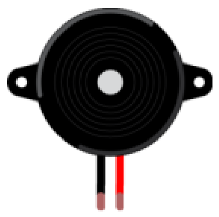


Lesson 10 Active Buzzer

Introduction

A buzzer is a great tool in your experiments whenever you want to make sounds.

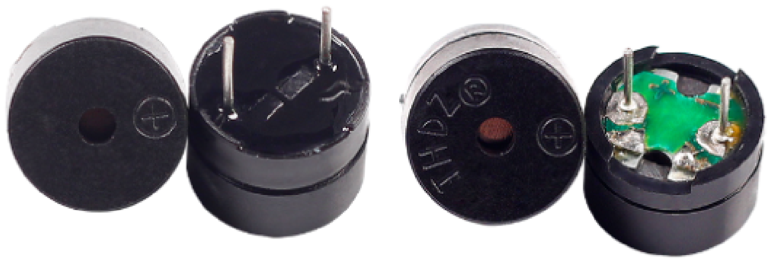
Newly Added Components

1 * NPN S8050	1 * Buzzer(Active)	1 * Resistor(1kΩ)
		

Principle

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

The difference between an active buzzer and a passive buzzer:



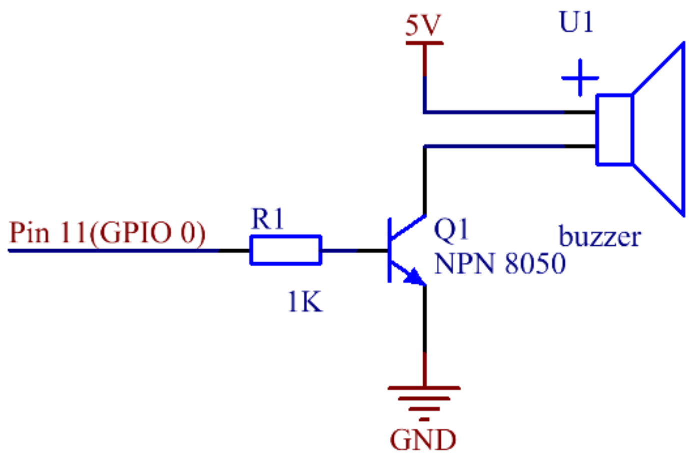
An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not tweet if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

In this experiment, we use an active buzzer.

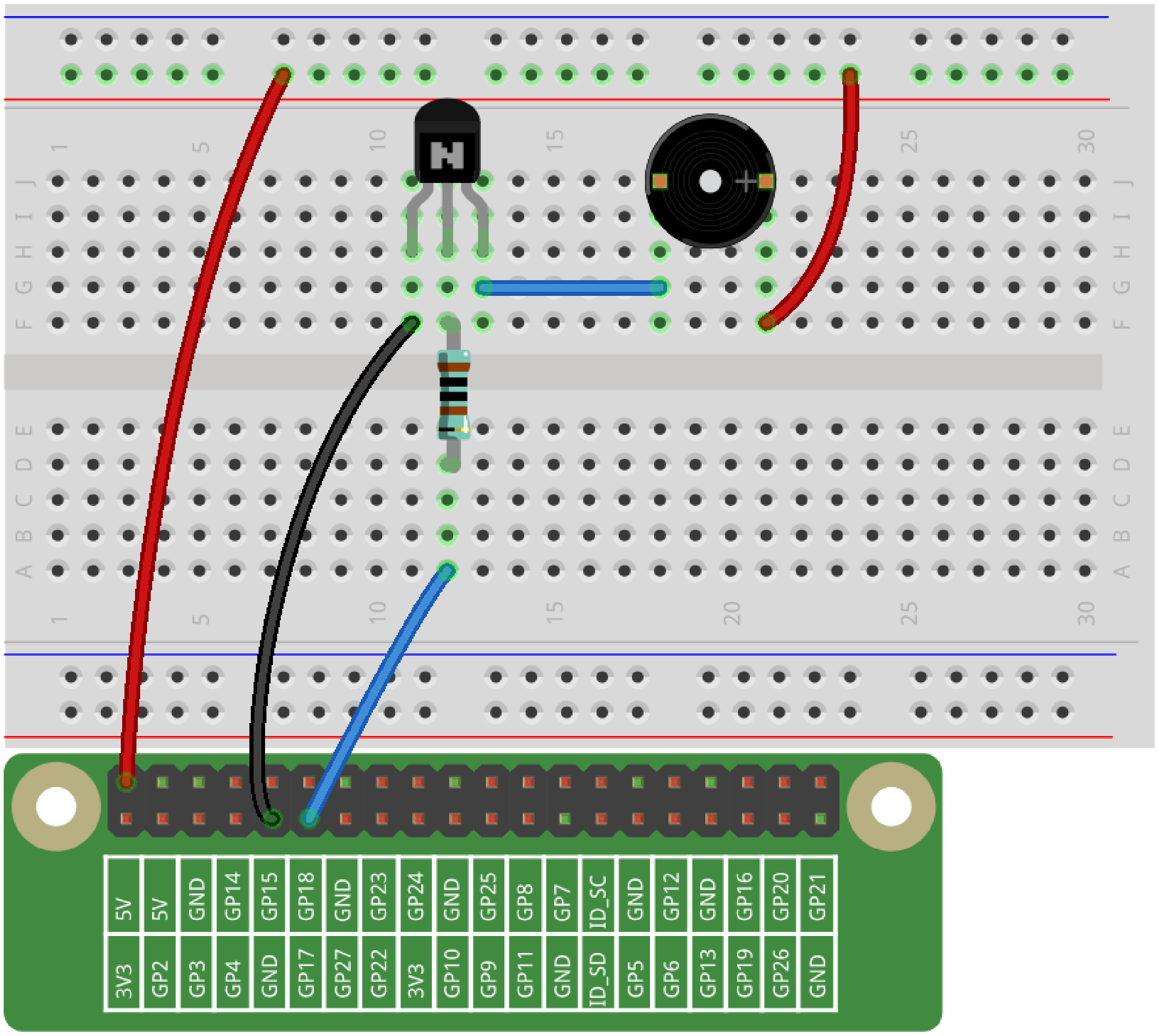
Schematic Diagram

When **Pin11** is input into high voltage, the transistor will be switched on, and the collector will output low level. When there is a level difference between the two pins of the buzzer, the buzzer is ringing. When **Pin11** inputs low power level, the transistor is cut off and the collector is at high level, here, both ends of the buzzer are at high level, the buzzer will be silent.

wiringPi	Physical	BCM
0	Pin11	17



Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_10_ActiveBuzzer
```

2. Compile the code.

```
gcc 10_ActiveBuzzer.c -lwiringPi
```

3. Run the executable file.

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

Now, you may hear the buzzer beep.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define BeepPin 0

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

    pinMode(BeepPin, OUTPUT);

    while(1){
        //beep on
        digitalWrite(BeepPin, HIGH);
        delay(100);
        //beep off
        digitalWrite(BeepPin, LOW);
        delay(100);
    }
    return 0;
}
```

Code Explanation

```
    pinMode(BeepPin, OUTPUT);
```

Set the pin connected to the buzzer to **OUTPUT** mode.

```
        digitalWrite(BeepPin, HIGH);
```

When BeepPin is at high level, the base pin(b pin) of the connected transistor inputs high level and the collector pin(c pin) output low level. That is, when the cathode of the buzzer is at low level and the anode of the buzzer is connected to a 5V high level, the buzzer sounds.

```
        digitalWrite(BeepPin, LOW);
```

The **BeepPin** is connected to the transistor and then to the cathode of the buzzer. When BeepPin is low level, the base pin (b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer is silent.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 10_ActiveBuzzer.py
```

Now, you should hear the buzzer beep.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

BeepPin = 17

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

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

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

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

Code Explanation

```
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
```

Initialize the pin connected to the buzzer to output mode and set it to the default low level.

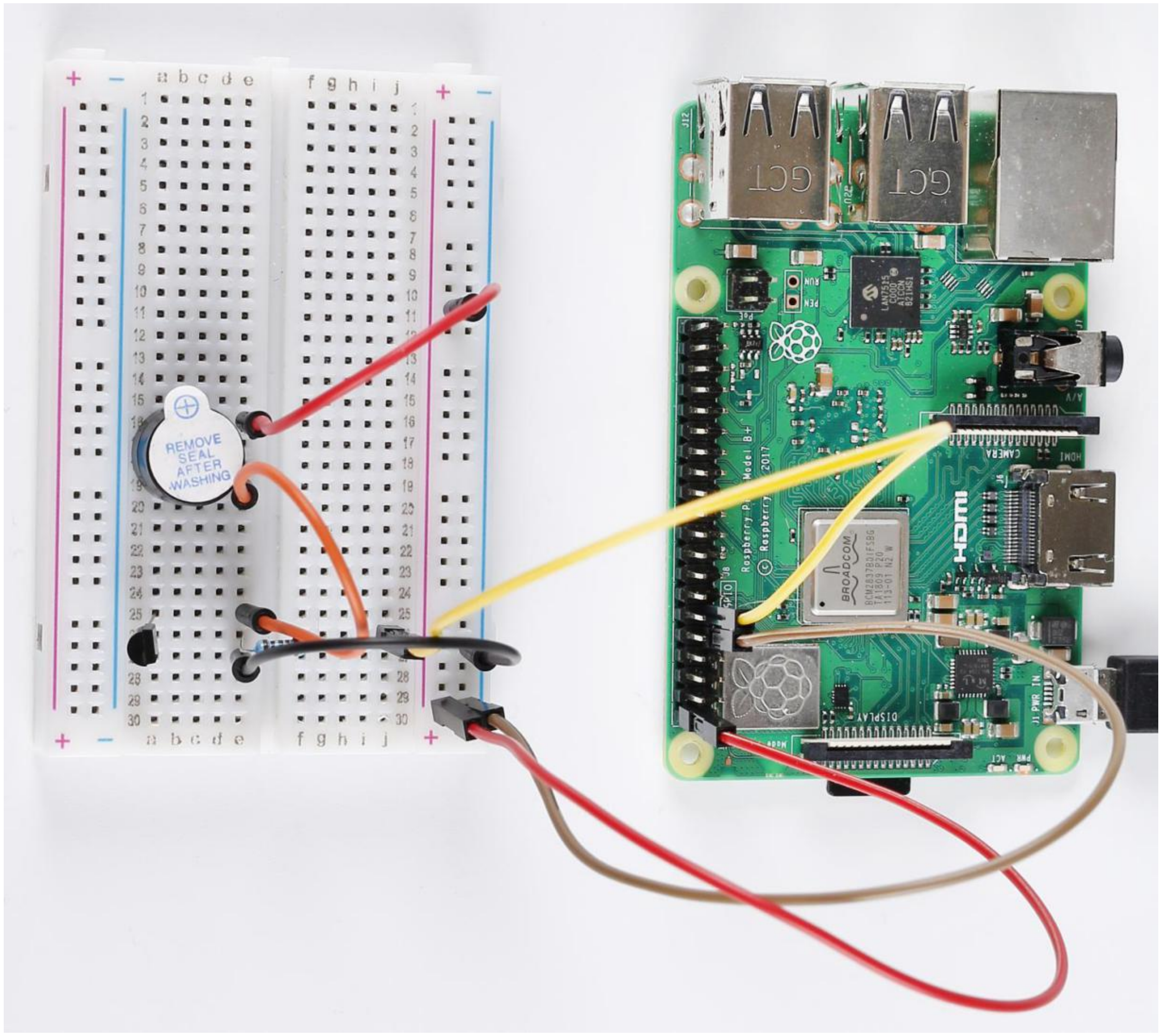
```
        GPIO.output(BeepPin, GPIO.HIGH)
```

The **BeepPin** is connected to the transistor and then to the cathode of the buzzer. When BeepPin is at high level, the base pin(b pin) of the connected transistor inputs high level, then the collector pin(c pin) outputs low level; that is, when the cathode of the buzzer is at low level and the anode of the buzzer is connected to a 5V high level, the buzzer sounds.

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

When **BeepPin** is at low level, the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer is silent.

Phenomenon Picture

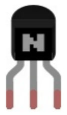
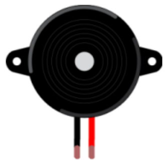


Lesson 11 Doorbell

Introduction

In this lesson, we will learn how to drive an active buzzer to build a simple doorbell.

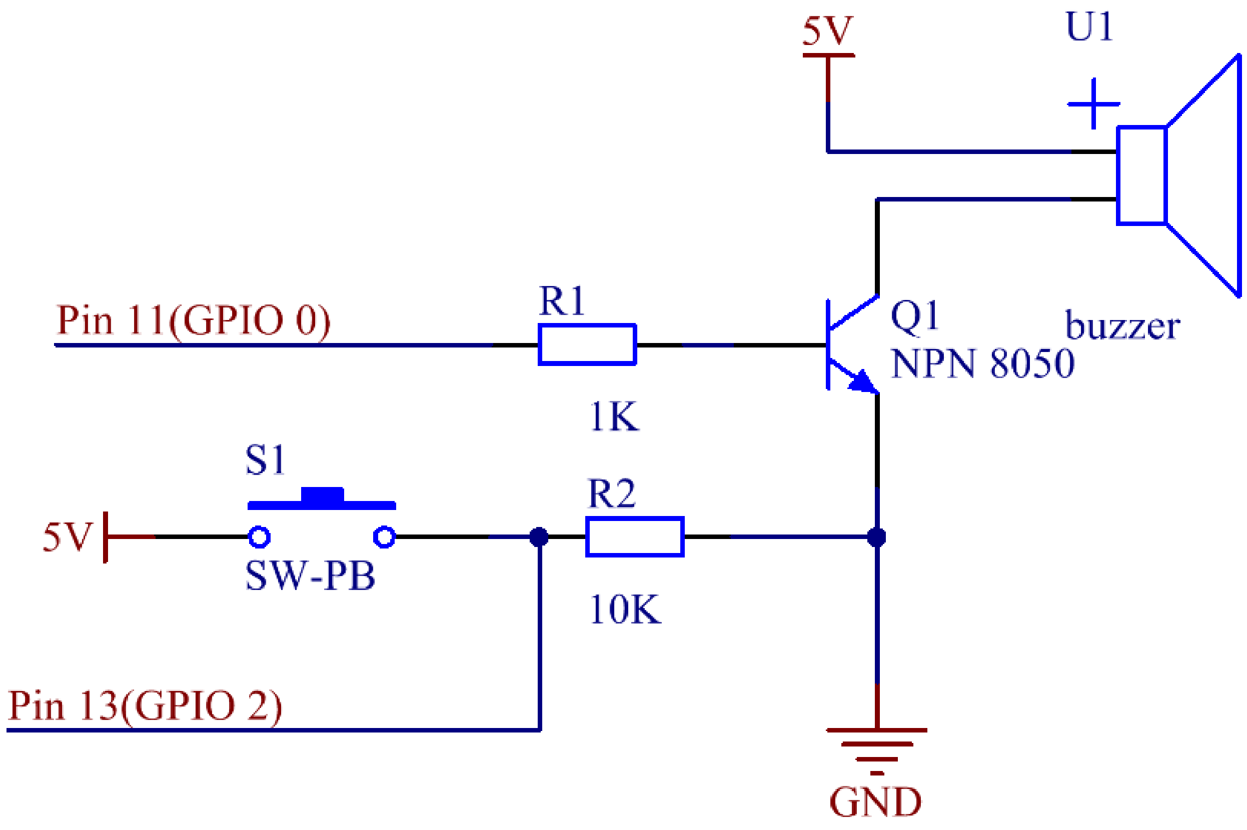
Newly Added Components

1 * NPN S8050	1 * Buzzer(Active)	1 * Resistor(1kΩ)	1 * Button
		1 * Resistor (10kΩ)	

Schematic Diagram

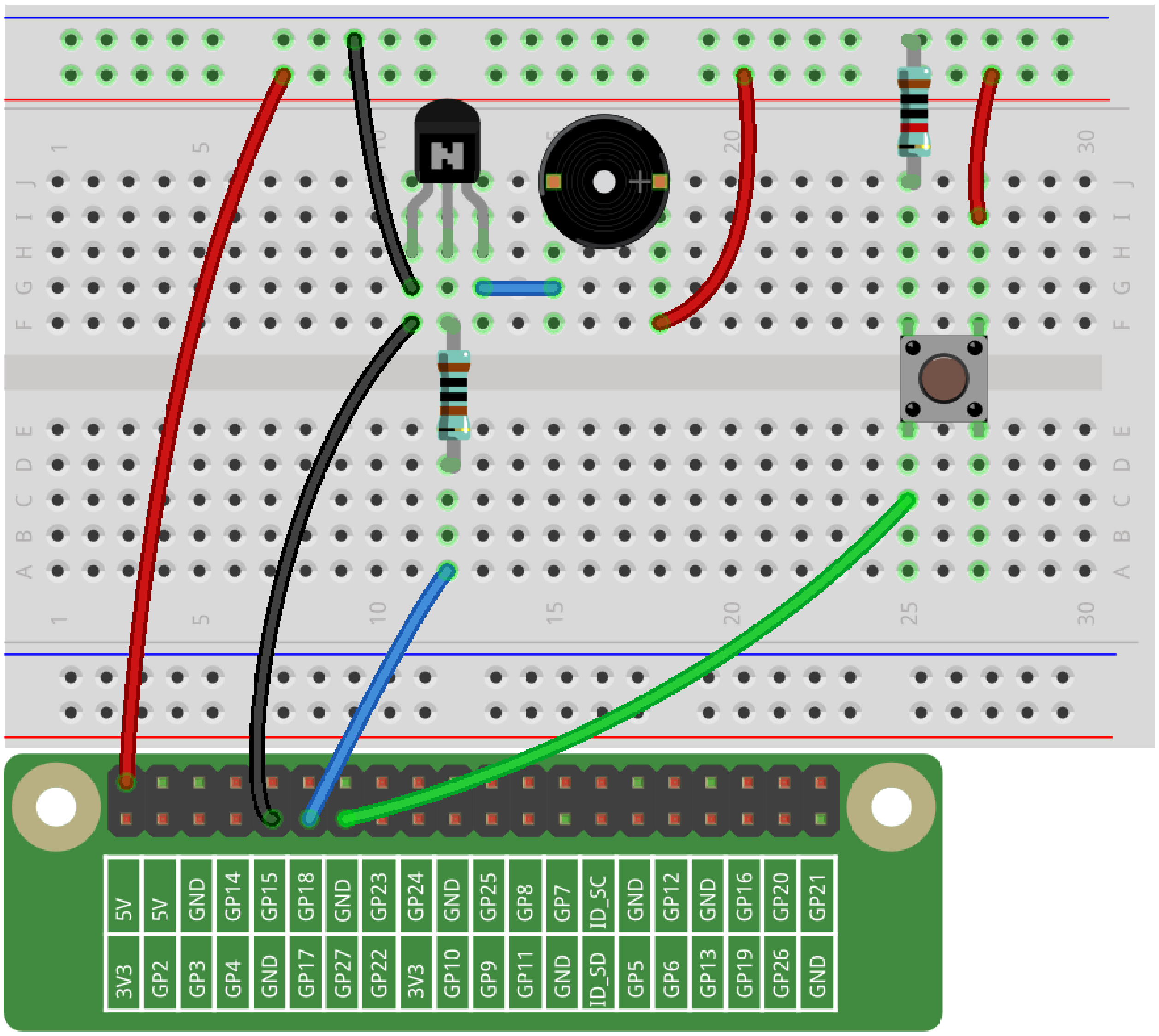
When the button is pressed, **Pin13** is connected to the **5V** power supply and reads out high level; therefore, the program responds to make **Pin11** output the high level so as to energize the transistor, and the collector will output low level that means the buzzer rings. When pin11 outputs at low level the buzzer will be silent.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27



Build the Circuit

Note: Long pins of buzzer is the Anode and the short pin is Cathode.



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_11_DoorBell
```

2. Compile the code.

```
gcc 11_DoorBell.c -lwiringPi
```

3. Run the executable file.

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

When the button is pressed, the buzzer makes a sound to simulate a doorbell.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define BeepPin 0
#define ButtonPin   2

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

    pinMode(BeepPin, OUTPUT);
    pinMode(ButtonPin, INPUT);
    pullUpDnControl(ButtonPin, PUD_DOWN);

    while(1){
        // Indicate that button has pressed down
        if(digitalRead(ButtonPin) == 1){
            delay(10);
            if(digitalRead(ButtonPin) == 1){
            //beep on
            printf("Buzzer on\n");
            digitalWrite(BeepPin, HIGH);
            delay(100);
            }
        }
        else{
            printf("Buzzer off\n");
            //beep off
            digitalWrite(BeepPin, LOW);
            delay(100);
        }
    }
    return 0;
}
```

Code Explanation

```
            delay(10);
```

Software removes button jitter. When the signal that the button is pressed is detected, a delay of 10ms is used to eliminate the possibility of false judgment. If both **if** conditions are met, confirm that the button is pressed, and then execute the program in if.

```
            if(digitalRead(ButtonPin) == 1){
            //beep on
            printf("Buzzer on\n");
            digitalWrite(BeepPin, HIGH);
            delay(100);
            }
```

If the button is recognized to be pressed, the **BeepPin** is at high level. The base pin(b pin) of the connected transistor inputs high level, while the collector pin(c pin) outputs low level. That is, the cathode of buzzer is at low level, and the anode is connected with a high level 5V. Then the buzzer rings.

```
        else{
            printf("Buzzer off\n");
            //beep off
            digitalWrite(BeepPin, LOW);
            delay(100);
        }
```

Otherwise, **BeepPin** is at low level, and the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, the level at both ends of the buzzer is high, and the buzzer does not ring.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 11_DoorBell.py
```

When the button is pressed, the buzzer makes a sound to simulate a doorbell.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

BeepPin = 17
BtnPin = 27

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BtnPin, GPIO.IN)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)

def main():
    while True:
        if GPIO.input(BtnPin) == 0:
            #Buzzer off
            print ('Buzzer Off')
            GPIO.output(BeepPin, GPIO.LOW)
            time.sleep(0.1)
        if GPIO.input(BtnPin) == 1:
            #Buzzer on
            print ('Buzzer On')
            GPIO.output(BeepPin, GPIO.HIGH)
            time.sleep(0.1)

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

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

Code Explanation

```python
        if GPIO.input(BtnPin) == 0:
            #Buzzer off
            print ('Buzzer Off')
            GPIO.output(BeepPin, GPIO.LOW)
            time.sleep(0.1)
```

If it is judged that the button is not pressed, BeepPin is at low level, and the base pin(b pin) of the connected transistor inputs low level, then the collector pin(c pin) outputs high level; that is, when the level at both ends of the connected buzzer is high, the buzzer does not ring.

```python
        if GPIO.input(BtnPin) == 1:
            #Buzzer off
            print ('Buzzer On')
```

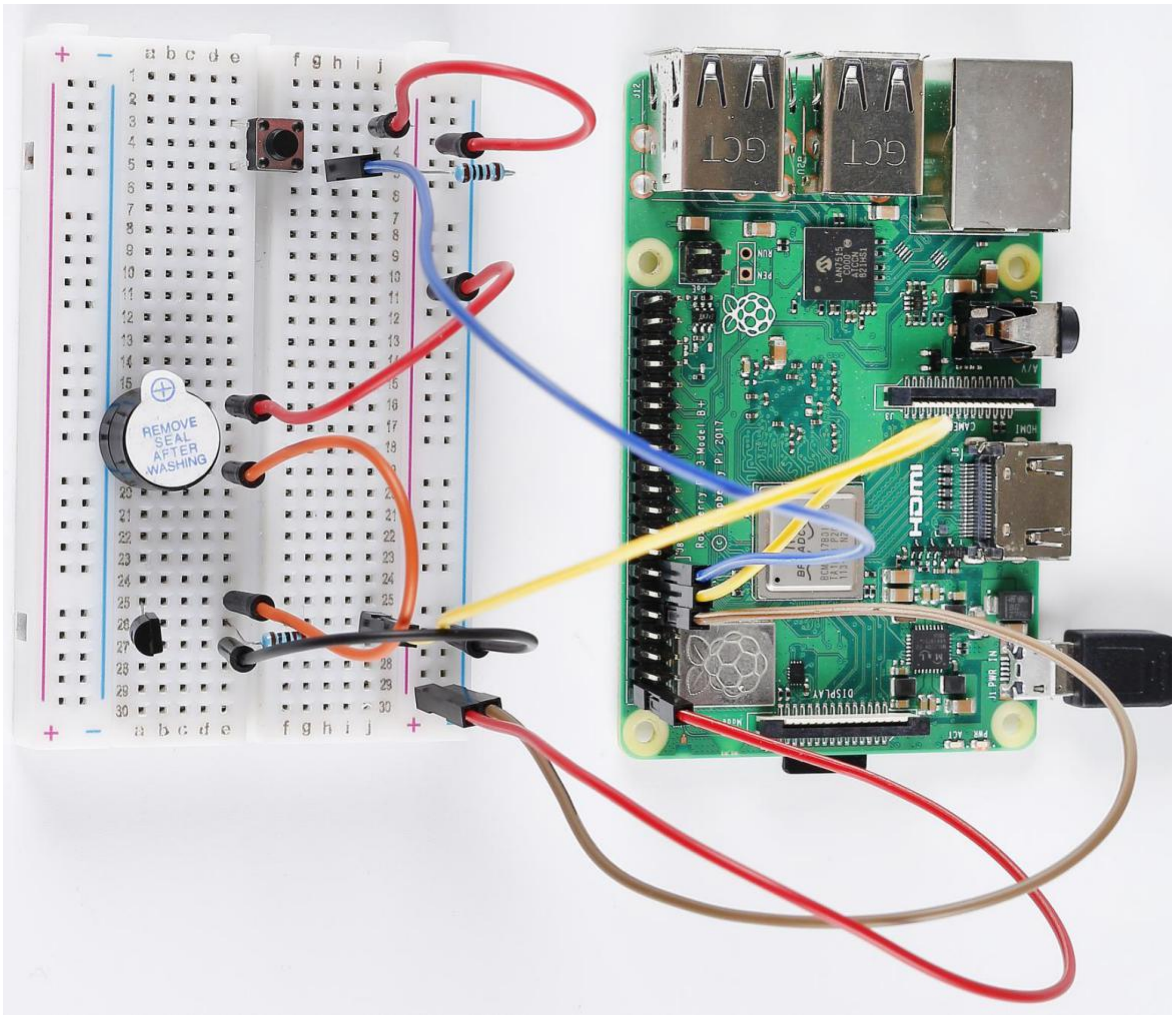
(continues on next page)

(continued from previous page)

```
            GPIO.output(BeepPin, GPIO.HIGH)
             time.sleep(0.1)
```

If the button is recognized to be pressed, the BeepPin is at high level. The base pin(b pin) of the connected transistor inputs high level, while the collector pin(c pin) outputs low level. That is, the cathode of buzzer is at low level, and the anode is connected with a high level 5V. Then the buzzer rings.

Phenomenon Picture

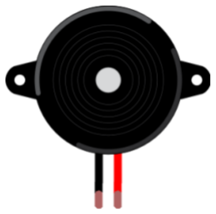
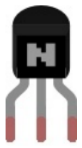


Lesson 12 Passive Buzzer

Introduction

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

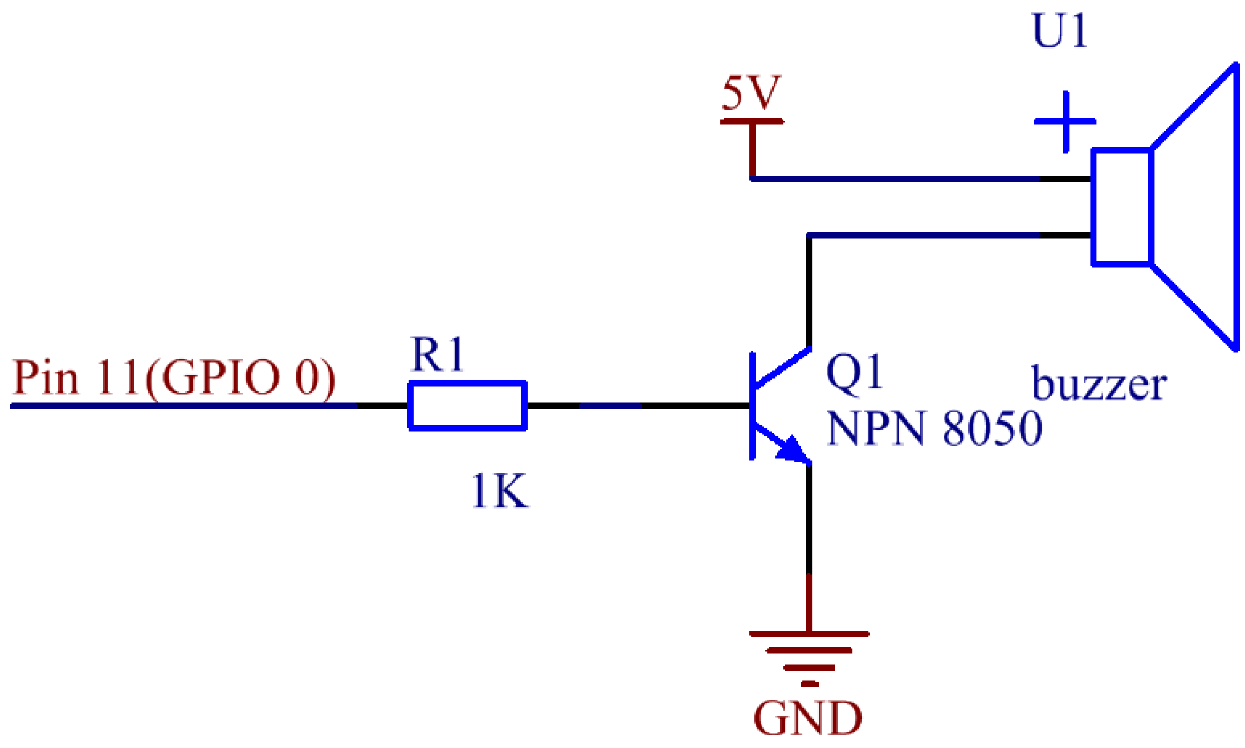
Newly Added Components

1 * Resistor(1kΩ)	1 * Buzzer(Passive)	1 * NPN S8050
		

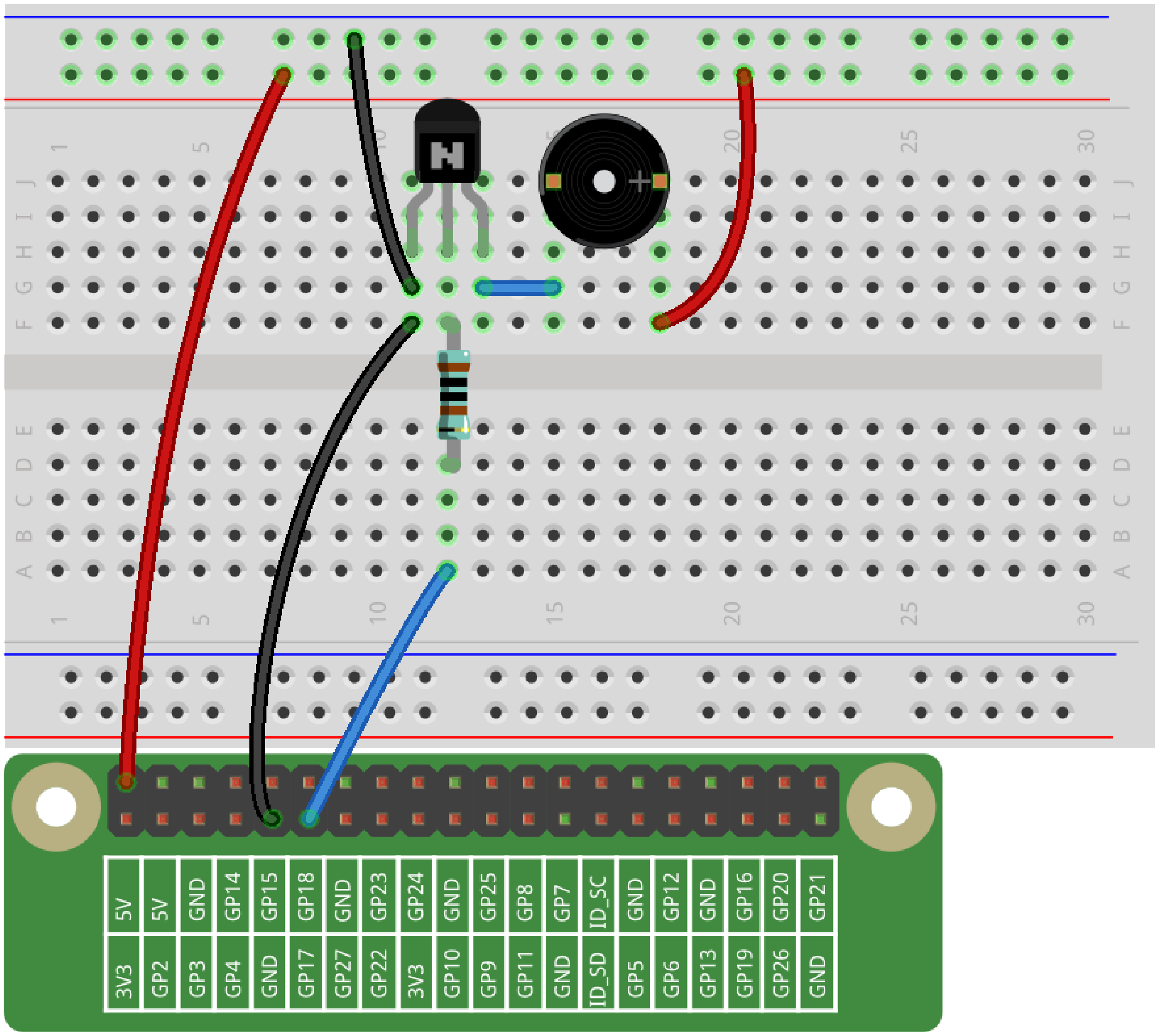
Schematic Diagram

The base pin(b pin) of the transistor is connected to pin11, the collector pin(c pin) to the cathode pin of the buzzer, and the emitter pin(e pin) to GND. The anode of the buzzer is connected to 5 v power supply. When pin11 inputs high voltage, the transistor will be switched on, and the collector will output low level. When there is a level difference between the two pins of the buzzer, the buzzer rings. When pin11 inputs low power level, the transistor is cut off, and the collector is at high level, and both ends of the buzzer are at high level, so the buzzer is silent.

wiringPi	Physical	BCM
0	Pin11	17


U1
5V
+
Pin 11(GPIO 0)
R1
1K
Q1
NPN 8050
buzzer
GND

Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_12_PassiveBuzzer
```

2. Compile the code.

```
gcc 12_PassiveBuzzer.c -lwiringPi
```

3. Run the executable file.

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

Now, the buzzer automatically plays music on a loop.

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

#define BuzPin    0

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

#define  CH1  525
#define  CH2  589
#define  CH3  661
#define  CH4  700
#define  CH5  786
#define  CH6  882
#define  CH7  990

int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};
int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};

int main(void)
{
    int i, j;

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

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

    while(1){
        printf("music is being played...\n");
        for(int i=0;i<sizeof(song)/4;i++){
            softToneWrite(BuzPin, song[i]);
            delay(beat[i] * 250);
        }
```

(continues on next page)

(continued from previous page)

```
    }
    return 0;
}
```

Code Explanation

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

WiringPi includes a software-driven sound handler capable of outputting a simple tone/square wave signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is 100S. That gives a maximum frequency of 1/0.0002 = 5000Hz. Within these limitations, simple tones on a high impedance speaker or piezo sounder is possible.

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

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

```
int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};
int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};
```

Define a section of music and the corresponding beat. The number in **beat[]** refers to the beat of each note in the **song** (0.5s for each beat).

```
    if(softToneCreate(BuzPin) == -1){
```

softToneCreate( ) creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the **wiringPiSetup()** function you used. The return value is 0 for success. This is used to determine whether it is successful for the software to control tone pin; if it fails, it will not execute the program.

```
        for(int i=0;i<sizeof(song)/4;i++){
            softToneWrite(BuzPin, song[i]);
            delay(beat[i] * 250);
        }
```

Employ a for statement to play song_1.In the judgment condition, **i<sizeof(song_1)/4**"devide by 4" is used because the array song_1[] is an array of the data type of integer, and each element takes up four bytes.

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

To enable each note to play for beat * 500ms, the function **delay(beat_1[i] * 500)** is called.

The prototype of softToneWrite(BuzPin, song_1[i])

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

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

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 12_PassiveBuzzer.py
```

Now, the buzzer automatically plays music on a loop.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

Buzzer = 17

CL = [0, 131, 147, 165, 175, 196, 211, 248]        # Frequency of Low C notes
CM = [0, 262, 294, 330, 350, 393, 441, 495]        # Frequency of Middle C notes
CH = [1, 525, 589, 661, 700, 786, 882, 990]        # Frequency of High C notes

song = [    CH[5], CH[2], CM[6], CH[2], CH[3], CH[6],CH[0], CH[3], # Notes of song
            CH[5], CH[3], CM[6], CH[2],CH[0]]

beat = [    1,1,1,1,1,1,2,1,1,1,1,1,3    ]

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(Buzzer, GPIO.OUT)
    global Buzz



def loop():
    while True:
        print ('\n    Playing song...')
        for i in range(1, len(song)):
            if  song[i] == 1 :
                time.sleep(beat[i] *0.25)
            else:
                Buzz = GPIO.PWM(Buzzer, song[i])
                Buzz.start(50)
                time.sleep(beat[i] * 0.25)
                Buzz.stop()
        time.sleep(1)             # Wait a second for next song.

def destory():
```

(continues on next page)

(continued from previous page)

```
    Buzz.stop()
    GPIO.output(Buzzer, LOW)
    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.
        destory()
```

Code Explanation

```
CL = [0, 131, 147, 165, 175, 196, 211, 248]
CM = [0, 262, 294, 330, 350, 393, 441, 495]
CH = [1, 525, 589, 661, 700, 786, 882, 990]
```

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

```
int song[] = {CH5,CH2,CM6,CH2,CH3,CH6,0,CH3,CH5,CH3,CM6,CH2,0};
int beat[] = {1,1,1,1,1,1,2,1,1,1,1,1,3};
```

Define a section of music and the corresponding beats. The number in **beat[]** refers to the beat of each note in the **song** (0.5s for each beat).

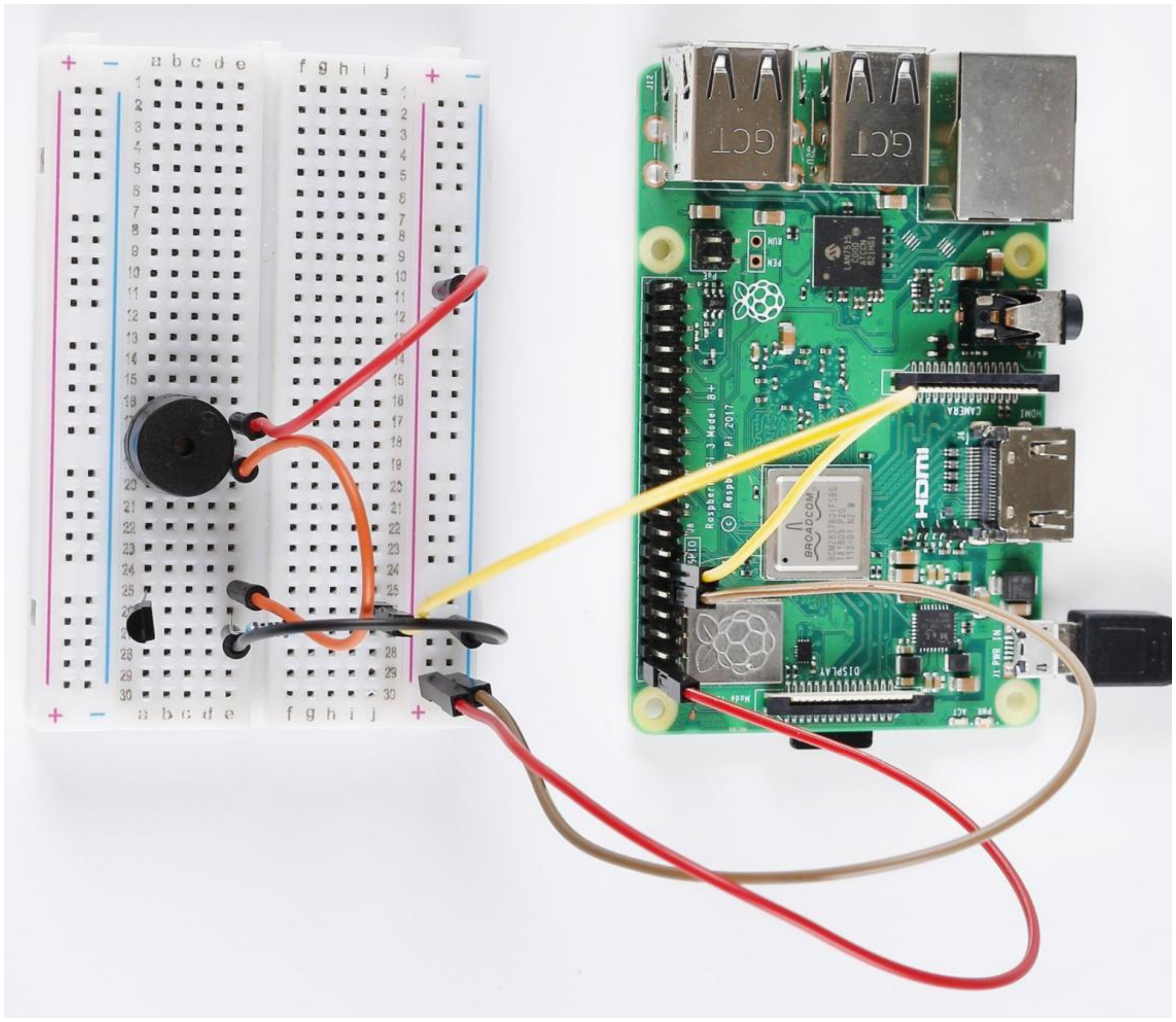
```
    Buzz = GPIO.PWM(Buzzer, song[i])
    Buzz.start(50)
```

Define pin Buzzer as PWM pin, then set its frequency to 786(song[0]) and **Buzz.start(50)** is used to run PWM. What's more, set the duty cycle to 50%.

```
def loop():
   while True:
       print ('\n    Playing song...')
       for i in range(1, len(song)):
           if  song[i] == 1 :
               time.sleep(beat[i] *0.25)
           else:
               Buzz = GPIO.PWM(Buzzer, song[i])
               Buzz.start(50)
               time.sleep(beat[i] * 0.25)
               Buzz.stop()
  time.sleep(1)
```

Play music in the while loop. As i increases gradually, the buzzer plays following the note in song[].

Phenomenon Picture

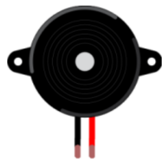
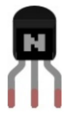


Lesson 13 Button Piano

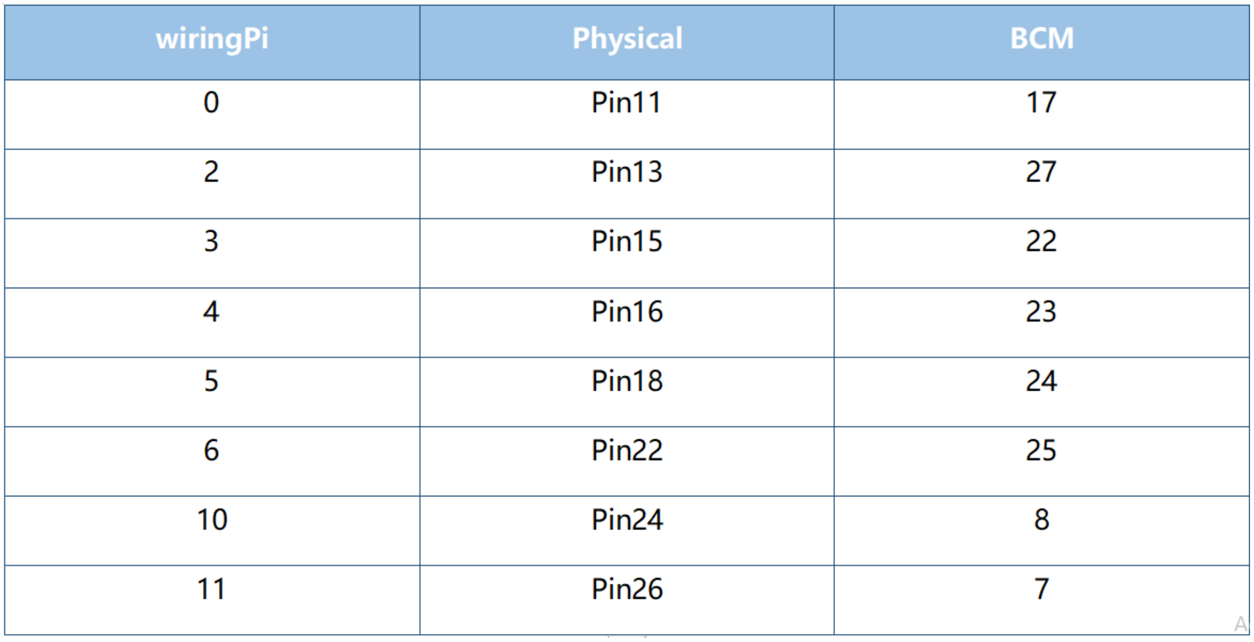
Introduction

In our past lesson, we learned how to use PWM waves to drive a passive buzzer to ring. In this lesson, we make a simple keyboard by applying a passive buzzer. Let's get started!

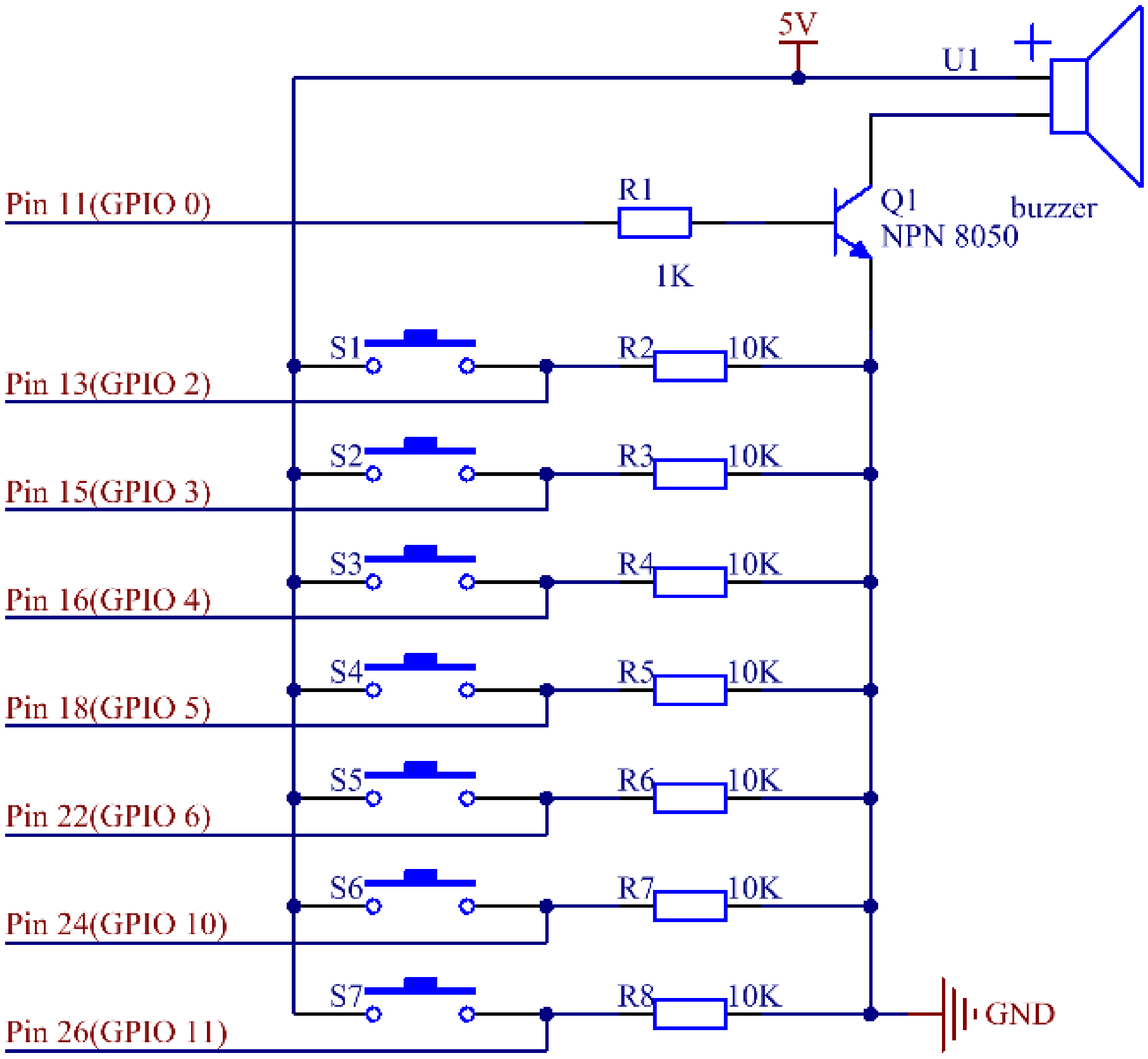
Newly Added Components

8 * Resistor (10kΩ)	7 * Button	1 * Buzzer(Passive)	1 * NPN S8050
			

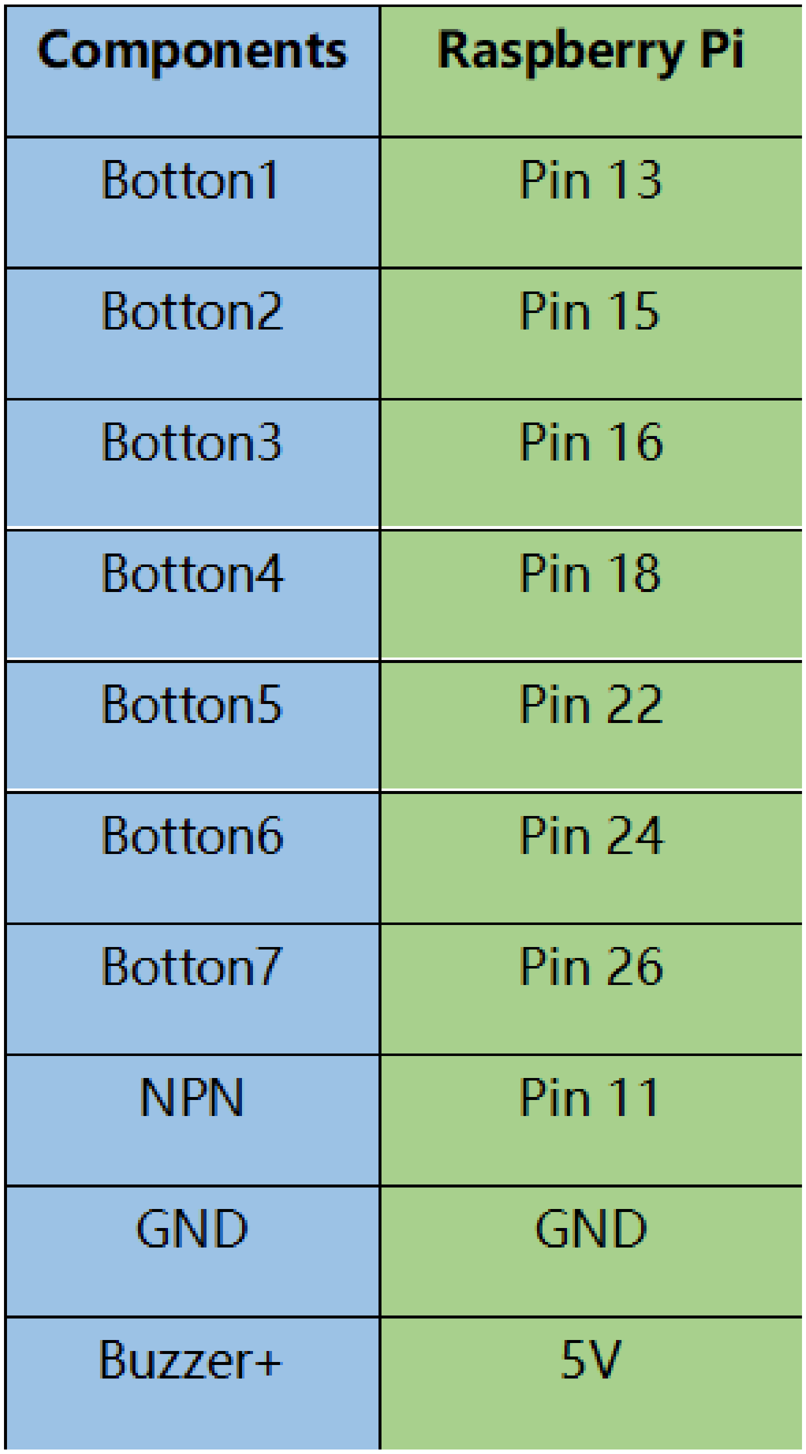
Schematic Diagram



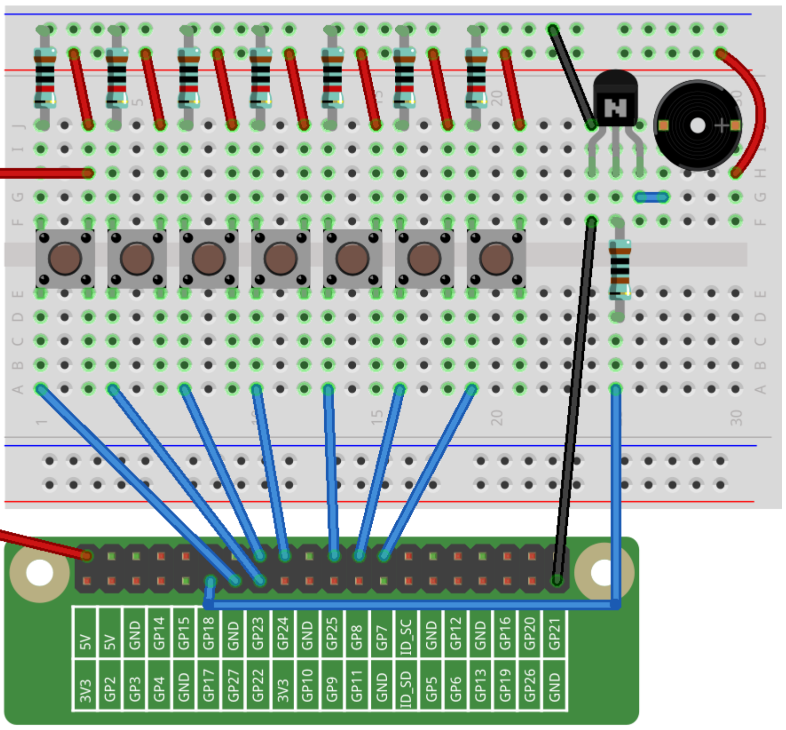
wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24
6	Pin22	25
10	Pin24	8
11	Pin26	7


5V
U1
+
buzzer
R1
1K
Q1
NPN 8050
Pin 11(GPIO 0)
Pin 13(GPIO 2)
Pin 15(GPIO 3)
Pin 16(GPIO 4)
Pin 18(GPIO 5)
Pin 22(GPIO 6)
Pin 24(GPIO 10)
Pin 26(GPIO 11)
S1
S2
S3
S4
S5
S6
S7
R2 10K
R3 10K
R4 10K
R5 10K
R6 10K
R7 10K
R8 10K
GND

Build the Circuit



Components	Raspberry Pi
Botton1	Pin 13
Botton2	Pin 15
Botton3	Pin 16
Botton4	Pin 18
Botton5	Pin 22
Botton6	Pin 24
Botton7	Pin 26
NPN	Pin 11
GND	GND
Buzzer+	5V



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_13_Button_Piano
```

2. Compile the code.

```
gcc 13_ButtonPiano.c -lwiringPi
```

3. Run the executable file.

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

Now press the seven buttons, and the buzzer will emit the notes: DO, RE, MI, FA, SO, LA, TI. You can play a song with these seven buttons.

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

#define BuzPin    0

const int Tone[] = {262,294,330,350,393,441,495};//define DO, RE, MI, FA, SO, LA, TI
int beat[] = {1,1,1,1,1,1,1};
const int Btn[] = {2,3,4,5,6,10,11};//define 7 buttons

int main(void)
{
    int i, j;

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

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

    //set the buttons mode
    for(int j=0;j<7;j++)
    {
        pinMode(Btn[j], INPUT);
    }

    while(1){
        //printf("Please press button to play the piano\n");
        // Indicate that button has pressed down
        for(i=0;i<7;i++)
        {
            if(digitalRead(Btn[i])==1)
            {
                delay(10);//Prevent the button' s vibration
                if(digitalRead(Btn[i])==1)
                {
                    softToneWrite(BuzPin, Tone[i]);
                    delay(beat[i]*250);
                    printf("1");
                }
            }
            else
                softToneWrite(BuzPin, 0);
            if(i==7)
                i=0;
        }
    }
    return 0;
}
```

Code Explanation

```
const int Tone[] = {262,294,330,350,393,441,495};
int beat[] = {1,1,1,1,1,1,1};
```

In the array **Tone[]**, define the frequencies of DO, RE, MI, FA, SO, LA, TI and the number in **beat[]** refers to the beat of each note in this song(0.5s for each beat).

```
    for(int j=0;j<7;j++)
    {
        pinMode(Btn[j], INPUT);
    }
```

Set the mode of all buttons to input mode in the for loop.

```
        for(i=0;i<7;i++)
        {
            if(digitalRead(Btn[i])==1)
            {
                delay(10);//Prevent the button' s vibration
                if(digitalRead(Btn[i])==1)
                {
                    softToneWrite(BuzPin, Tone[i]);
                    delay(beat[i]*250);
                    printf("1");
                }
            }
```

Use a **for** loop to check all the buttons. When one button in array **Btn[i]** is detected to be pressed, the buzzer will respond to the corresponding note in array **Tone[i]**.

```
            else
                softToneWrite(BuzPin, 0);
            if(i==7)
                i=0;
        }
```

If no button is pressed, turn off the buzzer.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 13_ButtonPiano.py
```

Now press the seven buttons, and the buzzer will emit the notes: DO, RE, MI, FA, SO, LA, TI. You can play a song with these seven buttons.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

Buzzer = 17
BtnPin = [18,27,22,23,24,25,8,7]

CL = [0, 131, 147, 165, 175, 196, 211, 248]        # Frequency of Low C notes
CM = [0, 262, 294, 330, 350, 393, 441, 495]        # Frequency of Middle C notes
CH = [1, 525, 589, 661, 700, 786, 882, 990]        # Frequency of High C notes

song = [     0,CM[1],CM[2],CM[3],CM[4],CM[5],CM[6],CM[7]    ]
beat = [    1,1, 1, 1, 1, 1, 1,  1]

def setup():
    GPIO.setmode(GPIO.BCM)
    for i in range(1, len(BtnPin)):
        GPIO.setup(BtnPin[i],GPIO.IN)
    GPIO.setup(Buzzer, GPIO.OUT)

def loop():
    global Buzz
    while True:
        #print ('\n    Please playing piano...')
        for i in range(1, len(BtnPin)):
            if GPIO.input(BtnPin[i]) == 1:
                Buzz = GPIO.PWM(Buzzer, song[i])
                Buzz.start(50)
                time.sleep(beat[i] * 0.25)
                Buzz.stop()

def destory():
    Buzz.stop()
    GPIO.output(Buzzer, 0)
    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.
        destory()
```

Code Explanation

```
CL = [0, 131, 147, 165, 175, 196, 211, 248]        # Frequency of Low C notes
CM = [0, 262, 294, 330, 350, 393, 441, 495]        # Frequency of Middle C notes
CH = [1, 525, 589, 661, 700, 786, 882, 990]        # Frequency of High C notes
```

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

```
song = [     0,CM[1],CM[2],CM[3],CM[4],CM[5],CM[6],CM[7]     ]
beat = [    1,1, 1, 1, 1, 1, 1,   1]
```

Define a section of music and the corresponding beats. The number in beat[] refers to the beat of each note in the song(0.5s for each beat).

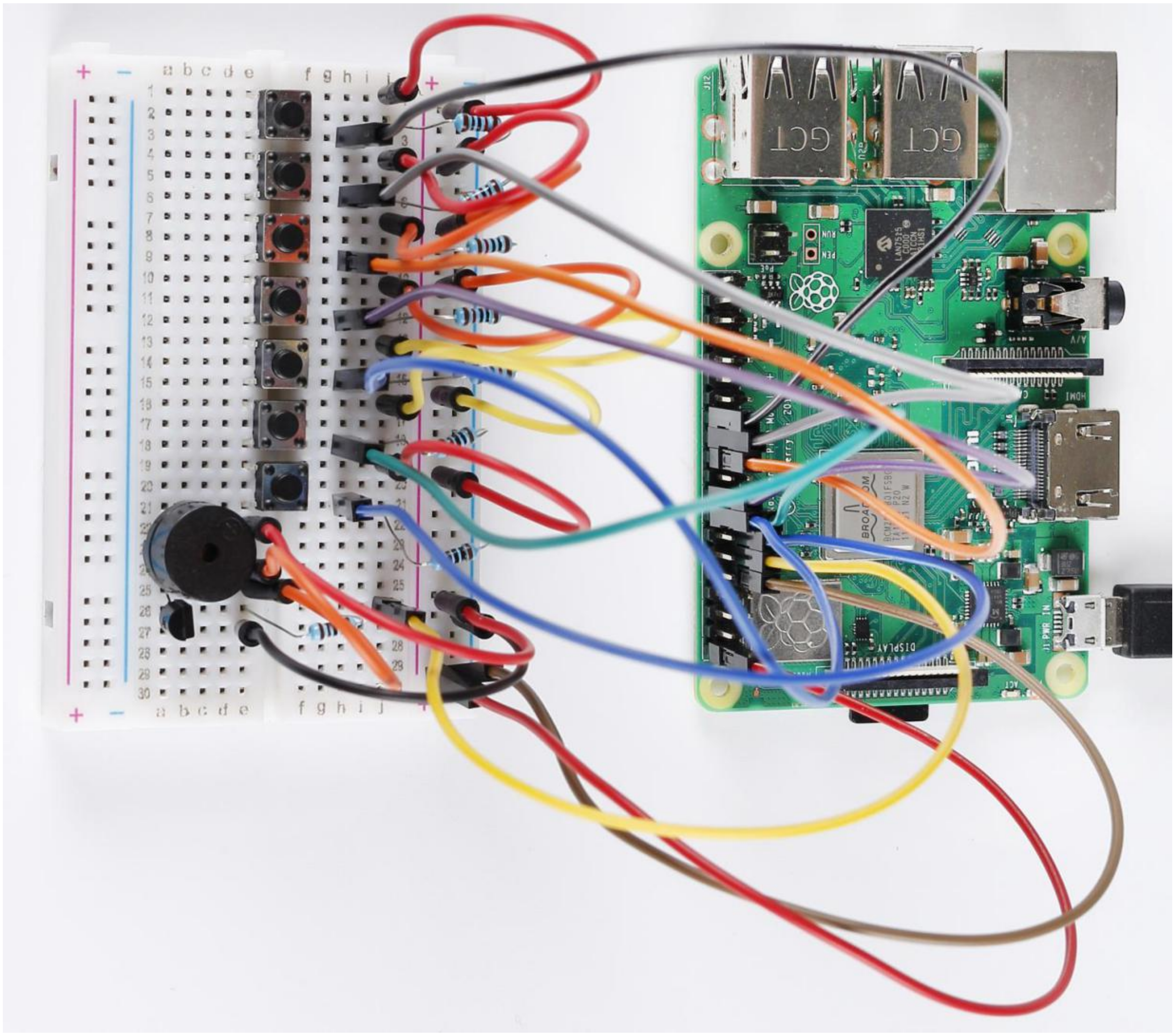
```
    for i in range(1, len(BtnPin)):
        GPIO.setup(BtnPin[i],GPIO.IN)
```

Set the mode of all buttons to input mode in the for loop.

```
        for i in range(1, len(BtnPin)):
            if GPIO.input(BtnPin[i]) == 1:
                Buzz = GPIO.PWM(Buzzer, song[i])
                Buzz.start(50)
                time.sleep(beat[i] * 0.25)
                Buzz.stop()
```

Use a for loop to check all the buttons. When one button in array **button[i]** is detected to be pressed, the buzzer will respond to the corresponding note in array **song[i]**.

Phenomenon Picture

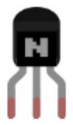
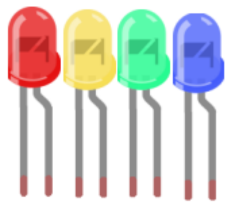
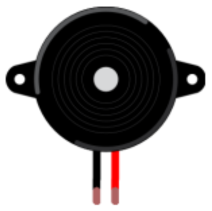


Lesson 14 Quiz Buzzer System

Introduction

In quiz shows, especially entertainment activities (e.g. competitive answering activities), organizers often apply a quiz buzzer system in order to accurately, fairly and visually determine the seat number of a responder. In this lesson, we will use some buttons, buzzers, and LEDs to make a quiz buzzer system.

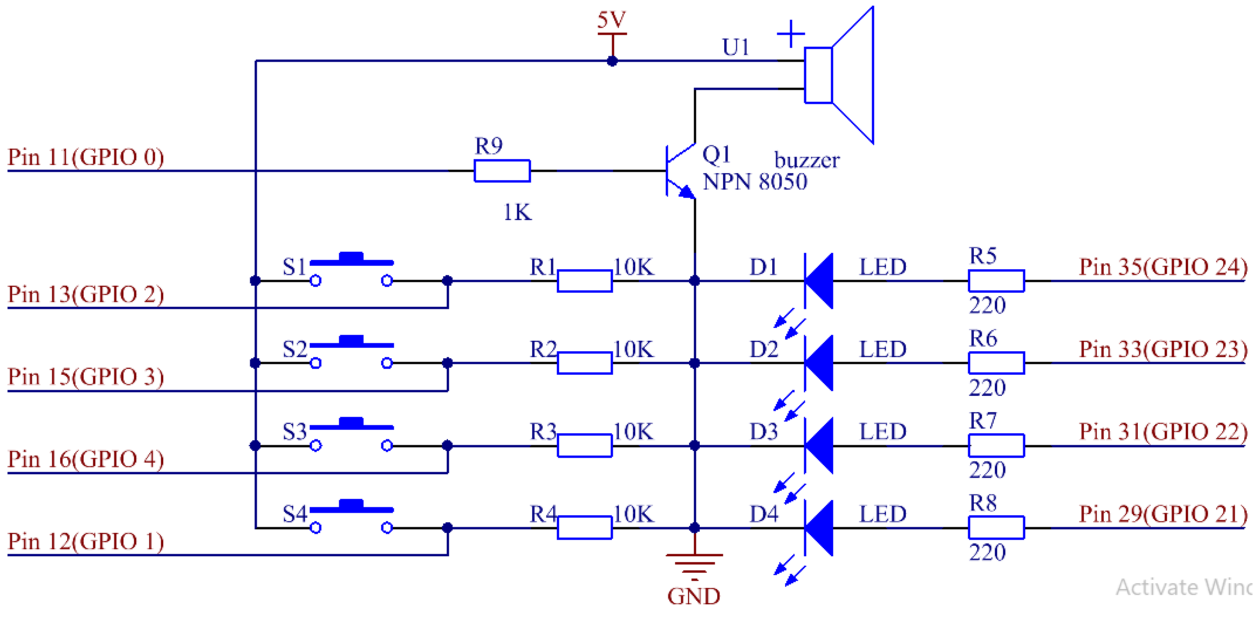
Newly Added Components

1 * NPN S8050 	4 * Resistor (220Ω)	1 * Resistor(1kΩ)	4 * Resistor (10kΩ)
4 * Led LED 	4 * Button	1 * Active Buzzer 	

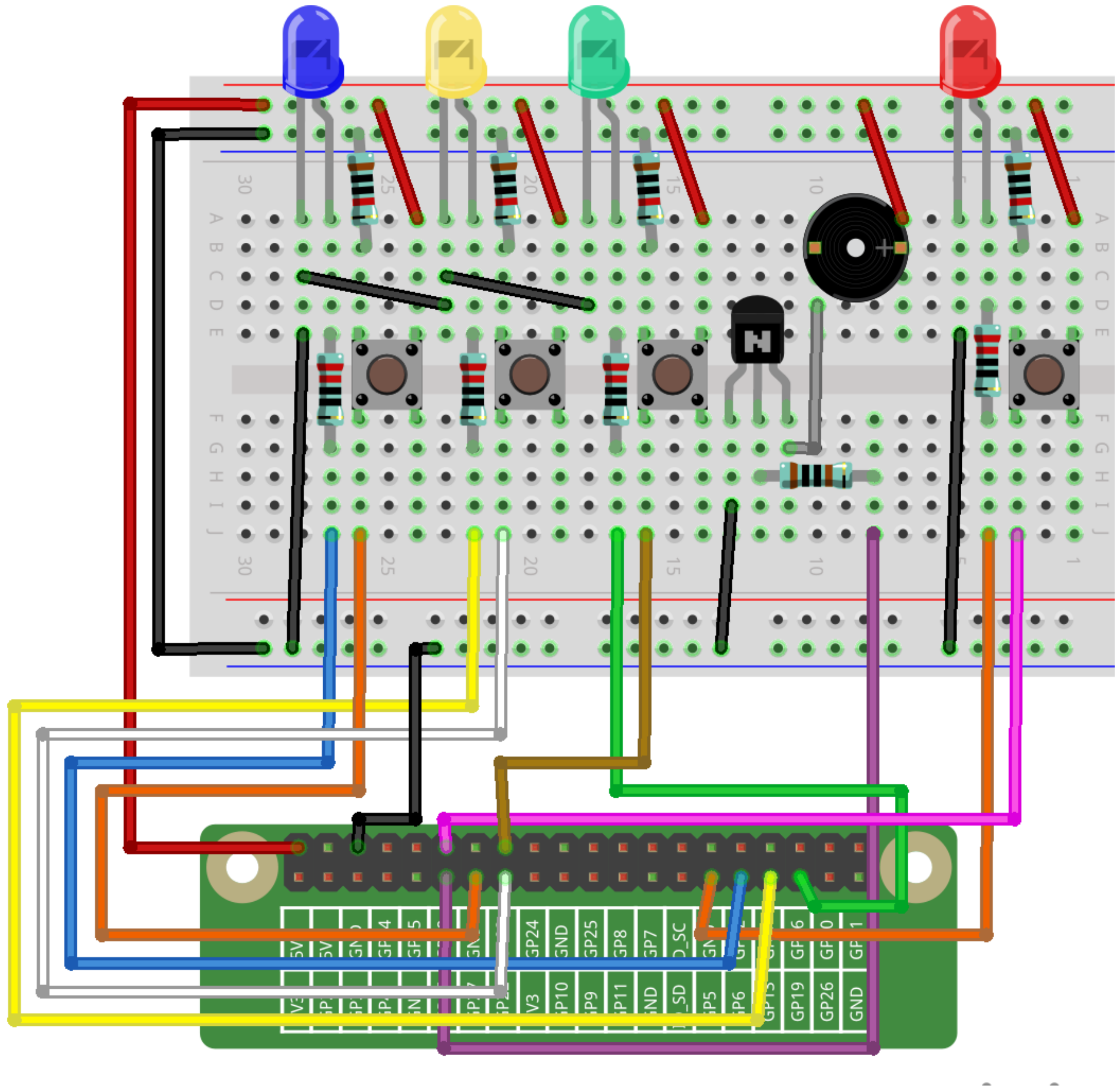
Schematic Diagram

Button 2, 3 and 4 are answer buttons, and button 1 is the reset button. If button 2 is pressed first, the buzzer will beep, the corresponding LED will light up and all the other LEDs will go out. If you want to start another round, press button 1 to reset.

wiringPi	Physical	BCM
0	Pin11	17
2	Pin13	27
3	Pin15	22
4	Pin16	23
1	Pin12	18
21	Pin29	5
22	Pin31	6
23	Pin33	13
24	Pin35	19


5V
U1
buzzer
Pin 11(GPIO 0)
R9
1K
Q1
NPN 8050
S1
R1
10K
D1
LED
R5
220
Pin 35(GPIO 24)
Pin 13(GPIO 2)
S2
R2
10K
D2
LED
R6
220
Pin 33(GPIO 23)
Pin 15(GPIO 3)
S3
R3
10K
D3
LED
R7
220
Pin 31(GPIO 22)
Pin 16(GPIO 4)
S4
R4
10K
D4
LED
R8
220
Pin 29(GPIO 21)
Pin 12(GPIO 1)
GND

Build the Circuit



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_14_AnswerMachine
```

2. Compile the code.

```
gcc 14_AnswerMachine.c -lwiringPi
```

3. Run the executable file.

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

Now, first press button 4 to get started. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define BeepPin 0
#define ResetBtnPin 1
const int BtnPin[] = {2,3,4};
const int LedPin[] = {21,22,23,24};

void Alarm()
{
    for(int i=0;i<50;i++){
    digitalWrite(BeepPin,HIGH); //the buzzer sound
    delay(2); //delay 2ms
    digitalWrite(BeepPin,LOW);  //without sound
    delay(2);
  }
}

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

    pinMode(BeepPin, OUTPUT);
    for(int j=1;j<4;j++)
    {
        pinMode(LedPin[j], OUTPUT);
        digitalWrite(LedPin[j],LOW);
    }
    pinMode(LedPin[0], OUTPUT);
    digitalWrite(LedPin[0],HIGH);
    for(int k;k<3;k++)
    {
        pinMode(BtnPin[k], INPUT);
    }

    int flag = 1;

    while(1){
        // if reset button is pressed
```

(continues on next page)

(continued from previous page)

```
        if(digitalRead(ResetBtnPin) == 1)
        {
            flag = 1;
            digitalWrite(LedPin[0], HIGH);//Reset Led turns on
            digitalWrite(LedPin[1],LOW);
            digitalWrite(LedPin[2],LOW);
            digitalWrite(LedPin[3],LOW);
        }
        if(flag==1)
        {
            //If the button1 press the first
            if(digitalRead(BtnPin[0]) == 1)
            {
                flag = 0;
                digitalWrite(LedPin[0],LOW);
                Alarm();  //buzzer sound
                digitalWrite(LedPin[1],HIGH);//turn the LED1 on only
                digitalWrite(LedPin[2],LOW);
                digitalWrite(LedPin[3],LOW);
                while(digitalRead(ResetBtnPin));
            }
            if(digitalRead(BtnPin[1]) == 1)
            {
                flag = 0;
                digitalWrite(LedPin[0],LOW);
                Alarm();  //buzzer sound
                digitalWrite(LedPin[1],LOW);
                digitalWrite(LedPin[2],HIGH);//turn the LED2 on only
                digitalWrite(LedPin[3],LOW);
                while(digitalRead(ResetBtnPin));
            }
            if(digitalRead(BtnPin[2]) == 1)
            {
                flag = 0;
                digitalWrite(LedPin[0],LOW);
                Alarm();  //buzzer sound
                digitalWrite(LedPin[1],LOW);
                digitalWrite(LedPin[2],LOW);
                digitalWrite(LedPin[3],HIGH);//turn the LED3 on only
                while(digitalRead(ResetBtnPin));
            }
        }
    }
    return 0;
}
```

Code Explanation

```
void Alarm()
{
    for(int i=0;i<50;i++){
    digitalWrite(BeepPin,HIGH); //the buzzer sound
    delay(2);
    digitalWrite(BeepPin,LOW);  //without sound
    delay(2);
  }
}
```

Define a function to control the buzzer. The buzzer rings when this function is called in the main function.

```
    int flag = 1;
```

Define a flag to judge whether the answer device is in the state of answering. When flag = 0, it indicates that someone is currently scrambling, and others cannot continue to answer first; when flag = 1, it means that the reset button has been pressed, and a new round of answer rush can be conducted.

```
        if(digitalRead(ResetBtnPin) == 1)
        {
            flag = 1;
            digitalWrite(LedPin[0], HIGH);//Reset Led turns on
            digitalWrite(LedPin[1],LOW);
            digitalWrite(LedPin[2],LOW);
            digitalWrite(LedPin[3],LOW);
        }
```

If the reset button is detected to have been pressed, it means that the answer begins. Now set flag to 1 and let the referee LED light up, the rest of the LED lights out.

```
            if(digitalRead(BtnPin[0]) == 1)
            {
                flag = 0;
                digitalWrite(LedPin[0],LOW);
                Alarm();  //buzzer sound
                digitalWrite(LedPin[1],HIGH);//turn the LED1 on only
                digitalWrite(LedPin[2],LOW);
                digitalWrite(LedPin[3],LOW);
                while(digitalRead(ResetBtnPin));
            }
```

In the process of quick answering, if the first button is recognized to have been pressed, the flag is set to 0, and then no other buttons are detected. At this time, the buzzer alarms, indicating that someone has successfully responsed, and the corresponding LED lights up. The identification codes of the remaining buttons are explained as above.

```
                    while(digitalRead(ResetBtnPin));
```

Having executed the instruction of successful quick answer, it enters the loop to judge whether the button reset is pressed. Here, if the button reset is pressed, then the next round of quickfire answering begins.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 14_AnswerMachine.py
```

Now, first press button 4 to get started. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

BeepPin = 17
ResetBtnPin = 18
BtnPin =(27,22,23)
LedPin =(5,6,13,19)

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(ResetBtnPin, GPIO.IN)
    GPIO.setup(LedPin[0], GPIO.OUT, initial=GPIO.HIGH)
    for i in range(1,4):
        GPIO.setup(LedPin[i], GPIO.OUT, initial=GPIO.LOW)
    for i in range(0,3):
        GPIO.setup(BtnPin[i], GPIO.IN)

def Alarm():
    for i in range(0,50):
        GPIO.output(BeepPin,GPIO.HIGH)
        time.sleep(0.003)
        GPIO.output(BeepPin,GPIO.LOW)
        time.sleep(0.003)

def loop():
    flag = 1
    while True:
        if GPIO.input(ResetBtnPin) == 1:
            flag = 1
            GPIO.output(LedPin[0],GPIO.HIGH)
            GPIO.output(LedPin[1],GPIO.LOW)
            GPIO.output(LedPin[2],GPIO.LOW)
            GPIO.output(LedPin[3],GPIO.LOW)
```

(continues on next page)

(continued from previous page)

```
        if flag == 1:
            if GPIO.input(BtnPin[0]) == 1:
                flag = 0
                GPIO.output(LedPin[0],GPIO.LOW)
                Alarm()
                GPIO.output(LedPin[1],GPIO.HIGH)
                GPIO.output(LedPin[2],GPIO.LOW)
                GPIO.output(LedPin[3],GPIO.LOW)
            elif GPIO.input(BtnPin[1]) == 1:
                flag = 0
                GPIO.output(LedPin[0],GPIO.LOW)
                Alarm()
                GPIO.output(LedPin[1],GPIO.LOW)
                GPIO.output(LedPin[2],GPIO.HIGH)
                GPIO.output(LedPin[3],GPIO.LOW)
            elif GPIO.input(BtnPin[2]) == 1:
                flag = 0
                GPIO.output(LedPin[0],GPIO.LOW)
                Alarm()
                GPIO.output(LedPin[1],GPIO.LOW)
                GPIO.output(LedPin[2],GPIO.LOW)
                GPIO.output(LedPin[3],GPIO.HIGH)

def destroy():
    # Turn off buzzer
    GPIO.output(BeepPin, GPIO.LOW)
    GPIO.output(LedPin[0],GPIO.LOW)
    GPIO.output(LedPin[1],GPIO.LOW)
    GPIO.output(LedPin[2],GPIO.LOW)
    GPIO.output(LedPin[3],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 Explanation

```
def Alarm():
    for i in range(0,50):
        GPIO.output(BeepPin,GPIO.HIGH)
        time.sleep(0.003)
        GPIO.output(BeepPin,GPIO.LOW)
        time.sleep(0.003)
```

Define a function to control the buzzer. The buzzer rings when this function is called in the function **main**.

```
    flag = 1
```

Define a flag bit to judge whether the responder is in the state of answering. When **flag** = 0, it indicates that someone is currently scrambling, and others cannot continue to answer first; when **flag** = 1, it means that the reset button has been pressed, and a new round of answer rush can be conducted.

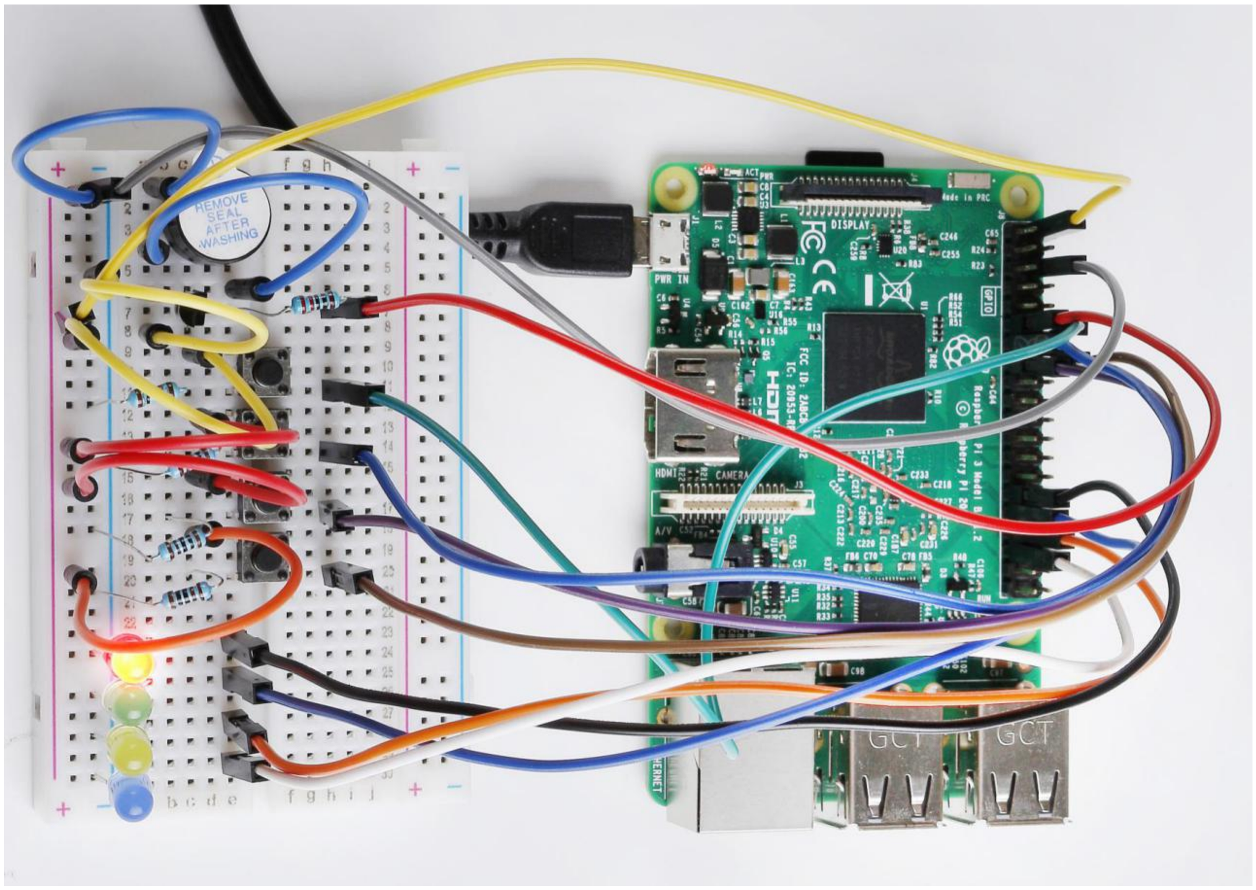
```
        if GPIO.input(ResetBtnPin) == 1:
            flag = 1
            GPIO.output(LedPin[0],GPIO.HIGH)
            GPIO.output(LedPin[1],GPIO.LOW)
            GPIO.output(LedPin[2],GPIO.LOW)
            GPIO.output(LedPin[3],GPIO.LOW)
```

If the recognition that reset button has been pressed is done, it means that answer begins. Now, set flag to 1, and let the referee LED light up, other LEDs light out.

```
            if GPIO.input(BtnPin[0]) == 1:
                flag = 0
                GPIO.output(LedPin[0],GPIO.LOW)
                Alarm()
                GPIO.output(LedPin[1],GPIO.HIGH)
                GPIO.output(LedPin[2],GPIO.LOW)
                GPIO.output(LedPin[3],GPIO.LOW)
```

In the process of quick answering, if the first button is recognized to have been pressed, the flag is set to 0, and then no other buttons are detected. At this time, the buzzer alarms, indicating that there is a successful response, and the corresponding LED lights up. The identification codes of the remaining buttons are explained as above.

Phenomenon Picture

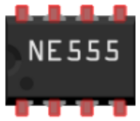


Lesson 15 NE555 Timer

Introduction

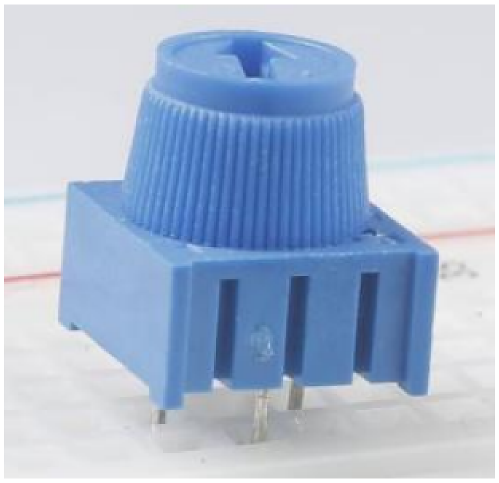
The NE555 Timer, a mixed circuit composed of analog and digital circuits, integrates analog and logical functions into an independent IC, thus tremendously expanding the applications of analog integrated circuits. It is widely used in various timers, pulse generators, and oscillators. In this experiment, the Raspberry Pi is used to test the frequencies of square waves generated by the 555 oscillating circuit and show them on terminal windows.

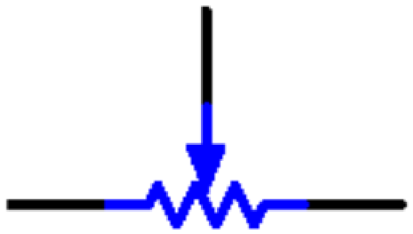
Newly Added Components

1 * Potentiometer (10KΩ)	2 * 104 ceramic capacitor	1 * NE555	1 * Resistor(10kΩ)
			

Principle

Potentiometer





Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation. Potentiometer usually consists of resistor and movable brush. When the brush is moving along the resistor, there is a certain resistance or voltage output depending on the displacement.

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

1.Serving as a voltage divider Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the distance it moves.

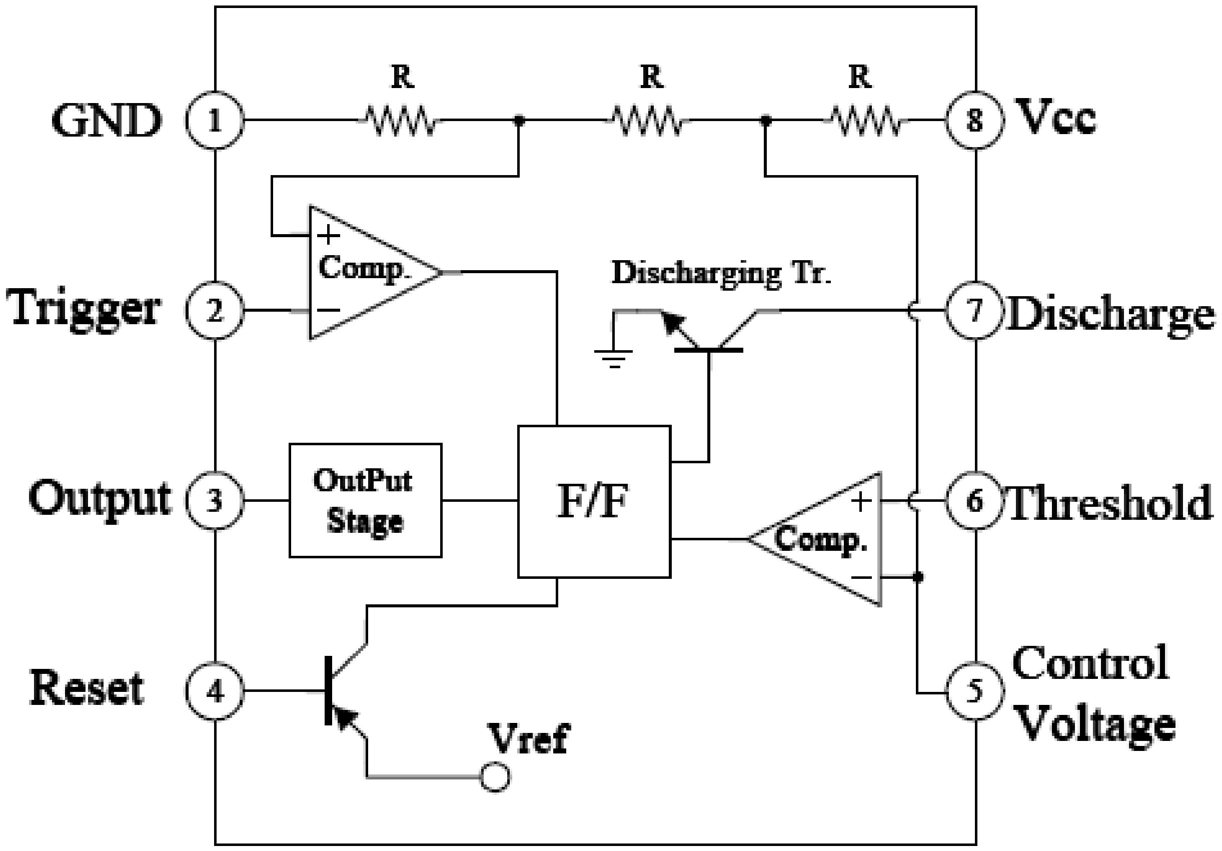
2.Serving as a rheostat When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value cused by moving contact.

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

555 IC

The 555 IC was originally used as a timer, hence the name 555 time base circuit. It is now widely used in various electronic products because of its reliability, convenience, and low price. The 555 is a complex hybrid circuit with dozens of components such as a divider, comparator, basic R-S trigger, discharge tube, and buffer.

Pins and functions:



As shown in the picture, the pins are set dual in-line with the 8-pin package.

- Pin 1 (**GND**): the ground
- Pin 2 (**TRIGGER**): when the voltage at the pin reduces to 1/3 of the VCC (or the threshold defined by the control board), the output terminal sends out a High level
- Pin 3 (**OUTPUT**): outputs High or Low, two states 0 and 1 decided by the input electrical level; maximum output current approx. 200mA at High
- Pin 4 (**RESET**): when a Low level is received at the pin, the timer will be reset and the output will return to Low level; usually connected to positive pole or neglected
- Pin 5 (**CONTROL VOLTAGE**): to control the threshold voltage of the chip (if it skips connection, by default, the threshold voltage is 1/3 VCC and 2/3 VCC)
- Pin 6 (**THRESHOLD**): when the voltage at the pin increases to 2/3 VCC (or the threshold defined by the control board), the output terminal sends out a High level.
- Pin 7 (**DISCHARGE**): output synchronized with Pin 3, with the same logical level; but this pin does not output current, so pin 3 is the real High (or Low) when pin 7 is the virtual High (or Low); connected to the open collector (OC) inside to discharge the capacitor.
- Pin 8 (**VCC**): positive terminal for the NE555 timer IC, ranging +4.5V to +16V

The NE555 timer works under the monostable, astable and bistable modes. In this experiment, apply it under the astable mode, which means it works as an oscillator, as shown below:

Cap

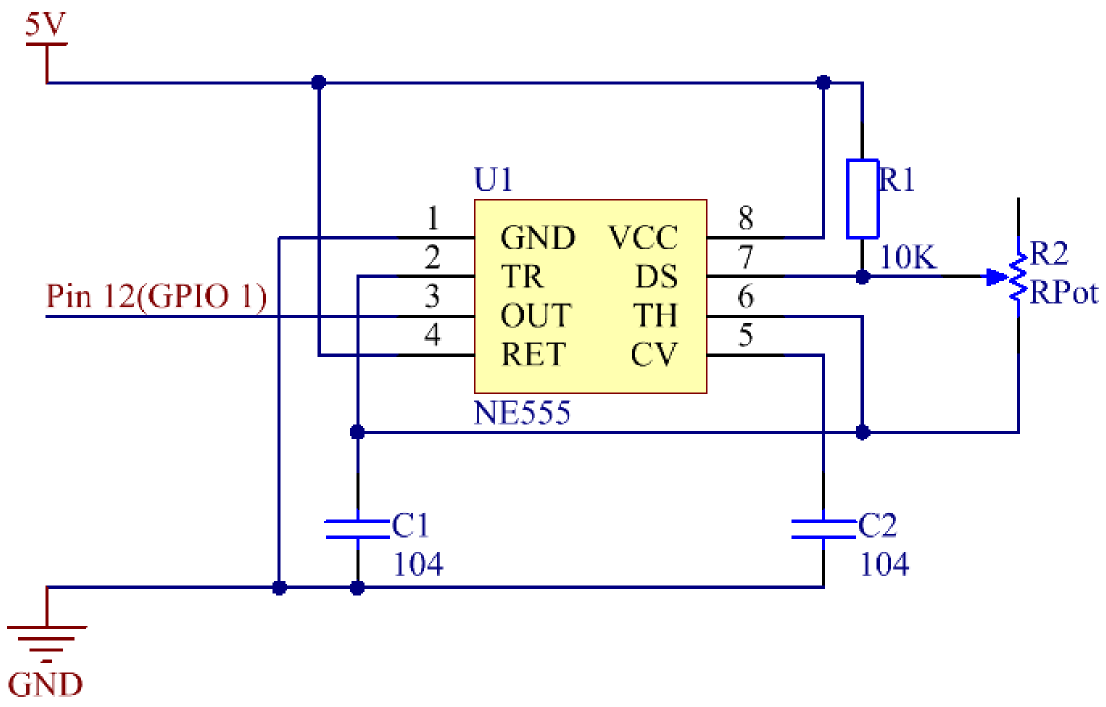
A ceramic capacitor is a capacitor that is made of ceramic material and works as a dielectric. It is coated with a metal film on the surface of the ceramic and sintered at a high temperature. The ceramic capacitor is commonly used in high-stability oscillator circuits as loops, bypass capacitors, and pad capacitors. It is a non-polar capacitor, so this capacitor does not need to distinguish between positive and negative during installation.

In the circuit of this lesson, the main function of the ceramic capacitor, high-frequency filtering is to remove some clutter that may occur in the working process of the NE555 chip, so that the waveform is more stable.

Schematic Diagram

Build the circuit according to the following schematic diagram.

wiringPi	Physical	BCM
1	Pin12	18



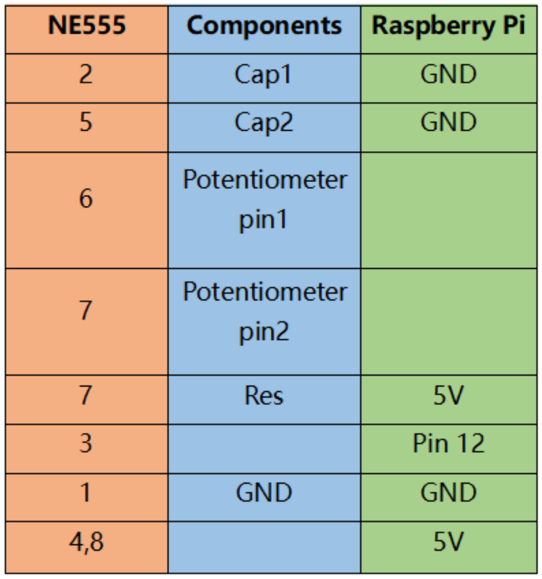
Working Process:

The oscillator starts to shake once the circuit is power on. During energizing, since the voltage at C1 cannot change abruptly, which means pin 2 is Low level initially, set the timer to 1, so pin 3 is High level. The capacitor C1 charges via R1 and R2 in a time span:

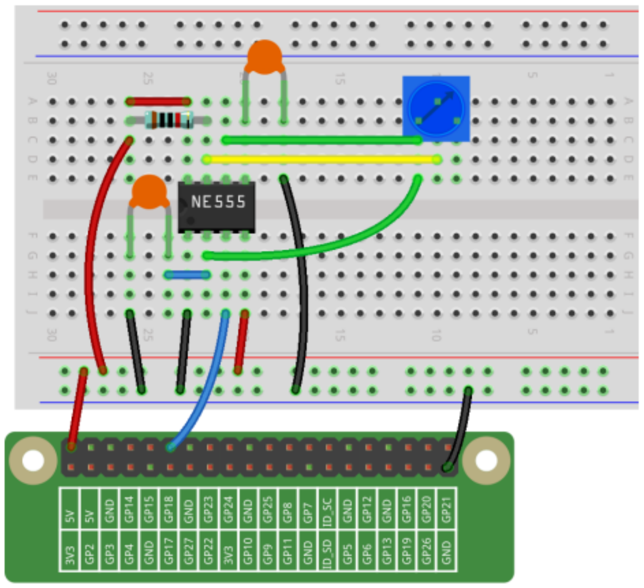
When the voltage at C1 reaches the threshold 2/3Vcc, the timer is reset and pin 3 is Low level. Then C1 discharges via R2 till 2/3Vcc in a time span:

Then the capacitor is recharged and the output voltage flips again:

Build the Circuit



NE555	Components	Raspberry Pi
2	Cap1	GND
5	Cap2	GND
6	Potentiometer pin1	
7	Potentiometer pin2	
7	Res	5V
3		Pin 12
1	GND	GND
4,8		5V



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_15_NE555_Timer
```

2. Compile the code.

```
gcc 15_NE555_Timer.c -lwiringPi
```

3. Run the executable file.

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

When the code is running, you will see the number of pulses on the display screen and the level of pin3 in NE555 at this time.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

#define  OutPin  1

static volatile int globalCounter = 0 ;

void exInt0_ISR(void)  //GPIO 1 interrupt service routine
{
    ++globalCounter;
}

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

    delay(2000);
    pinMode(OutPin,INPUT);
    pullUpDnControl(OutPin,PUD_UP);
    wiringPiISR(OutPin, INT_EDGE_FALLING, &exInt0_ISR);

    while(1){
        printf("Current pluse number is : %d, %d\n", globalCounter,
↪digitalRead(OutPin));
        delay(100);
    }
    return 0;
}
```

Code Explanation

```c
static volatile int globalCounter = 0 ;
```

Define a variable to record the number of pulses, and initialize the number of pulses to 0.

```c
void exInt0_ISR(void)
{
    ++globalCounter;
}
```

Set an external interrupt function and **globalCounter** will automatically +1 when an interrupt occurs.

```c
 pinMode(OutPin,INPUT);
 pullUpDnControl(OutPin,PUD_UP);
```

Set the out pin of NE555 to **INPUT** mode, then let the pin be in pull-up state (1).

```
wiringPiISR(OutPin, INT_EDGE_FALLING, &exInt0_ISR);
```

Set an interrupt in **OutPin**, when the value of **OutPin** changes from 1 to 0. Then call the exInt0_ISR() function to let the variable **globalCounter** add 1.

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

Print out the number of pulses, **globalCounter** and the value of **OutPin** at this time.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 15_NE555.py
```

When the code is running, you can see the number of pulses on the display.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

SigPin = 18

g_count = 0

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

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # wait for rasing

def main():
    while True:
        print ('g_count = %d' % g_count)
        time.sleep(0.01)

```

(continues on next page)

(continued from previous page)

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

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

Code Explanation

```python
6. g_count = 0
```

Define a variable to record the number of pulses, and initialize the number of pulses to **0**.

```python
7.def count(ev=None):
8.global g_count
9.g_count += 1
```

This function will change the value of the global variable **g_count**.

```python
14.      GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

Set the **SigPin** to input mode and pull up to high level(3.3V).

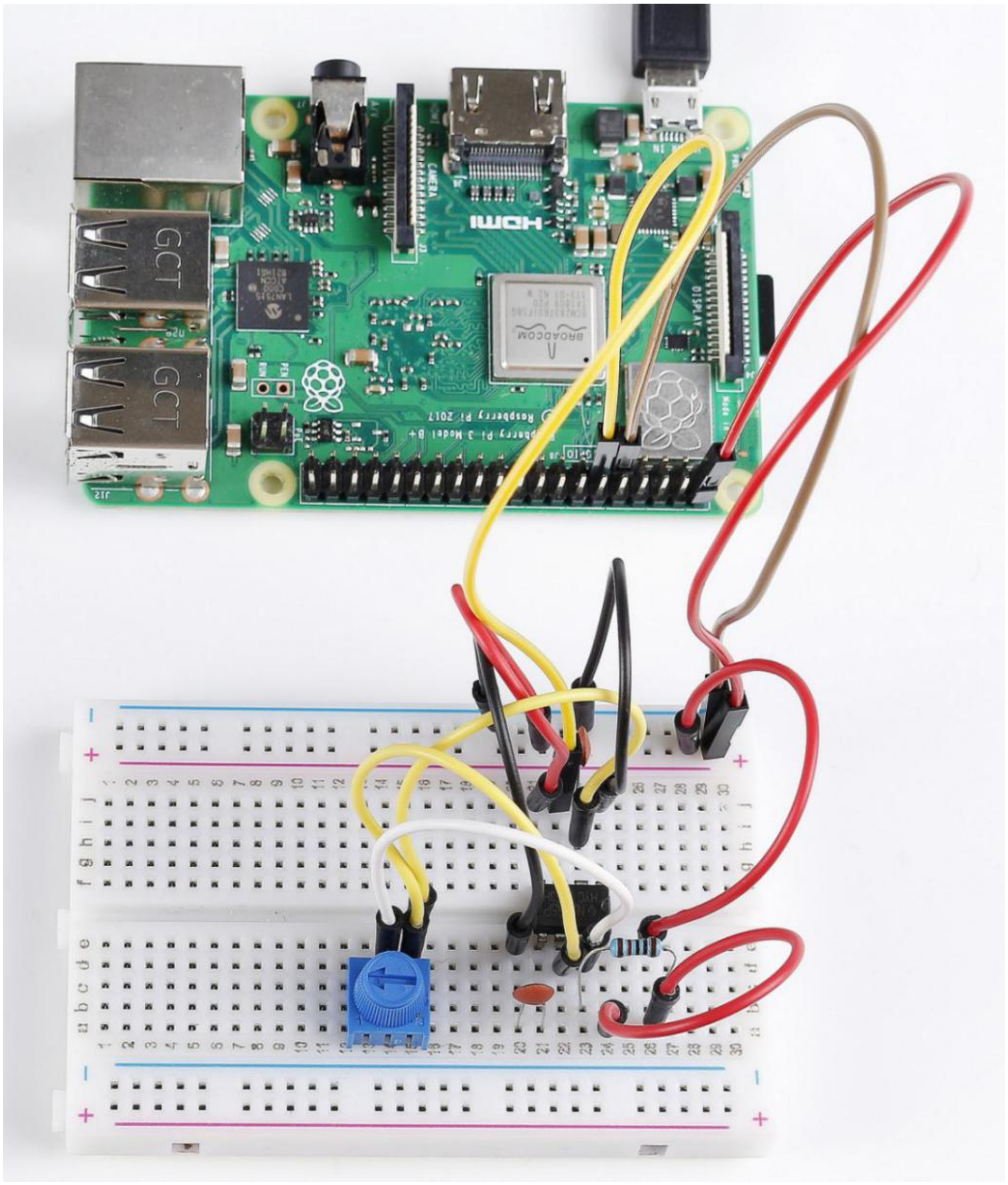
```python
15.  GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count)
```

Set an interrupt in **SigPin**, when the value of **SigPin** changes from 0 to 1. Then call the **count()** function to let the variable **g_count** add 1.

```python
18.    while True:
19.        print ('g_count = %d' % g_count)
20.        time.sleep(0.01)
```

Print out the value of the number of pulse g_count at an interval of 0.01s.

Phenomenon Picture



Lesson 16 Servo

Introduction

Servo is a type of geared motor that can only rotate 180 degrees. It is controlled by sending electrical pulses from your board. These pulses tell the servo what position it should move to.

A servo has three wires: the brown wire is GND, the red one is VCC, and the orange one is signal line.

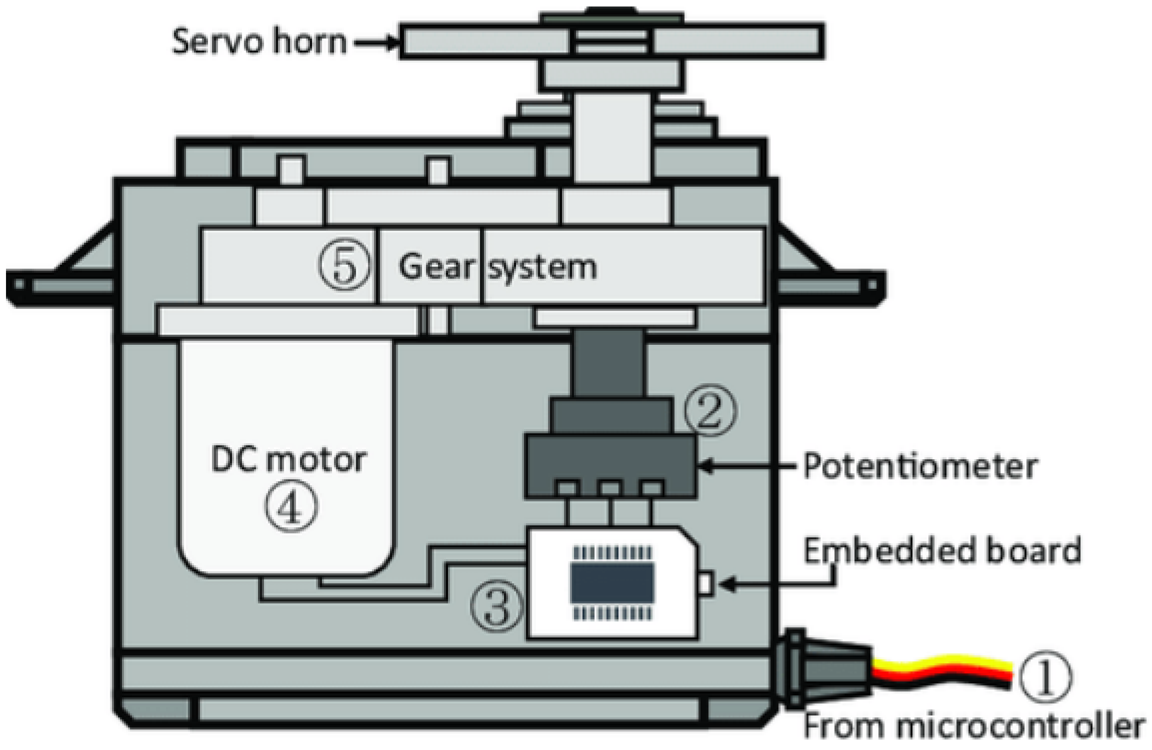
Newly Added Components

Principle

Servo

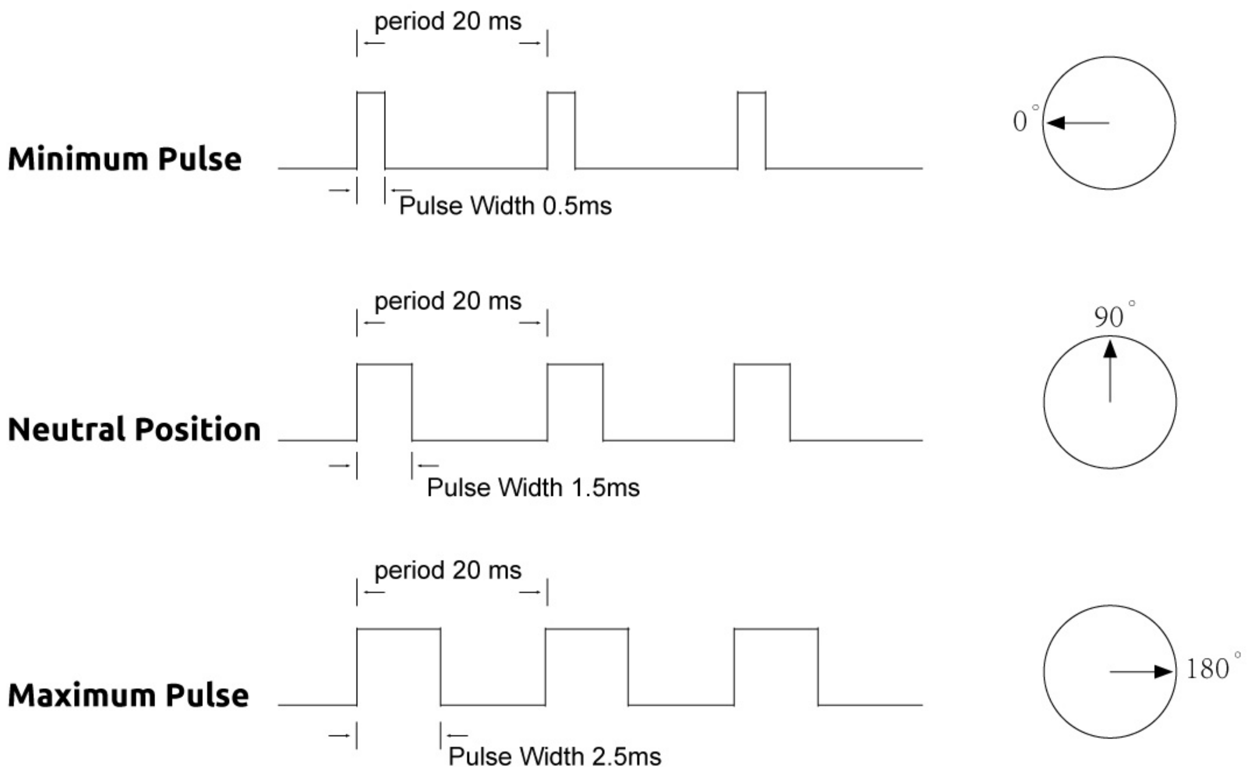
A servo is generally composed of the following parts: case, shaft, gear system, potentiometer, DC motor, and embedded board.

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



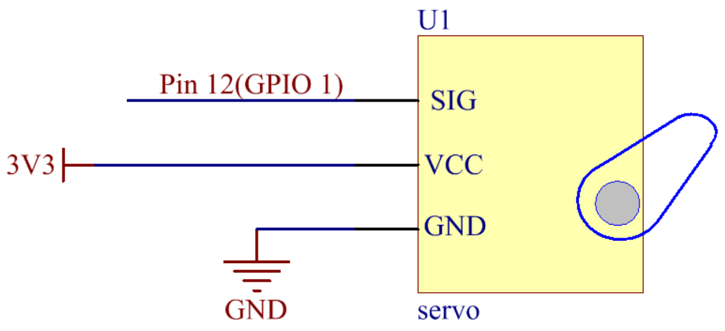
The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse width Modulation. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position).

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



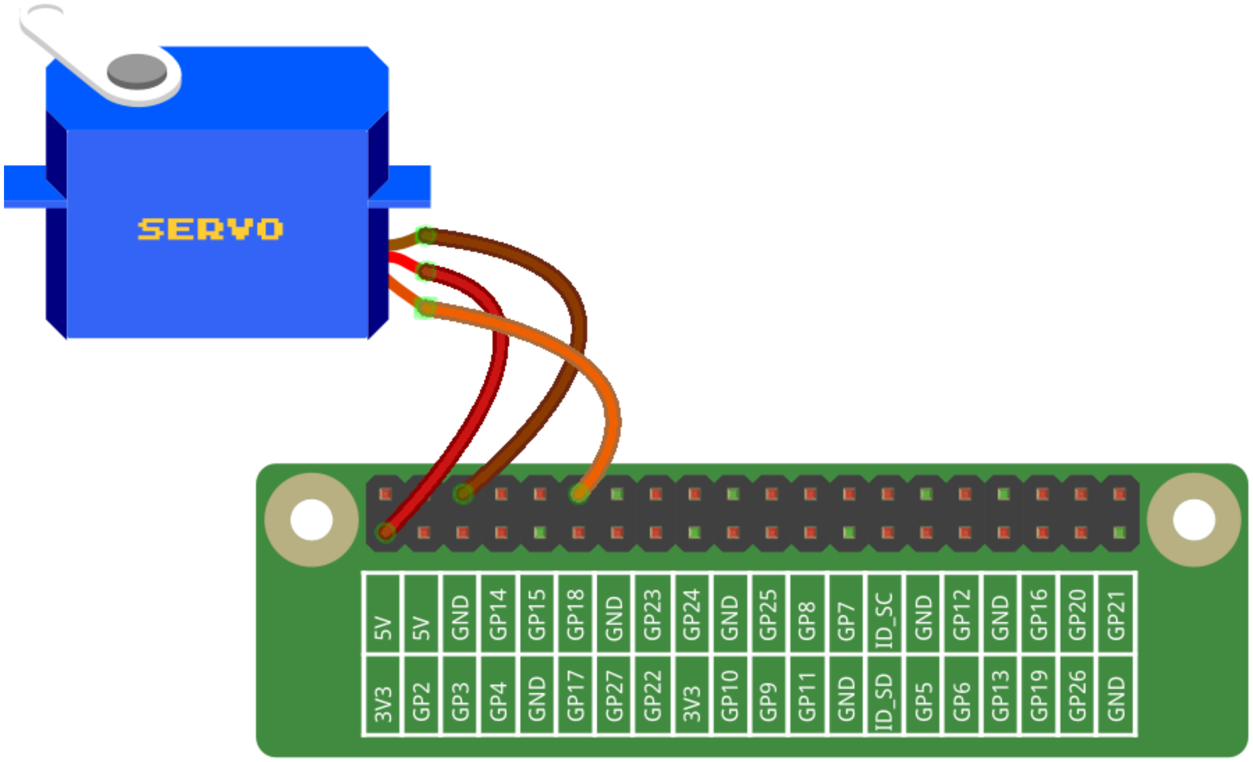
Schematic Diagram

wiringPi	Physical	BCM
1	Pin12	18



Build the Circuit

Note: Connect the brown to GND, Red to VCC, Orange to pin12 of the control board.



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_16_Servo
```

2. Compile the code.

```
gcc 16_Servo.c -lwiringPi
```

3. Run the executable file.

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

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

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

#define ServoPin    1
long Map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void setAngle(int pin, int angle){    //Specif a certain rotation angle (0-180) for the servo
    if(angle < 0)
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,map(angle,0,180,5,25));
}

int main(void)
{
    int i;

    if(wiringPiSetup() == -1){ //when initialize wiring faiservo,print message to screen
        printf("setup wiringPi failed !");
        return 1;
    }
    softPwmCreate(servoPin,  0, 200);       //initialize PMW pin of servo
    while(1){
        for(i=0;i<181;i++){
            setAngle(ServoPin,i);
            delay(1);
        }
        delay(500);
        for(i=181;i>-1;i--){
            setAngle(ServoPin,i);
            delay(1);
        }
        delay(500);
    }
    return 0;
}
```

Code Explanation

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

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

```c
void setAngle(int pin, int angle){    //Specif a certain rotation angle (0-180) for the servo
    if(angle < 0)
```

(continues on next page)

(continued from previous page)

```
        angle = 0;
    if(angle > 180)
        angle = 180;
    softPwmWrite(pin,map(angle,0,180,5,25));
}
```

Define a function to limit the angle of the servo to 0 to 180 in order to set the angle of servo.

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

This function can change the duty cycle of the PWM pin.

To make the servo rotate to 0 ~ 180 °, the pulse width should change within the range of 0.5ms ~ 2.5ms when the period is 20ms; in the function, **softPwmCreate()**, we have set that the period is 200x100us=20ms, thus we need to map 0 ~ 180 to 5x100us ~ 25x100us.

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

The function is to use softwares to create a PWM pin, **servoPin**, then the initial pulse widths of them are set to **0**, and the period of PWM is **200** x100us.

```
        for(i=0;i<181;i++){
            setAngle(ServoPin,i);
            delay(1);
        }
```

In a **for** loop, we want servo to rotate from 0 degrees to 180 degrees.

```
        for(i=181;i>-1;i--){
            setAngle(ServoPin,i);
            delay(1);
        }
```

In a **for** loop, we want servo to rotate from 180 degrees to 0 degrees.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 16_Servo.py
```

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

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

SERVO_MIN_PULSE = 500
SERVO_MAX_PULSE = 2500

ServoPin = 18

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

def setup():
    global p
    GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by BCM
    GPIO.setup(ServoPin, GPIO.OUT)   # Set ServoPin's mode is output
    GPIO.output(ServoPin, GPIO.LOW)  # Set ServoPin to low
    p = GPIO.PWM(ServoPin, 50)     # set Frequecy to 50Hz
    p.start(0)                     # Duty Cycle = 0

def setAngle(angle):      # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it

def loop():
    while True:
        for i in range(0, 181, 5):   #make servo rotate from 0 to 180 deg
            setAngle(i)     # Write to servo
            time.sleep(0.002)
        time.sleep(1)
        for i in range(180, -1, -5): #make servo rotate from 180 to 0 deg
            setAngle(i)
            time.sleep(0.001)
        time.sleep(1)

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

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

Code Explanation

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

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

```
p = GPIO.PWM(ServoPin, 50)
p.start(0)
```

Set the **servoPin** to PWM pin, then the frequency to **50** hz, and the period to 20ms. p.start(0): Run the PWM functionand set the initial value to **0**.

```
def setAngle(angle):      # make the servo rotate to specific angle (0-180 degrees)
    angle = max(0, min(180, angle))
    pulse_width = map(angle, 0, 180, SERVO_MIN_PULSE, SERVO_MAX_PULSE)
    pwm = map(pulse_width, 0, 20000, 0, 100)
    p.ChangeDutyCycle(pwm)#map the angle to duty cycle and output it
```

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

```
p.ChangeDutyCycle(pwm)
```

This function can change the duty cycle of the PWM. To render a range **0 ~ 180°** to the servo, the pulse width of the servo is set to **0.5ms-2.5ms**.

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

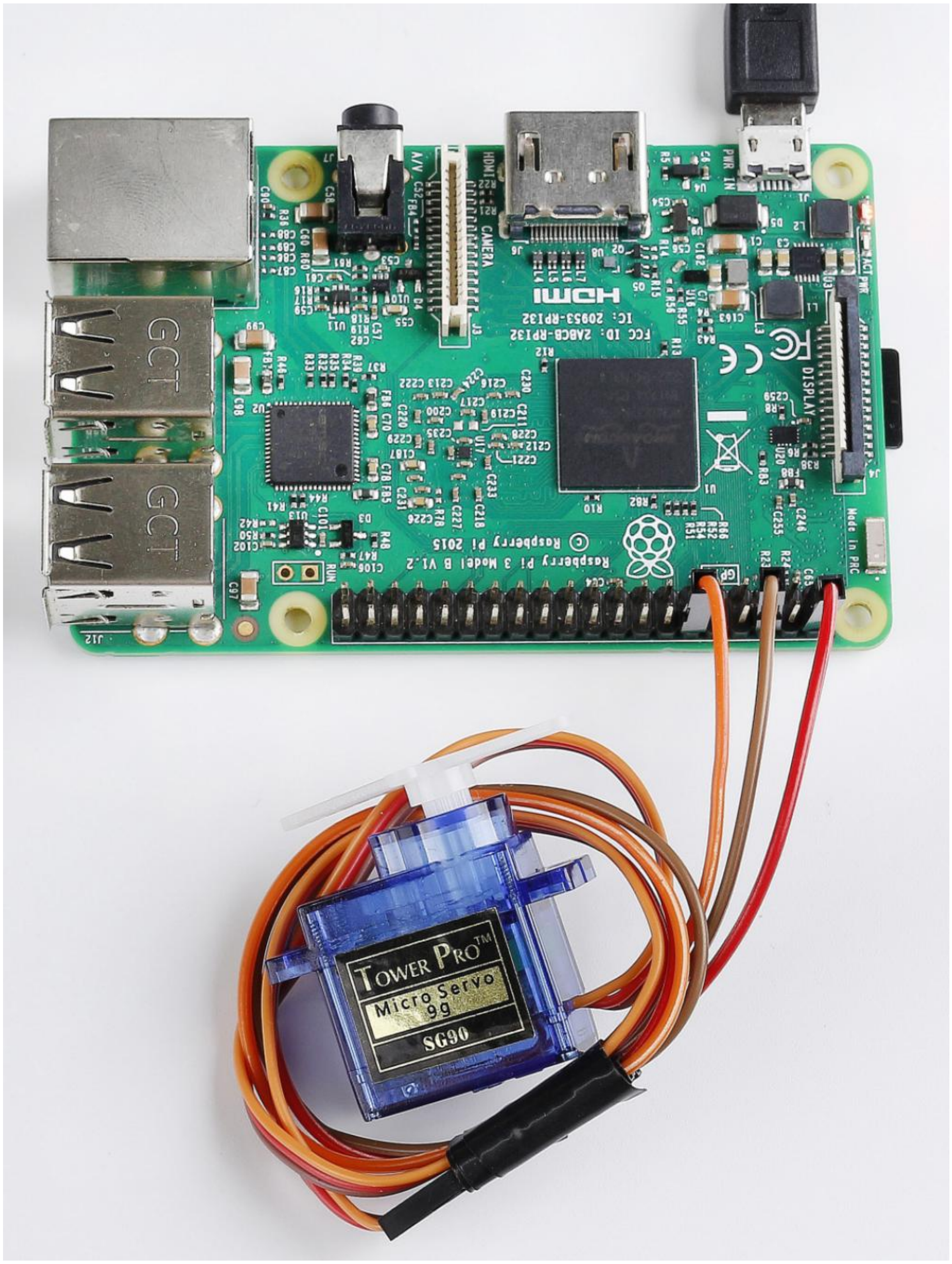
```
        for i in range(0, 181, 5):   #make servo rotate from 0 to 180 deg
            setAngle(i)     # Write to servo
            time.sleep(0.002)
```

In a **for** loop, we want servo to rotate from **0** degrees to **180** degrees.

```
        for i in range(180, -1, -5): #make servo rotate from 180 to 0 deg
            setAngle(i)
            time.sleep(0.001)
```

In a **for** loop, we want servo to rotate from **180** degrees to **0** degrees.

Phenomenon Picture

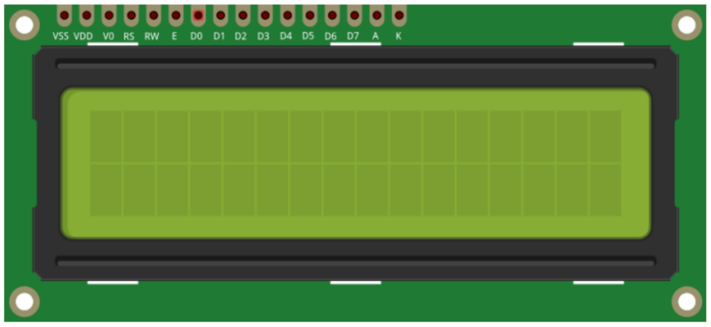


Lesson 17 LCD1602

Introduction

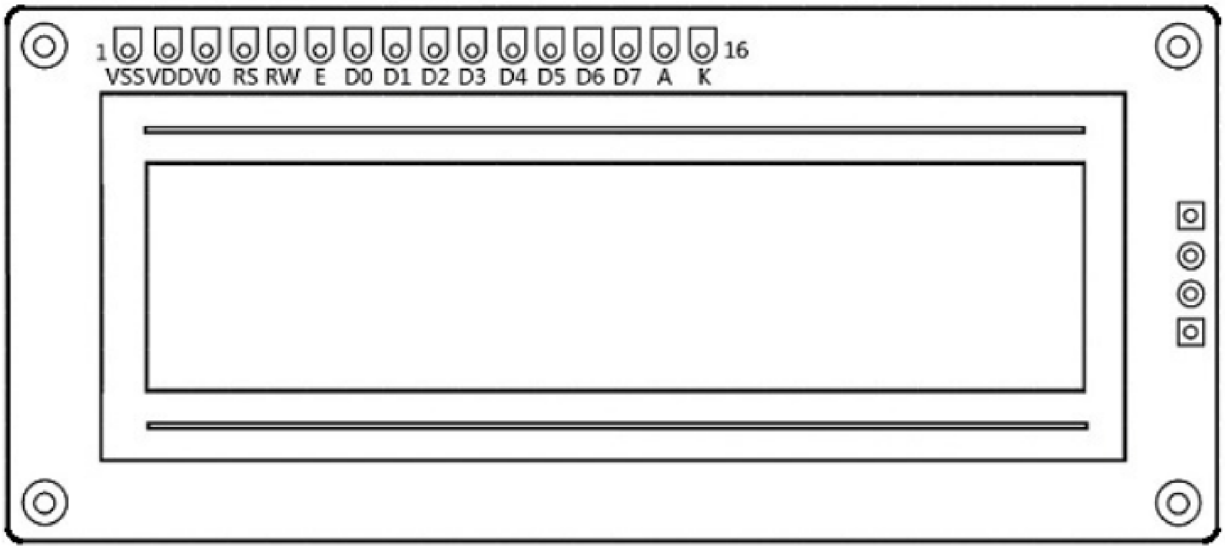
In this lesson, we will learn how to use an LCD1602 to display characters and strings. LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters and so on. It's composed of 5x7 or 5x11 dot matrix positions; each position can display one character. Now let's check more details!

Newly Added Components

1 * Potentiometer (10kΩ)	1 * LCD1602
	

Principle

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the Raspberry Pi are almost completely occupied. If you want to connect more sensors, there will be no ports available. Therefore, the four-port connection is used here for better application.



Pins of LCD1602 and their Functions

VSS: connected to ground.

VDD: connected to a +5V power supply.

VO: to adjust the contrast.

RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

R/W: A Read/Write pin to select between reading and writing mode.

E: An enabling pin that reads the information when High level (1) is received. The instructions are run when the signal changes from High level to Low level.

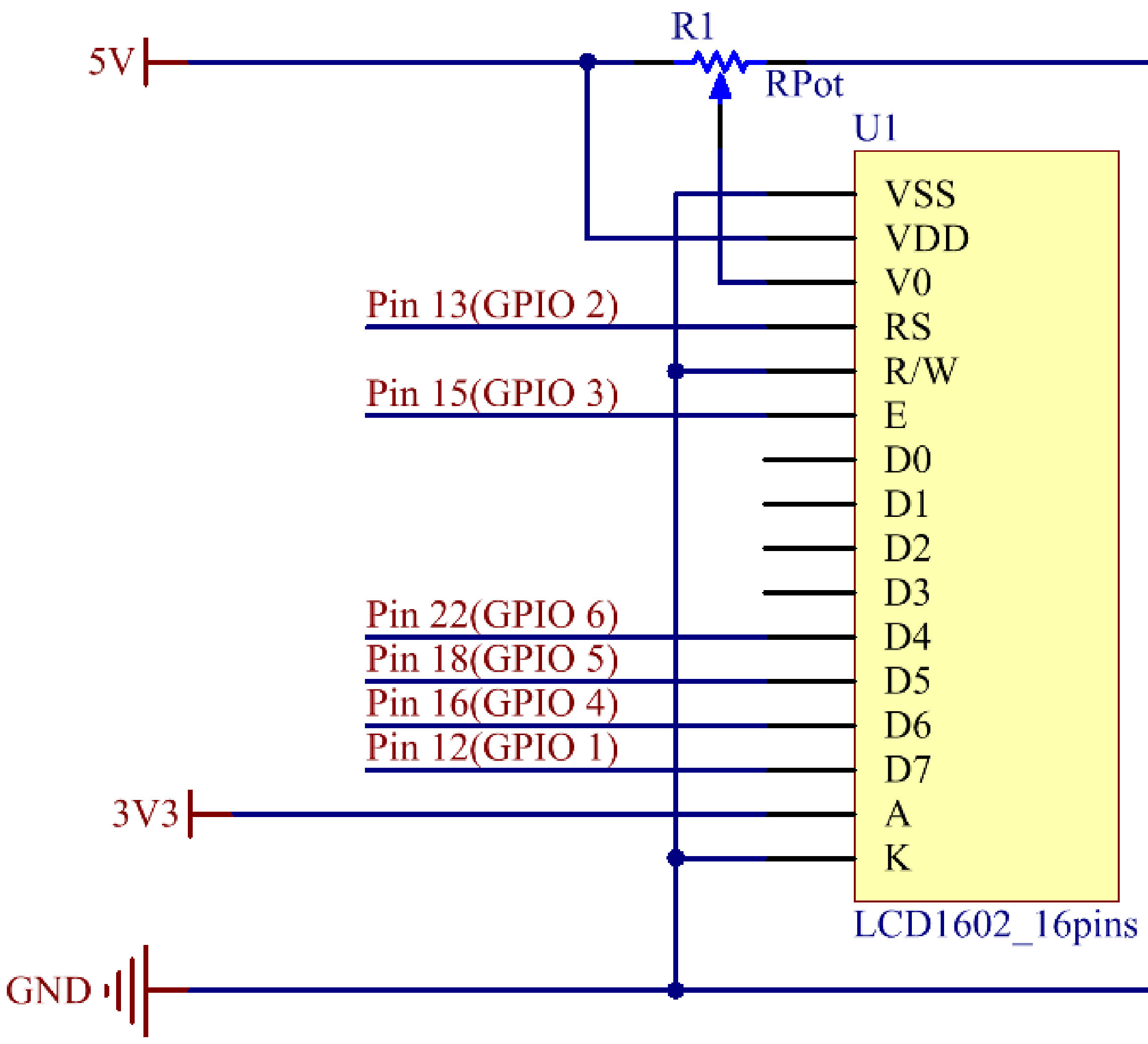
D0-D7: to read and write data.

A and K: Pins that control the LCD backlight. Connect K to GND and A to 3.3v. Open the backlight and you will see clear characters in a comparatively dark environment.

Schematic Diagram

Connect **K** to **GND** and **A** to **3.3 V**, and then the backlight of the LCD1602 will be turned on. Connect **VSS** to **GND** and the **LCD1602** to the power source. Connect **VO** to the middle pin of the potentiometer - with it you can adjust the contrast of the screen display. Connect **RS** to **Pin 13** and **R/W** pin to **GND**. Connect **E** to **Pin 15** and the characters displayed on the LCD1602 are controlled by **D4-D7**. For programming, it is optimized by calling function libraries.

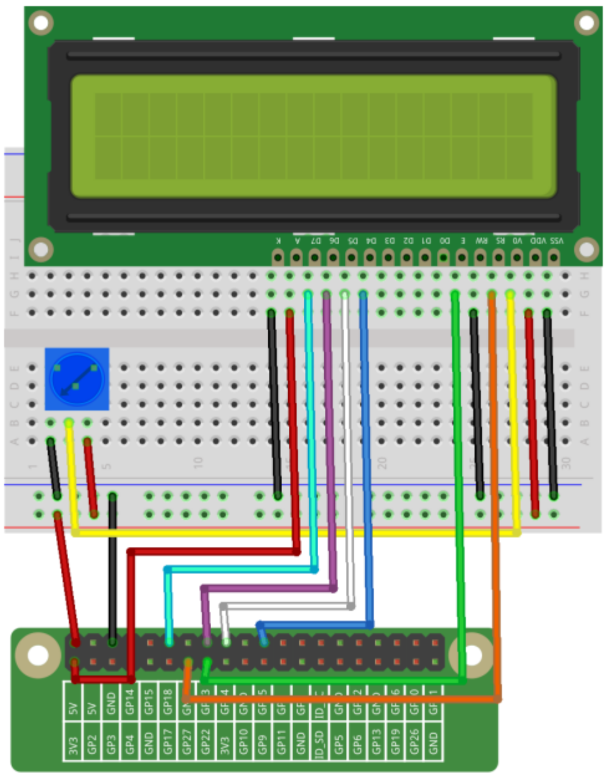
wiringPi	Physical	BCM
1	Pin12	18
2	Pin13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24
6	Pin22	25



Build the Circuit

Note: Make sure the pins are connected correctly. Otherwise, characters will not be displayed properly. You may need to adjust the potentiometer till the LCD1602 can display clearly.

LCD1602	Components	RPi
VSS		GND
VDD		5V
V0	Potentiometer pin2	
RS		Pin 13
R/W		GND
E		Pin 15
D4		Pin 22
D5		Pin 18
D6		Pin 16
D7		Pin 12
A		3V3
K		GND
	Potentiometer pin1	5V
	Potentiometer pin3	GND



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_17_LCD1602
```

2. Compile the code.

```
gcc 17_Lcd1602.c -lwiringPiDev -lwiringPi
```

Note: In order to use the LCD driver in the wiringPi devLib, you need to use -lwiringPiDev at compile time.

3. Run the executable file.

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

You may see the “SunFounder” and “hello, world” appear one by one on the LCD.

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

const unsigned char Buf[] = "---SUNFOUNDER---";
const unsigned char myBuf[] = "  sunfounder.com";

int main(void)
{
    int fd;
    int i;

    if(wiringPiSetup() == -1){
        exit(1);
    }

    fd = lcdInit(2,16,4,2,3, 0,0,0,0,6,5,4,1); //see /usr/local/include/lcd.h
    printf("%d", fd);
    if (fd == -1){
        printf("lcdInit 1 failed\n") ;
        return 1;
    }

    delay(1000);
    lcdClear(fd);
    lcdPosition(fd, 0, 0);
    lcdPuts(fd, "Welcome To--->");
    lcdPosition(fd, 0, 1);
    lcdPuts(fd, "sunfounder.com");
    delay(1000);
    lcdClear(fd);

    while(1){
        lcdClear(fd);
        for(i=0; i<16; i++){
            lcdPosition(fd, i, 0);
            lcdPutchar(fd, *(myBuf+i));
            delay(100);
        }
        for(i=0;i<sizeof(Buf)-1;i++){
            lcdPosition(fd, i, 1);
            lcdPutchar(fd, *(Buf+i));
            delay(200);
        }
        delay(500);
    }
    return 0;
}
```

Code Explanation

```
#include <lcd.h>
```

This is a library that integrates lcd1602 functional functions, in which functions are defined such as lcdClear(), lcdPosition(), lcdPuts(), and so on. These functions can be called directly after importing into the library.

```
fd = lcdInit(2,16,4,2,3, 0,0,0,0,6,5,4,1); //see /usr/local/include/lcd.h
printf("%d", fd);
if (fd == -1){
    printf("lcdInit 1 failed\n") ;
return 1;
```

Initialize the lcd1602. The prototype of lcdInit() is as follows:

int lcdInit (int rows, int cols, int bits, int rs, int strb, int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7) ;

This is the main initialisation function and must be called before you use any other LCD functions.

Rows and **cols** are the rows and columns on the display (e.g. 2, 16 or 4,20). **Bits** is the number of bits wide on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the display RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the Pi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault. (Usually incorrect parameters)

```
lcdClear(fd);
```

This function is used to clear the lcd screen. After calling this function, all information displayed on the screen will be cleared.

```
lcdPosition(fd, 0, 0);
```

Set the position of the cursor at row 0 and col 0 (in fact it's the first line and first column) for subsequent text entry.

The prototype of **lcdpostion** function is as follows:

```
lcdPosition (int handle, int x, int y) ;
```

Set the position of the cursor for subsequent text entry. **x** is the column and **0** is the left-most edge. **y** is the line and **0** is the top line.

```
lcdPuts(fd, "Welcome To--->");
```

Display **"Welcome To—>"** at the specified location of LCD1602.

```
    for(i=0; i<16; i++){
        lcdPosition(fd, i, 0);
        lcdPutchar(fd, *(myBuf+i));
        delay(100);
    }
```

Use the **lcdPosition()** function to place the cursor at col i and row 0(the top line) for subsequent text entry. Then the characters in the array **myBuf []** are displayed one by one to the LCD1602.

* is the address of myBuf, the real address of characters stored in memory. After calling lcdPutchar(fd, *(myBuf+ I)), the program will find the real address of the character, read the information stored in the address, and display it on the LCD screen.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 17_Lcd1602.py
```

You may see the “SunFounder” and “hello, world” appear one by one on the LCD.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

from time import sleep

class LCD:
    # commands
    LCD_CLEARDISPLAY                = 0x01
    LCD_RETURNHOME              = 0x02
    LCD_ENTRYMODESET                = 0x04
    LCD_DISPLAYCONTROL              = 0x08
    LCD_CURSORSHIFT                 = 0x10
    LCD_FUNCTIONSET                 = 0x20
    LCD_SETCGRAMADDR                = 0x40
    LCD_SETDDRAMADDR                = 0x80

    # flags for display entry mode
    LCD_ENTRYRIGHT          = 0x00
    LCD_ENTRYLEFT           = 0x02
    LCD_ENTRYSHIFTINCREMENT         = 0x01
    LCD_ENTRYSHIFTDECREMENT         = 0x00

    # flags for display on/off control
    LCD_DISPLAYON           = 0x04
    LCD_DISPLAYOFF          = 0x00
    LCD_CURSORON            = 0x02
    LCD_CURSOROFF           = 0x00
    LCD_BLINKON             = 0x01
    LCD_BLINKOFF            = 0x00

    # flags for display/cursor shift
    LCD_DISPLAYMOVE         = 0x08
    LCD_CURSORMOVE          = 0x00
```

(continues on next page)

(continued from previous page)

```
    # flags for display/cursor shift
    LCD_DISPLAYMOVE         = 0x08
    LCD_CURSORMOVE          = 0x00
    LCD_MOVERIGHT           = 0x04
    LCD_MOVELEFT            = 0x00

    # flags for function set
    LCD_8BITMODE            = 0x10
    LCD_4BITMODE            = 0x00
    LCD_2LINE                       = 0x08
    LCD_1LINE                       = 0x00
    LCD_5x10DOTS            = 0x04
    LCD_5x8DOTS             = 0x00

    def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO = None):
        # Emulate the old behavior of using RPi.GPIO if we haven't been given
        # an explicit GPIO interface to use
        if not GPIO:
            import RPi.GPIO as GPIO
            self.GPIO = GPIO
            self.pin_rs = pin_rs
            self.pin_e = pin_e
            self.pins_db = pins_db

            self.used_gpio = self.pins_db[:]
            self.used_gpio.append(pin_e)
            self.used_gpio.append(pin_rs)

            self.GPIO.setwarnings(False)
            self.GPIO.setmode(GPIO.BCM)
            self.GPIO.setup(self.pin_e, GPIO.OUT)
            self.GPIO.setup(self.pin_rs, GPIO.OUT)

            for pin in self.pins_db:
                self.GPIO.setup(pin, GPIO.OUT)

        self.write4bits(0x33) # initialization
        self.write4bits(0x32) # initialization
        self.write4bits(0x28) # 2 line 5x7 matrix
        self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor
        self.write4bits(0x06) # shift cursor right

        self.displaycontrol = self.LCD_DISPLAYON | self.LCD_CURSOROFF | self.LCD_
↪BLINKOFF

        self.displayfunction = self.LCD_4BITMODE | self.LCD_1LINE | self.LCD_5x8DOTS
        self.displayfunction |= self.LCD_2LINE

        """ Initialize to default text direction (for romance languages) """
        self.displaymode =  self.LCD_ENTRYLEFT | self.LCD_ENTRYSHIFTDECREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode) #  set the entry␣
↪mode

        self.clear()

    def begin(self, cols, lines):
        if (lines > 1):
```

(continues on next page)

(continued from previous page)

```
            self.numlines = lines
            self.displayfunction |= self.LCD_2LINE
            self.currline = 0

    def home(self):
        self.write4bits(self.LCD_RETURNHOME) # set cursor position to zero
        self.delayMicroseconds(3000) # this command takes a long time!

    def clear(self):
        self.write4bits(self.LCD_CLEARDISPLAY) # command to clear display
        self.delayMicroseconds(3000)        # 3000 microsecond sleep, clearing the␣
↪display takes a long time

    def setCursor(self, col, row):
        self.row_offsets = [ 0x00, 0x40, 0x14, 0x54 ]

        if ( row > self.numlines ):
            row = self.numlines - 1 # we count rows starting w/0

        self.write4bits(self.LCD_SETDDRAMADDR | (col + self.row_offsets[row]))

    def noDisplay(self):
        # Turn the display off (quickly)
        self.displaycontrol &= ~self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def display(self):
        # Turn the display on (quickly)
        self.displaycontrol |= self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noCursor(self):
        # Turns the underline cursor on/off
        self.displaycontrol &= ~self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def cursor(self):
        # Cursor On
        self.displaycontrol |= self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def DisplayLeft(self):
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE | self.LCD_
↪MOVELEFT)

    def scrollDisplayRight(self):
```

(continues on next page)

(continued from previous page)

```
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE | self.LCD_
↪MOVERIGHT);

    def leftToRight(self):
        # This is for text that flows Left to Right
        self.displaymode |= self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode);

    def rightToLeft(self):
        # This is for text that flows Right to Left
        self.displaymode &= ~self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def autoscroll(self):
        # This will 'right justify' text from the cursor
        self.displaymode |= self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def noAutoscroll(self):
        # This will 'left justify' text from the cursor
        self.displaymode &= ~self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def write4bits(self, bits, char_mode=False):
        # Send command to LCD
        self.delayMicroseconds(1000) # 1000 microsecond sleep
        bits=bin(bits)[2:].zfill(8)
        self.GPIO.output(self.pin_rs, char_mode)
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i], True)
        self.pulseEnable()
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4,8):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i-4], True)
        self.pulseEnable()

    def delayMicroseconds(self, microseconds):
        seconds = microseconds / float(1000000)     # divide microseconds by 1␣
↪million for seconds
        sleep(seconds)

    def pulseEnable(self):
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)           # 1 microsecond pause - enable pulse must␣
↪be > 450ns
        self.GPIO.output(self.pin_e, True)
        self.delayMicroseconds(1)           # 1 microsecond pause - enable pulse must␣
↪be > 450ns
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)           # commands need > 37us to settle
```

(continues on next page)

(continued from previous page)

```
    def message(self, text):
        # Send string to LCD. Newline wraps to second line
        for char in text:
            if char == '\n':
                self.write4bits(0xC0) # next line
            else:
                self.write4bits(ord(char),True)

    def destroy(self):
        self.GPIO.cleanup(self.used_gpio)

def print_msg():
    print ("========================================")
    print ("|                LCD1602               |")
    print ("|    ------------------------------    |")
    print ("|         D4 connect to BCM25          |")
    print ("|         D5 connect to BCM24          |")
    print ("|         D6 connect to BCM23          |")
    print ("|         D7 connect to BCM18          |")
    print ("|         RS connect to BCM27          |")
    print ("|         CE connect to BCM22          |")
    print ("|          RW connect to GND           |")
    print ("|                                      |")
    print ("|           Control LCD1602            |")
    print ("|                                      |")
    print ("|                            SunFounder|")
    print ("========================================\n")
    print ('Program is running...')
    print ('Please press Ctrl+C to end the program...')
    #input ("Press Enter to begin\n")

def main():
    global lcd
    print_msg()
    lcd = LCD()
    line0 = "  sunfounder.com"
    line1 = "---SUNFOUNDER---"

    lcd.clear()
    lcd.message("Welcome to --->\n  sunfounder.com")
    sleep(3)

    msg = "%s\n%s" % (line0, line1)
    while True:
        lcd.begin(0, 2)
        lcd.clear()
        for i in range(0, len(line0)):
            lcd.setCursor(i, 0)
            lcd.message(line0[i])
            sleep(0.1)
        for i in range(0, len(line1)):
            lcd.setCursor(i, 1)
            lcd.message(line1[i])
            sleep(0.1)
        sleep(1)

if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    try:
        main()
    except KeyboardInterrupt:
        lcd.clear()
        lcd.destroy()
```

Note: Because the source code contains so many definitions, we only list few code here. Please download the complete code from the address marked in the document.

Code Explanation

```
    line0 = "  sunfounder.com"
    line1 = "---SUNFOUNDER---"
```

Define 2 lines of characters that will be displayed on the LCD 1602.

```
    lcd.message("Welcome to --->\n  sunfounder.com")
```

On LCD1602, **"Welcome to —>n sunfounder.com"** pops up.

```
        lcd.begin(0, 2)
```

Initializes the LCD screen and specifies the dimensions (width and height) of the display. begin() function needs to be called before any other LCD library commands.

```
        lcd.clear()
```

This function is used to clear the lcd screen. After calling this function, all information displayed on the screen will be cleared.

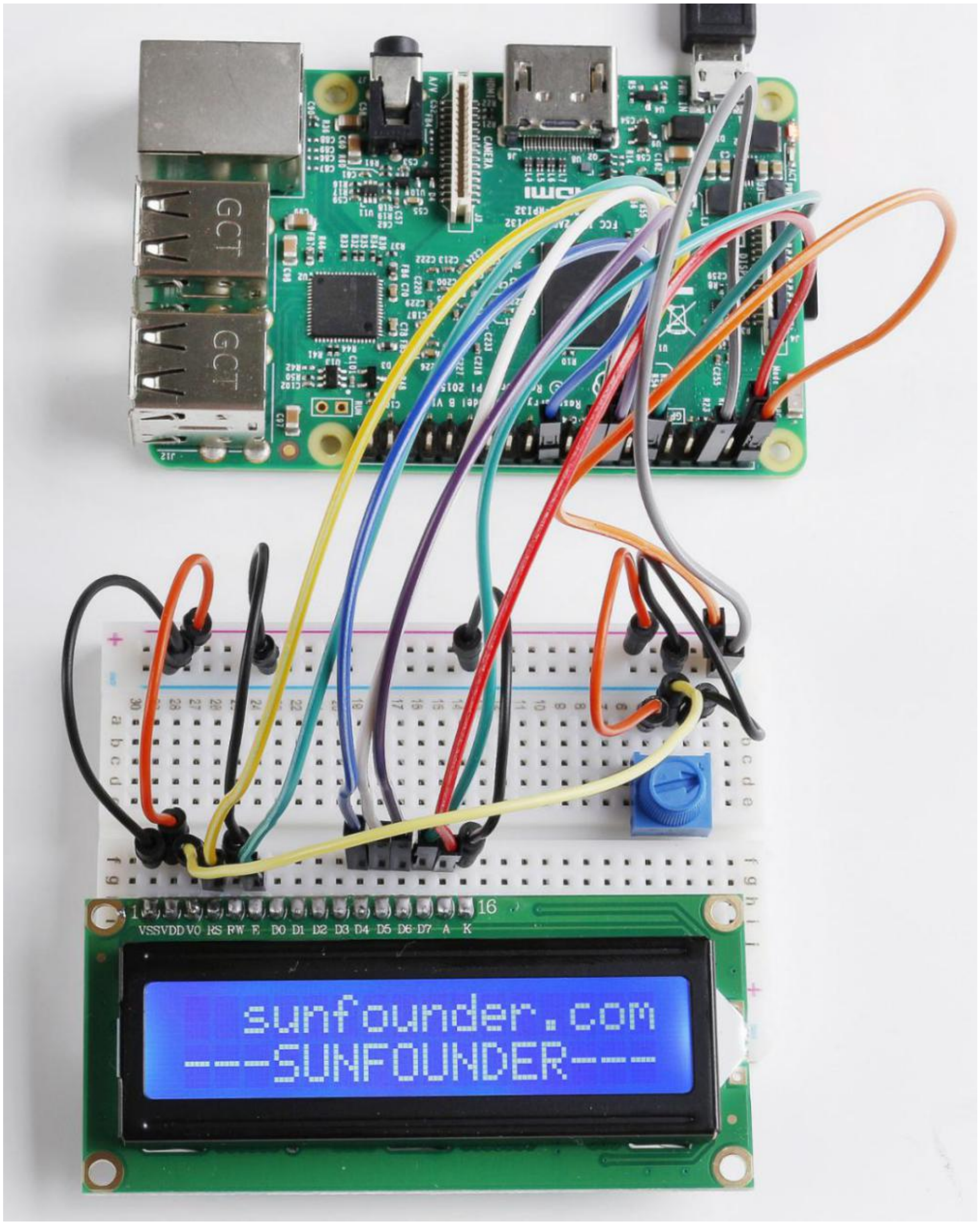
```
            lcd.setCursor(i, 0)
```

Set the position of the cursor at col i and row 0 (the first line) for subsequent text entry.

```
            lcd.message(line0[i])
```

The characters in the array **line0[]** will be displayed at the specified location one by one.

Phenomenon Picture

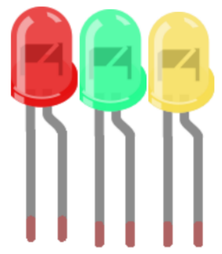
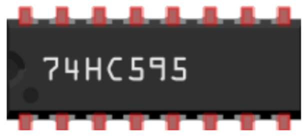


Lesson 18 Driving LEDs by 74HC595

Introduction

In this lesson, we will learn how to use 74HC595 to make eight LEDs blink regularly. Now let's get started!

Newly Added Components

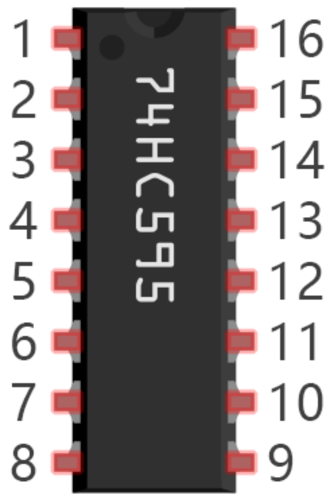
8 * LED	1 * 74HC595	8 * Resistor (220Ω)
		

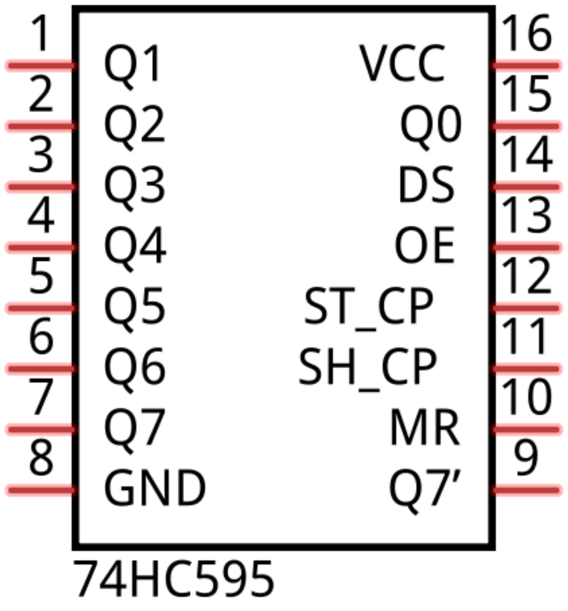
Principle

74HC595

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

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





Pins of 74HC595 and their Functions:

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

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

MR: Reset pin, active at low level;

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

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

OE: Output enable pin, active at low level.

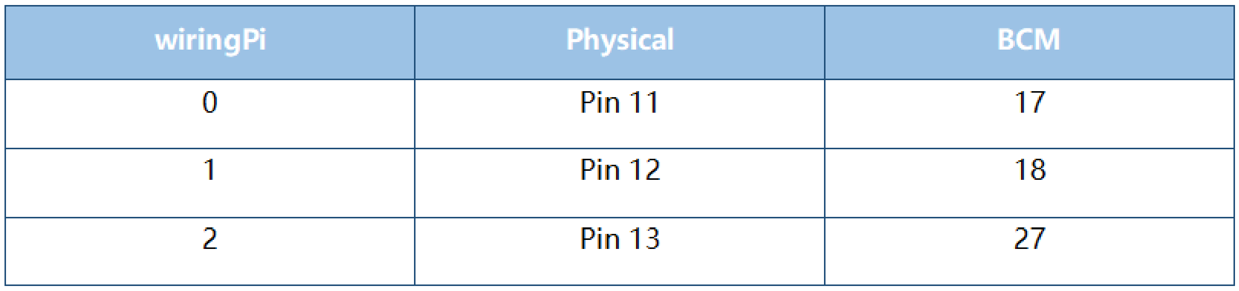
DS: Serial data input pin.

VCC: Positive supply voltage.

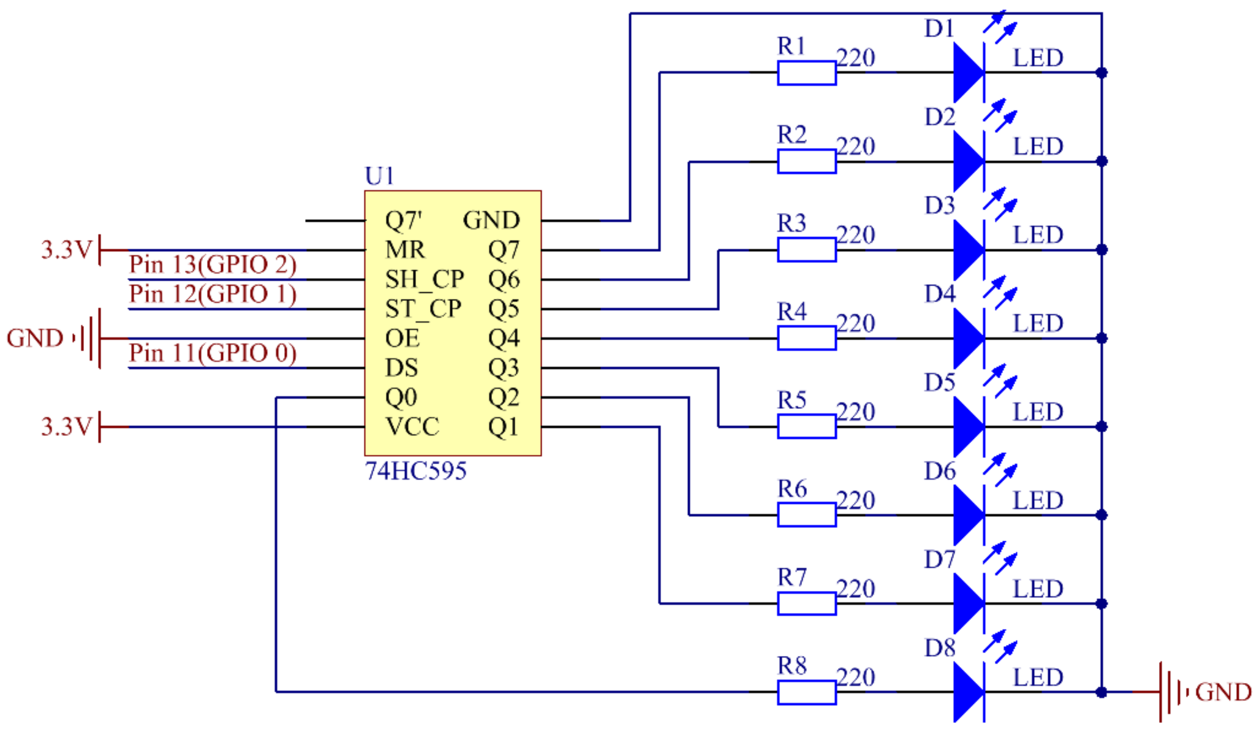
GND: Ground.

Schematic Diagram

In the experiment **MR** is connected to **3.3V** (HIGH Level) and **OE** to **GND** (LOW Level). Therefore, the data is input into the rising edge of **SHcp** and enters the memory register through the rising edge. In the rising edge of the **SHcp**, the data in the shift register moves successively one bit in one time, i.e. data in **Q1** moves to **Q2**, and so forth. In the rising edge of **STcp**, data in the shift register moves into the memory register. All data will be moved to the memory register 8 times. Then the data in the memory register is output to the bus (**Q0-Q7**).



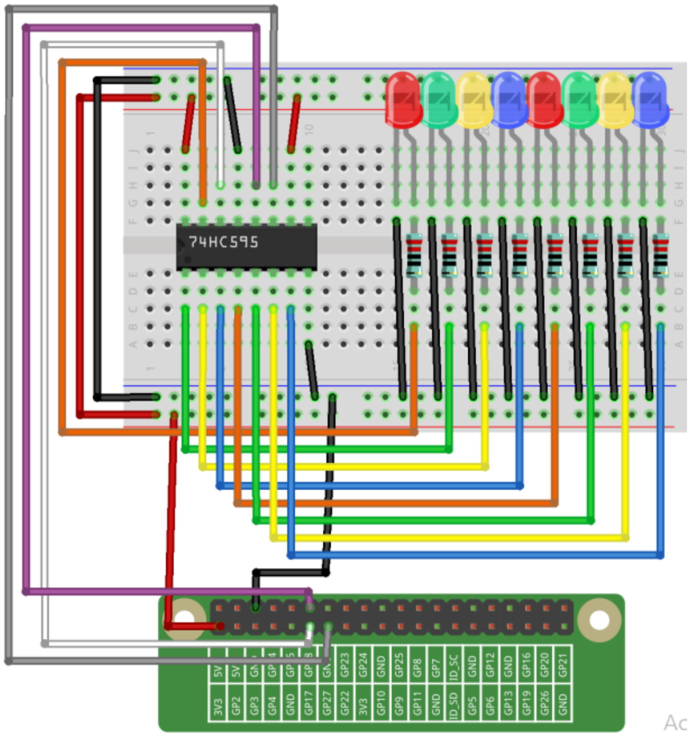
wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27



Build the Circuit

Note: Recognize the direction of the chip according to the concave on it.

LED	74HC595	Raspberry Pi
LED1	Q7	
LED2	Q6	
LED3	Q5	
LED4	Q4	
LED5	Q3	
LED6	Q2	
LED7	Q1	
LED8	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_18_Driving_Leds_by_74hc595
```

2. Compile the code.

```
gcc 18_74hc595.c -lwiringPi
```

3. Run the executable file.

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

As the code runs, you can see these eight LEDs are lit up from left to right, and then all LEDs light up and flash 3 times. After that, these eight LEDs are lit from right to left, then they all turn on before flashing 3 times. This loop continues in this way.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

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

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

unsigned char LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};

void pulse(int pin){
    digitalWrite(pin, 0);
    digitalWrite(pin, 1);
}

void SIPO(unsigned char byte){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));
        pulse(SRCLK);
    }
}

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

    digitalWrite(SDI, 0);
```

(continues on next page)

(continued from previous page)

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

int main(void){
    int i;

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

    init();

    while(1){
        for(i=0;i<8;i++){
            SIPO(LED[i]);
            pulse(RCLK);
            delay(150);
        }
        delay(500);

        for(i=0;i<3;i++){
            SIPO(0xff);
            pulse(RCLK);
            delay(100);
            SIPO(0x00);
            pulse(RCLK);
            delay(100);
        }
        delay(500);

        for(i=0;i<8;i++){
            SIPO(LED[8-i-1]);
            pulse(RCLK);
            delay(150);
        }
        delay(500);

        for(i=0;i<3;i++){
            SIPO(0xff);
            pulse(RCLK);
            delay(100);
            SIPO(0x00);
            pulse(RCLK);
            delay(100);
        }
        delay(500);
    }
    return 0;
}
```

Code Explanation

```
void pulse(int pin){
    digitalWrite(pin, 0);
    digitalWrite(pin, 1);
}
```

Define an pulse function to generate an pulse.

```
void SIPO(unsigned char byte){
    int i;
    for(i=0;i<8;i++){
        digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));
        pulse(SRCLK);
    }
}
```

The function **SIPO** is used to assign the byte data to **SDI(DS)** by bits.

Among them, the inequality in statement **digitalWrite()** ((byte & (0x80>>i))>0) is used to confirm each value written into the register and it realizes the function by Shift operator (>>).

For example, if byte=0x01:

When the condition "i=0" is met, 0x80(1000 0000)>>0 becomes 0x80(1000 0000), if byte&0x80=0, the inequality is false, and output 0 (false).

If "i=1" is true, 0x80>>1 changes into 0x40(0100 0000); when byte&0x40=0, output 0.

Deduce the rest from this, when and only when "i=8" is met, 0x80>>8 is 0x01(0000 0001), byte&0x01=1, and output 1(true).

Pulse(SRCLK) generates a rising edge pulse on input pin of shift register to shift the 8 bit data on SDI to shift register successively.

In a word, this **for** loop produces 8 times to shift the 8 bits of 0000 0001 to shift register.

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

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

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

```
        for(i=0;i<8;i++){
            SIPO(LED[i]);
            pulse(RCLK);
            delay(150);
        }
```

Use the **for** loop to count 8 times in cycle, and write a 1-bit data to the SDI each time.

When i=0, LED[0]=0x01(0000 0001), through the function SIPO(LED[0]), shifts the 8 bits of 0x01 to shift register successively. Pulse(SRCLK) generates a rising edge signal on input pin of storage register to shift the 0x01 on shift register to storage register at once. Then the data in the memory register are output to the bus (Q7-Q0), so you'll see the LED on Q0 is lit up. After loops, output all eight elements in the array LED[i] to the bus (Q7-Q0), and you'll see eight LEDs turning on from left to right.

```
    for(i=0;i<3;i++){
        SIPO(0xff);
        pulse(RCLK);
        delay(100);
        SIPO(0x00);
        pulse(RCLK);
        delay(100);
    }
```

In this part, the **for** loop is used to three times repeat the program in **for()** statement. SIPO(0xff) means 8 LEDs are lit up, SIPO(0x00) represents 8 LEDs turn off. That is, let 8 LEDs turn off 3 times simultaneously.

```
    for(i=0;i<8;i++){
        SIPO(LED[8-i-1]);
        pulse(RCLK);
        delay(150);
    }
```

By the same token, this for loop allows 8 LEDs be lit up one by one in reverse order. Here, **i** gradually increases from 0, and 8-i-1 gradually decreases. SIPO(LED[8-i-1]) can be used to call the data in the LED[] array from back to front so that you can get 8 LEDs lit up one by one in reverse order.

```
    for(i=0;i<3;i++){
        SIPO(0xff);
        pulse(RCLK);
        delay(100);
        SIPO(0x00);
        pulse(RCLK);
        delay(100);
    }
```

Then, make the eight LEDs turn on or off 3 times simultaneously.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 18_74HC595.py
```

As the code runs, you can see these eight LEDs are lit up from left to right, and then all LEDs light up and flash 3 times. After that, these eight LEDs are lit from right to left, then they all turn on before flashing 3 times. This loop continues in this way.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

#================================================
#
#   This program is for SunFounder SuperKit for Rpi.
#
#   Extend use of 8 LED with 74HC595.
#
#   Change the  WhichLeds and sleeptime value under
#   loop() function to change LED mode and speed.
#
#=================================================

import RPi.GPIO as GPIO
import time

SDI   = 17
RCLK  = 18
SRCLK = 27

#===============   LED Mode Defne ================
#   You can define yourself, in binay, and convert it to Hex
#   8 bits a group, 0 means off, 1 means on
#   like : 0101 0101, means LED1, 3, 5, 7 are on.(from left to right)
#   and convert to 0x55.

LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80]    #original mode
BLINK = [0xff,0x00,0xff,0x00,0xff,0x00]          #blink
#=================================================

def print_message():
    print ("========================================")
    print ("|           LEDs with 74HC595          |")
    print ("|    ------------------------------    |")
    print ("|         SDI connect to GPIO 0        |")
    print ("|         RCLK connect to GPIO 1       |")
    print ("|        SRCLK connect to GPIO 2       |")
    print ("|                                      |")
    print ("|       Control LEDs with 74HC595      |")
    print ("|                                      |")
    print ("|                            SunFounder|")
    print ("========================================\n")
    print ('Program is running...')
    print ('Please press Ctrl+C to end the program...')
    #input ("Press Enter to begin\n")

def setup():
    GPIO.setmode(GPIO.BCM)    # Number GPIOs by its BCM location
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
```

(continues on next page)

(continued from previous page)

```
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)

# Shift the data to 74HC595
def hc595_shift(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)

def main():
    print_message()
    mode = LED0
    sleeptime = 0.15        # Change speed, lower value, faster speed
    blink_sleeptime = 0.15

    while True:
        # Change LED status from mode
        for onoff in mode:
            hc595_shift(onoff)
            time.sleep(sleeptime)

        for onoff in BLINK:
            hc595_shift(onoff)
            time.sleep(blink_sleeptime)

        # Change LED status from mode reverse
        for onoff in reversed(mode):
            hc595_shift(onoff)
            time.sleep(sleeptime)

        for onoff in BLINK:
            hc595_shift(onoff)
            time.sleep(blink_sleeptime)

def destroy():
    GPIO.cleanup()

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

Code Explanation

```
LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80]     #original mode
```

Use array to define LED flashing mode, you can also customize several hexadecimals to light up 8 LEDs.

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

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LED lights are set to output mode, default low level.

```
def hc595_shift(dat):
```

Define a function **hc595_shift()** to output the 8 bits of **dat** to Q0-Q7.

```
for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
```

Assign the **dat** to SDI(DS) according to bits. Pin **SRCLK** will convert from low to high, and generate a rising edge pulse, then shift the data in pin SDI to shift register. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

```
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)
```

Pin **RCLK** converts from low to high and generate a rising edge, then shift data from shift register to storage register. Finally the data in the memory register is output to the bus (Q0-Q7).

```
        for onoff in mode:
            hc595_shift(onoff)
            time.sleep(sleeptime)
```

Here we use a onoff variable to control the LED that changes within the range of mode, and hc595_shift (onoff) means lighting up LED one by one. For example, when mode is the first datum in LED0, or 0x01, onoff = mode = 0x01 = 00000001. In this course, the LED is lit by high level. To put it another way, it is Hc595_shift (onoff) = hc595_shift (00000001) that lights up the last LED. Along the same vein, when the value of mode is the second datum of LED0 (onoff = 0x02 = 00000010), the second last LED turns on.

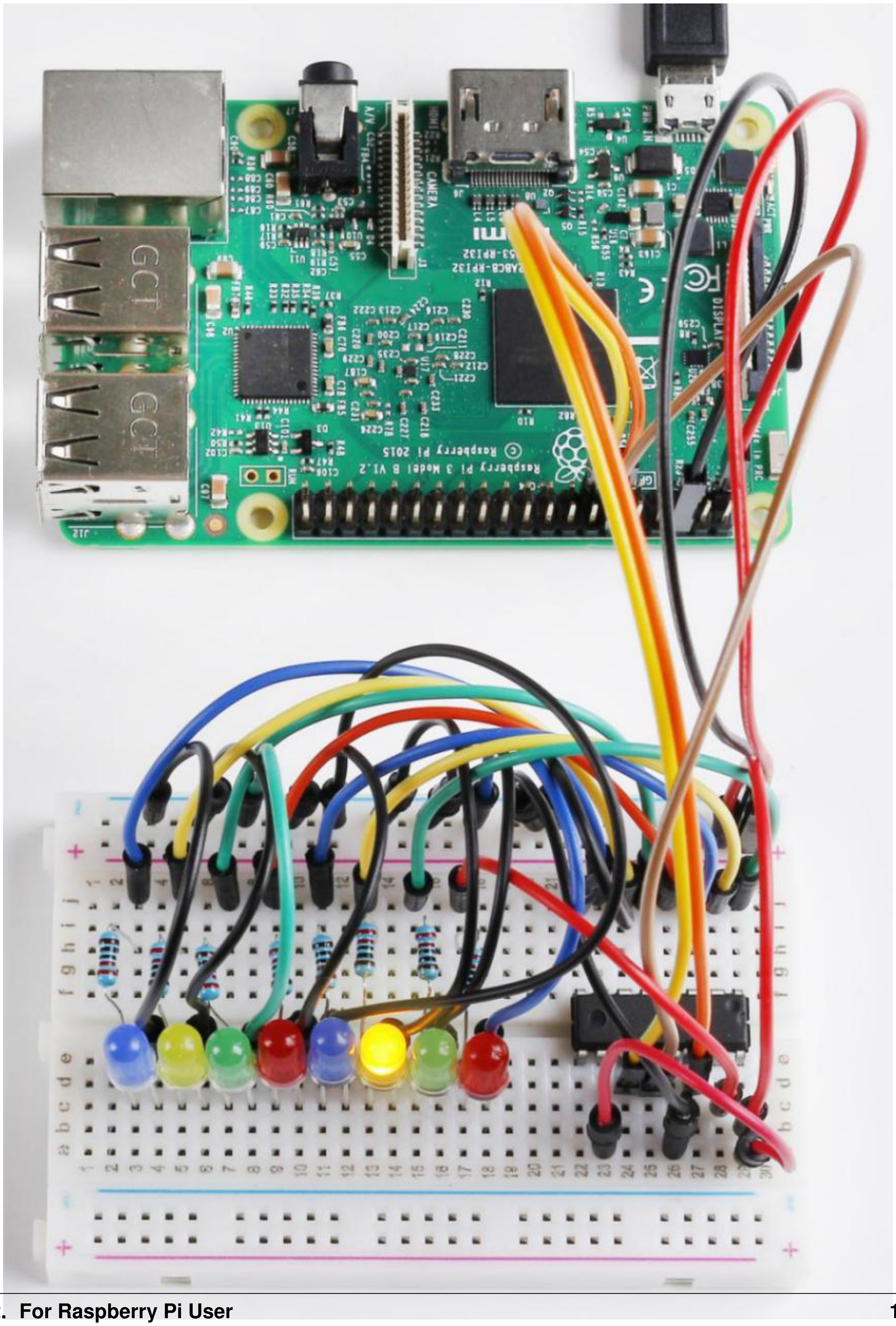
```
        for onoff in reversed(mode):
            hc595_shift(onoff)
            time.sleep(sleeptime)
```

According to the same principle, a reversed is used here to get LEDs lit up in reverse order.

```
        for onoff in BLINK:
            hc595_shift(onoff)
            time.sleep(blink_sleeptime)
```

In the same way, light up 8 LEDs; exactly, 8 LEDs are turned on or off 3 times synchronously in the same pattern as that of the LEDs in the BLINK array.

Phenomenon Picture

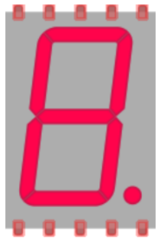


Lesson 19 7-segment

Introduction

Generally, there are two ways to drive a single 7-Segment Display. One way is to connect its 8 pins directly to eight ports on the Raspberry Pi. Or you can connect the 74HC595 to three ports of the Raspberry Pi and then the 7- segment display to the 74HC595. In this experiment, we will use the latter. By this way, we can save five ports – considering the board's limited ports, and this is very important. Now let's get started!

Newly Added Components

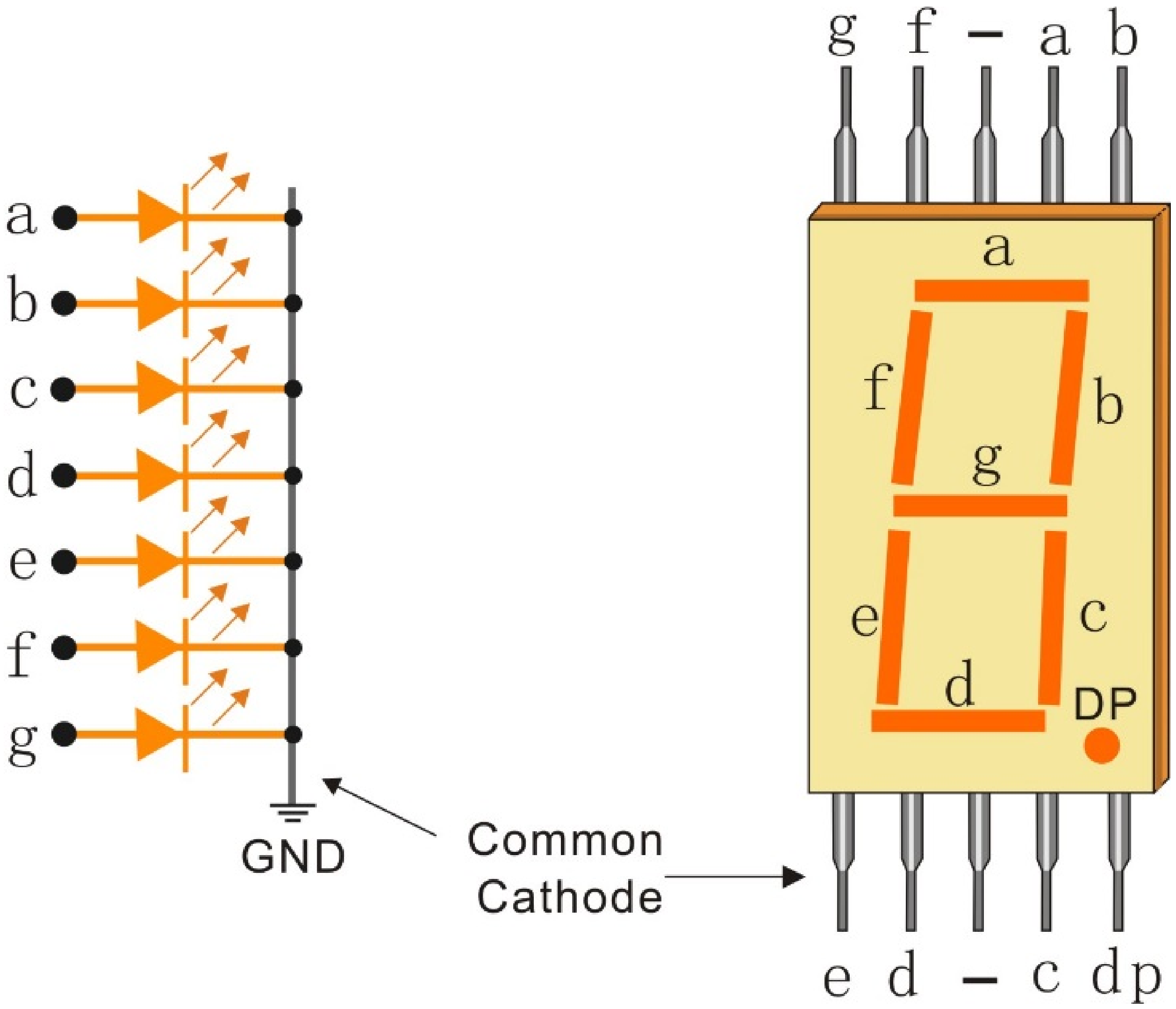
1 * 7-Segment Display	1 * 74HC595	1 * Resistor (220Ω)
		

Principle

7-Segment Display

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

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



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

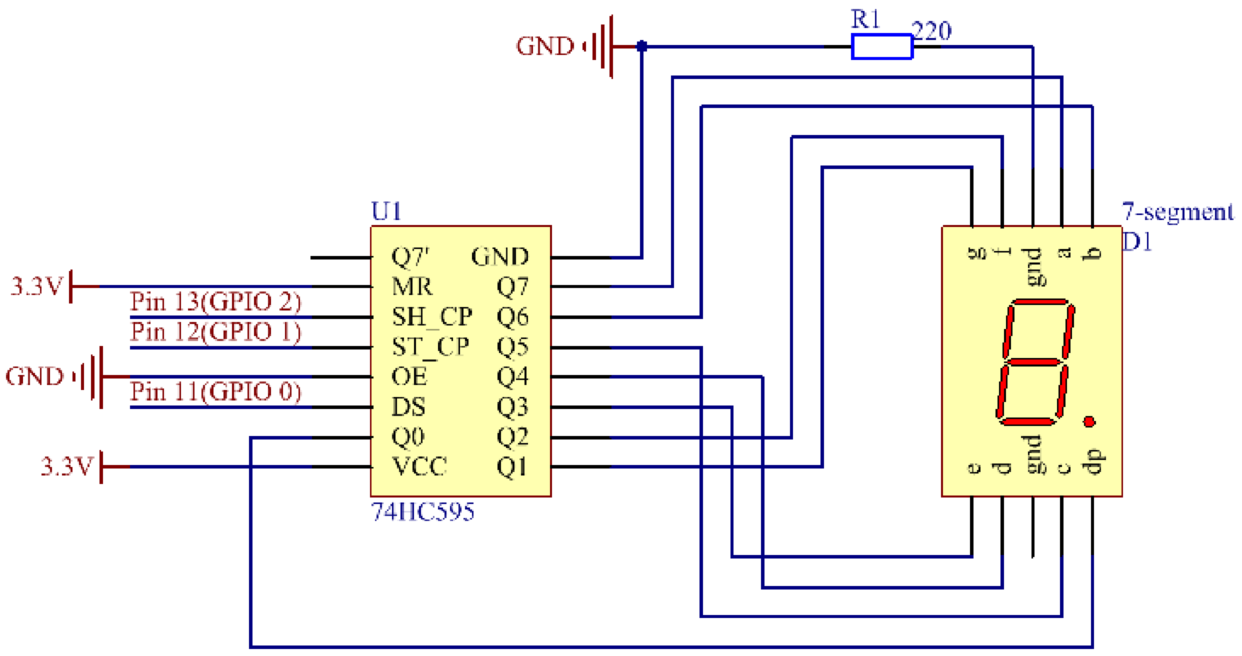
Display Codes

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

Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77
1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39
3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d	.	10000000	0x80
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

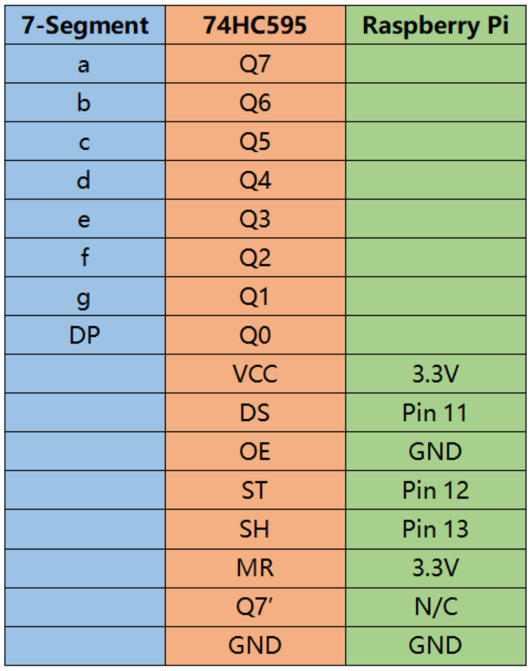
Schematic Diagram

wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27

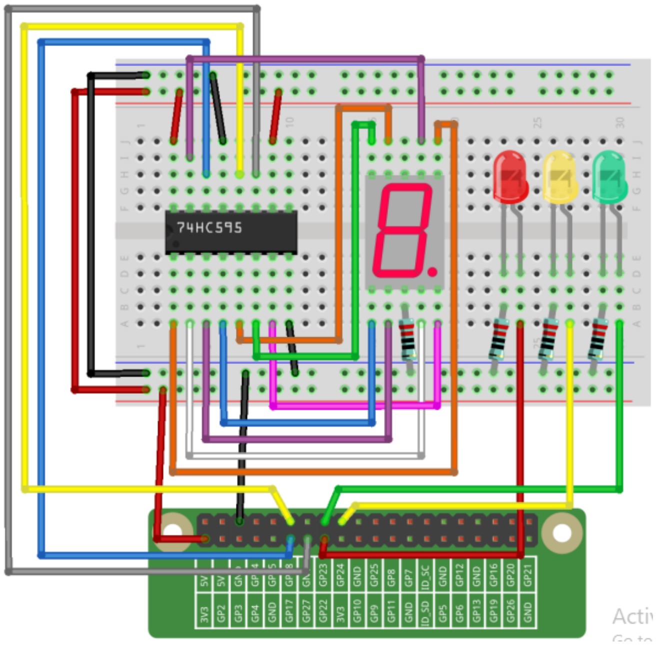


Build the Circuit

Note: Recognize the direction of the chip according to the concave on it.



7-Segment	74HC595	Raspberry Pi
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_19_7-segment
```

2. Compile the code.

```
gcc 19_7-Segment.c -lwiringPi
```

3. Run the executable file.

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

You may see 0 to 9 and A to F on the 7-Segment Display.

Note: If it does not work after running, or there is an error prompt: “wiringPi.h: No such file or directory”, please refer to *C code is not working?*.

Code

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

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

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

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

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

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

        digitalWrite(RCLK, 1);
        delay(1);
        digitalWrite(RCLK, 0);
}

int main(void){
    int i;

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

    init();

    while(1){
        for(i=0;i<17;i++){
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
    return 0;
}
```

Code Explanation

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

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

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

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

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

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

Assign the **dat** value to SDI(DS) by bits. Then shift them to the shift register by bits. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

```
        digitalWrite(RCLK, 1);
        delay(1);
        digitalWrite(RCLK, 0);
```

Pin RCLK converts from low to high and generates a rising edge, then shifts data from shift register to storage register. Finally the data in the memory register are output to the bus (Q0-Q7).

```
        for(i=0;i<17;i++){
            hc595_shift(SegCode[i]);
            delay(500);
        }
```

In the **for** loop, output 16 values from array **Segcode[]** to 7-Segment Display.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 19_7-Segment.py
```

You may see 0 to 9 and A to F on the 7-Segment Display.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

```
import RPi.GPIO as GPIO
import time

# Set up pins
SDI   = 17
RCLK  = 18
SRCLK = 27

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

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

# Shift the data to 74HC595
def hc595_shift(dat):
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)

def main():
    while True:
        # Shift the code one by one from segCode list
        for code in segCode:
            hc595_shift(code)
            time.sleep(0.5)

def destroy():
    GPIO.cleanup()

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

Code Explanation

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

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level.

```
def hc595_shift(dat):
```

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

```
    for bit in range(0, 8):
        GPIO.output(SDI, 0x80 & (dat << bit))
        GPIO.output(SRCLK, GPIO.HIGH)
        time.sleep(0.001)
        GPIO.output(SRCLK, GPIO.LOW)
```

Assign the **dat** value to SDI(DS) by bits. Then shift them to the shift register by bits. Execute the loop 8 times to shift the 8 bits of **dat** to the shift register in proper order.

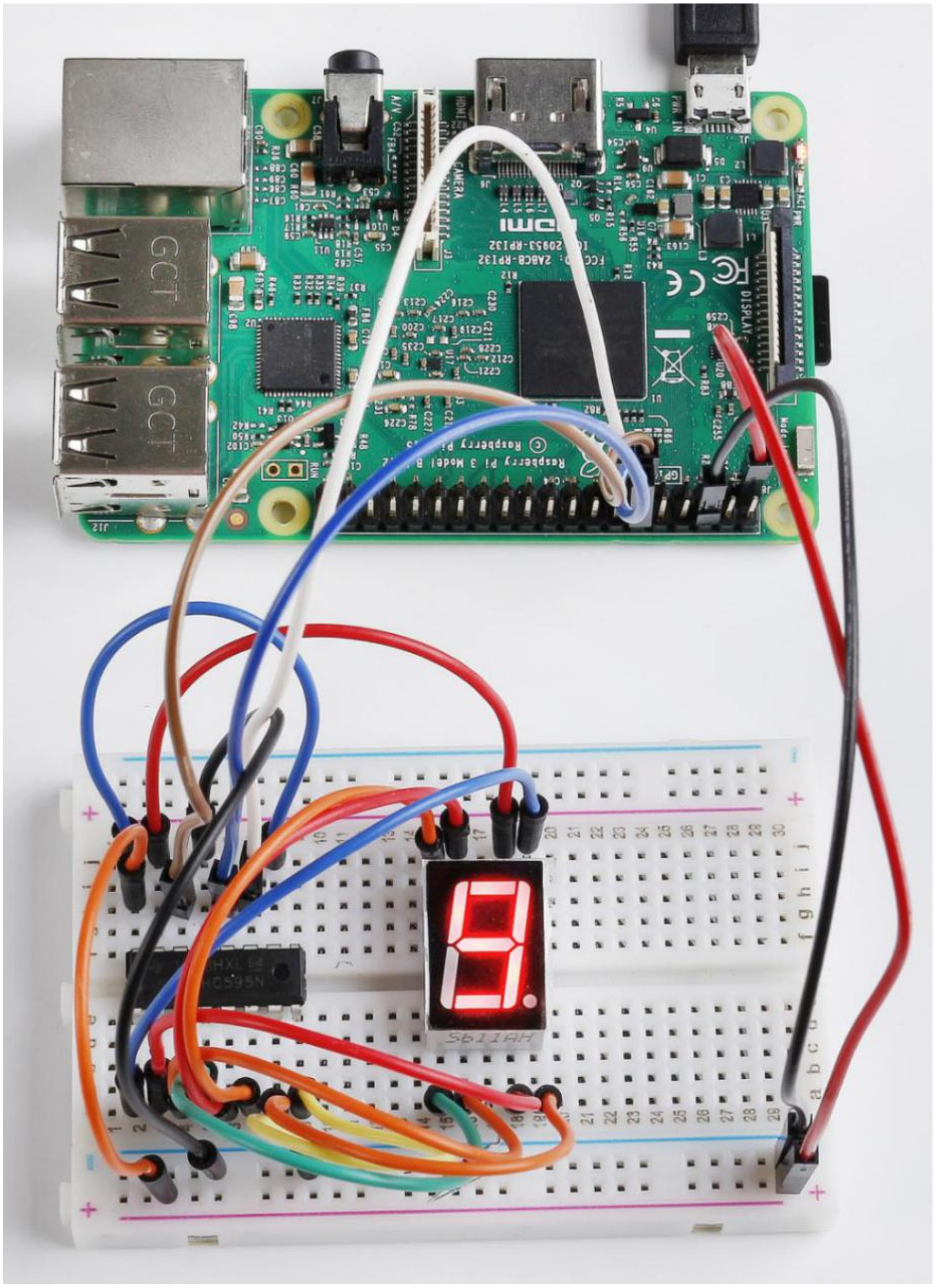
```
    GPIO.output(RCLK, GPIO.HIGH)
    time.sleep(0.001)
    GPIO.output(RCLK, GPIO.LOW)
```

Pin **RCLK** converts from low to high and generates a rising edge, then shifts data from shift register to storage register. Finally the data in the memory register are output to the bus (Q0-Q7).

```
        for code in segCode:
            hc595_shift(code)
            time.sleep(0.5)
```

In the **for** loop, output 16 values from array **Segcode []** to 7-Segment Display.

Phenomenon Picture

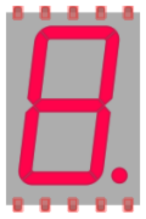
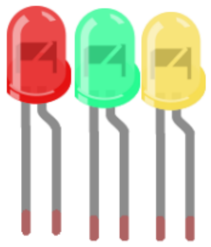


Lesson 20 Traffic Light

Introduction

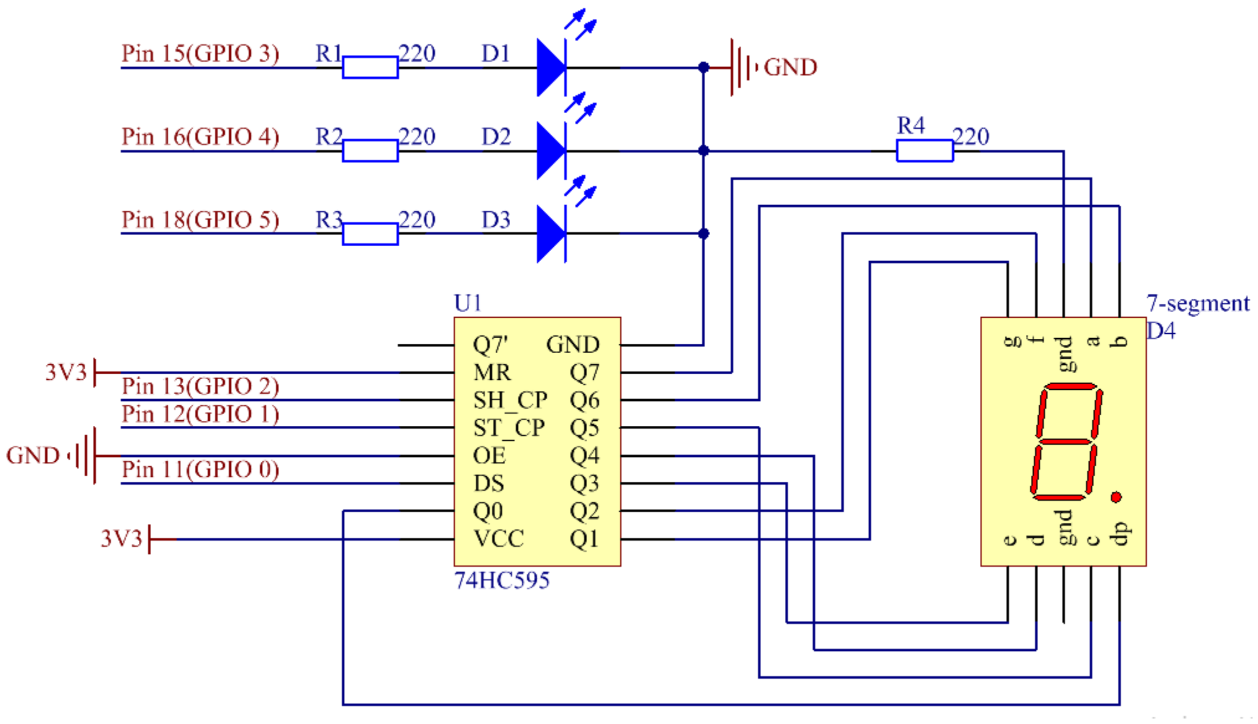
In last lesson, we learned how to use a 74HC595 chip to drive a 7-Segment Display. Based on that, we can apply it more widely now, such as making a simple traffic light. Now let's get started!

Newly Added Components

1 * 7-Segment Display	1 * 74HC595	4 * Resistor (220 Ω)	3 * LED
			

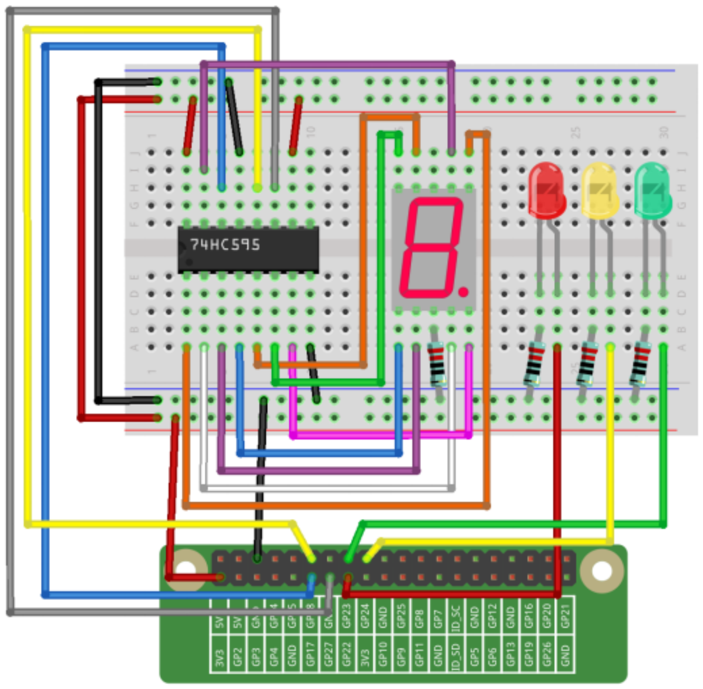
Schematic Diagram

wiringPi	Physical	BCM
0	Pin 11	17
1	Pin 12	18
2	Pin 13	27
3	Pin15	22
4	Pin16	23
5	Pin18	24



Build the Circuit

7-Segment	74HC595	Raspberry Pi
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	3.3V
	DS	Pin 11
	OE	GND
	ST	Pin 12
	SH	Pin 13
	MR	3.3V
	Q7'	N/C
	GND	GND
Red LED		Pin 15
Green LED		Pin 16
Yellow		Pin 18



For C Language Users

Command

1. Go to the folder of the code.

```
cd /home/pi/electronic-kit/for-raspberry-pi/c/Lesson_20_TrafficLight
```

2. Compile the code.

```
gcc 20_TrafficLight.c -lwiringPi
```

3. Run the executable file.

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

You can see the following phenomenon of traffic lights. The red LED lights up for 9 seconds, green LED for 5s, and yellow LED for 3s.

Note: If it does not work after running, or there is an error prompt: "wiringPi.h: No such file or directory", please refer to *C code is not working?*.

Code

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define      SDI      0   //serial data input(DS)
#define      RCLK     1   //memory clock input(STCP)
#define      SRCLK    2    //shift register clock input(SHCP)
const int ledPin[]={3,4,5};    //Define 3 LED pin(Red, Green, Yellow)
unsigned char SegCode[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,
↪0x7c,0x39,0x5e,0x79,0x71,0x80};

int greentime = 5;
int yellowtime = 3;
int redtime = 9;
int colorState = 0;
char *lightColor[]={"Red","Green","Yellow"};
int counter = 9;

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

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

      for(int i=0;i<3;i++){
```

(continues on next page)

(continued from previous page)

```
        pinMode(ledPin[i],OUTPUT);
        digitalWrite(ledPin[i],LOW);
    }
}

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

void timer(int  sig){       //Timer function
    if(sig == SIGALRM){
        counter --;
        alarm(1);
        if(counter == 0){
            if(colorState == 0) counter = greentime;
            if(colorState == 1) counter = yellowtime;
            if(colorState == 2) counter = redtime;
            colorState = (colorState+1)%3;
        }
        printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);
    }
}

void display(int num)
{
    hc595_shift(SegCode[num%10]);
    delay(1);
}

void lightup(int state)
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],LOW);
    }
        digitalWrite(ledPin[state],HIGH);
}

int main(void)
{
    int i;

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

    init();
```

(continues on next page)

(continued from previous page)

```
    signal(SIGALRM,timer);  //configure the timer
    alarm(1);               //set the time of timer to 1s
    while(1){
        display(counter);
        lightup(colorState);
    }
    return 0;
}
```

Code Explanation

```
int greentime = 5;
int yellowtime = 3;
int redtime = 9;
```

Define the duration of lighting of three LEDs. Since what we use is a 7-Segment Display here, we shorten the length of seconds of lighting of traffic lights, setting the green LED to light up for 5 seconds, the yellow LED to 3 seconds, and the red LED to 9 seconds.

```
int colorState = 0;
int counter = 9;
```

The variable **colorState** corresponds to the state of the traffic lights, and we only need to do a simple calculation of **colorState** to indicate the order change of the state of the traffic lights. The Variable counter is used to count down the time to each traffic light status and will be output on a 7-Segment Display.

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

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

      for(int i=0;i<3;i++){
        pinMode(ledPin[i],OUTPUT);
        digitalWrite(ledPin[i],LOW);
    }
}
```

Initialize pins. Set all control pins of 74HC595 to output mode and initialize them to low level. At the same time, the LEDs are set to output mode, default low level.

```
void timer(int  sig){
    if(sig == SIGALRM){
        counter --;
        alarm(1);
        if(counter == 0){
            if(colorState == 0) counter = greentime;
            if(colorState == 1) counter = yellowtime;
            if(colorState == 2) counter = redtime;
            colorState = (colorState+1)%3;
        }
```

(continues on next page)

(continued from previous page)

```
        printf("counter : %d \t light color: %s \n",counter,lightColor[colorState]);
    }
}
```

On this timer, counter decreases gradually with every second passing, and when it goes to 0, the state of the traffic light changes accordingly.

```
void lightup(int state)
{
    for(int i=0;i<3;i++){
        digitalWrite(ledPin[i],LOW);
    }
        digitalWrite(ledPin[state],HIGH);
}
```

The function is to turn off all the lights first, and then light up the corresponding LED according to the value of the traffic light state.

For Python Language Users

Command

1. Go to the folder of the code.

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

2. Run the code.

```
sudo python3 20_TrafficLight.py
```

You can see the following phenomenon of traffic lights. The red LED lights up for 9 seconds, green LED for 5s, and yellow LED for 3s.

Code

Note: You can **Modify/Reset/Copy/Run/Stop** the code below. But before that, you need to go to source code path like `electronic-kit/for-raspberry-pi/python`. After modifying the code, you can run it directly to see the effect.

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

#define the pins connect to 74HC595
SDI   = 17      #serial data input(DS)
RCLK  = 18      #memory clock input(STCP)
SRCLK = 27       #shift register clock input(SHCP)
number = (0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x80)
```

(continues on next page)

(continued from previous page)

```
ledPin =(22,23,24)

greenLight = 5
yellowLight = 3
redLight = 9
lightColor=("Red","Green","Yellow")

colorState=0
counter = 9
t = 0

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT)
    GPIO.setup(RCLK, GPIO.OUT)
    GPIO.setup(SRCLK, GPIO.OUT)
    for pin in ledPin:
        GPIO.setup(pin,GPIO.OUT)


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

def display(num):
    hc595_shift(0xff)
    hc595_shift(number[num%10])
    time.sleep(0.003)


def timer():        #timer function
    global counter
    global colorState
    global t
    t = threading.Timer(1.0,timer)
    t.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
        colorState=(colorState+1)%3
    print ("counter : %d    color: %s "%(counter,lightColor[colorState]))

def lightup(state):
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.LOW)
    GPIO.output(ledPin[state], GPIO.HIGH)
```

(continues on next page)

(continued from previous page)

```
def loop():
    global t
    global counter
    global colorState
    t = threading.Timer(1.0,timer)
    t.start()
    while True:
        display(counter)
        lightup(colorState)

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

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

Code Explanation

```
greenLight = 5
yellowLight = 3
redLight = 9
```

Define the duration of lighting of three LEDs. Since what we use is a 7-Segment Display here, we shorten the length of seconds of lighting of traffic LEDs, setting the green LED to light up for 5 seconds, the yellow LED to 3 seconds, and the red LED to 9 seconds.

```
colorState=0
counter = 9
```

The variable **colorState** corresponds to the state of the traffic LEDs, and we only need to do a simple calculation of **colorState** to indicate the order change of the state of the traffic LEDs.

counter is used to count down the time to each traffic LED status and will be output on a 7-Segment Display.

```
def timer():        #timer function
    global counter
    global colorState
    global t
    t = threading.Timer(1.0,timer)
    t.start()
    counter-=1
    if (counter is 0):
        if(colorState is 0):
            counter= greenLight
        if(colorState is 1):
            counter=yellowLight
        if (colorState is 2):
            counter=redLight
```

(continues on next page)

(continued from previous page)

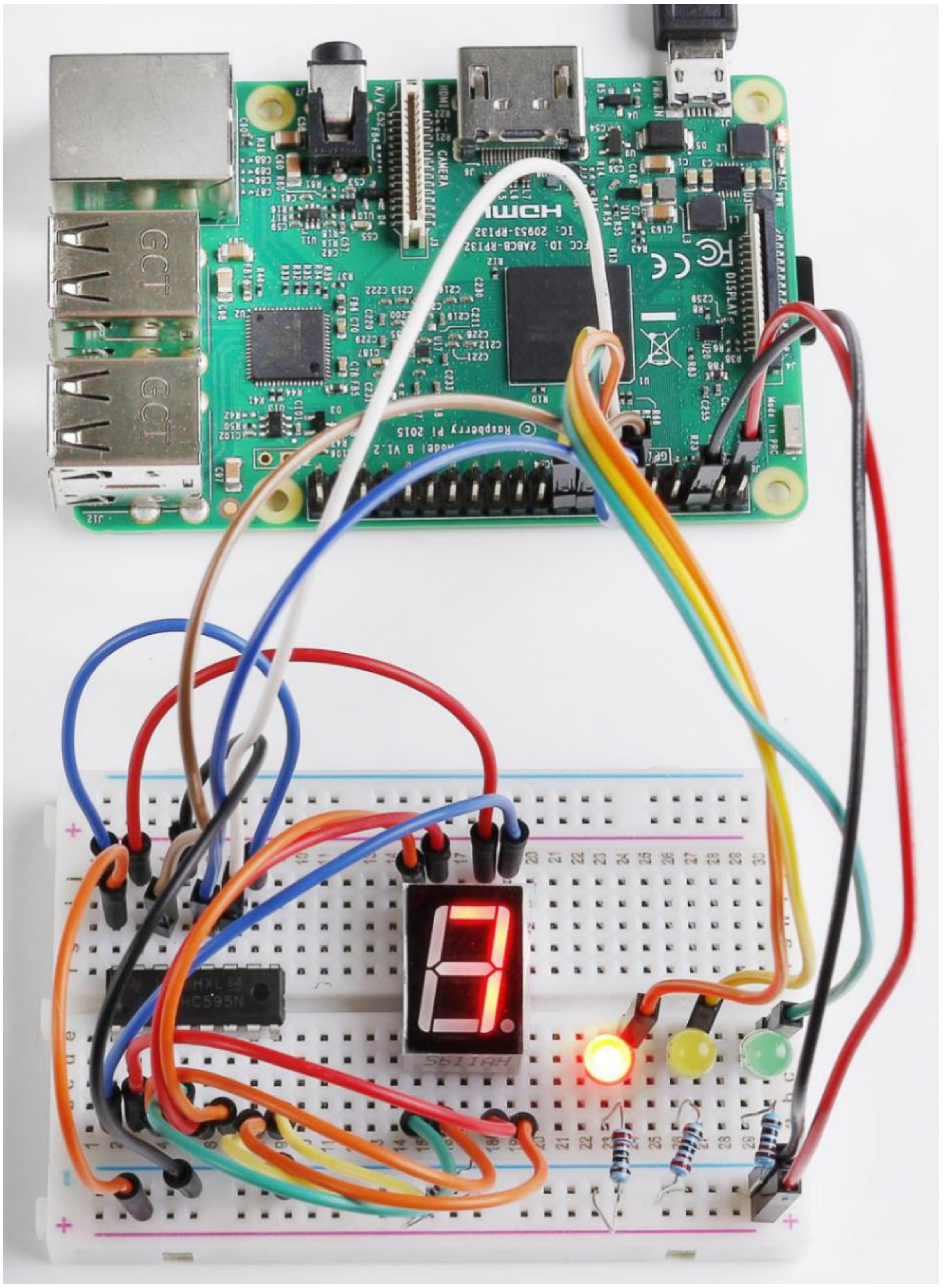
```
        colorState=(colorState+1)%3
    print ("counter : %d    color: %s "%(counter,lightColor[colorState]))
```

On this timer, counter decreases gradually with every second passing, and when it goes to 0, the state of the traffic LED changes accordingly.

```
def lightup(state):
    for i in range(0,3):
        GPIO.output(ledPin[i], GPIO.LOW)
    GPIO.output(ledPin[state], GPIO.HIGH)
```

The function is to turn off all the LEDs first, and then light up the corresponding LED according to the value of the traffic LED state.

Phenomenon Picture



1.2.6 Appendix

Remote Desktop

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

VNC

You can use the function of remote desktop through VNC.

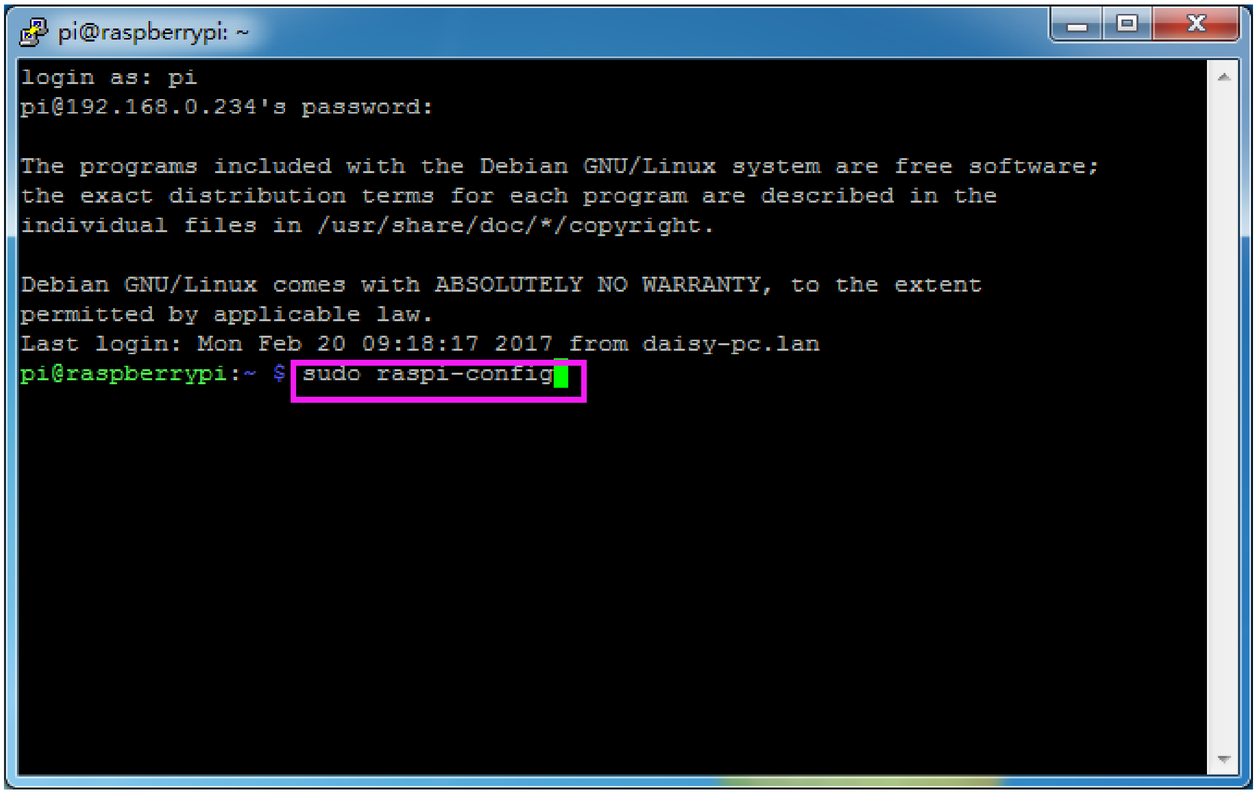
Enable VNC Service

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

Step 1

Input the following command:

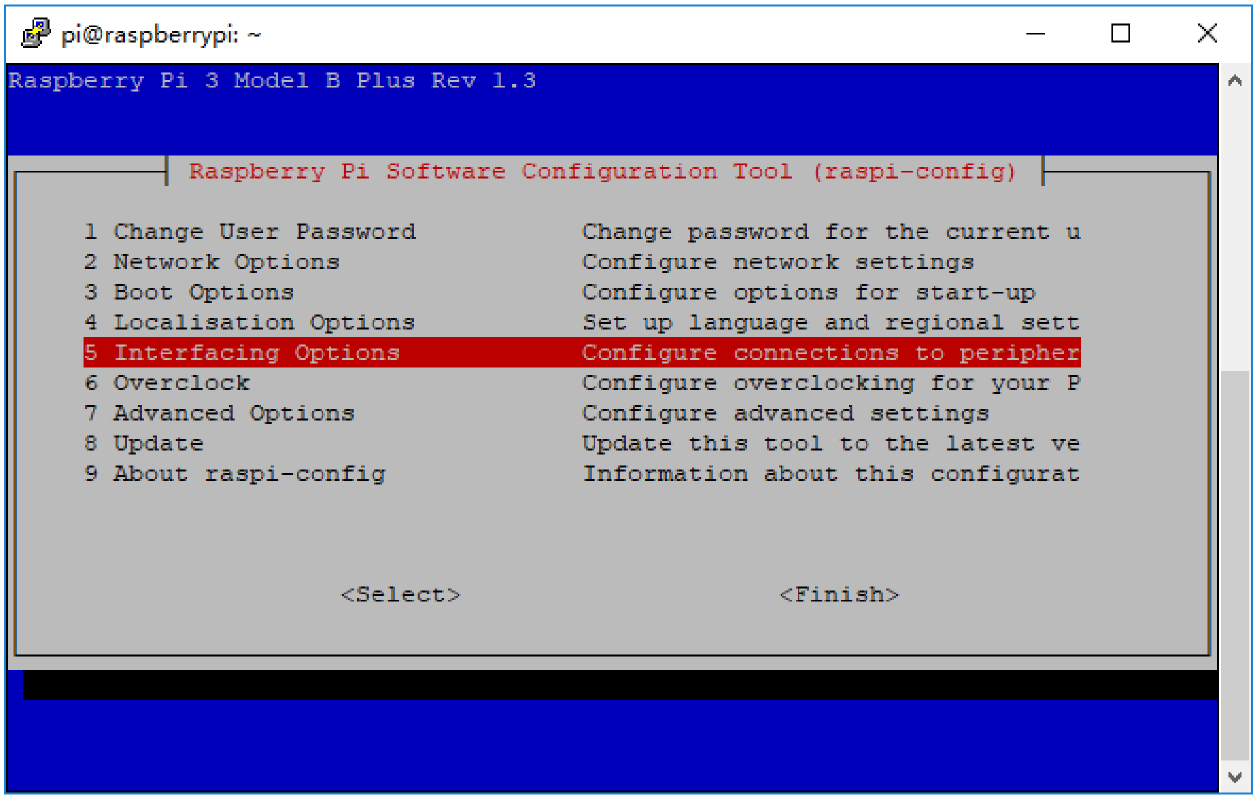
sudo raspi-config



```
login as: pi
pi@192.168.0.234's password:

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 20 09:18:17 2017 from daisy-pc.lan
pi@raspberrypi:~ $ sudo raspi-config
```

Step 2

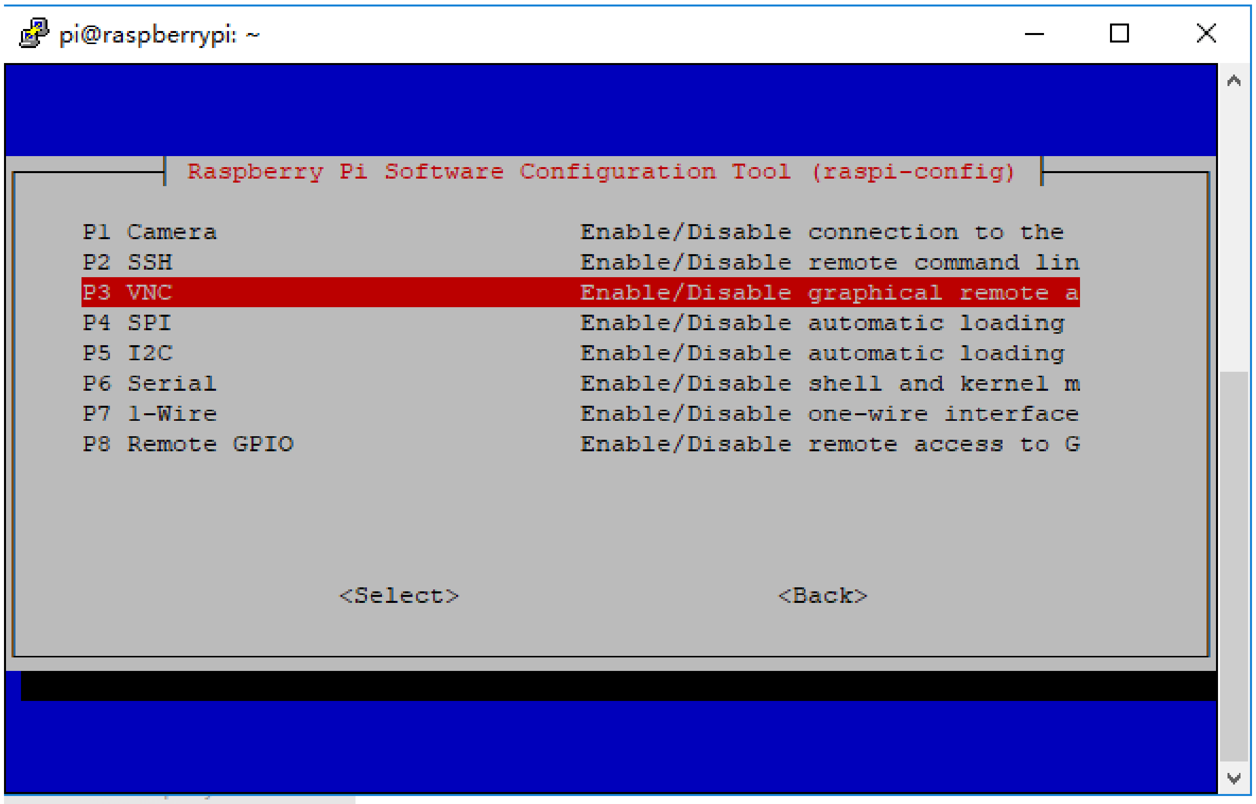
On the config interface, select **"Interfacing Options"** by the up, down, left and right keys on the keyboard.



```
Raspberry Pi 3 Model B Plus Rev 1.3

          Raspberry Pi Software Configuration Tool (raspi-config)

   1 Change User Password           Change password for the current u
   2 Network Options                Configure network settings
   3 Boot Options                   Configure options for start-up
   4 Localisation Options           Set up language and regional sett
   5 Interfacing Options            Configure connections to peripher
   6 Overclock                      Configure overclocking for your P
   7 Advanced Options               Configure advanced settings
   8 Update                         Update this tool to the latest ve
   9 About raspi-config             Information about this configurat

                  <Select>                     <Finish>
```

Step 3

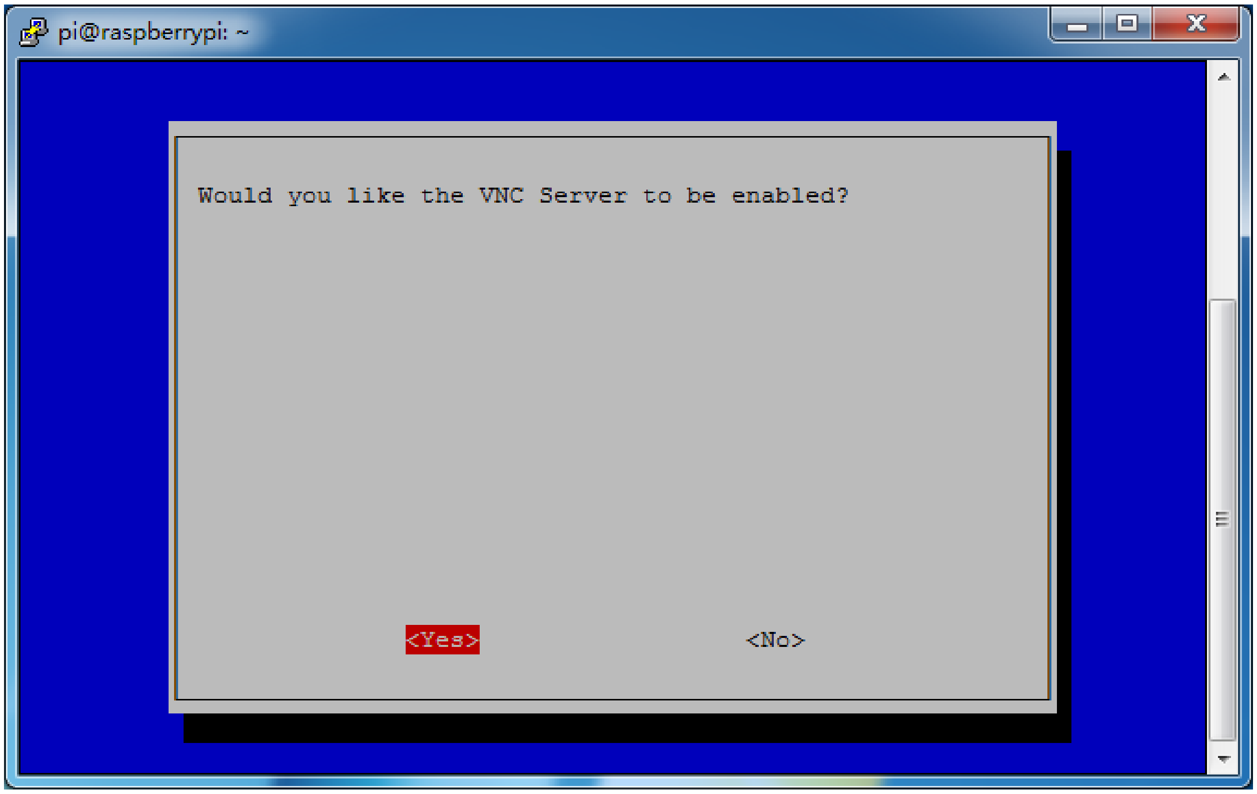
Select **VNC**.



```
          Raspberry Pi Software Configuration Tool (raspi-config)

   P1 Camera                        Enable/Disable connection to the
   P2 SSH                           Enable/Disable remote command lin
   P3 VNC                           Enable/Disable graphical remote a
   P4 SPI                           Enable/Disable automatic loading
   P5 I2C                           Enable/Disable automatic loading
   P6 Serial                        Enable/Disable shell and kernel m
   P7 1-Wire                        Enable/Disable one-wire interface
   P8 Remote GPIO                   Enable/Disable remote access to G

                  <Select>                     <Back>
```

Step 4

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



```
Would you like the VNC Server to be enabled?

            <Yes>                    <No>
```

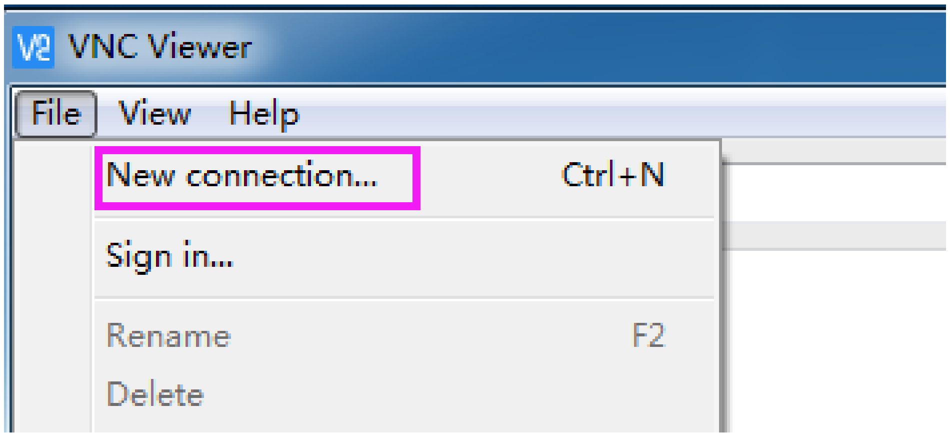
Login to VNC

Step 1

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

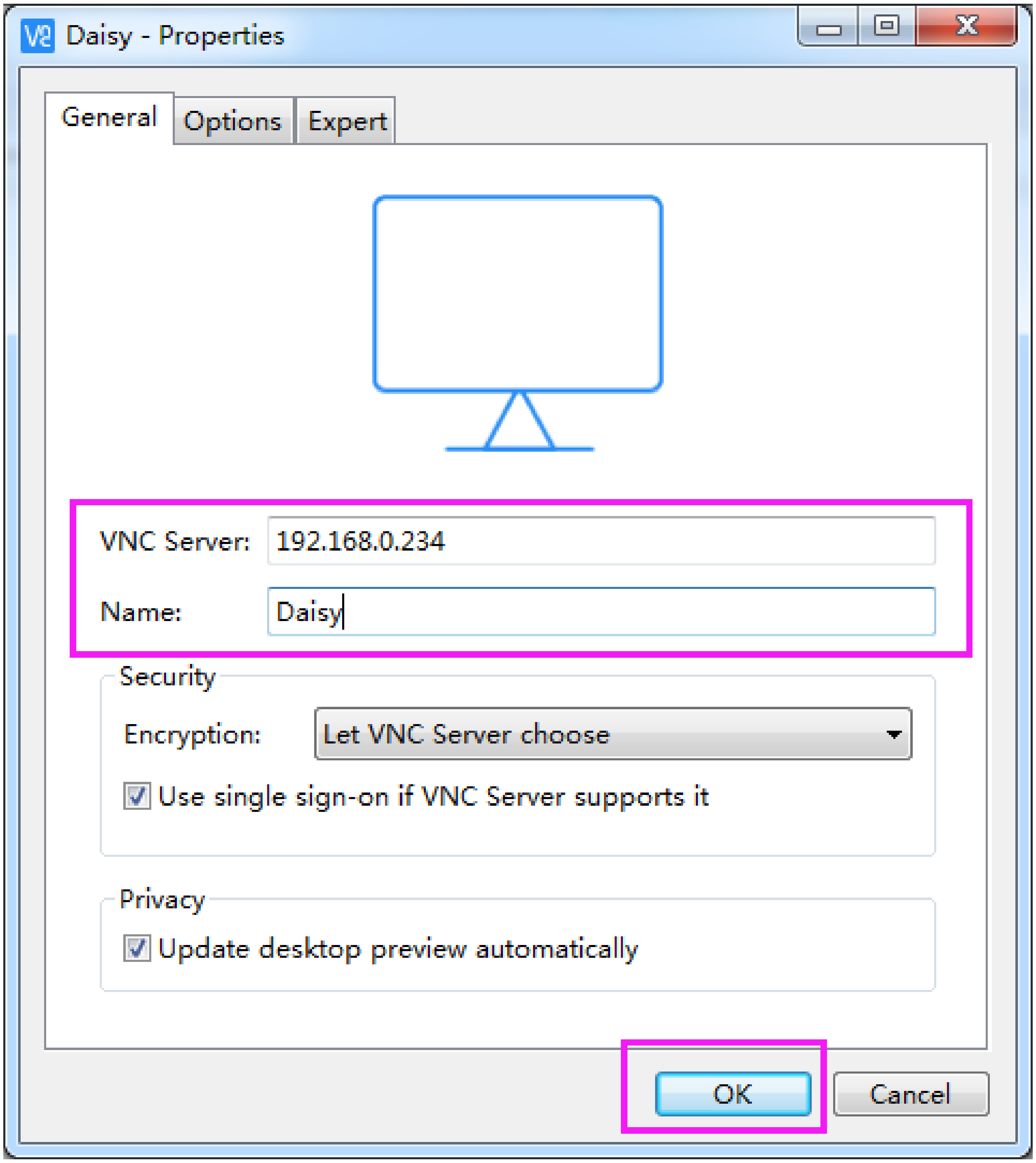
Step 2

Then select "**New connection**".



Step 3

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



VNC Server: 192.168.0.234

Name: Daisy

Security

Encryption: Let VNC Server choose

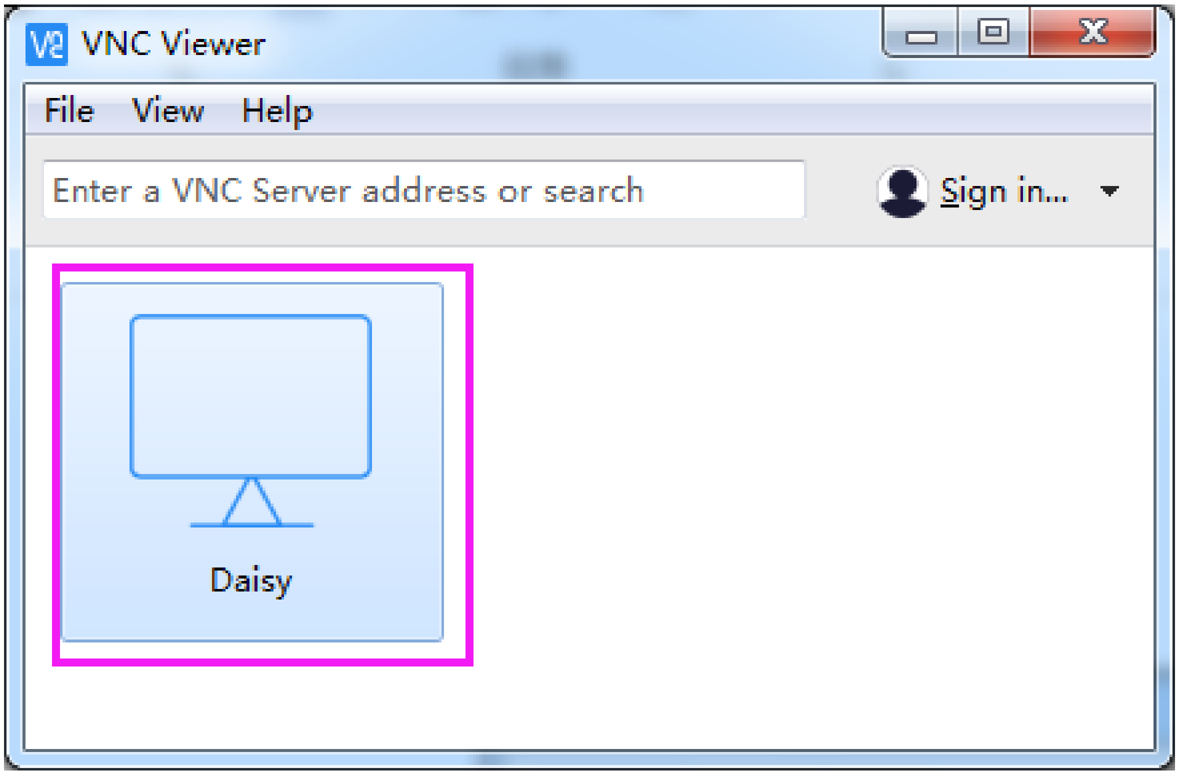
☑ Use single sign-on if VNC Server supports it

Privacy

☑ Update desktop preview automatically

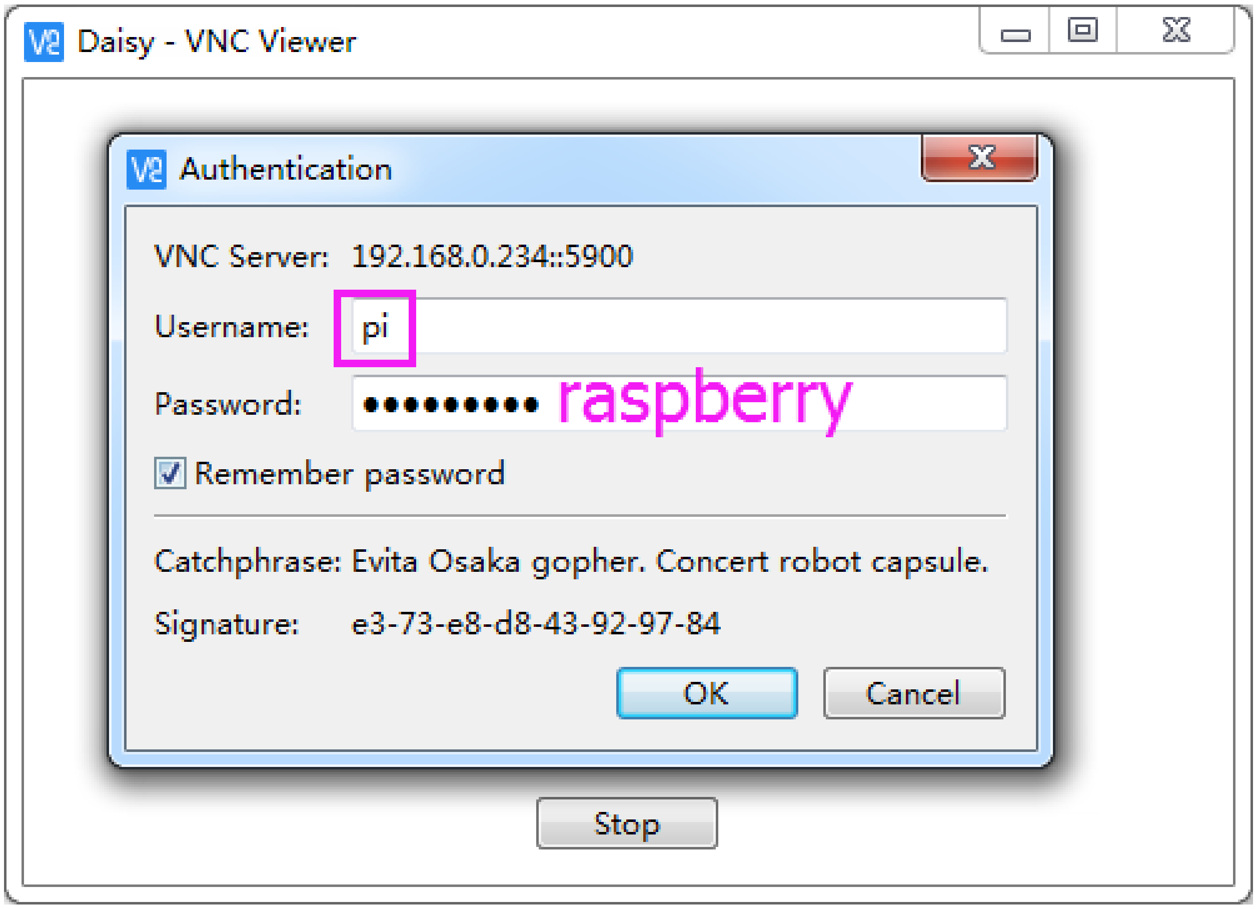
Step 4

Double click the **connection** just created:



Step 5

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


Daisy - VNC Viewer

Authentication

VNC Server: 192.168.0.234::5900

Username: pi

Password: •••••••••• raspberry

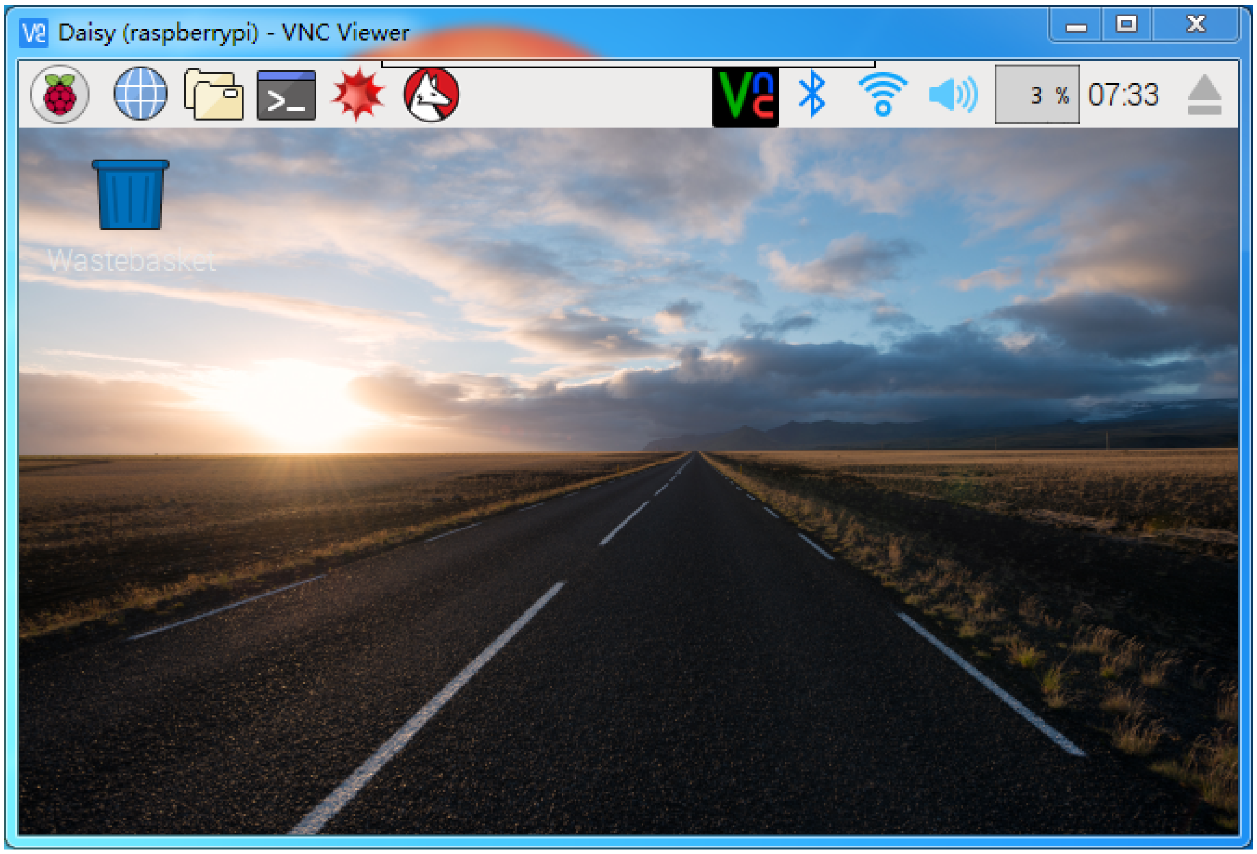
☑ Remember password

Catchphrase: Evita Osaka gopher. Concert robot capsule.

Signature: e3-73-e8-d8-43-92-97-84

Step 6

Now you can see the desktop of the Raspberry Pi:



XRDP

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

Install XRDP

Step 1

Login to Raspberry Pi by using SSH.

Step 2

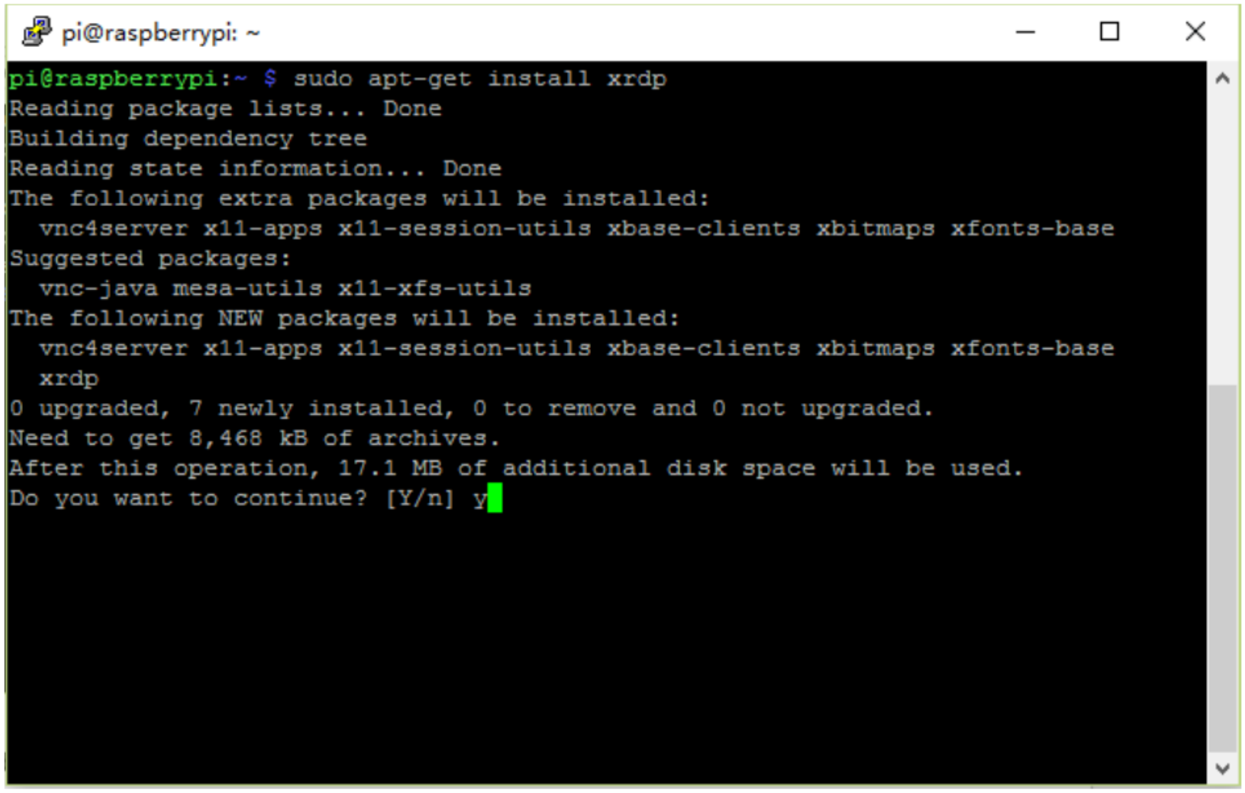
Input the following instructions to install XRDP.

```
sudo apt-get update

sudo apt-get install xrdp
```

Step 3

Later, the installation starts.

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



pi@raspberrypi: ~

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

Step 4

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

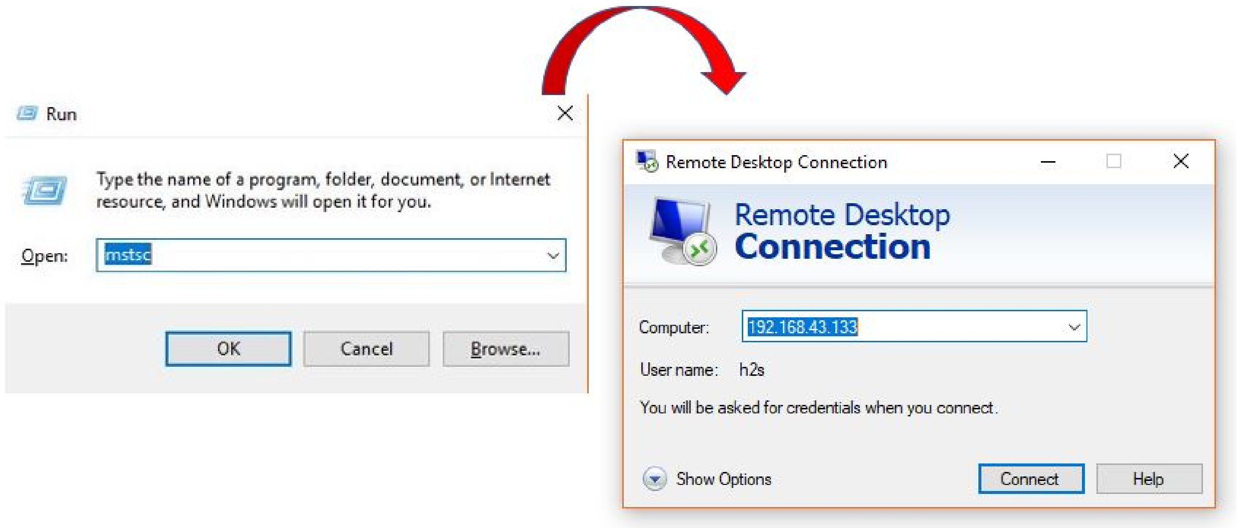
Login to XRDP

Step 1

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

Step 2

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


Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: mstsc

Remote Desktop Connection

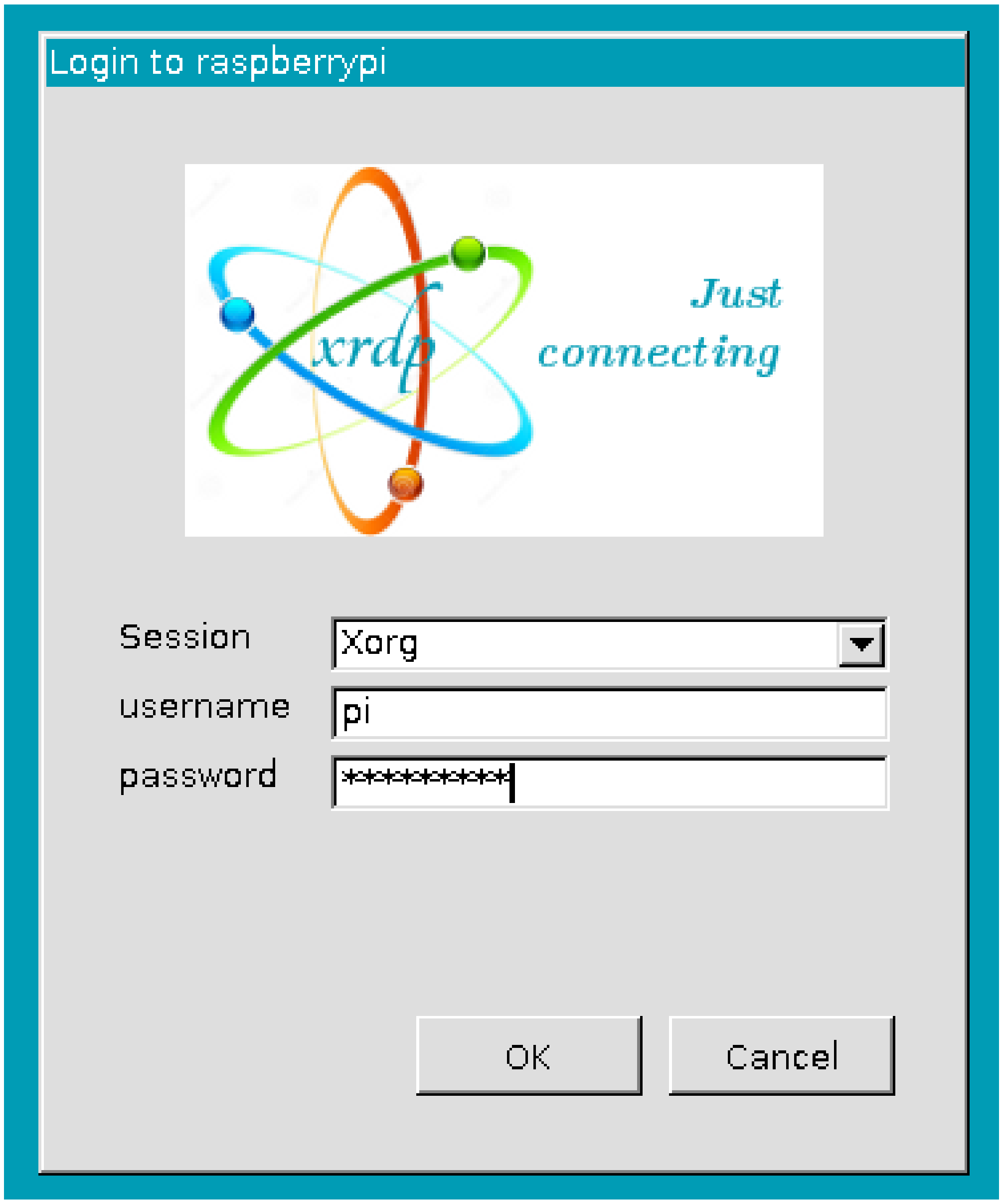
Computer: 192.168.43.133

User name: h2s

You will be asked for credentials when you connect.

Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click "OK". At the first time you log in, your username is "pi" and the password is "raspberry".



Session Xorg
username pi
password **********

Step 4

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

1.3 For Arduino User

1.3.1 Install and Introduce Arduino IDE

Description

Arduino is an open source platform with simple software and hardware. You can pick it up in short time even if you are a beginner. It provides an integrated development environment (IDE) for code compiling, compatible with multiple control boards. So you can just download the Arduino IDE, upload the sketches (i.e. the code files) to the board, and then you can see relative experimental phenomena. For more information, refer to http://www.arduino.cc.

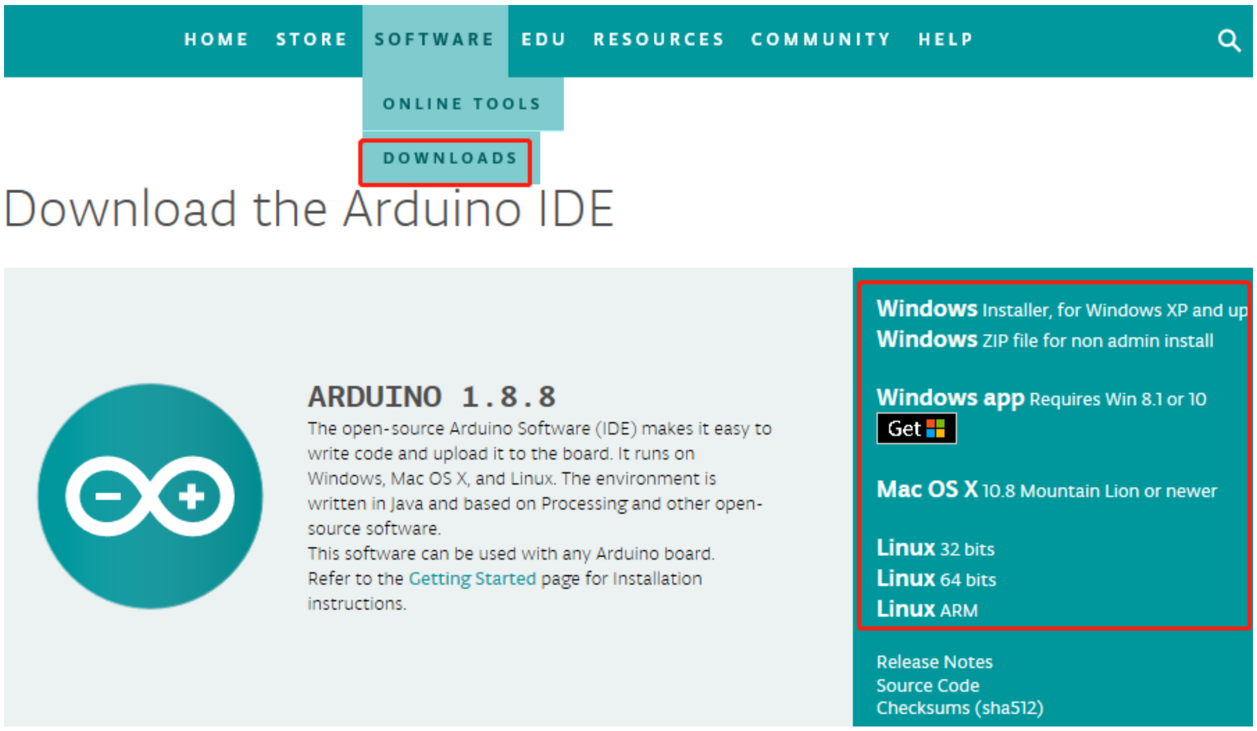
Install Arduino IDE

Here are the installation steps on the windows system.

For other systems, please refer to: Install Arduinio IDE in different system and FAQ.pdf

The code in this kit is written based on Arduino, so you need to install the IDE first. Skip it if you have done this.

Now go to arduino.cc and click SOFTWARE -> DOWNLOADs. on the page, check the software list on the right side.



Download the Arduino IDE

ARDUINO 1.8.8

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board.

Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM

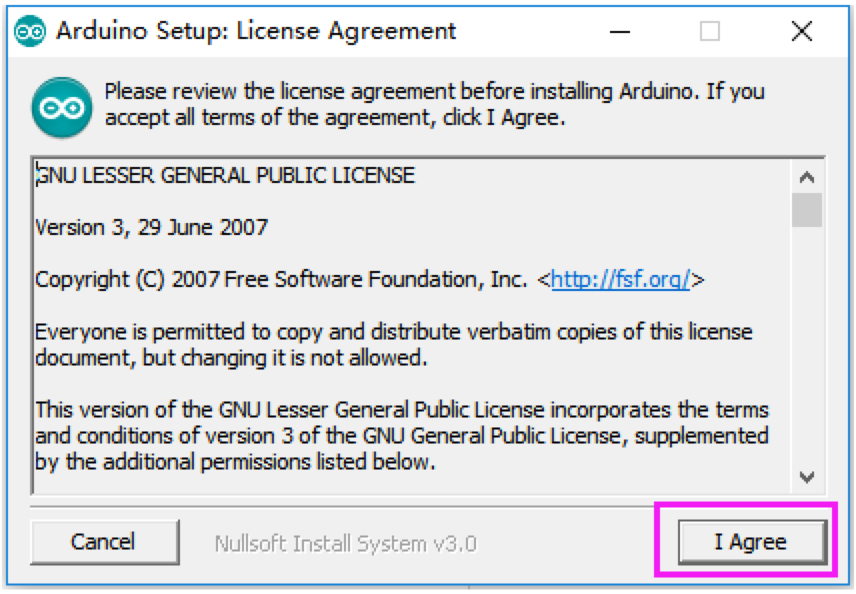
Release Notes
Source Code
Checksums (sha512)

Find the one that suits your operation system and click to download. There are two versions of Arduino for Windows: Installer or ZIP file. You're recommended to download the former.

For Installer File

Step 1: Find the .exe file just downloaded.

Step 2: Double click the file and a window will pop up as below. Click **I Agree**.



Arduino Setup: License Agreement

Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.

GNU LESSER GENERAL PUBLIC LICENSE

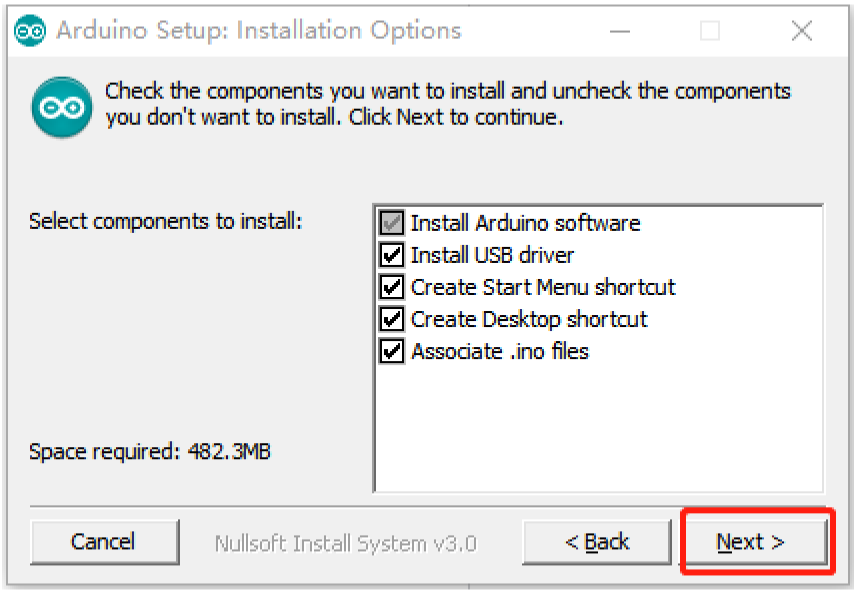
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

Step 3: Click **Next.**



Arduino Setup: Installation Options

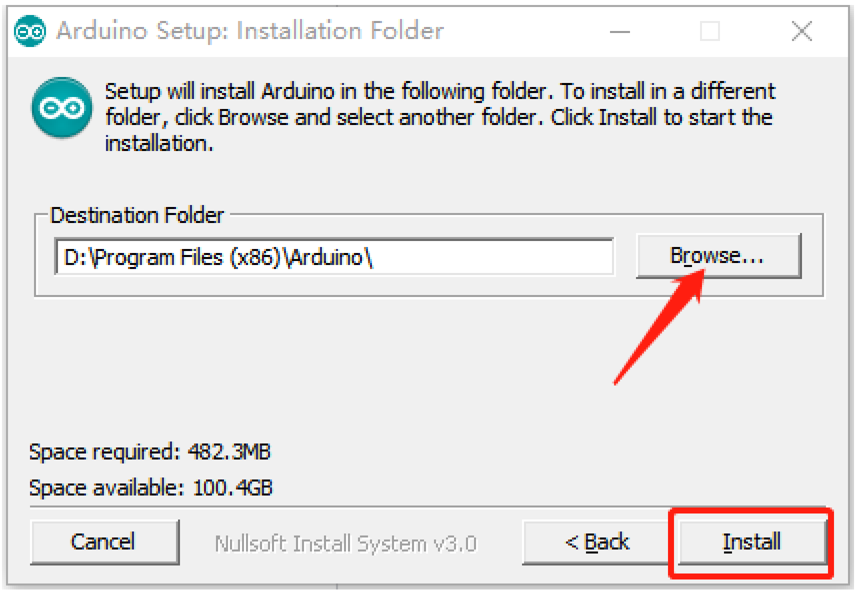
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 482.3MB

Step 4: Select the path to install. By default, it's set in the C disk. You can click **Browse** and choose other paths. Click **OK**. Then click Install.


Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder

D:\Program Files (x86)\Arduino\

Space required: 482.3MB

Space available: 100.4GB

Step 5: Meanwhile, it will prompts install the needed drivers, please select the 'Always trust software from "Arduino LLC" '. After the installation is done, click **Close**

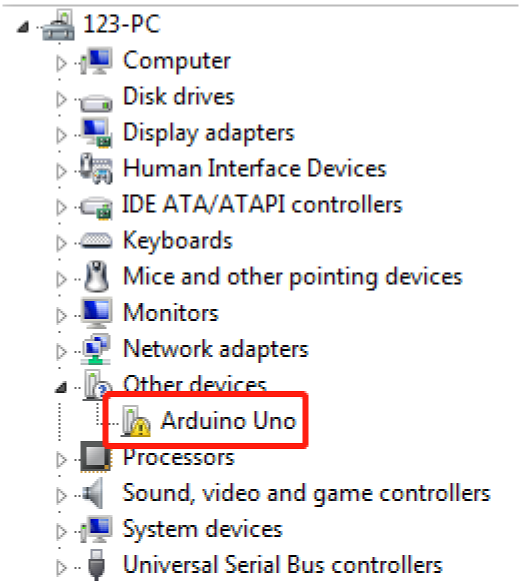
Note: The new IDE may prompt errors when you're compiling code under Windows XP. So if your computer is running on XP, you're suggested to install Arduino 1.0.5 or 1.0.6. Also you can upgrade your computer.

For ZIP File

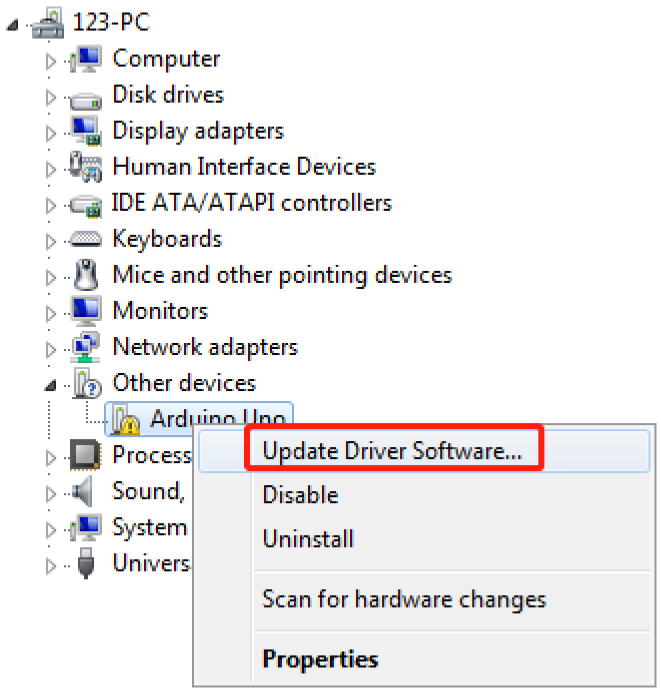
If you download the zip file before, when you connect the MCU to the computer, it may not be recognized. Then you need to install the driver manually. Take the following steps.

Step1: Plug in the board to the computer with a 5V USB cable. After a while, a prompt message of failed installation will appear.

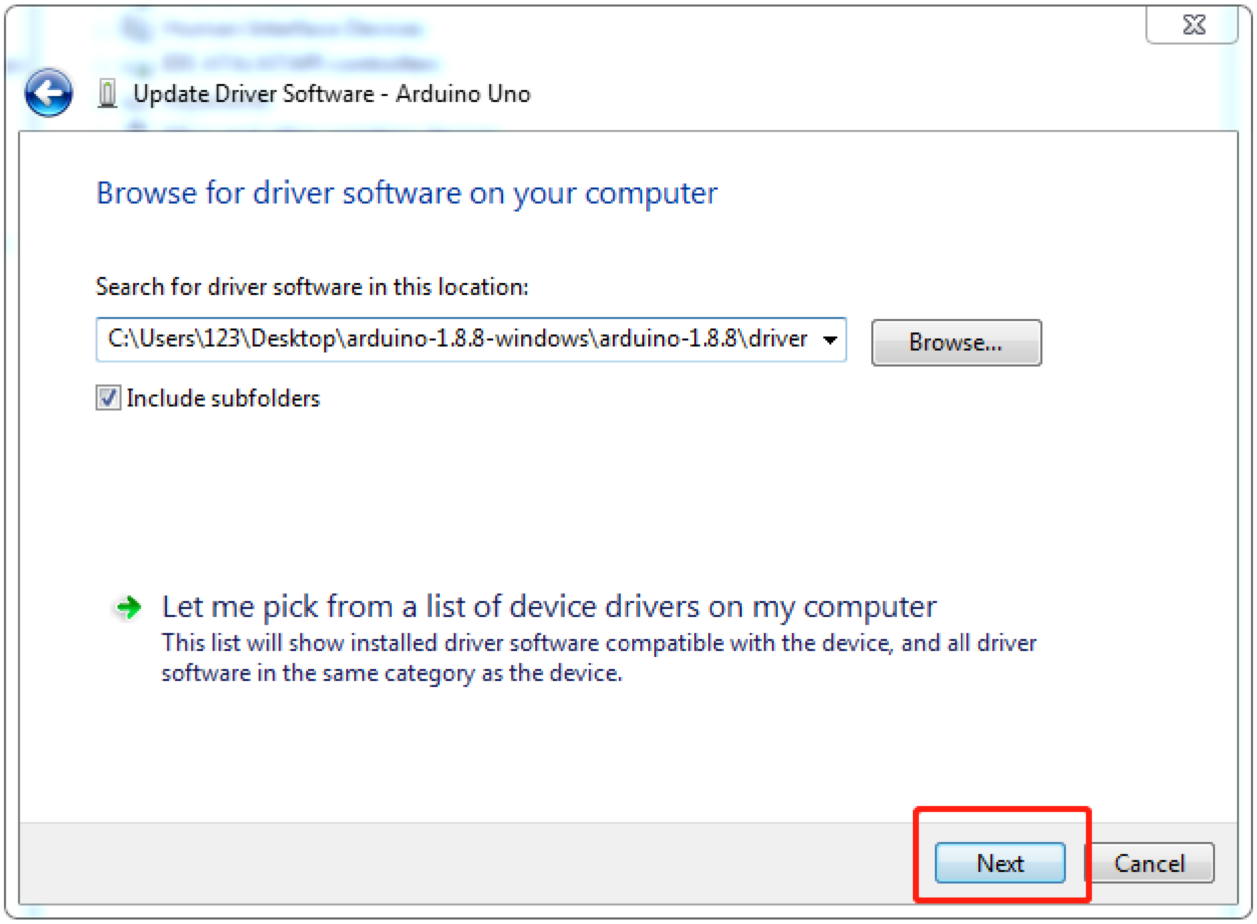
Step2: Go to the **Device Manager**. You will find under other devices, Arduino Uno with an exclamation mark appear, which means the computer did not recognize the board.



Step3: Right click on **Arduino Uno** and select **Update Driver Software**.



Step4: Choose the second option, **Browse my computer for Driver software**.



Browse for driver software on your computer

Search for driver software in this location:

C:\Users\123\Desktop\arduino-1.8.8-windows\arduino-1.8.8\driver

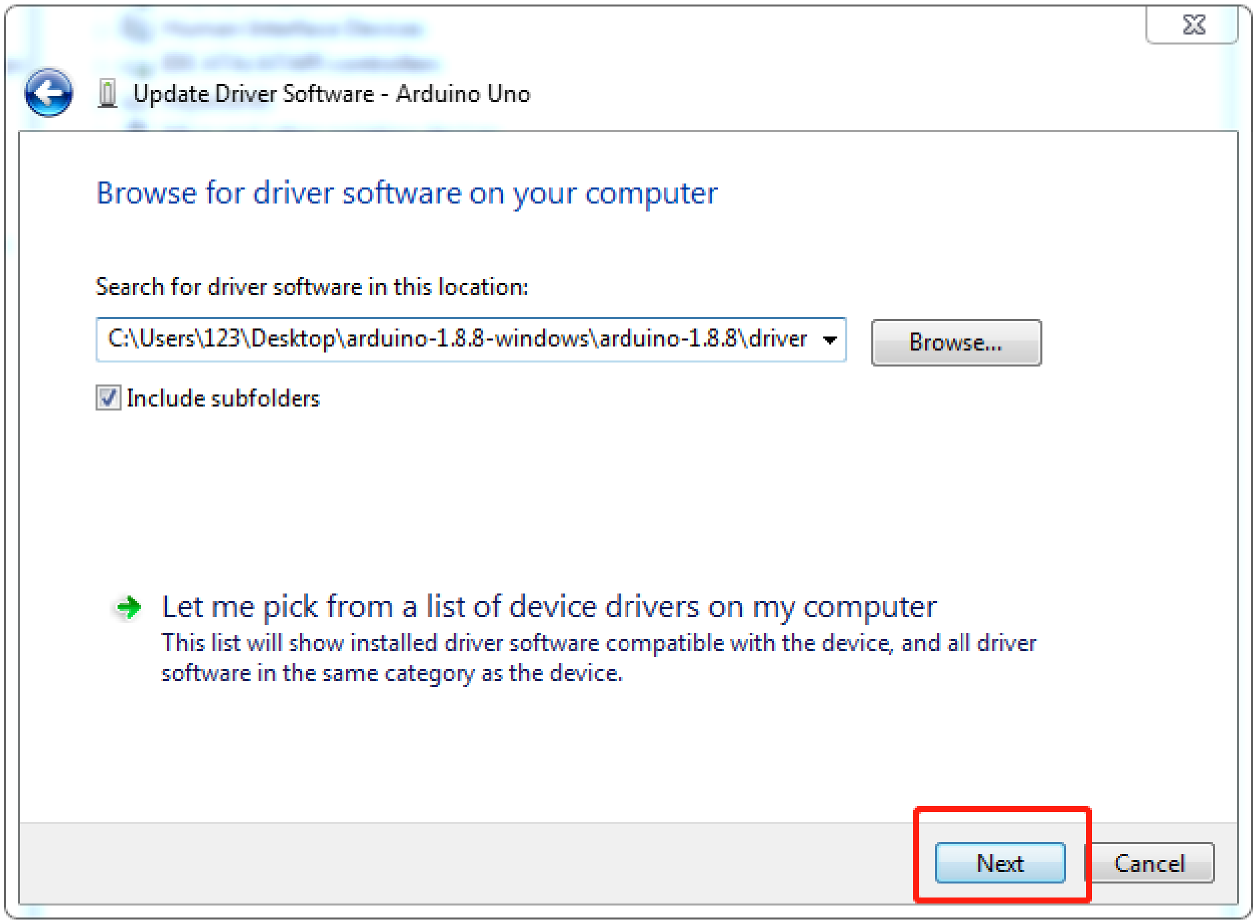
Browse...

[✓] Include subfolders

Let me pick from a list of device drivers on my computer

This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Step5: A window pops up then. Click **Browse**. Then go to the folder where you just extracted the file. Go to the *drivers* folder and click **OK** -> **Next**.



Update Driver Software - Arduino Uno

Browse for driver software on your computer

Search for driver software in this location:

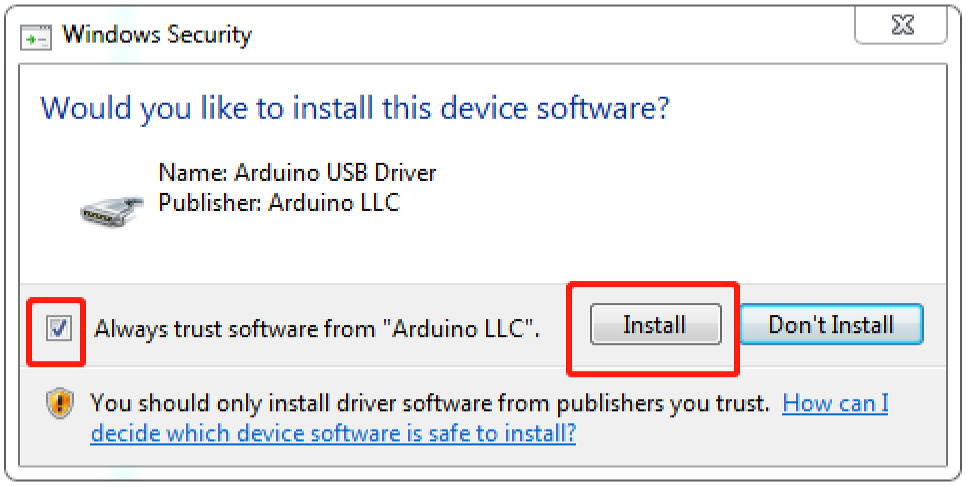
C:\Users\123\Desktop\arduino-1.8.8-windows\arduino-1.8.8\driver

[✓] Include subfolders

Let me pick from a list of device drivers on my computer

This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Step6: Select 'Always trust software from "Arduino LLC" ' then click Install.



Windows Security

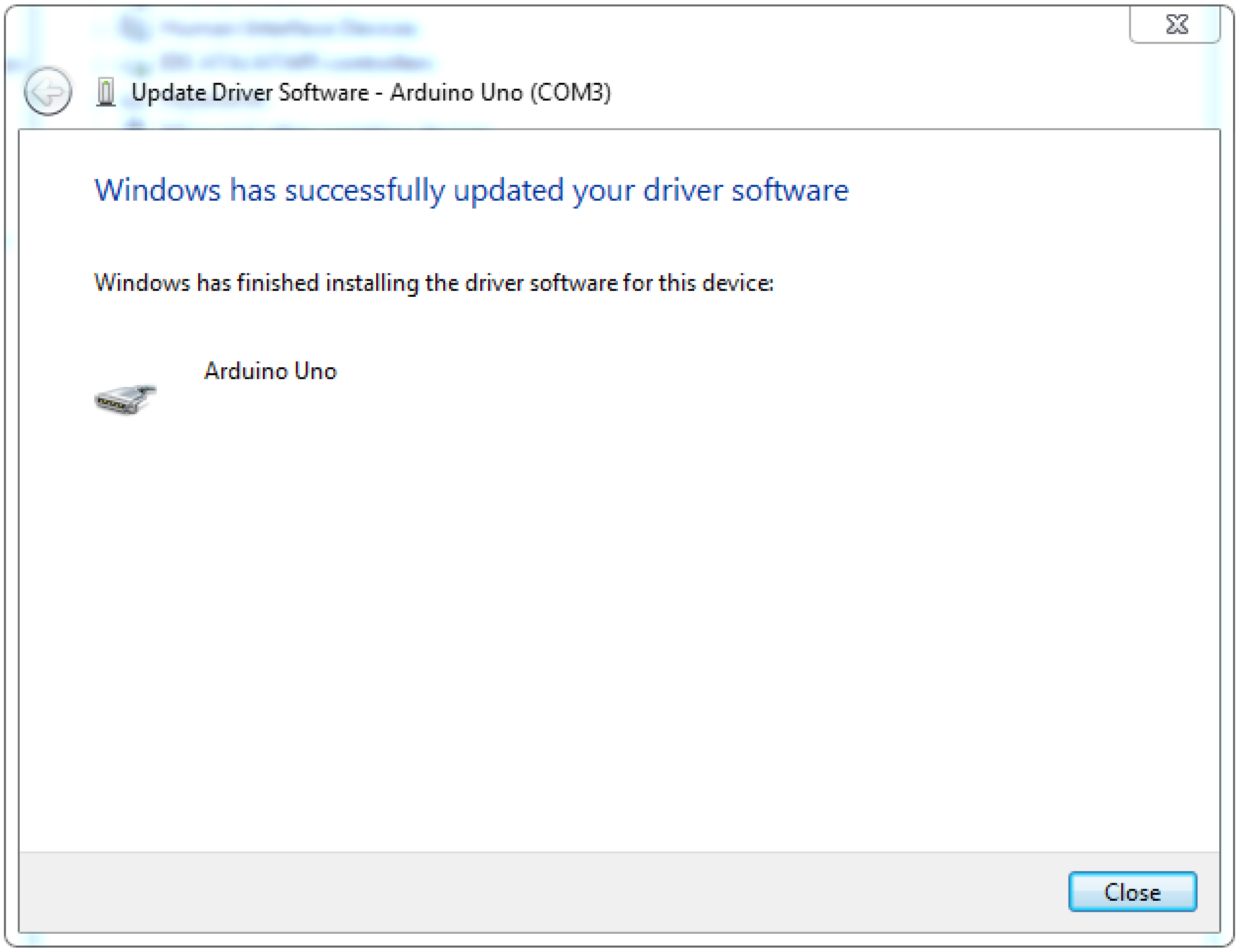
Would you like to install this device software?

Name: Arduino USB Driver
Publisher: Arduino LLC

[✓] Always trust software from "Arduino LLC".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

It may need a sec. Then the system prompts you the driver has been installed successfully. So the computer can recognize the board now. Click **Close**.



Update Driver Software - Arduino Uno (COM3)

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

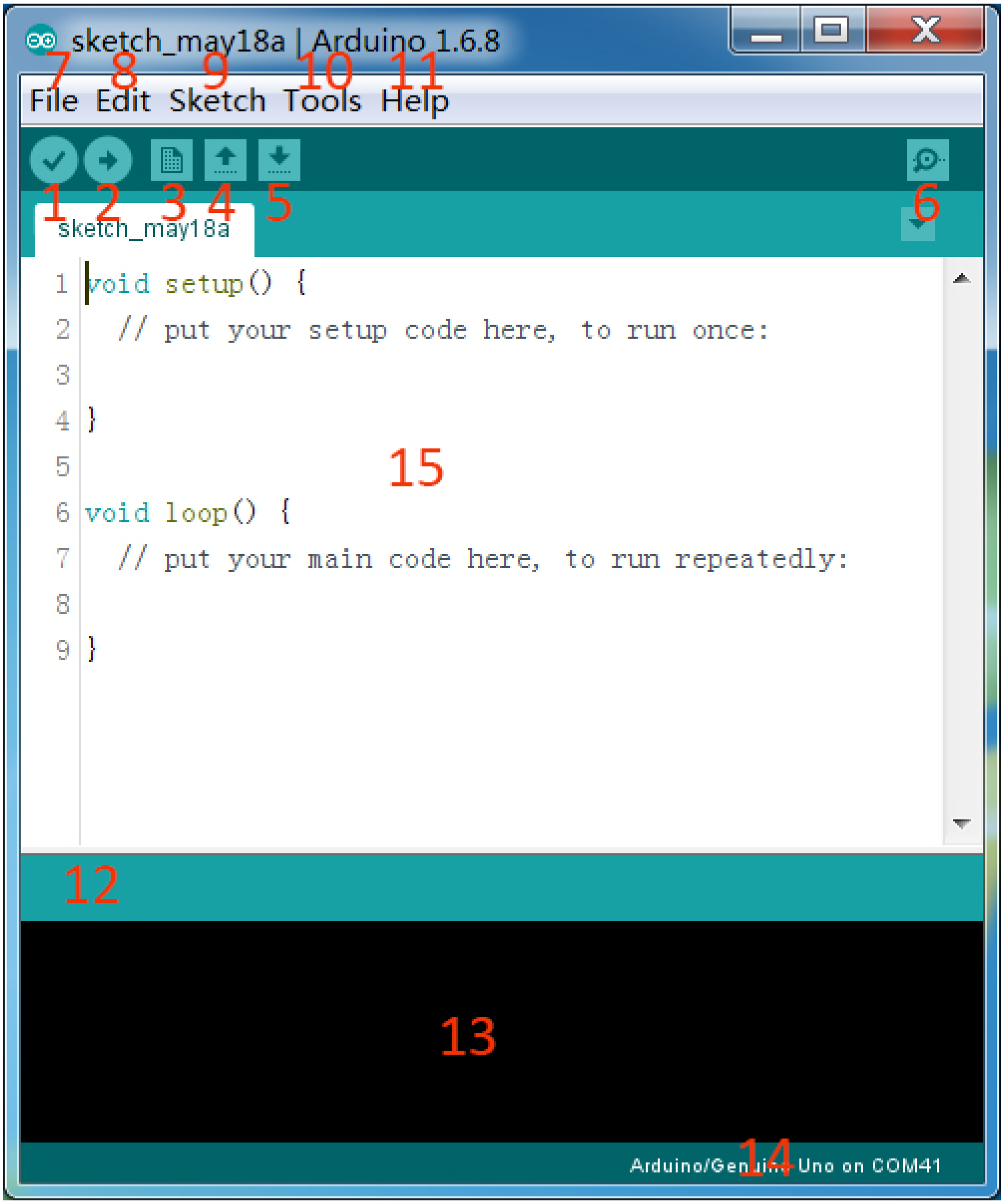
Arduino Uno

Open the Arduino Software (IDE)

Double-click the Arduino icon (arduino.exe) created by the installation process. Then the Arduino IDE will appear. Let's check details of the software.

1. **Verify**: Compile your code. Any syntax problem will be prompted with errors.
2. **Upload**: Upload the code to your board. When you click the button, the RX and TX LEDs on the board will flicker fast and won't stop until the upload is done.
3. **New**: Create a new code editing window.
4. **Open**: Open an .ino sketch.
5. **Save**: Save the sketch.

6. **Serial Monitor**: Click the button and a window will appear. It receives the data sent from your control board. It is very useful for debugging.
7. **File**: Click the menu and a drop-down list will appear, including file creating, opening, saving, closing, some parameter configuring, etc.
8. **Edit**: Click the menu. On the drop-down list, there are some editing operations like Cut, Copy, Paste, Find, and so on, with their corresponding shortcuts.
9. **Sketch**: Includes operations like Verify, Upload, Add files, etc. More important function is Include Library – where you can add libraries.
10. **Tool**: Includes some tools – the most frequently used Board (the board you use) and Port (the port your board is at). Every time you want to upload the code, you need to select or check them.
11. **Help**: If you're a beginner, you may check the options under the menu and get the help you need, including operations in IDE, introduction information, troubleshooting, code explanation, etc.
12. In this message area, no matter when you compile or upload, the summary message will always appear.
13. Detailed messages during compile and upload. For example, the file used lies in which path, the details of error prompts.
14. **Board and Port**: Here you can preview the board and port selected for code upload. You can select them again by **Tools** -> **Board / Port** if any is incorrect.
15. The editing area of the IDE. You can write code here.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

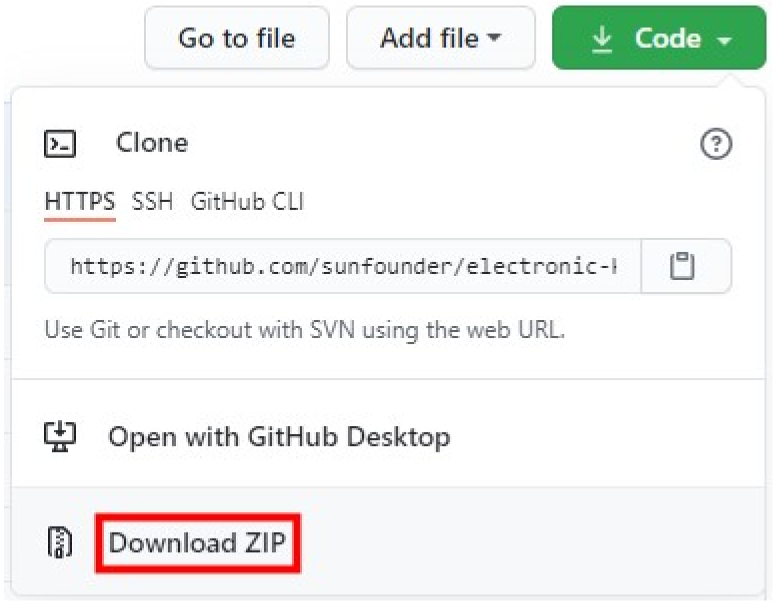
1.3.2 Download the Code

We have uploaded the Arduino codes of this kit to the Arduino Cloud, you can see and download it in every lesson.

If you want to download all the codes to your local folder at once, please visit the link below:

https://github.com/sunfounder/electronic-kit

This is a github repository, which contains information about Arduino and Raspberry Pi. You can download it via the **Download ZIP** button.


Go to file
Add file
Code
Clone
HTTPS
SSH
GitHub CLI
https://github.com/sunfounder/electronic-k
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

The Arduino code path is as follows: `electronic-kit/for-Arduino/code/`.

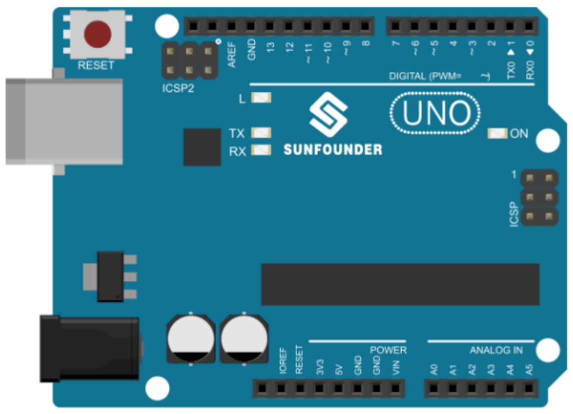
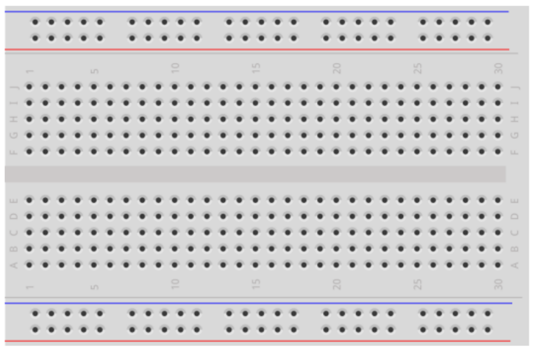
1.3.3 Lessons

Lesson 1 Blinking LED

Introduction

You should've learnt how to install Arduino IDE and add libraries before. Now you can start with a simple experiment to learn the basic operation and code in the IDE.

Component

1 * Uno Board 	1 * Breadboard 

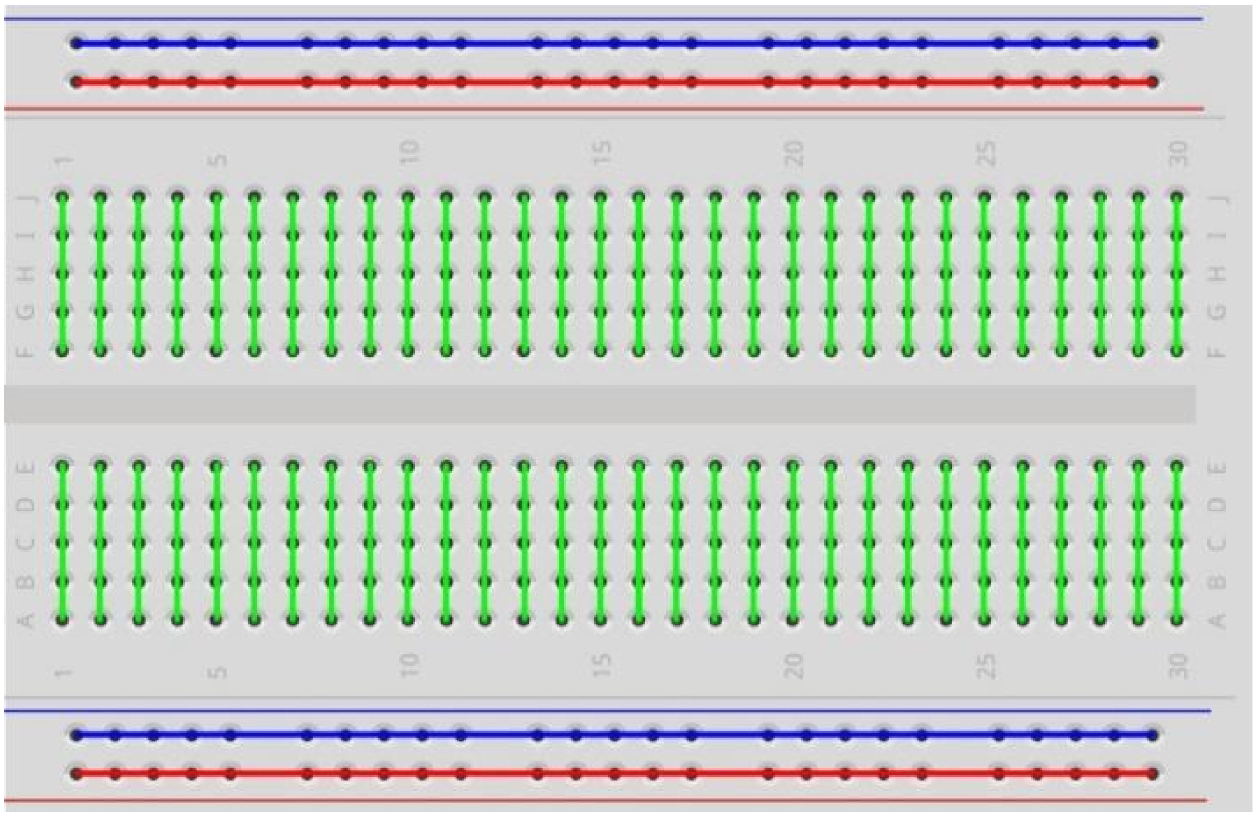
1 * Resistor (220Ω)	1 * LED
1 * USB Cable	Several Jumper Wires

Component Introduction

Breadboard

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

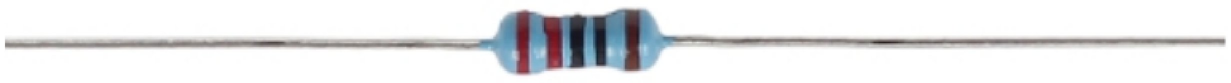
This is the internal structure of a full+ breadboard. Although there are holes on the breadboard, internally some of them are connected with metal strips.

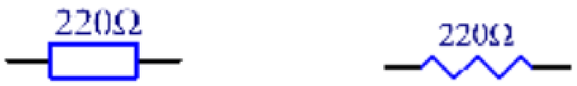


Resistor

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

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

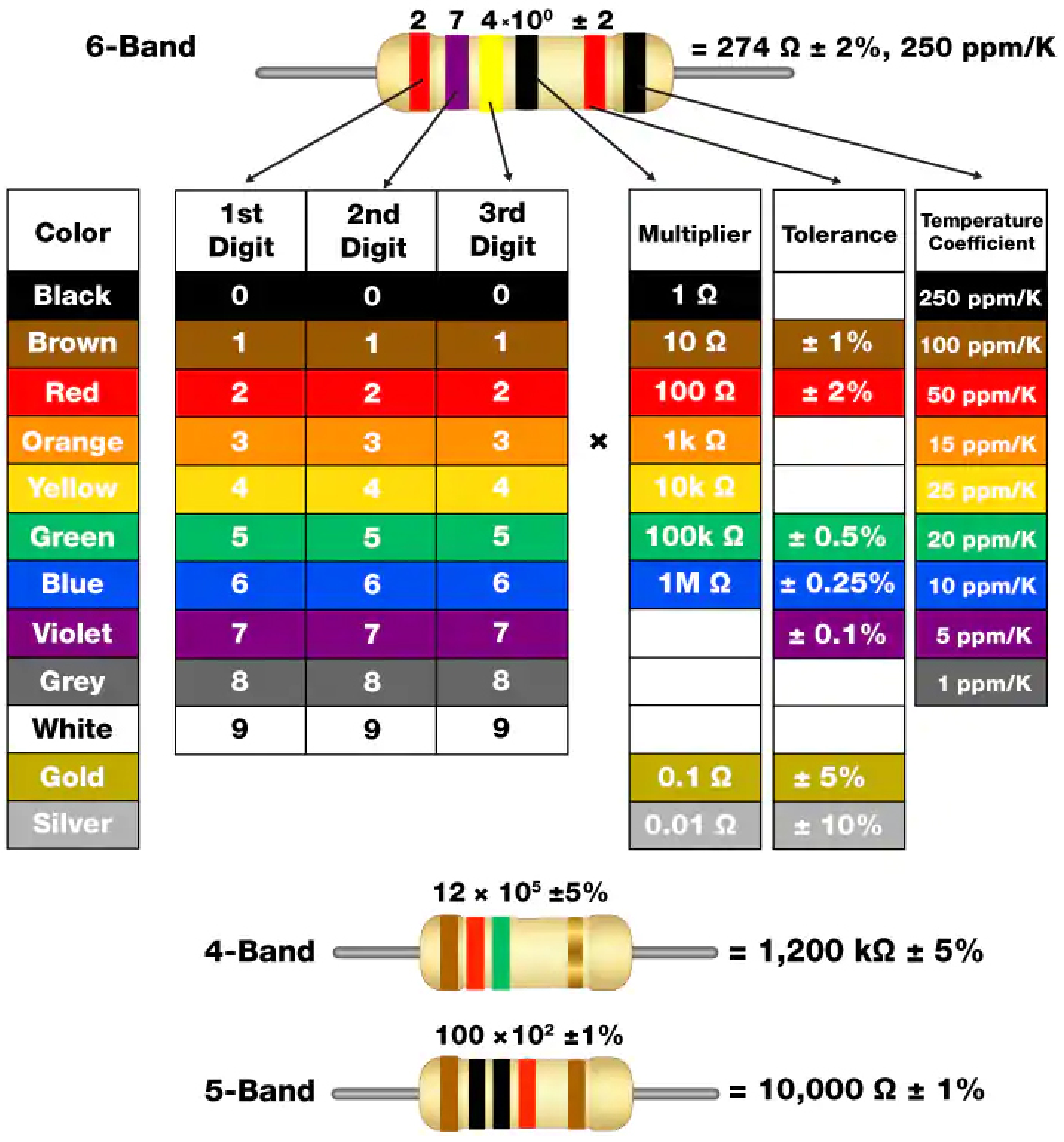




The resistance can be marked directly, in color code, and by character. The resistors offered in this kit are marked by different colors. Namely, the bands on the resistor indicate the resistance.

When using a resistor, we need to know its resistance first. Here are two methods: you can observe the bands on the resistor, or use a multimeter to measure the resistance. You are recommended to use the first method as it is more convenient and faster. If you are not sure about the value, use the multimeter.

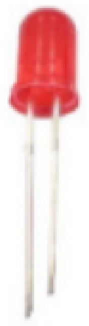
As shown in the card, each color stands for a number.

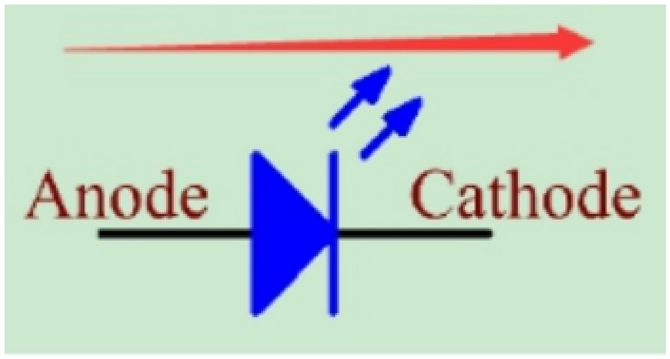


Color	1st Digit	2nd Digit	3rd Digit	Multiplier	Tolerance	Temperature Coefficient
Black	0	0	0	1 Ω		250 ppm/K
Brown	1	1	1	10 Ω	± 1%	100 ppm/K
Red	2	2	2	100 Ω	± 2%	50 ppm/K
Orange	3	3	3	1k Ω		15 ppm/K
Yellow	4	4	4	10k Ω		25 ppm/K
Green	5	5	5	100k Ω	± 0.5%	20 ppm/K
Blue	6	6	6	1M Ω	± 0.25%	10 ppm/K
Violet	7	7	7		± 0.1%	5 ppm/K
Grey	8	8	8			1 ppm/K
White	9	9	9			
Gold				0.1 Ω	± 5%	
Silver				0.01 Ω	± 10%	

LED

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





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

An LED has two pins. The longer one is the anode, and shorter one, the cathode. Pay attention not to connect them inversely. There is fixed forward voltage drop in the LED, so it cannot be connected with the circuit directly because the supply voltage can outweigh this drop and cause the LED to be burnt. The forward voltage of the red, yellow, and green LED is 1.8 V and that of the white one is 2.6 V. Most LEDs can withstand a maximum current of 20 mA, so we need to connect a current limiting resistor in series.

The formula of the resistance value is as follows:

$R = (V_{supply} - V_D)/I$

R stands for the resistance value of the current limiting resistor, Vsupply for voltage supply, VD for voltage drop and I for the working current of the LED.

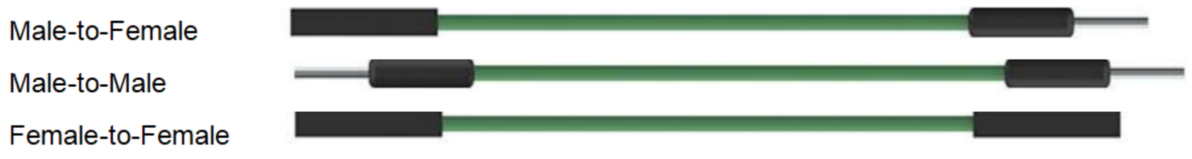
If we provide 5 voltage for the red LED, the minimum resistance of the current limiting resistor should be: (5V-1.8v)/20mA = 160. Therefore, you need a 160 or larger resistor to protect the LED. You are recommended to use the 220 resistor offered in the kit.

Jumper Wires

Wires that connect two terminals are called jumper wires. There are various kinds of jumper wires. Here we focus on those used in breadboard. Among others, they are used to transfer electrical signals from anywhere on the breadboard to the input/output pins of a microcontroller.

Jump wires are fitted by inserting their “end connectors” into the slots provided in the breadboard, beneath whose surface there are a few sets of parallel plates that connect the slots in groups of rows or columns depending on the area. The “end connectors” are inserted into the breadboard, without soldering, in the particular slots that need to be connected in the specific prototype.

There are three types of jumper wire: Female-to-Female, Male-to-Male, and Male-to-Female.

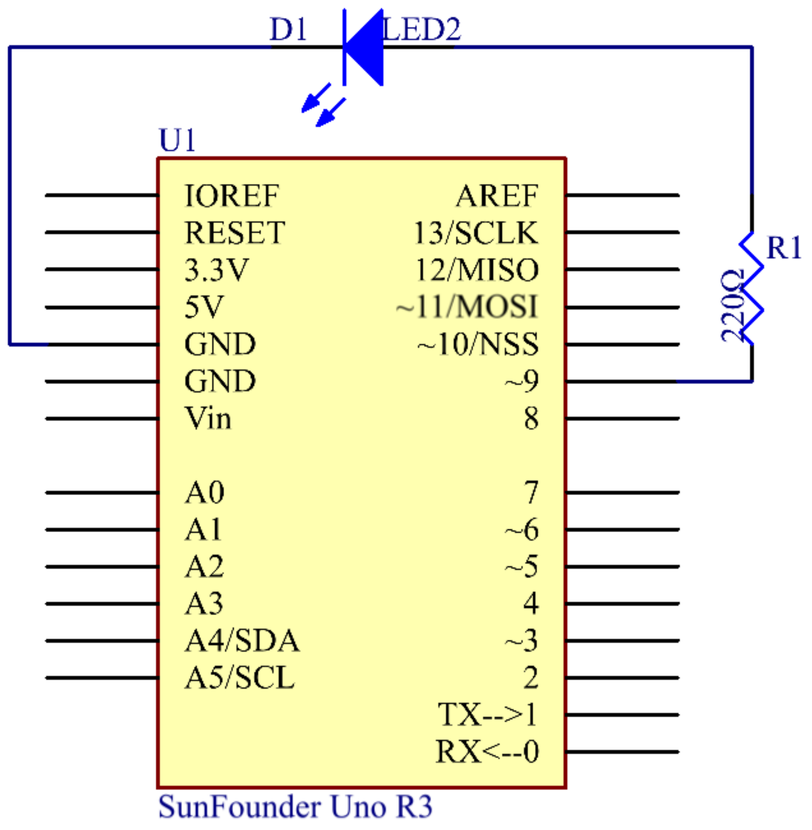


More than one type of them may be used in a project. The color of the jump wires is different but it doesn’t mean their function is different accordingly; it’s just designed so to better identify the connection between each circuit.

Principle:

Connect one end of the 220ohm resistor to pin 9 of the Uno and the other end to the anode (the long pin) of the LED, and the cathode (the short pin) of the LED to GND. When the pin 9 outputs high level, the current gets through the current limiting resistor to the anode of the LED. And since the cathode of the LED is connected to GND, the LED will light up. When pin 9 outputs low level, the LED goes out.

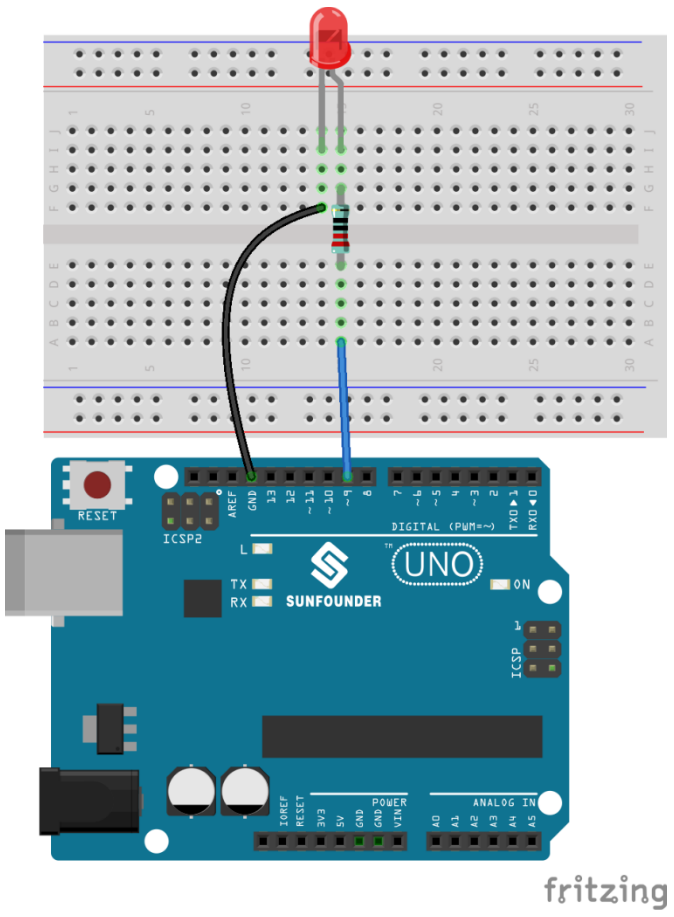
The schematic diagram:



Experimental Procedures

Step 1: Build the circuit (the pin with a curve is the anode of the LED).

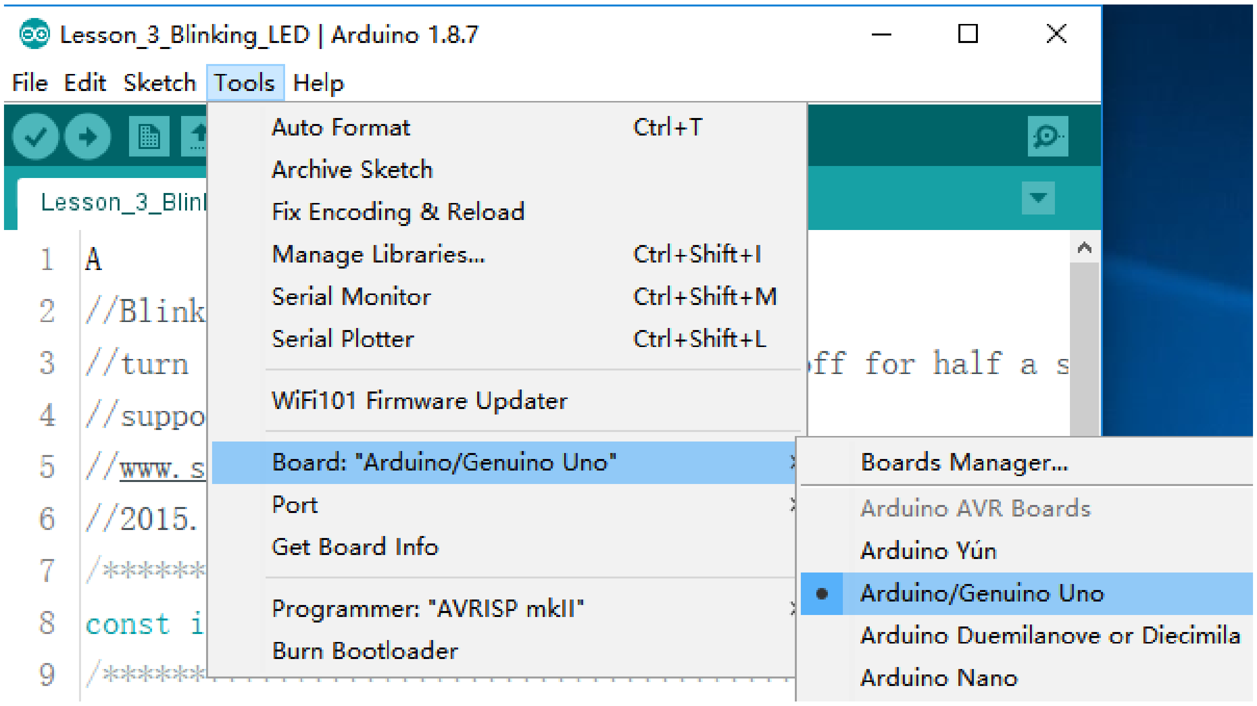
Then plug the board into the computer with a 5V USB cable.



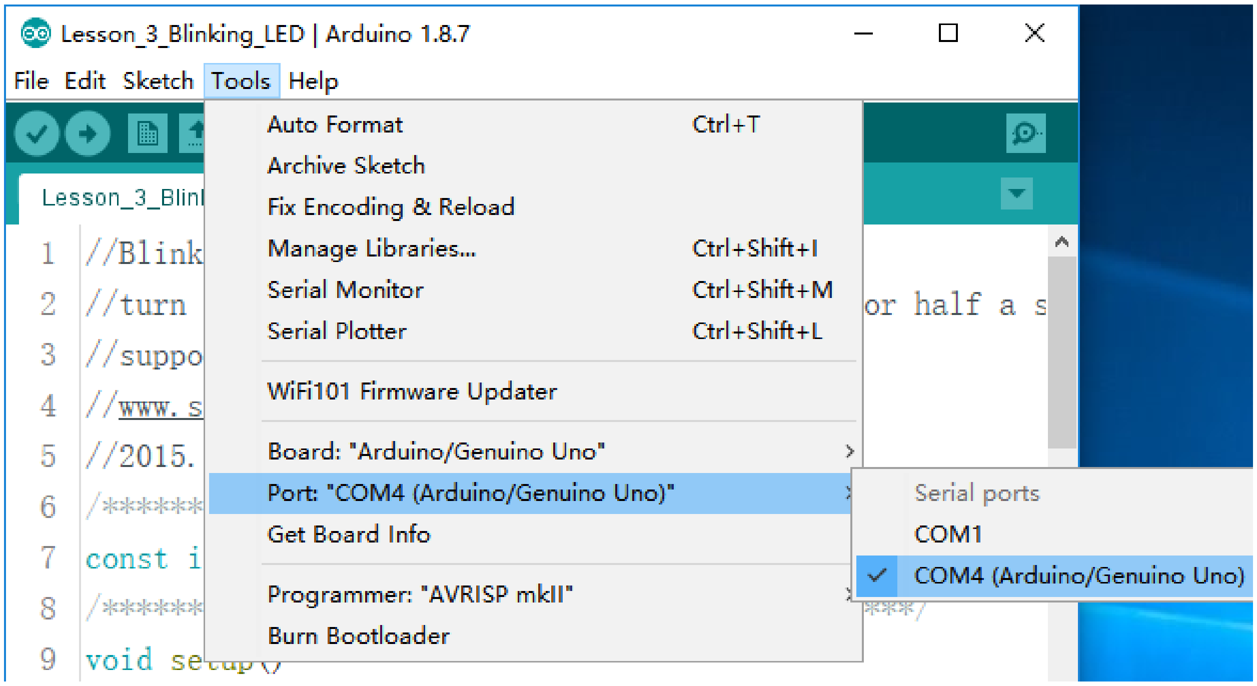
Step 2: Open the Lesson_1_Blinking_LED.ino code file in the path of *electronic-kit\for-Arduino\code\Lesson_1_Blinking_LED*

Step 3: Select the Board and Port

Before uploading the code, you need to select the **Board** and **Port**. Click **Tools** ->**Board** and select **Arduino/Genuino Uno**.

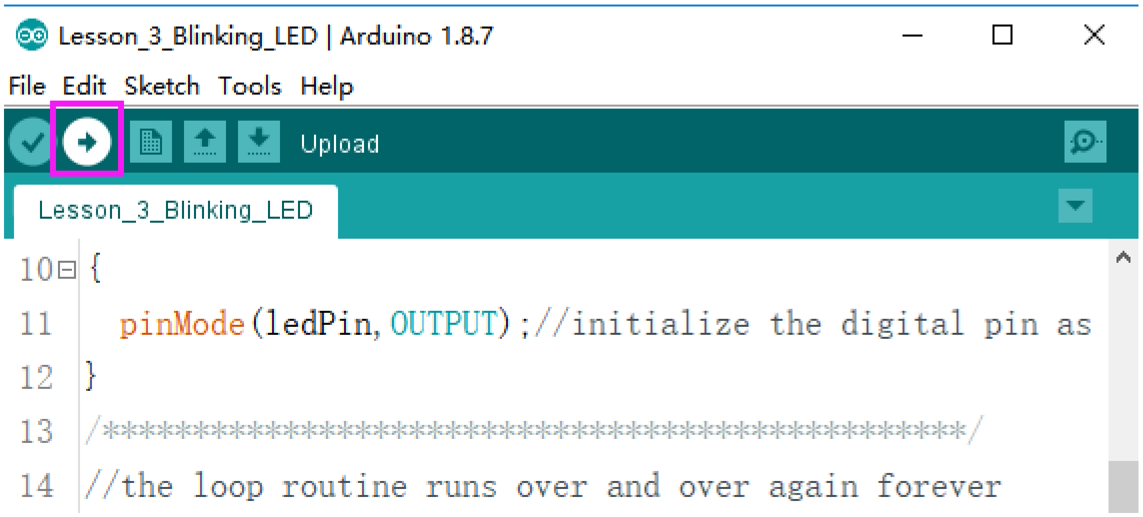

Lesson_3_Blinking_LED | Arduino 1.8.7
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Boards Manager...
Arduino AVR Boards
Arduino Yún
● Arduino/Genuino Uno
Arduino Duemilanove or Diecimila
Arduino Nano

Then select **Tools ->Port**. Your port should be different from mine.


Lesson_3_Blinking_LED | Arduino 1.8.7
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM4 (Arduino/Genuino Uno)"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
✓ COM4 (Arduino/Genuino Uno)

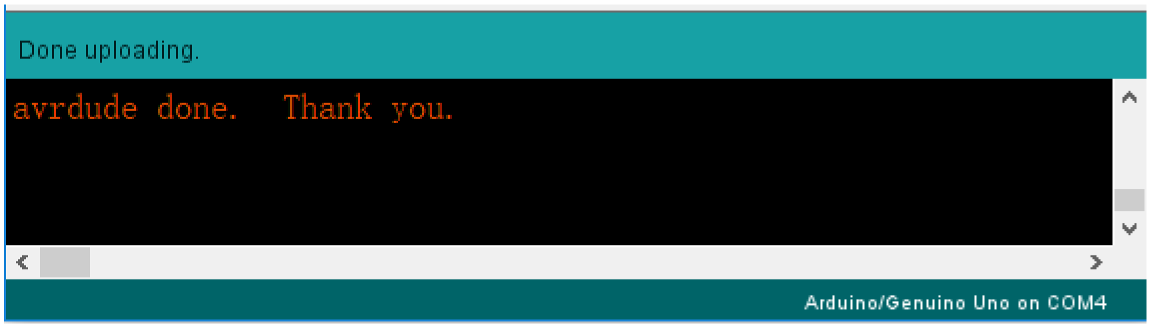
Step 4: Upload the sketch to the Uno board.

Click the **Upload** icon to upload the code to the control board.


Lesson_3_Blinking_LED | Arduino 1.8.7
File Edit Sketch Tools Help
Upload
Lesson_3_Blinking_LED

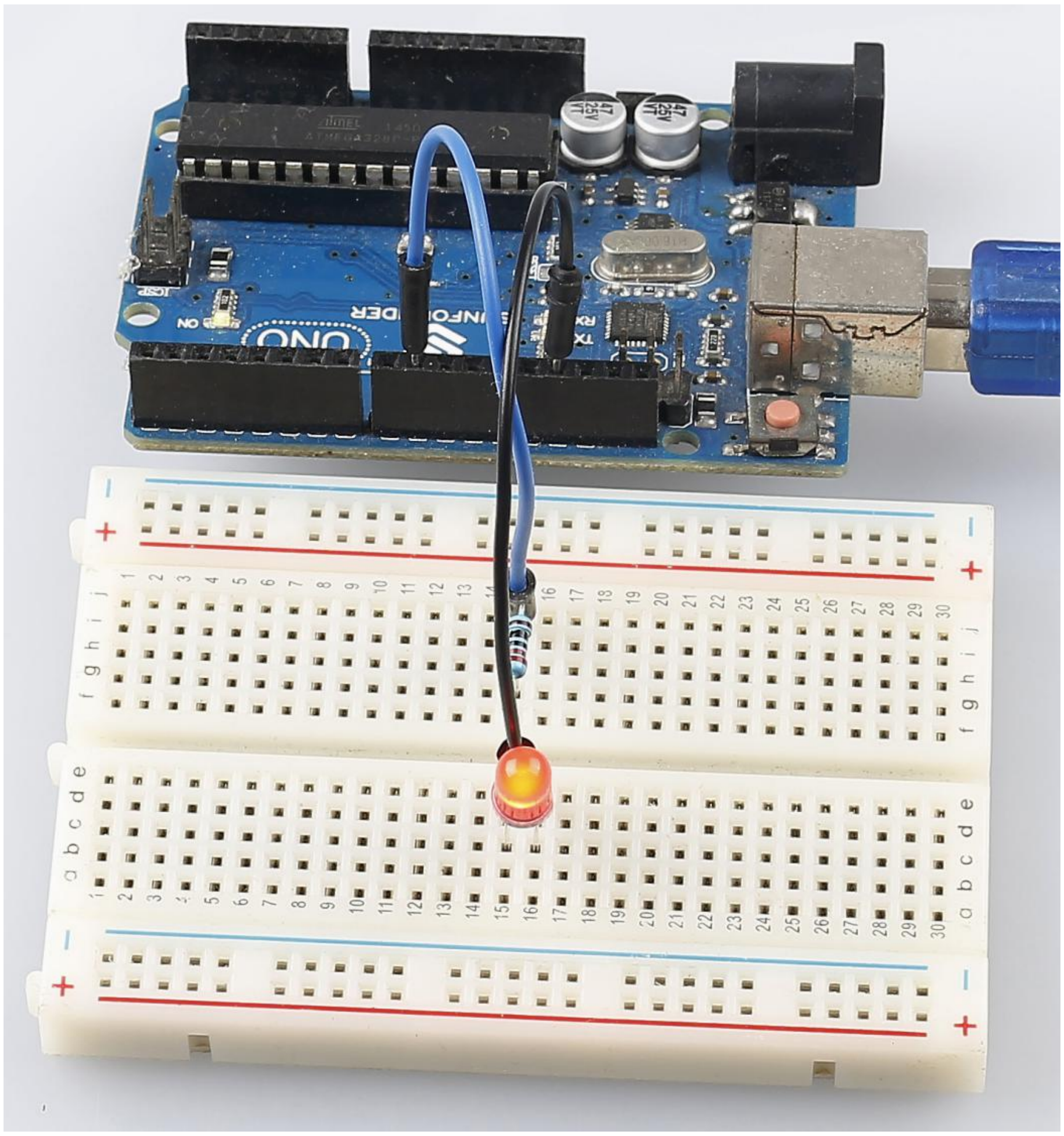
```
{
  pinMode(ledPin,OUTPUT);//initialize the digital pin as
}
/**************************************************/
//the loop routine runs over and over again forever
```

If "Done uploading" appears at the bottom of the window, it means the sketch has been successfully uploaded.



```
avrdude done.  Thank you.
```

You should now see the LED blinking.



Code

Code Analysis 1-1 Define variables

```
const int ledPin = 9; // the number of the LED pin
```

You should define every variable before using in case of making mistakes. This line defines a constant variable *ledPin* for the pin 9. In the following code, *ledPin* stands for pin 9. You can also directly use pin 9 instead.

Code Analysis 1-2 setup() function

A typical Arduino program consists of two subprograms: *setup()* for initialization and loop() which contains the main body of the program.

The *setup()* function is usually used to initialize the digital pins and set them as input or output as well as the baud rate of the serial communication.

The *loop()* function contains what the MCU will run circularly. It will not stop unless something happens like power outages.

```
void setup()

{

    pinMode(ledPin,OUTPUT); // initialize the digital pin as an output

}
```

The setup() function here sets the *ledPin* as OUTPUT.

pinMode(Pin): Configures the specified pin to behave either as an input or an output.

The void before the setup means that this function will not return a value. Even when no pins need to be initialized, you still need this function. Otherwise there will be errors in compiling.

Code Analysis 1-3 loop function

```
void loop()

{

    digitalWrite(ledPin,HIGH); // turn the LED on

    delay(500); // wait for half a second

    digitalWrite(ledPin,LOW); // turn the LED off

    delay(500); // wait for half a second

}
```

This program is to set *ledPin* as HIGH to turn on the LED, with a delay of 500ms. Set *ledPin* as LOW to turn the LED off and also delay 500ms. The MCU will run this program repeatedly and you will see that the LED brightens for 500ms and then dims for 500ms. This on/off alternation will not stop until the control board runs out of energy.

digitWrite(Pin): Write a HIGH or a LOW value to a digital pin. When this pin has been set as output in *pinModel()*, its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

Experiment Summary

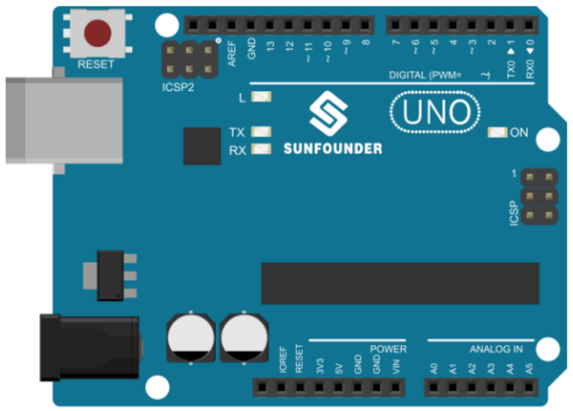
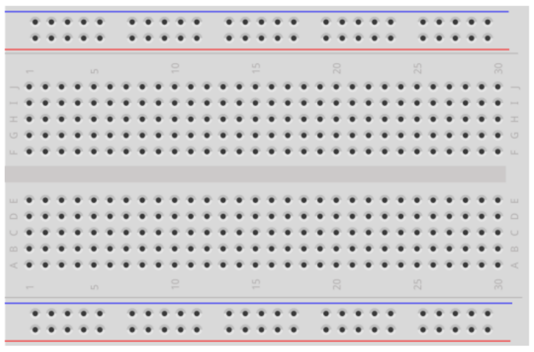
Through this experiment, you have learned how to turn on an LED. You can also change the blinking frequency of the LED by changing the *num* value in the delay function *delay (num)*. For example, change it to **delay (250)** and you will find that the LED blinks more quickly.

Lesson 2 Controlling LED by Button

Introduction

In this experiment, we will learn how to turn on/off an LED by using an I/O port and a button. The "I/O port" refers to the INPUT and OUTPUT port. Here the INPUT port of the Uno board is used to read the output of an external device. Since the board itself has an LED (connected to Pin 13), you can use this LED to do this experiment for convenience.

Components

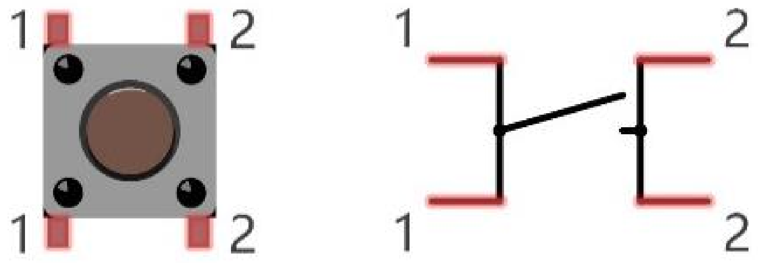
1 * Uno Board 	1 * Breadboard 
1 * Resistor (10kΩ)	1 * Button
1 * USB Cable	Several Jumper Wires

Experimental Principle

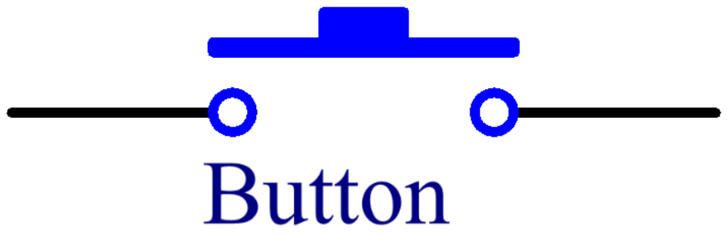
Button

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

Two pins on the left is connected, and the right is similar as the left, which is shown in the below:



The following is the internal structure of a button. The symbol on the right below is usually used to represent a button in circuits.

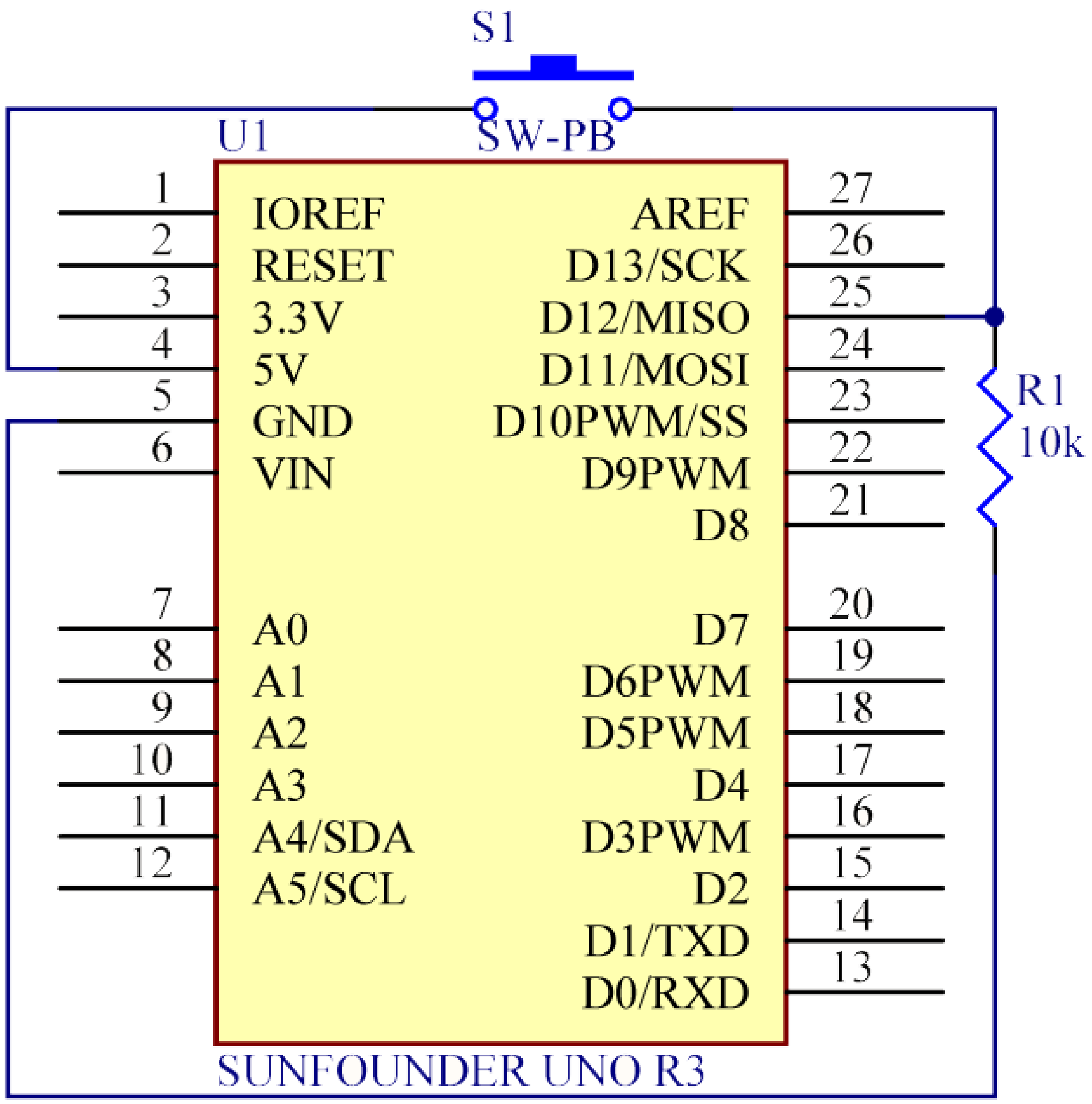


When the button is pressed, the 4 pins are connected, thus closing the circuit.

Principle:

Connect one end of the buttons to pin 12 which connects with a pull-down resistor (to eliminate jitter and output a stable level when the button is working). Connect the other end of the resistor to GND and one of the pins at the other end of the button to 5V. When the button is pressed, pin 12 is 5V (HIGH), then pin 13 is set (integrated with an LED) as HIGH at the same time. If the button release, the pin 12 changes to LOW and pin 13 is set to LOW. So we will see the LED lights up and goes out alternately as the button is pressed and released.

The schematic diagram



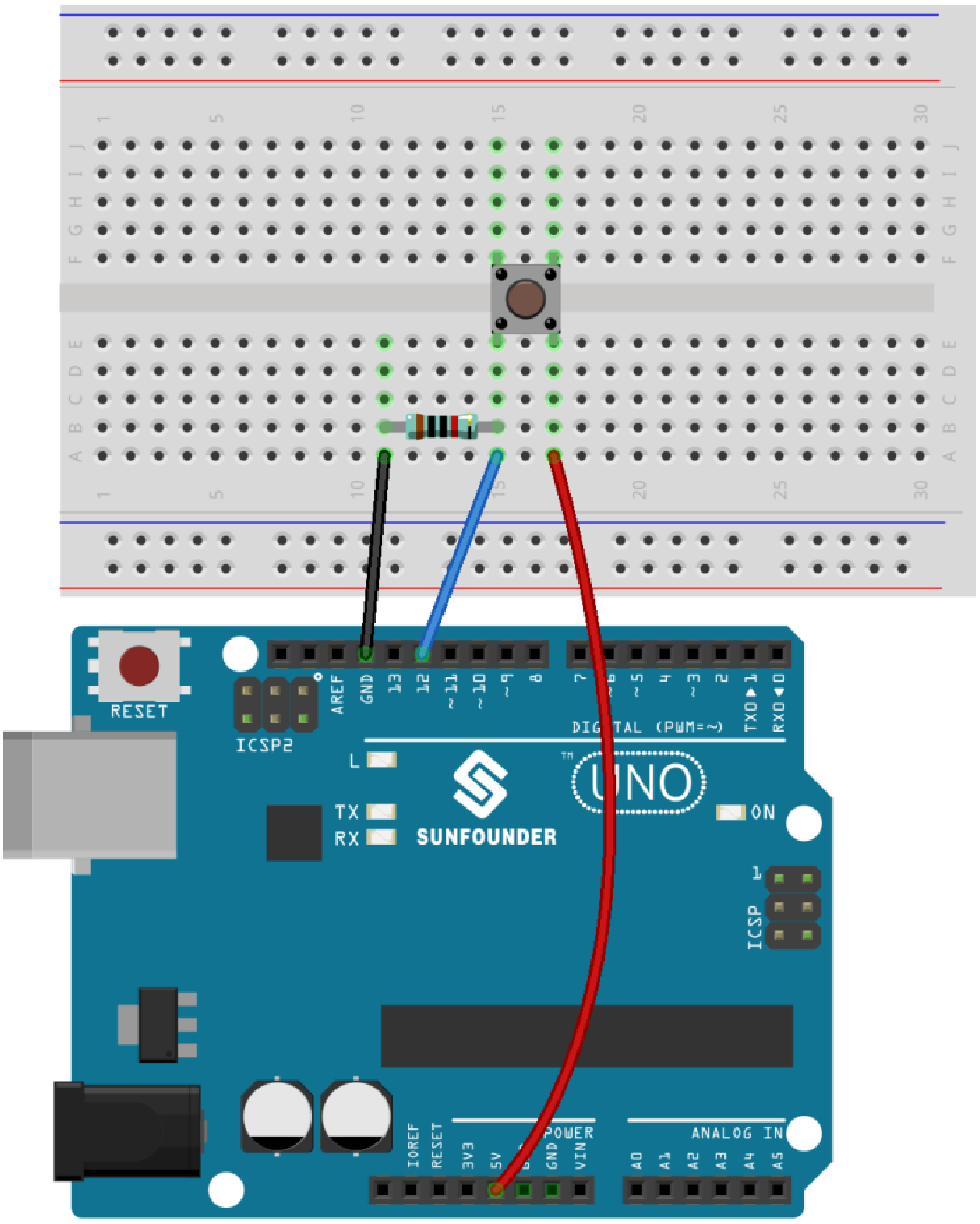
Experimental Procedures

Step 1: Build the circuit

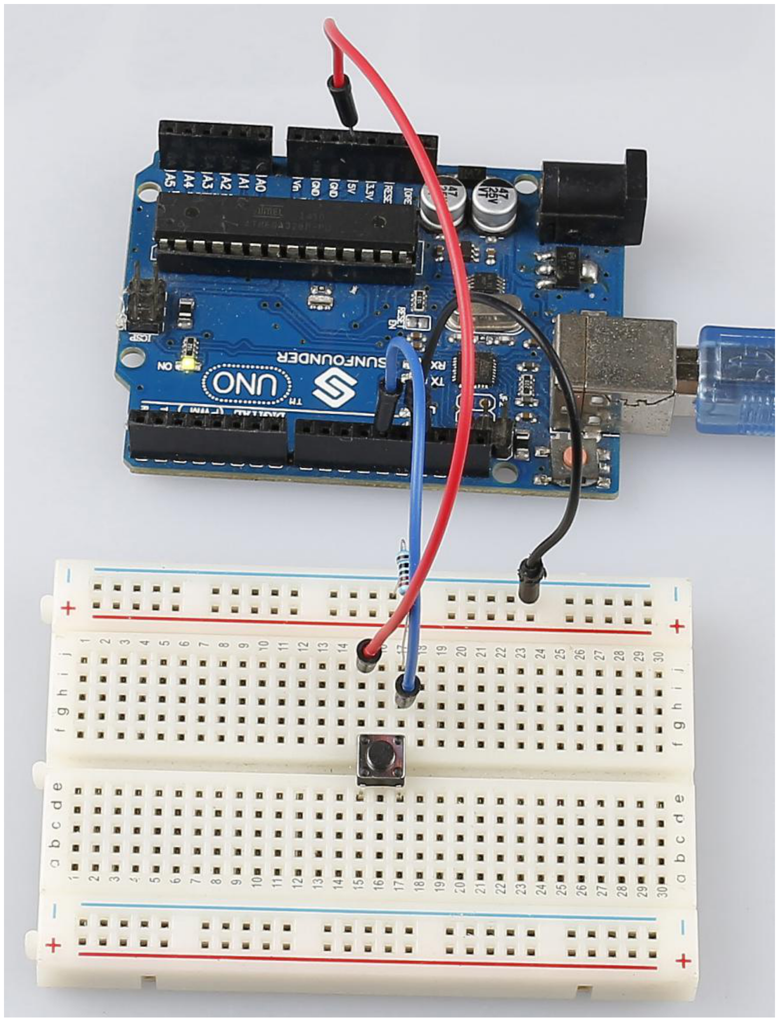
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, press the button, and the LED on the Uno board will light up.



Code

Code Analysis 2-1 Define variables

```
const int buttonPin = 12; // the button connect to pin 12

const int ledPin = 13; // the led connect to pin13

int buttonState = 0; // variable for reading the pushbutton status
```

Connect the button to pin 12. LED has been connected to pin 13. Define a variable *buttonState* to restore the state of the button.

Code Analysis 2-2 Set the input and output status of the pins

```
pinMode(buttonPin, INPUT); // initialize thebuttonPin as input

pinMode(ledPin, OUTPUT); // initialize the led pin as output
```

We need to know the status of the button in this experiment, so here set the *buttonPin* as INPUT; to set HIGH/LOW of the LED, we set *LedPin* as OUTPUT.

Code Analysis 2-3 Read the status of the button

```
buttonState = digitalRead(buttonPin);
```

buttonPin(Pin12) is a digital pin; here is to read the value of the button and store it in *buttonState*.

digitalRead (Pin): Reads the value from a specified digital pin, either HIGH or LOW.

Code Analysis 2-4 Turn on the LED when the button is pressed

```
if (buttonState == HIGH )

{

    digitalWrite(ledPin, HIGH); // turn the led on

}

else

{

    digitalWrite(ledPin, LOW); // turn the led off

}
```

In this part, when the **buttonState** is High level, write *ledPin* as High and the LED will be turned on. As one end of the button has been connected to 5V and the other end to pin 12, when the button is pressed, pin 12 is 5V (HIGH). And then determine with the *if* (conditional); if the conditional is true, then the LED will light up.

Else means that when the if(conditional) is determined as false, run the code in *else*.

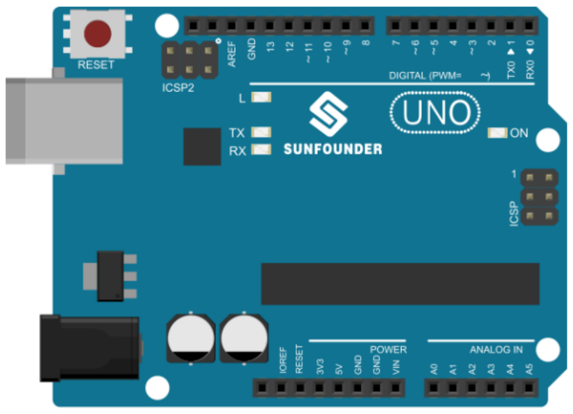
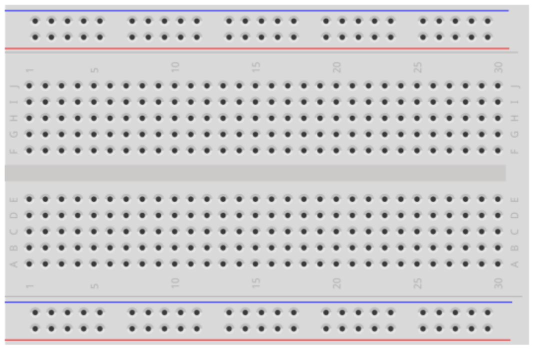
Experiment Summary

You can also change the code to: when the button is pressed, if (buttonState=HIGH). The LED goes out (digitalWrite(ledPin, LOW)). When the button is released (the else), the LED lights up ((digitalWrite(ledPin, HIGH)). You only need to replace the code in **if** with those in **else**.

Lesson 3 Controlling an LED by Potentiometer

Introduction

In this lesson, let’s see how to change the luminance of an LED by a potentiometer, and receive the data of the potentiometer in Serial Monitor to see its value change.

1 * Uno Board 	1 * Breadboard 

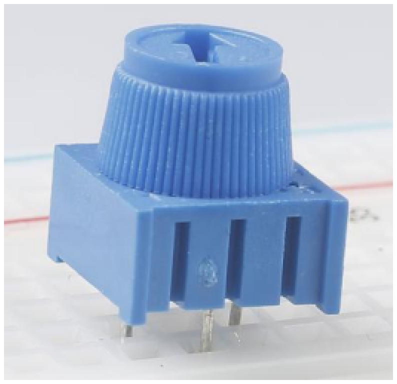
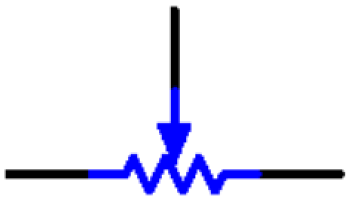
1 * Resistor (220Ω)	1* Potentiometer	1 * LED

1 * USB Cable	Several Jumper Wires

Experimental Principle

Potentiometer

Potentiometer is also a resistance component with 3 terminals and its resistance value can be adjusted according to some regular variation. Potentiometer usually consists of resistor and movable brush. When the brush is moving along the resistor, there is a certain resistance or voltage output depending on the displacement.

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

1. Serving as a voltage divider

   Potentiometer is a continuously adjustable resistor. When you adjust the shaft or sliding handle of the potentiometer, the movable contact will slide on the resistor. At this point, a voltage can be output depending on the voltage applied onto the potentiometer and the angle the movable arm has rotated to or the travel it has made.

2. Serving as a rheostat

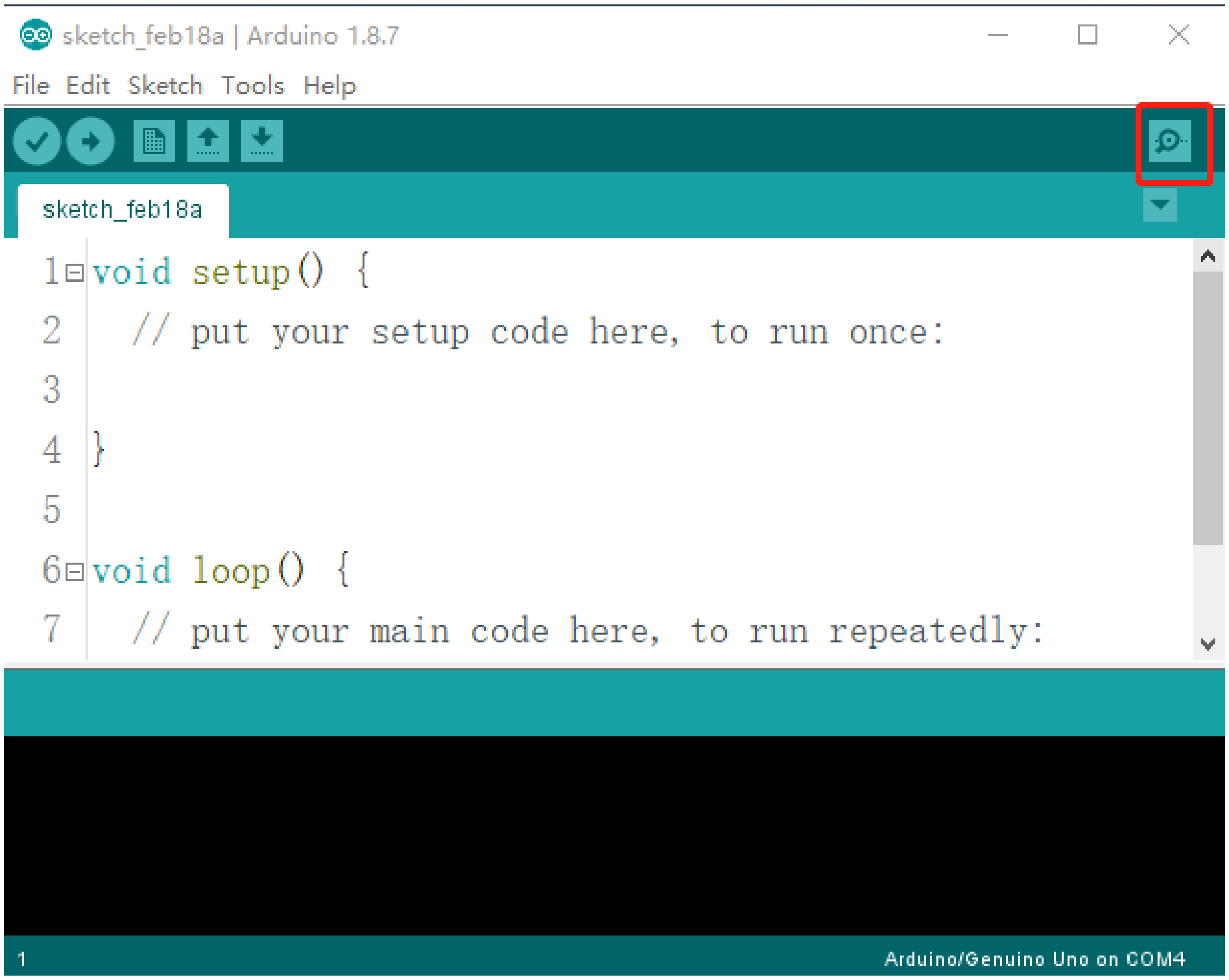
   When the potentiometer is used as a rheostat, connect the middle pin and one of the other 2 pins in the circuit. Thus you can get a smoothly and continuously changed resistance value within the travel of the moving contact.

3. Serving as a current controller

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

Serial Monitor

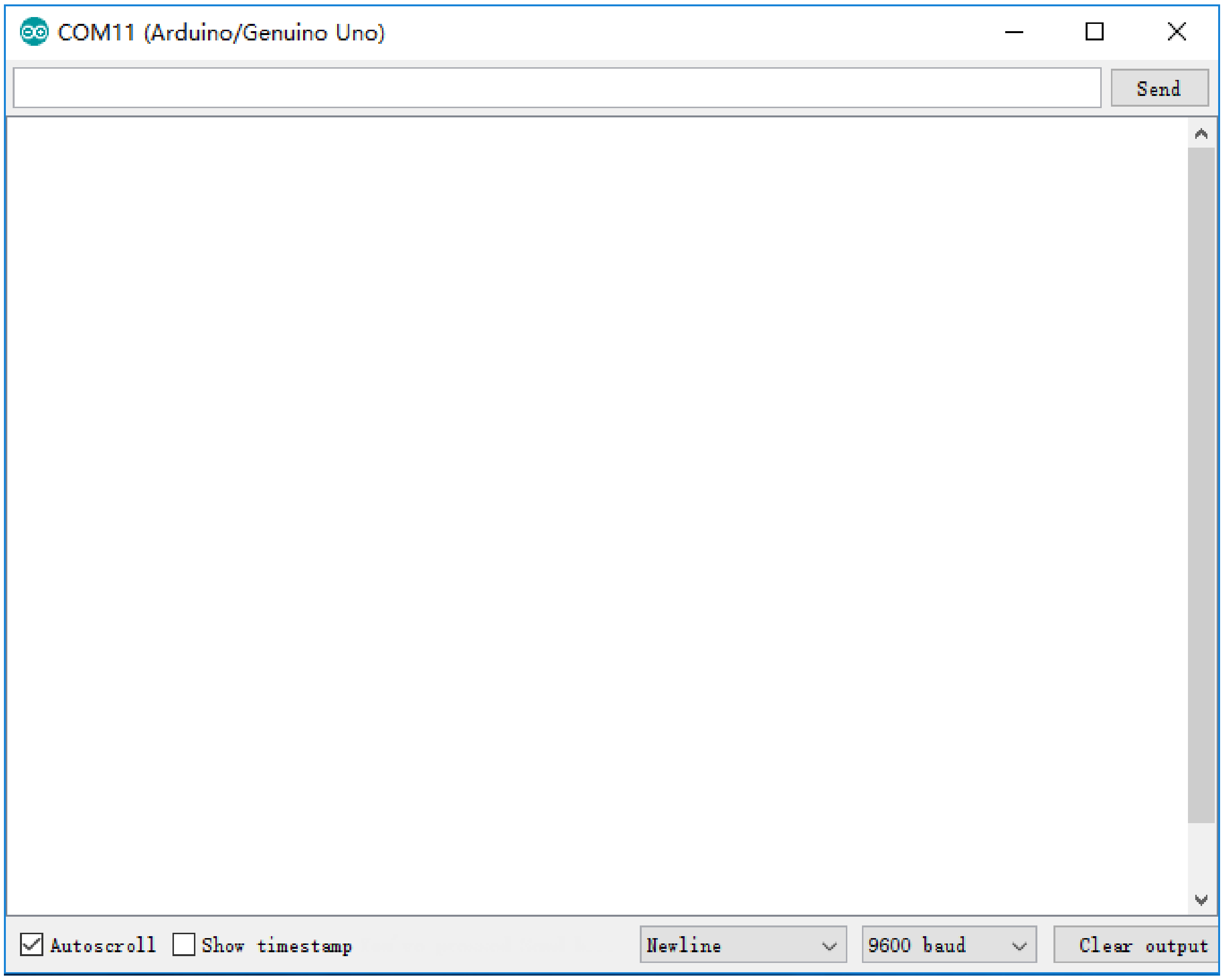
Serial Monitor is used for communication between the Mega 2560 board and a computer or other devices. It is a built-in software in the Arduino environment and you can click the button on the upper right corner to open it. You can send and receive data via the serial port on the control board and control the board by input from the keyboard.


sketch_feb18a | Arduino 1.8.7
File Edit Sketch Tools Help
sketch_feb18a

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
```

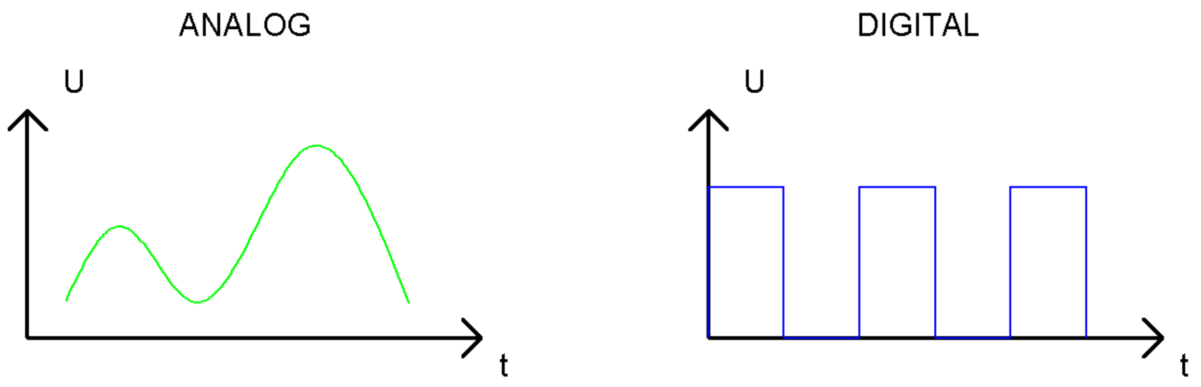
1
Arduino/Genuino Uno on COM4

Here, the Serial Monitor serves as a transfer station for communication between your computer and the Uno board. First, the computer transfers data to the Serial Monitor, and then the data is read by the Uno board. Finally, the Uno will perform related operations. Click the icon at the top right corner and a window will pop up as shown below:



Analog V.S. Digital

A linear potentiometer is an analog electronic component. So what's the difference between an analog value and a digital one? Simply put, digital means on/off, high/low level with just two states, i.e. either 0 or 1. But the data state of analog signals is linear, for example, from 1 to 1000; the signal value changes over time instead of indicating an exact number. Analog signals include those of light intensity, humidity, temperature, and so on.

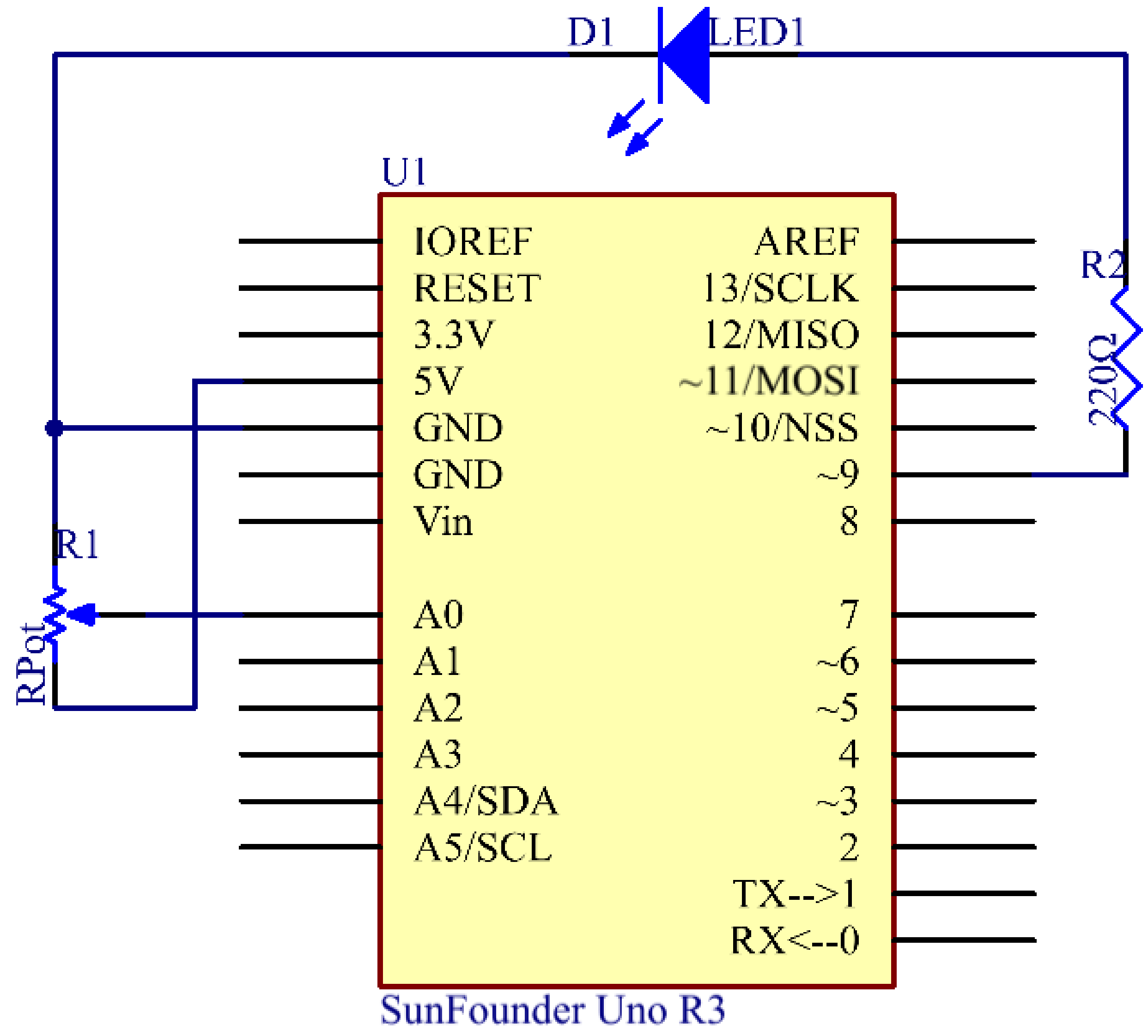


Principle:

In this experiment, the potentiometer is used as voltage divider, meaning connecting devices to all of its three pins.

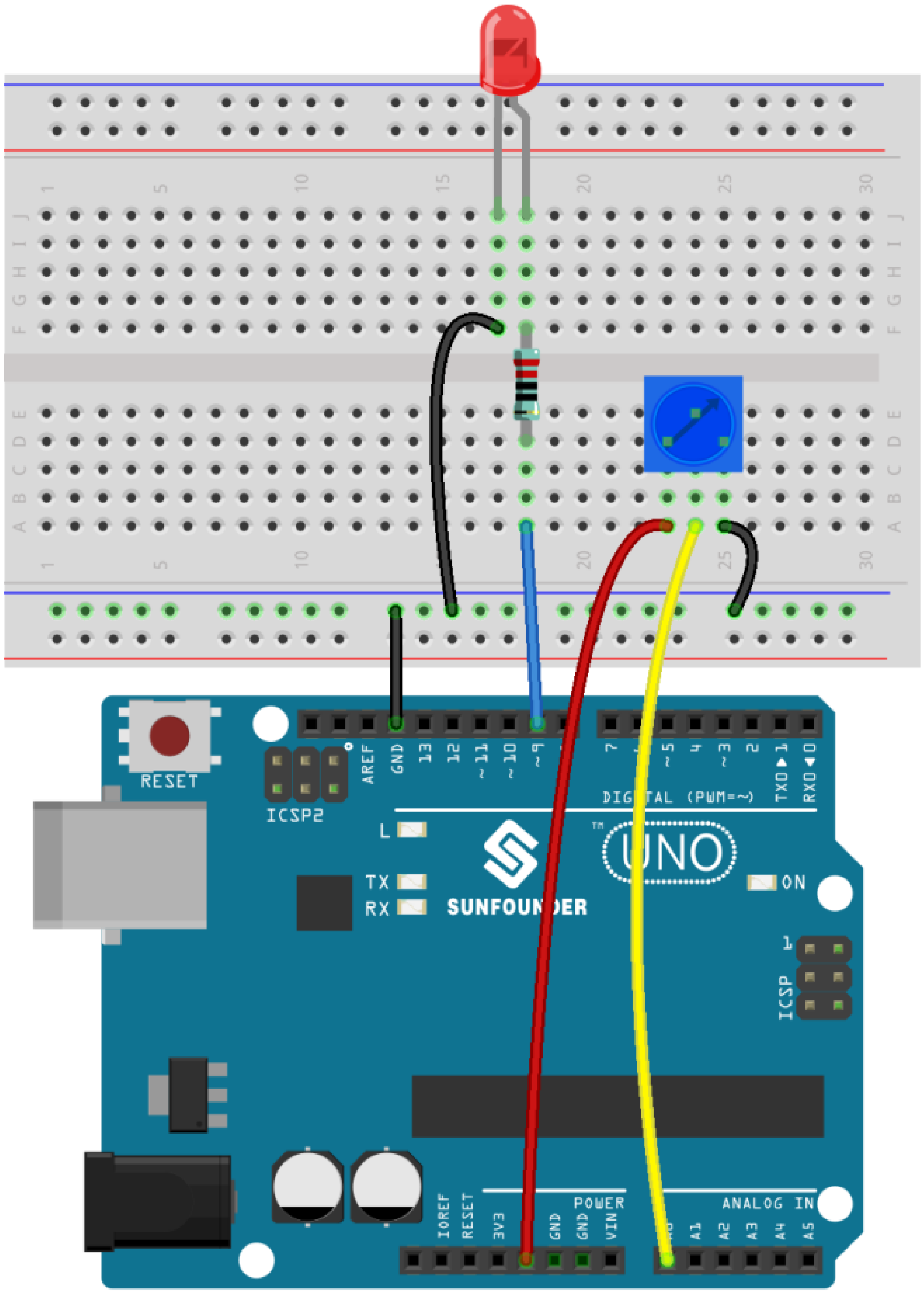
Connect the middle pin of the potentiometer to pin A0 and the other two pins to 5V and GND respectively. Therefore, the voltage of the potentiometer is 0-5V. Spin the knob of the potentiometer, and the voltage at pin A0 will change. Then convert that voltage into a digital value (0-1024) with the AD converter in the control board. Through programming, we can use the converted digital value to control the brightness of the LED on the control board.

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
7
~5
4
~3
2
TX▶1
RX◀0
DIGITAL (PWM=~)
L
TX
RX
SUNFOUNDER
UNO
ON
ICSP
IOREF
RESET
3V3
GND
GND
VIN
POWER
ANALOG IN
A1
A2
A3
A4
A5

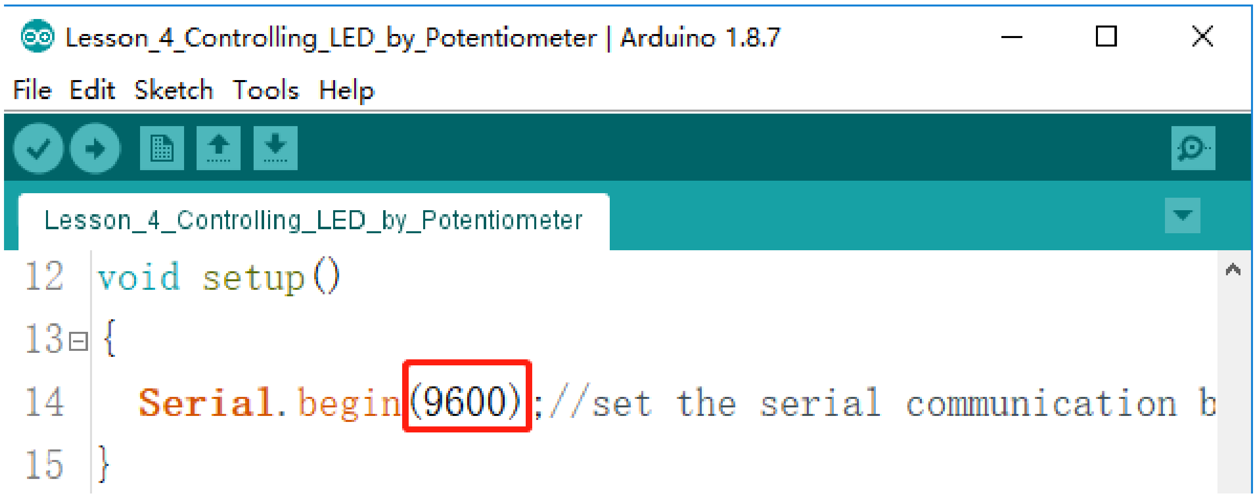
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Step5: Open the Serial Monitor.

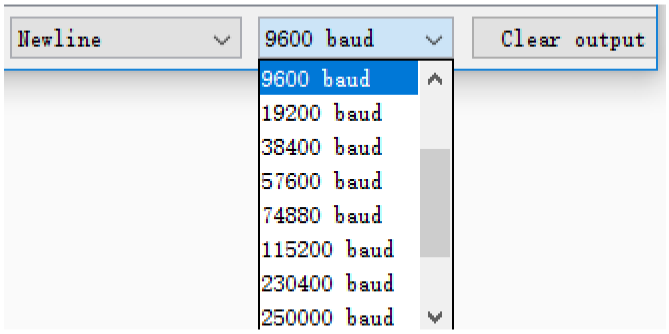
Find the Serial.begin() code to see what baud rate is set, here is 9600. Then click the top right corner icon to open the Serial Monitor.


Lesson_4_Controlling_LED_by_Potentiometer | Arduino 1.8.7
File Edit Sketch Tools Help
Lesson_4_Controlling_LED_by_Potentiometer

```
void setup()
{
  Serial.begin(9600);//set the serial communication b
}
```

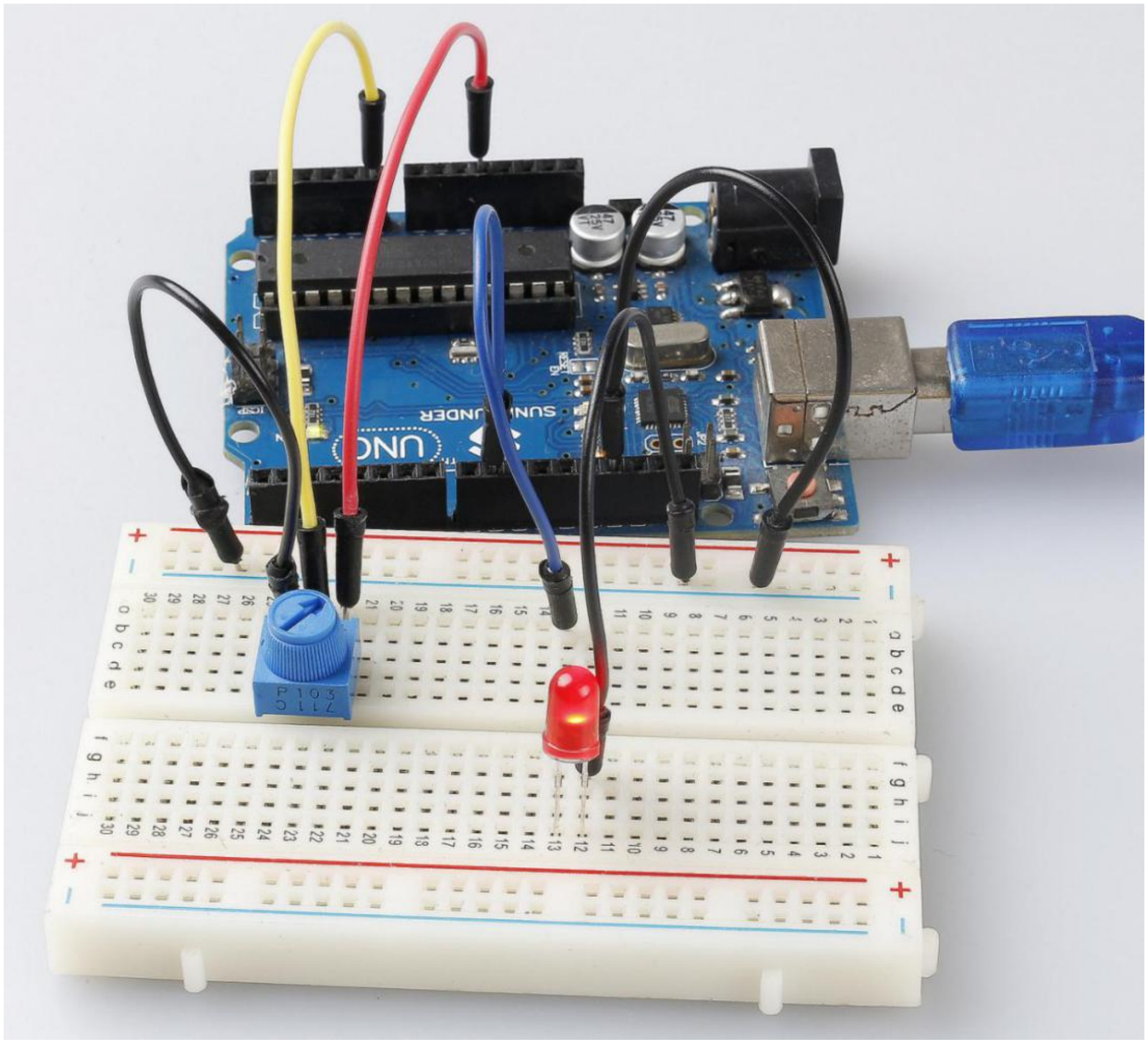
Step6: Set the baud rate to 9600.

The default baud rate for serial monitors is 9600, and if the code is also set to 9600, there is no need to change the baud rate bar.



Spin the shaft of the potentiometer and you should see the luminance of the LED change.

If you want to check the corresponding value changes, open the Serial Monitor and the data in the window will change with your spinning of the potentiometer knob.



Code

Code Analysis 3-1 Read the value from A0

```
inputValue = analogRead(analogPin); // read the value from the potentiometer
```

This line is to store the values A0 has read in the inputValue which has been defined before.

analog Read() reads the value from the specified analog pin. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023.

Code Analysis 3-2 Print values on Serial Monitor

```
Serial.print("Input: "); // print "Input"

Serial.println(inputValue); // print inputValue
```

Serial.print(): Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

Serial.print(): Commandant takes the same forms as Serial.print(), but it is followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').

Code Analysis 3-3 Map the values

```
outputValue = map(inputValue, 0, 1023, 0, 255);
// Convert from 0-1023 proportional to the number of a number of from 0 to 255
```

map(value, Fromm, from High, to Low, thigh) re-maps a number from one range to another. That is, a **value** of **Fromm** would get mapped to one of **to Low**, and a value of **from High** to one of **thigh**, values in-between to values in-between, etc.

As the range of *led Pin* (pin 9) is 0-255, we need to map 0-1023 with 0-255.

Display the output value in Serial Monitor in the same way. If you are not so clear about the *map()* functions, you can observe the data in the Serial Monitor and analyze it.

```
Serial.print("Output: "); // print "Output"

Serial.println(outputValue); // print outputValue
```

Code Analysis 3-4 Write the value of the potentiometer to LED

```
analogWrite(ledPin, outputValue); // turn the LED on depending on the output value
```

Write the output value to *led Pin* and you will see that the luminance of LED changes with your spinning of the potentiometer knob.

analog Write(): Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *incommode()* to set the pin as output before calling *analog Write()*.

Experiment Summary

This experiment can also be changed to others as you like. For example, use the potentiometer to control the time interval for the LED blinking. It is to use the value read from the potentiometer for delaying, as shown below. Have a try!

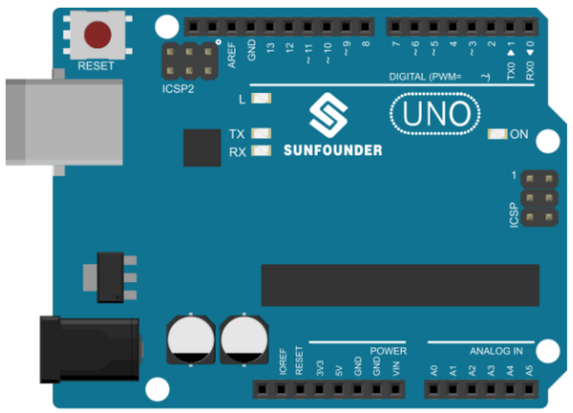
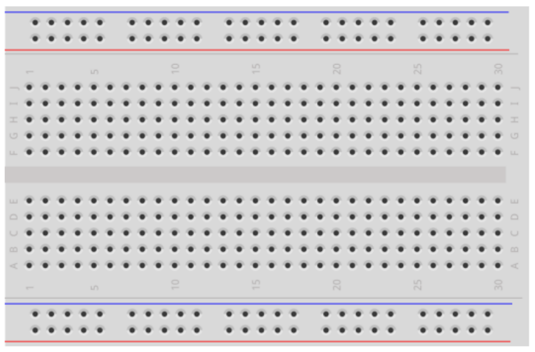
```
inputValue = analogRead(analogPin);//read the value from the sensor
digitalWrite(ledPin,HIGH);
delay(inputValue);
digitalWrite(ledPin,LOW);
delay(inputValue);
```

Lesson 4 Doorbell

Introduction

A buzzer is a great tool in your experiments whenever you want to make some sounds. In this lesson, we will learn how to drive an active buzzer to build a simple doorbell.

Components

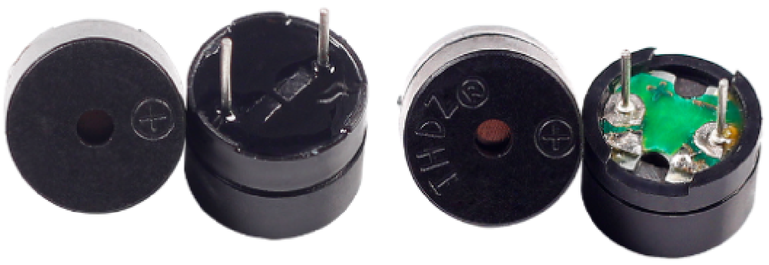
1 * Uno Board 	1 * Breadboard 

1 * 104 Capacitor	1*Buzzer(Active)	1 * Resistor (10k Ω)	1 * Button
1 * USB Cable		Several Jumper Wires	

Experimental Principle

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

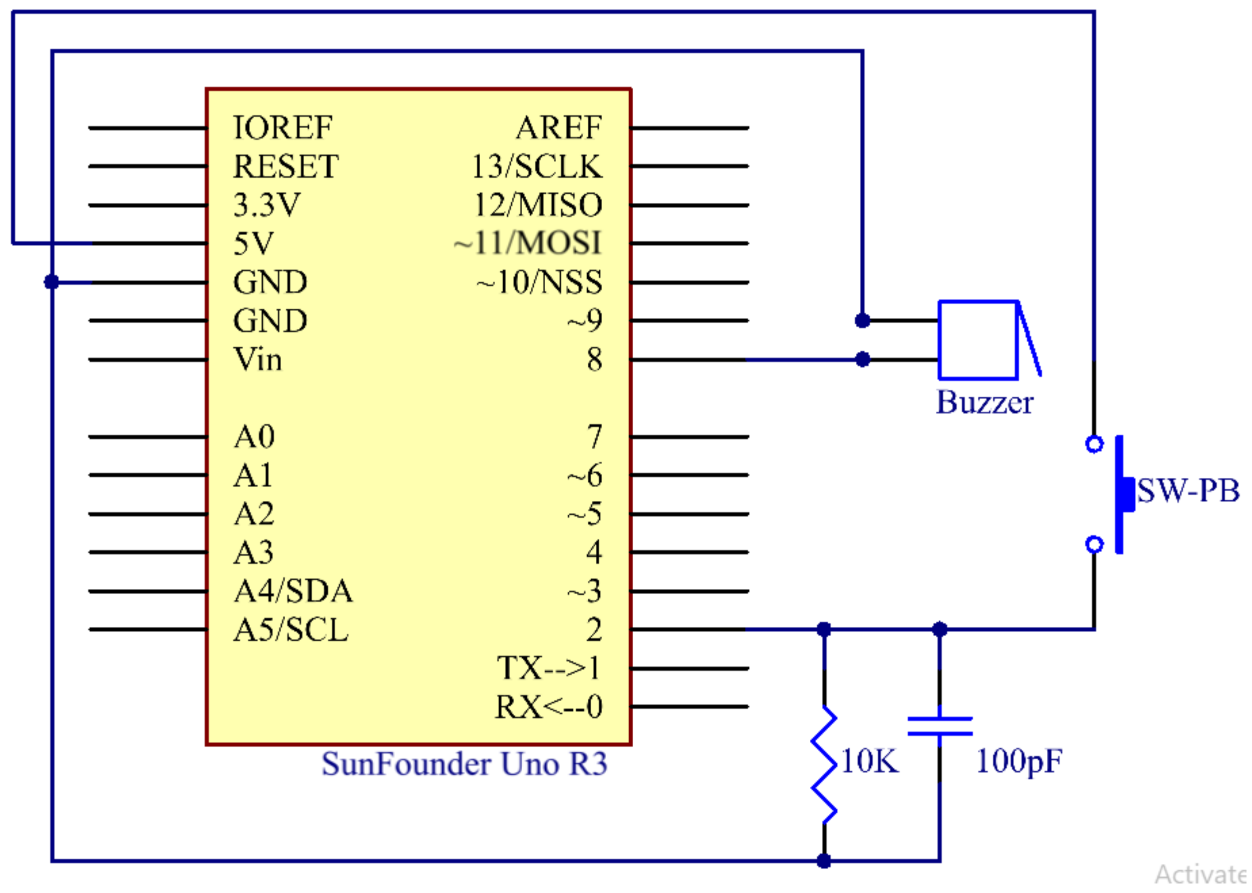
The difference between an active buzzer and a passive buzzer:



An active buzzer has a built-in oscillating source, so it will make sounds when electrified. But a passive buzzer does not have such source, so it will not tweet if DC signals are used; instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

In this experiment, we use an active buzzer.

The schematic diagram



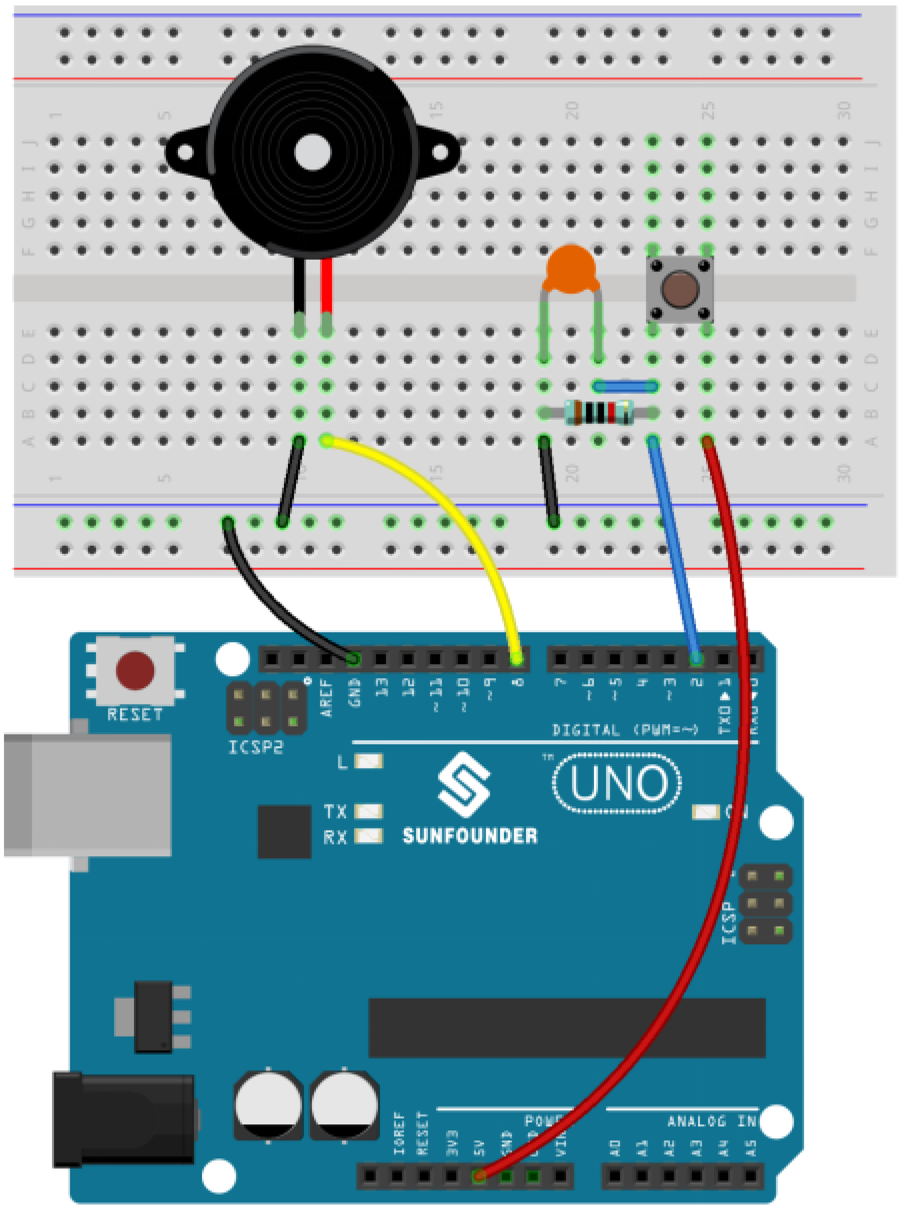
Experimental Procedures

Step 1: Build the circuit (Long pins of buzzer is the Anode and the short pin is Cathode).

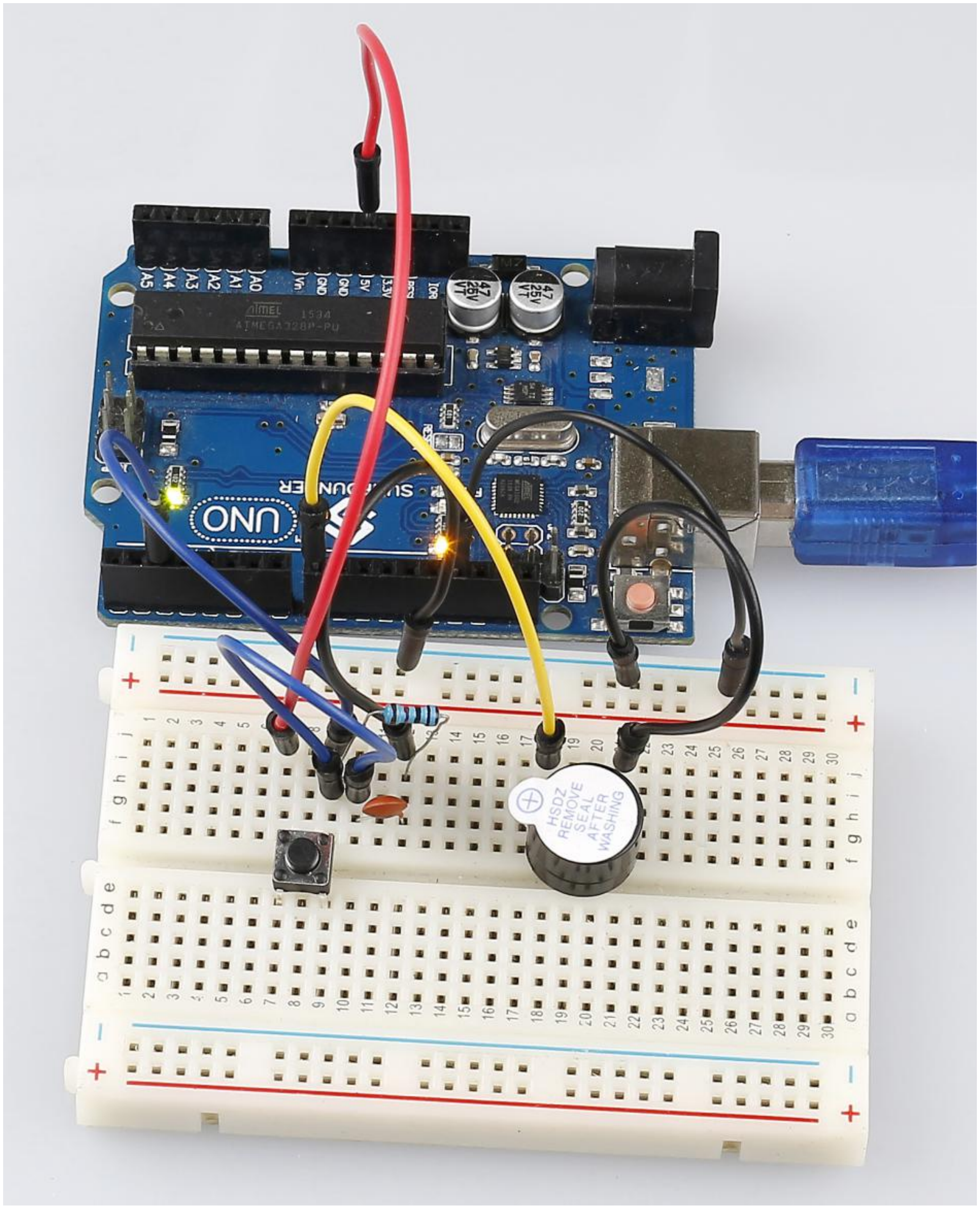
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, you should hear the buzzer beep.



Code

Code Analysis 4-1 Define variables

```
const int buttonPin = 2; //the button connect to pin2

const int buzzerPin = 8; //the led connect to pin8

/**********************************/

int buttonState = 0; //variable for reading the pushbutton status
```

Connect the button to pin 2 and buzzer to pin 8. Define a variable *buttonState* to restore the state of the button.

Code Analysis 4-2 Set the input and output status of the pins

```
void setup()

{

    pinMode(buttonPin, INPUT); //initialize the buttonPin as input

    pinMode(buzzerPin, OUTPUT); //initialize the buzzerpin as output

}
```

We need to know the status of the button in this experiment, so here set the *buttonPin* as INPUT; to set HIGH/LOW of the buzzer, we set *buzzerPin* as OUTPUT.

Code Analysis 4-3 Read the status of the button

```
buttonState = digitalRead(buttonPin);
```

buttonPin(Pin2) is a digital pin; here is to read the value of the button and store it in *buttonState*.

digitalRead (Pin): Reads the value from a specified digital pin, either HIGH or LOW.

Code Analysis 4-4 Turn on the LED when the button is pressed

```
if (buttonState == HIGH ) //When press the button, run the following code.

{

    for (i = 0; i < 50; i++) /*When i=o, which accords with the condition
    i<=50, i++ equals to 1 (here in i = i + 1, the two "i"s are not the
    same, but i_now = i_before + 1). Run the code in the curly braces:
    let the buzzer beep for 3ms and stop for 3ms. Then repeat 50 times.*/

    {
        digitalWrite(buzzerPin, HIGH); //Let the buzzer beep.
```

(continues on next page)

(continued from previous page)

```
        delay(3);//wait for 3ms

        digitalWrite(buzzerPin, LOW); //Stop the buzzer.

        delay(3);//wait for 3ms
    }

    for (i = 0; i < 80; i++) //Let the buzzer beep for 5ms and stop for 5ms, repeat␣
↪80 times.

    {
        digitalWrite(buzzerPin, HIGH);

        delay(5);//wait for 5ms

        digitalWrite(buzzerPin, LOW);

        delay(5);//wait for 5ms
    }
}
```

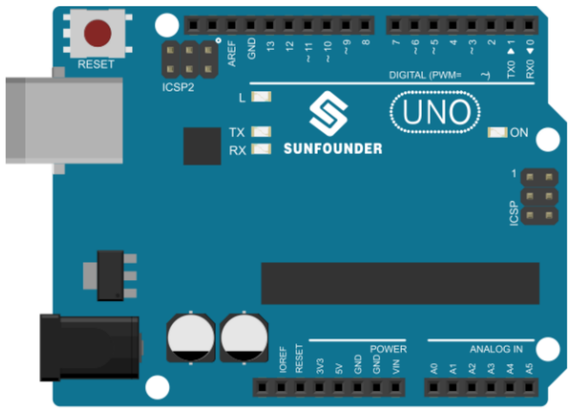
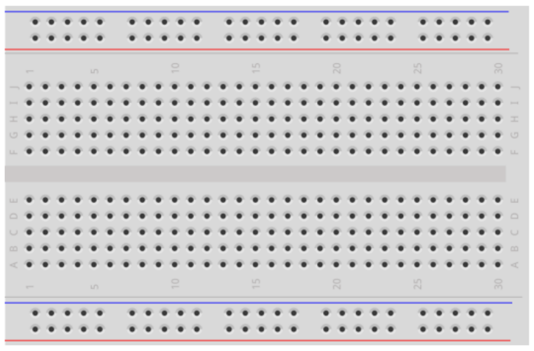
In this part, when the **buttonState** is High level, then let the buzzer beeping in different frequency which can simulate the doorbell.

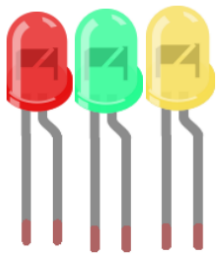
Lesson 5 Photoresistor

Introduction

In this lesson, you will learn to how to measure light intensity using a photo resistor. The resistance of a photo resistor changes with incident light intensity. If the light intensity gets higher, the resistance decreases; if it gets lower, the resistance increases.

Components

1 * Uno Board	1 * Breadboard
	

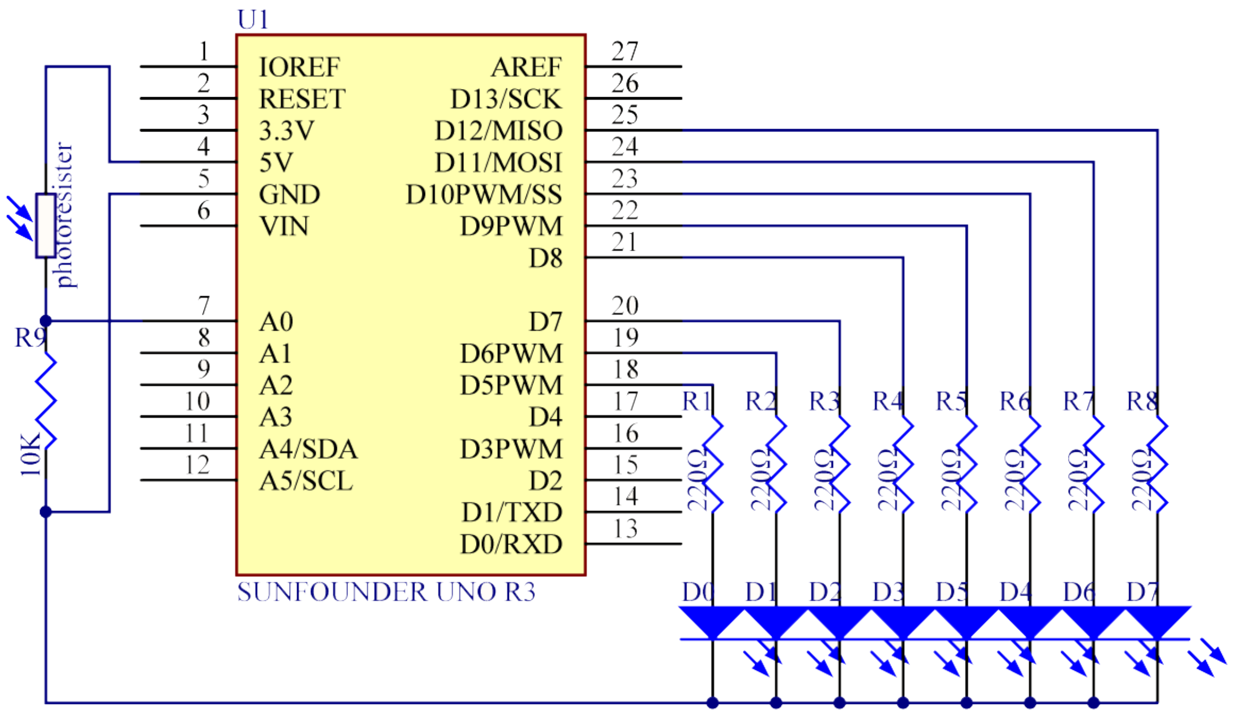
1 * Photo resistor	8 * LED 	8 * Resistor (220Ω)	1 * Resistor (10KΩ)
1 * USB Cable		Several Jumper Wires	

Experimental Principle

A photo resistor or photocell is a light-controlled variable resistor. The resistance of a photo resistor decreases with increasing incident light intensity; in other words, it exhibits photo conductivity. A photo resistor can be applied in light-sensitive detector circuits, and light- and darkness-activated switching circuits.

In this experiment, we will use 8 LEDs to show the light intensity. The higher the light intensity is, the more LEDs will light up. When the light intensity is high enough, all the LEDs will be on. When there is no light, all the LEDs will go out.

The schematic diagram:



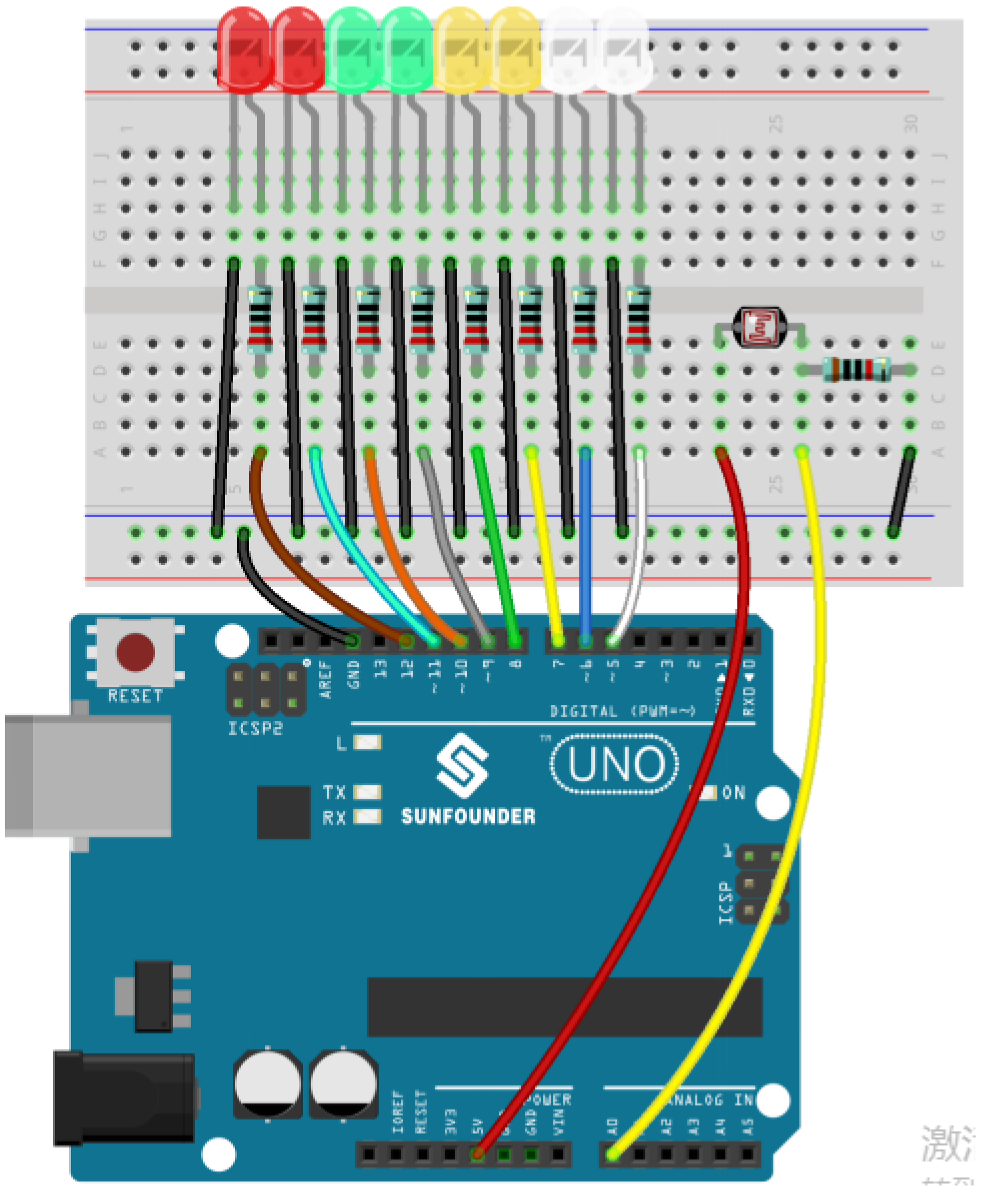
Experimental Procedures

Step 1: Build the circuit

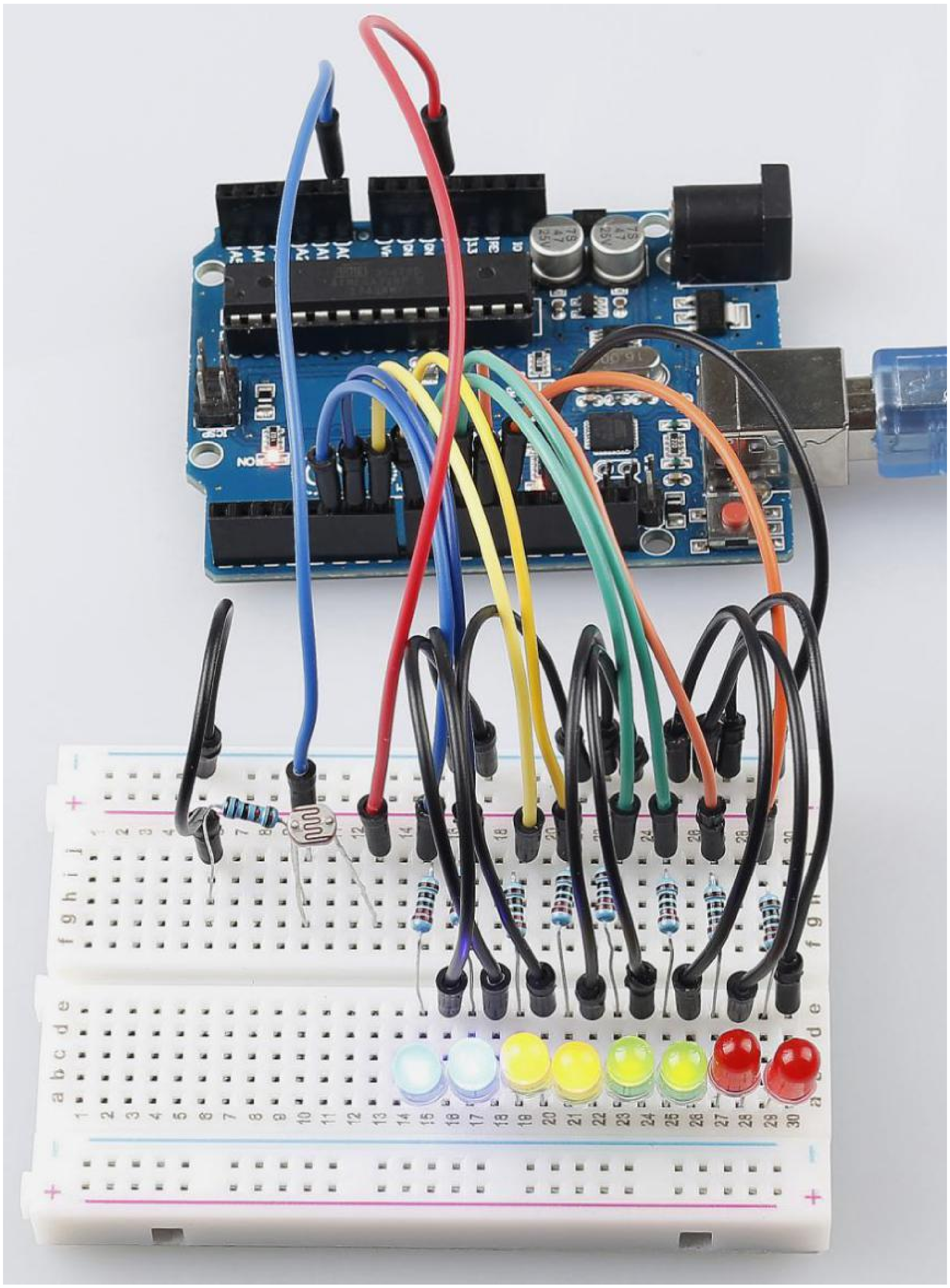
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, shine some light on the photo resistor, and you will see several LEDs light up. Shine more light and you will see more LEDs light up. When you place it in a dark environment, all the LEDs will go out.



Code

Code Analysis 5-1 Set the variables

```
const int NbrLEDs = 8;  // 8 leds

const int ledPins[] = {5, 6, 7, 8, 9, 10, 11, 12}; // 8 leds attach to pin 5-12␣
↪respectively

const int photocellPin = A0; // photoresistor attach to A0

int sensorValue = 0; // value read from the sensor

int ledLevel = 0; // sensor value converted into LED 'bars'
```

The 8 LEDs are connected to pin5-pin12, in this code, use a array to store the pins, ledPins[0] is equal to 5, ledPins[1] to 6 and so on.

Code Analysis 5-2 Set 8 pins to OUTPUT

```
for (int led = 0; led < NbrLEDs; led++)

{

    pinMode(ledPins[led], OUTPUT);// make all the LED pins outputs

}
```

Using the for() statement set the 8 pins to OUTPUT. The variable led is added from 0 to 8, and the pinMode() function sets pin5 to pin12 to OUTPUT in turn.

Code Analysis 5-3 Read the analog value of the photoresistor

```
sensorValue = analogRead(photocellPin); // read the value of A0
```

Read the analog value of the **photocellPin(A0)** and store to the variable **sensorValue.**

analogRead(): Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.

```
Serial.print("SensorValue: ");

Serial.println(sensorValue); // Print the analog value of the photoresistor
```

Use the Serial.print()function to print the analog value of the photoresistor. You can see them on the Serial Monitor.

Serial.print(): Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is.

Serial.println(): Thiscommand takes the same forms as Serial.print(), but it is followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').

Code Analysis 5-4 Map the analog value to 8 LEDs

```
ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs); // map to the number of LEDs

Serial.print("ledLevel: ");

Serial.println(ledLevel);
```

The map() command is used to map 0-1023 to 0-NbrLEDs(8), (1023-0)/(8-0)=127.875

0-127.875	128-255.75	256-383.6	384-511.5	512-639.4	640-767.25	768-895.1	896-1023
0	1	2	3	4	5	6	7

If sensorValue is 560, then the ledLevel is 4.

map(value, fromLow, fromHigh, toLow, toHigh) re-maps a number from one range to another. That is, a value of *fromLow* would get mapped to one of *toLow*, and a value of *fromHigh* to one of *toHigh*, values in-between to values in-between, etc.

Code Analysis 5-5 Light up the LEDs

```
for (int led = 0; led < NbrLEDs; led++)

{

    if (led <= ledLevel ) // When led is smaller than ledLevel, run the following␣
↪code.

    {

        digitalWrite(ledPins[led], HIGH); // turn on pins less than the level

    }

    else

    {

        digitalWrite(ledPins[led], LOW); // turn off pins higher than

    }

}
```

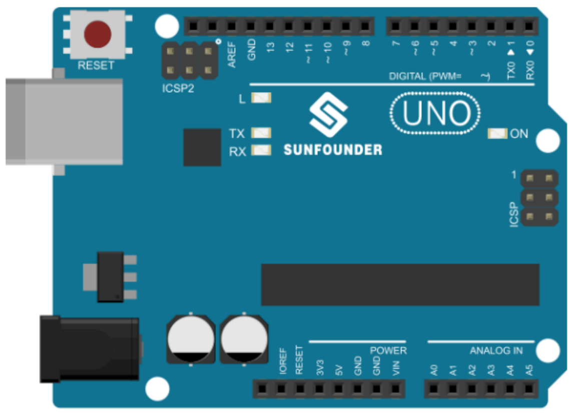
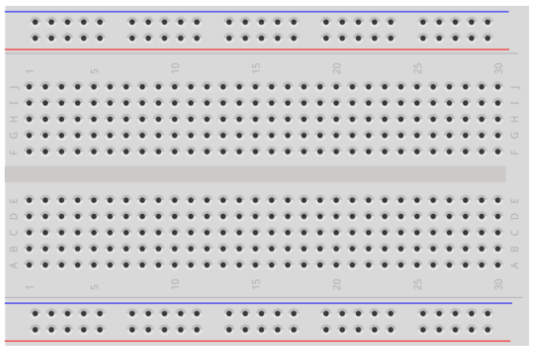
Light up the corresponding LEDs. Such as, when the ledLevel is 4, then light up the ledPins[0] to ledPins[4] and go out the ledPins[5] to ledPins[7].

Lesson 6 RGB LED

Introduction

Previously we've used the digital pin to control an LED brighten and dim. In this lesson, we will use PWM to control an RGB LED to flash various kinds of color. When different PWM values are set to the R, G, and B pins of the LED, its brightness will be different. When the three different colors are mixed, we can see that the RGB LED flashes different colors.

Components

1 * Uno Board 	1 * Breadboard 

1 * RGB LED	3 * Resistor (220Ω)
1 * USB Cable	Several Jumper Wires

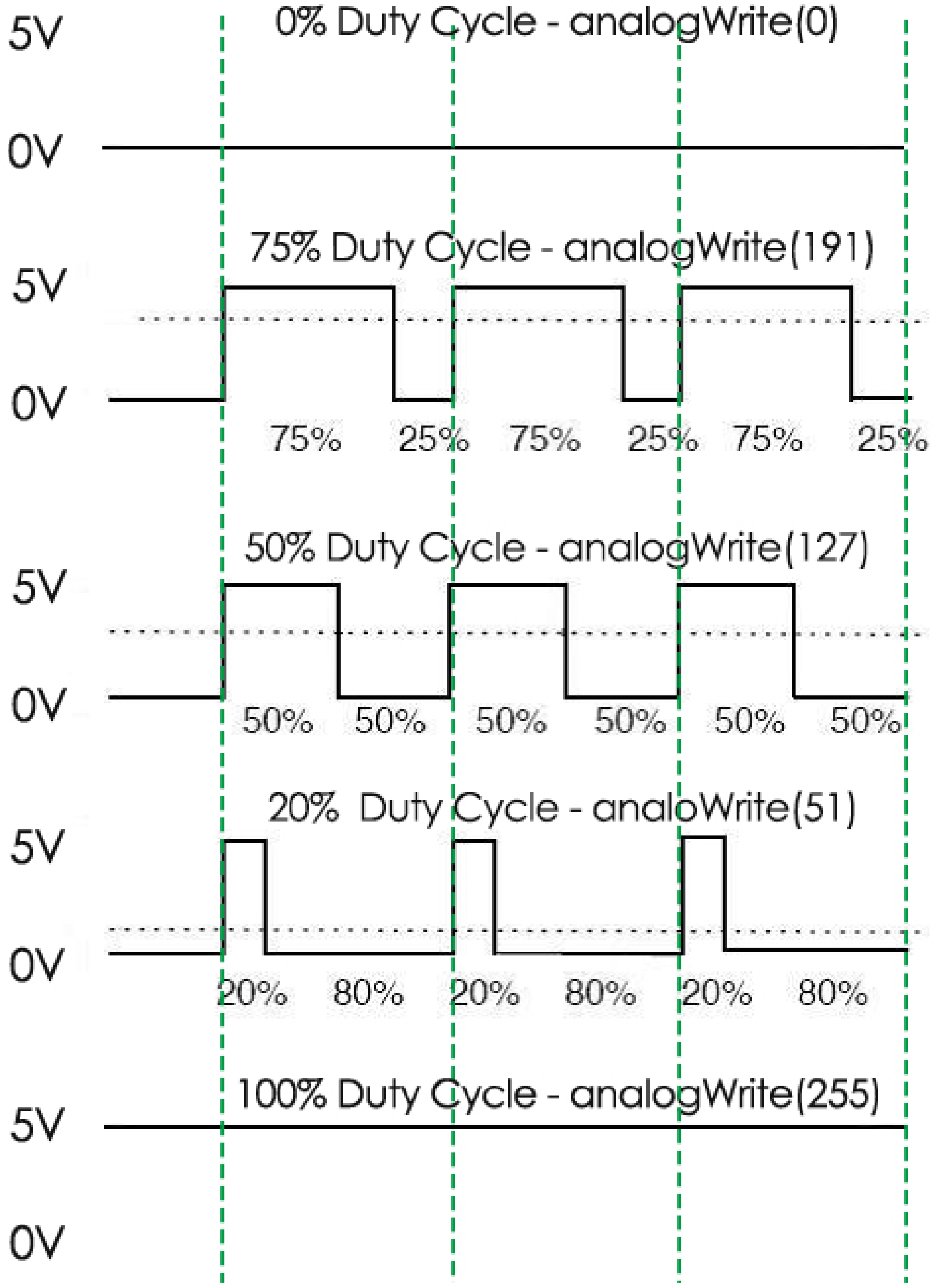
Experimental Principle

PWM

Pulse width modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called pulse width. To get varying analog values, you change, or modulate, that width. If you repeat this on-off pattern fast enough with some device, an LED for example, it would be like this:

the signal is a steady voltage between 0 and 5V controlling the brightness of the LED. (See the PWM description on the official website of Arduino).

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each.

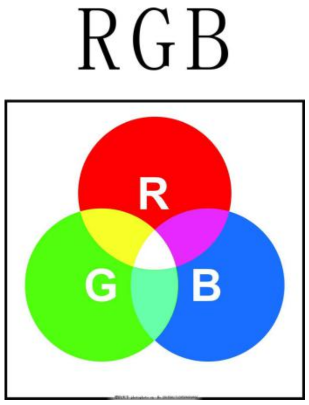

0% Duty Cycle - analogWrite(0)
5V
0V
75% Duty Cycle - analogWrite(191)
75% 25% 75% 25% 75% 25%
50% Duty Cycle - analogWrite(127)
50% 50% 50% 50% 50% 50%
20% Duty Cycle - analoWrite(51)
20% 80% 20% 80% 20% 80%
100% Duty Cycle - analogWrite(255)

A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.

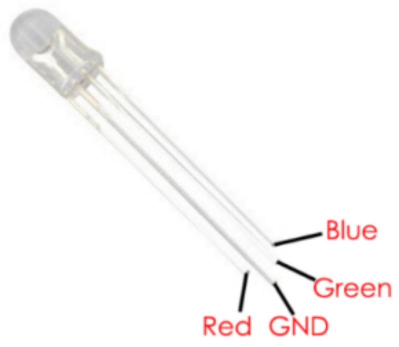
You will find that the smaller the PWM value is, the smaller the value will be after being converted into voltage. Then the LED becomes dimmer accordingly. Therefore, we can control the brightness of the LED by controlling the PWM value.

RGB LED

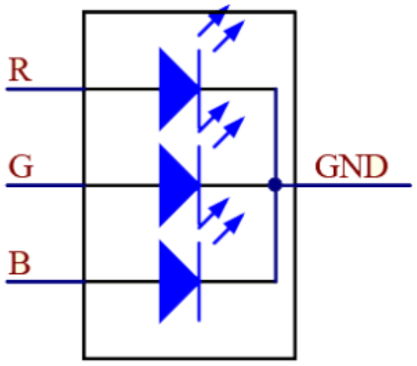
RGB LEDs emit light in various colors. An RGB LED packages three LEDs of red, green, and blue into a transparent or semitransparent plastic shell. It can display various colors by changing the input voltage of the three pins and superimpose them, which, according to statistics, can create 16,777,216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



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

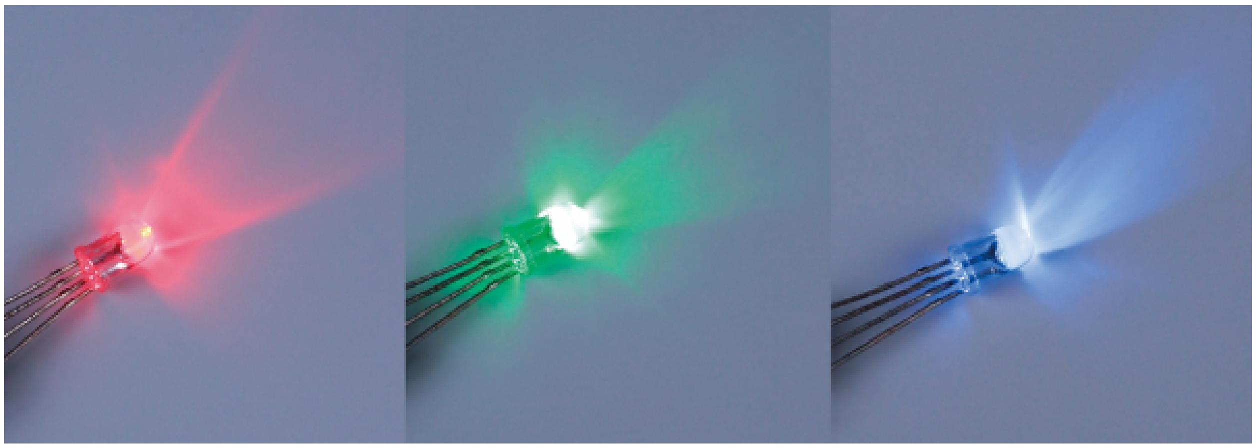


RGB LED



Circuit Symbol

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

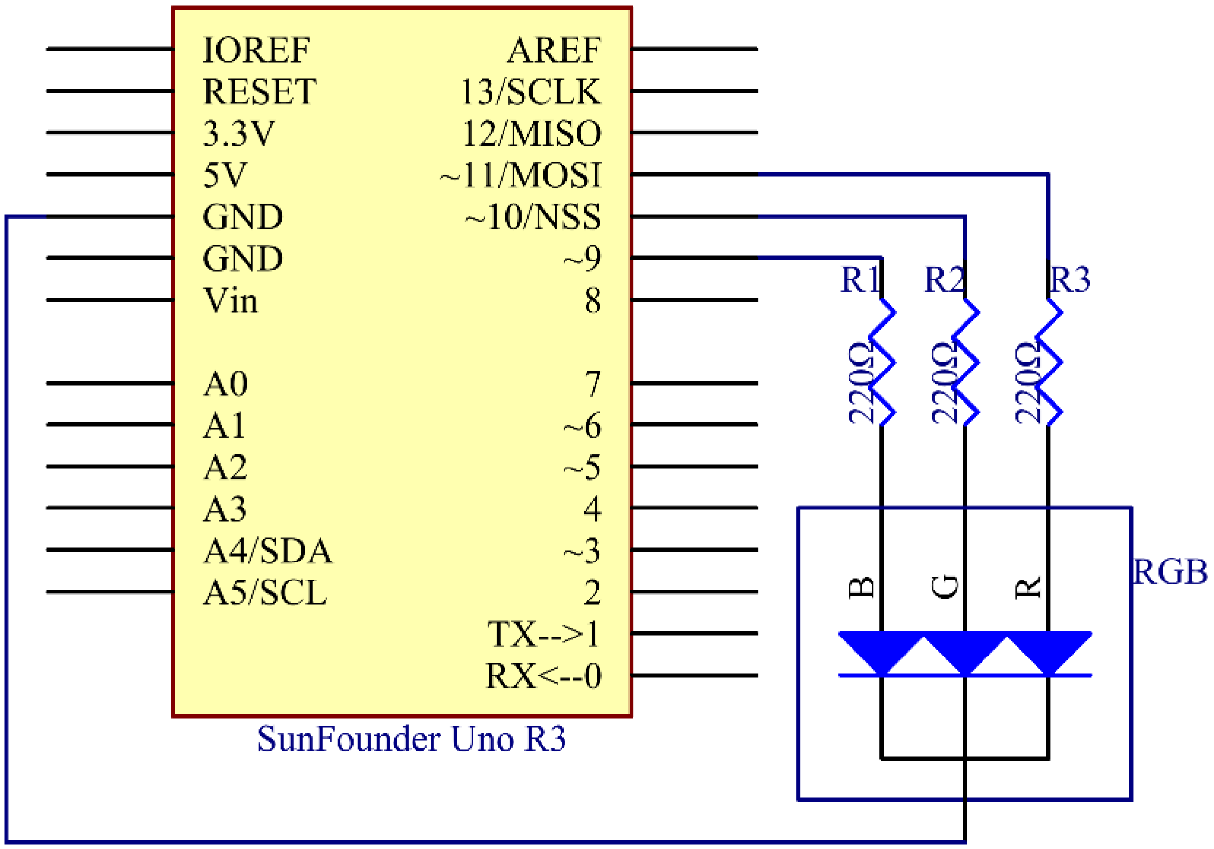


Or you can distinguish them in another way. As GND is the longest one and can be defined directly, you can test the other three pins by giving them a small voltage. In addition, you need to add the current limiting resistor to protect the component.

Principle:

On the Uno board, 3, 5, 6 and 9-11 is the PWM pins. Provide 8-bit PWM output with the analogWrite() function. You can connect any of these pins.Here we input a value between 0 and 255 to the three pins of the RGB LED to make it display different colors. After connecting the pins of R, G, and B to a current limiting resistor, connect them to the pin 9, pin 10, and pin 11 respectively. The longest pin (GND) of the LED connects to the GND of the Uno. When the three pins are given different PWM values, the RGB LED will display different colors.

The schematic diagram:



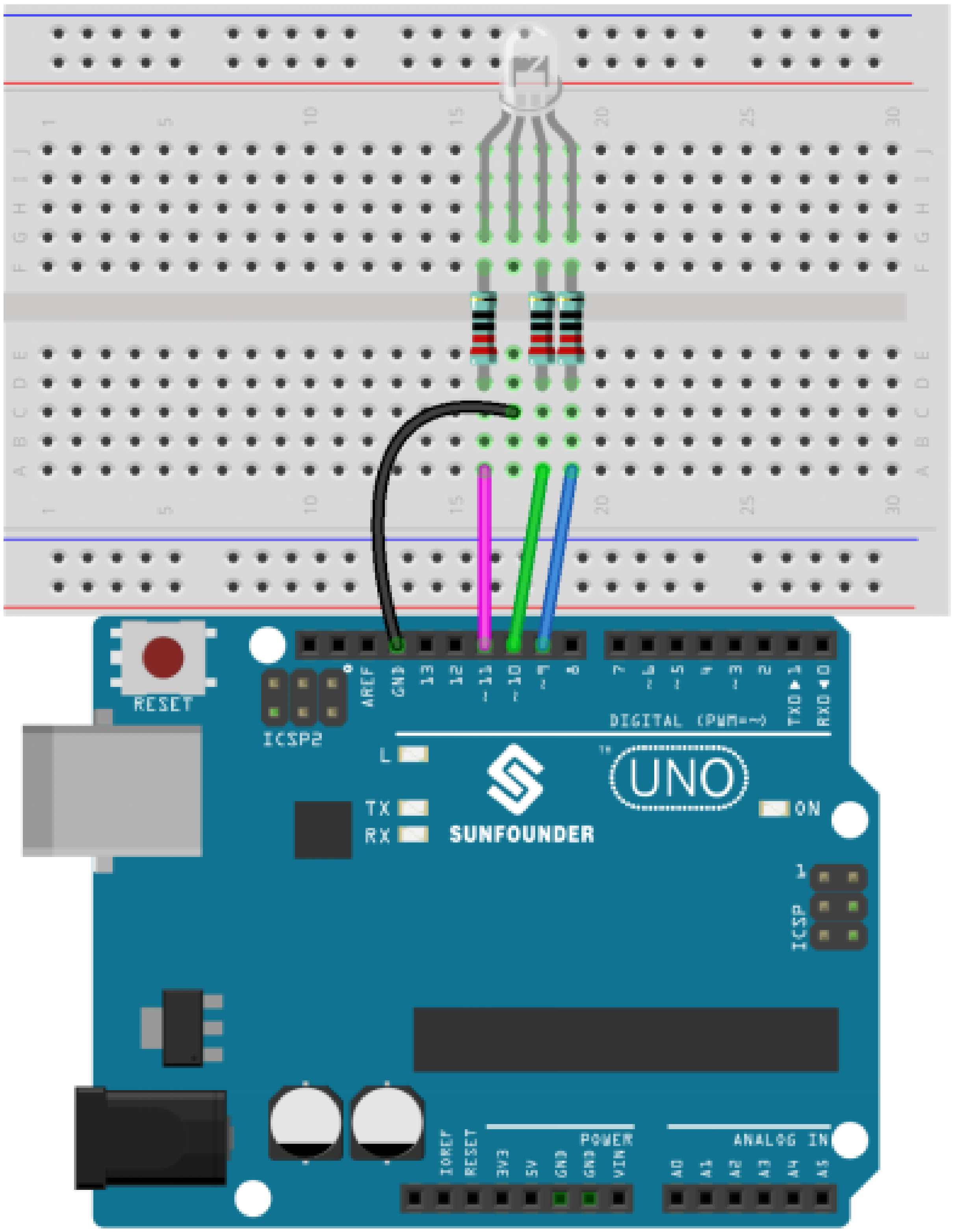
Experimental Procedures

Step 1: Build the circuit

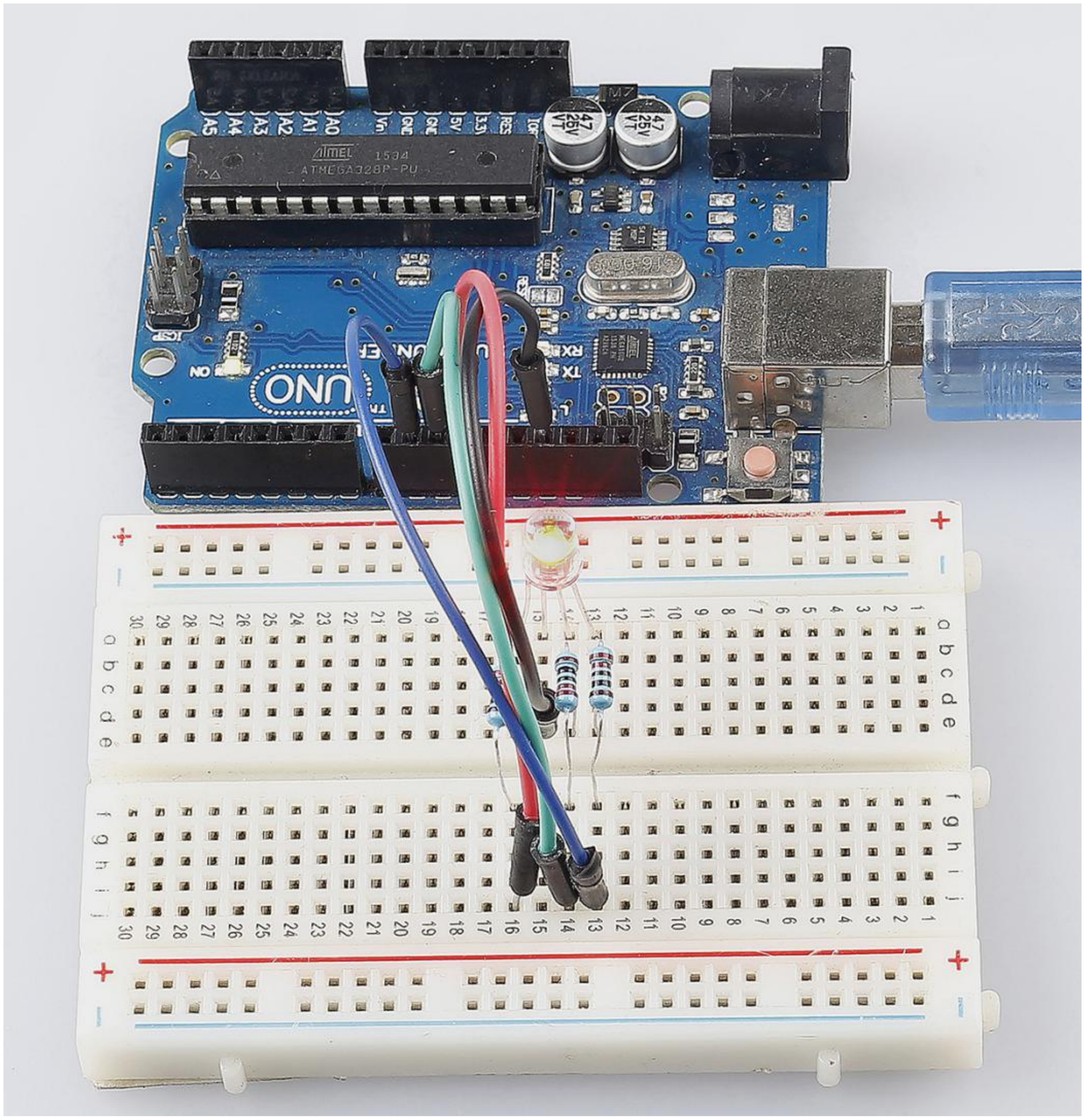
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Here you should see the RGB LED flash circularly red, green, and blue first, then red, orange, yellow, green, blue, indigo, and purple.



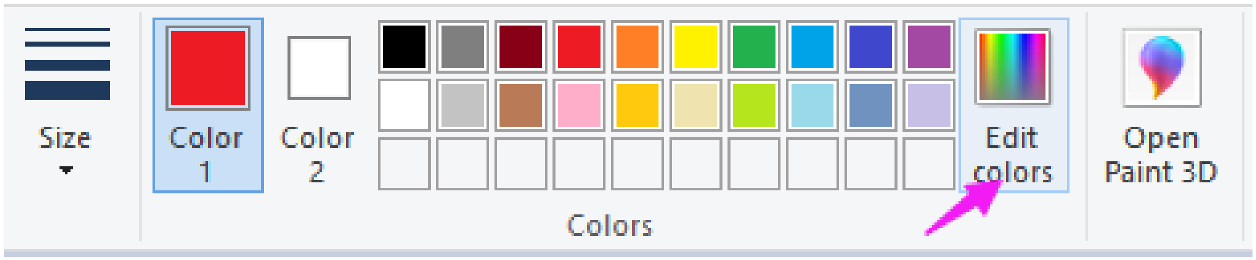
Code

Code Analysis 6-1 Set the color

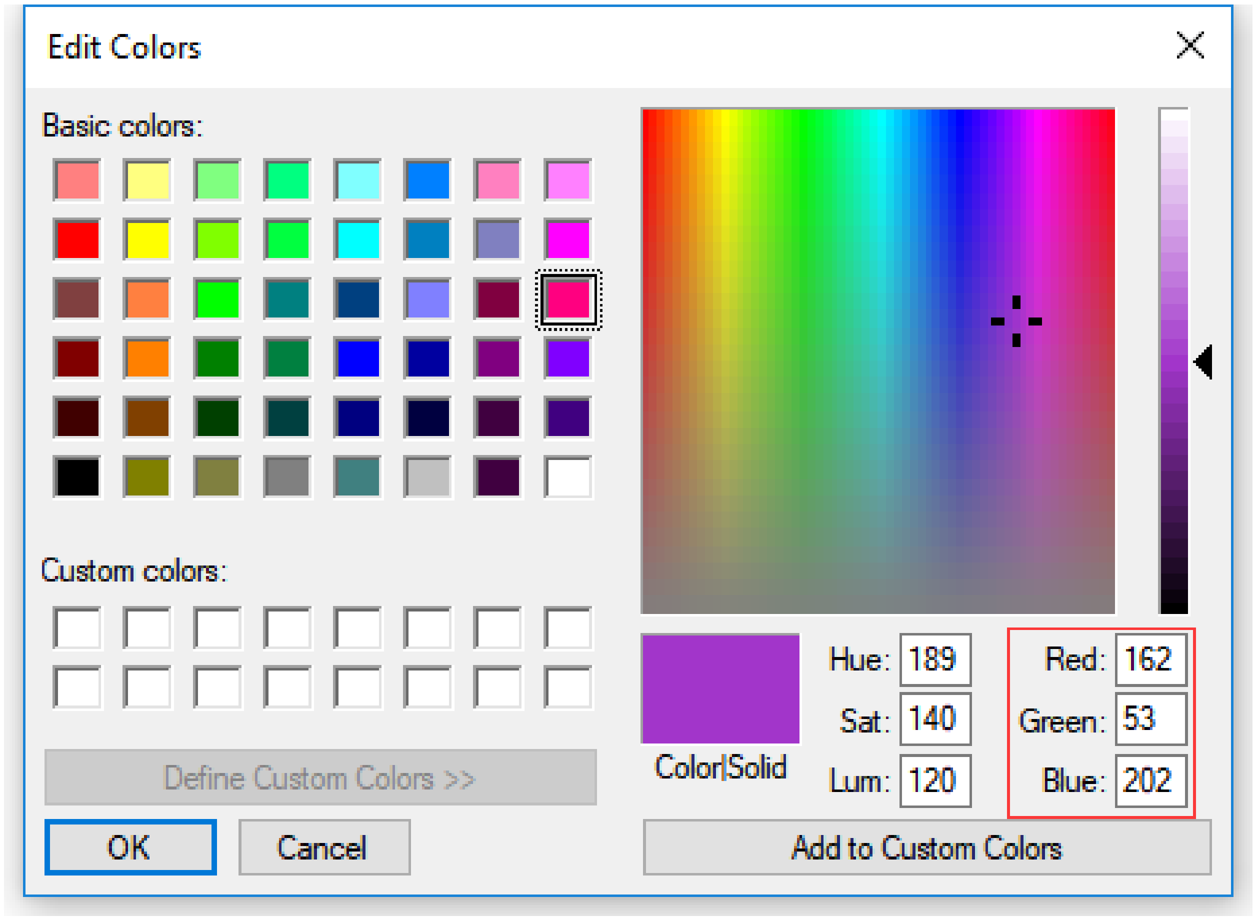
Here use the *color()* function to set the color of the RGB LED. In the code, it is set to flash 7 different colors.

You can use the paint tool on your computer to get the RGB value.

1) Open the paint tool on your computer and click to Edit colors.



2) Select one color, then you can see the RGB value of this color. Fill them in the code.



Hue: 189
Sat: 140
Lum: 120
Red: 162
Green: 53
Blue: 202

```
void loop() // run over and over again

{

    // Basic colors:

    color(255, 0, 0);   // turn the RGB LED red

    delay(1000);        // delay for 1 second

    color(0,255, 0);   // turn the RGB LED green

    delay(1000);        // delay for 1 second
```

(continues on next page)

(continued from previous page)

```
    color(0, 0, 255);  // turn the RGB LED blue

    delay(1000);         // delay for 1 second

    // Example blended colors:

    color(255,0,252); // turn the RGB LED red

    delay(1000);         // delay for 1 second

    color(237,109,0); // turn the RGB LED orange

    delay(1000);         // delay for 1 second

    color(255,215,0); // turn the RGB LED yellow

    delay(1000);         // delay for 1 second

    color(34,139,34); // turn the RGB LED green

    delay(1000);         // delay for 1 second

    color(0,112,255); // turn the RGB LED blue

    delay(1000);         // delay for 1 second

    color(0,46,90);      // turn the RGB LED indigo

    delay(1000);         // delay for 1 second

    color(128,0,128); // turn the RGB LED purple

    delay(1000);         // delay for 1 second

}
```

Code Analysis 6-2 color() function

```
void color (unsigned char red, unsigned char green, unsigned char blue)
// the color generating function

{

    analogWrite(redPin, red);

    analogWrite(greenPin, green);

    analogWrite(bluePin, blue);

}
```

Define three unsigned char variables, i.e., red, green and blue. Write their values to *redPin*, *greenPin* and *bluePin*. For example, color(128,0,128) is to write 128 to *redPin*, 0 to *greenPin* and 128 to *bluePin*. Then the result is the LED flashing purple.

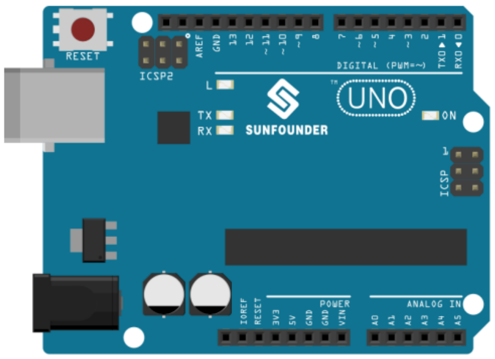
analogWrite(): Writes an analog value (PWM wave) to a pin. It has nothing to do with an analog pin, but is just for PWM pins. You do not need to call the *pinMode()* to set the pin as output before calling *analogWrite()*.

Lesson 7 Tilt Switch

Introduction

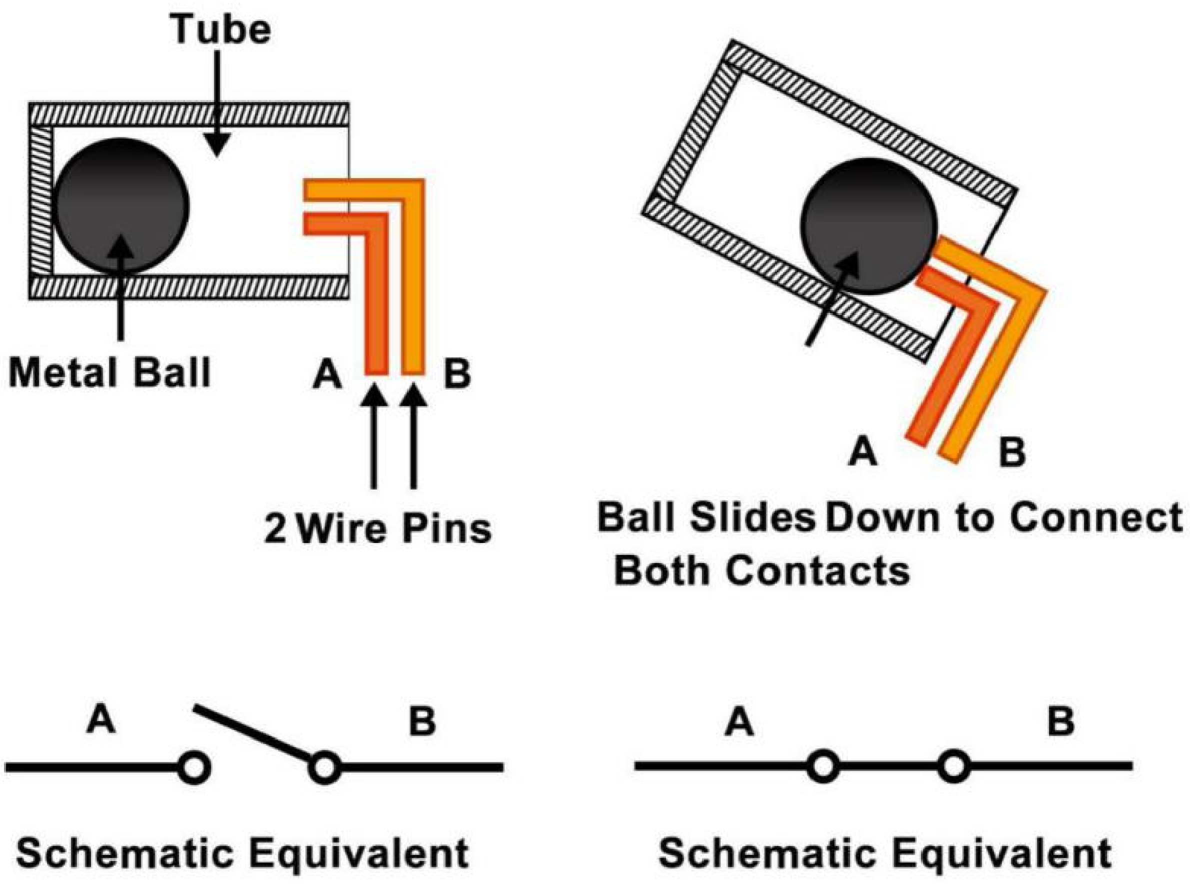
The tilt switch used here is a ball one with a metal ball inside. It is used to detect inclinations of a small angle.

Components

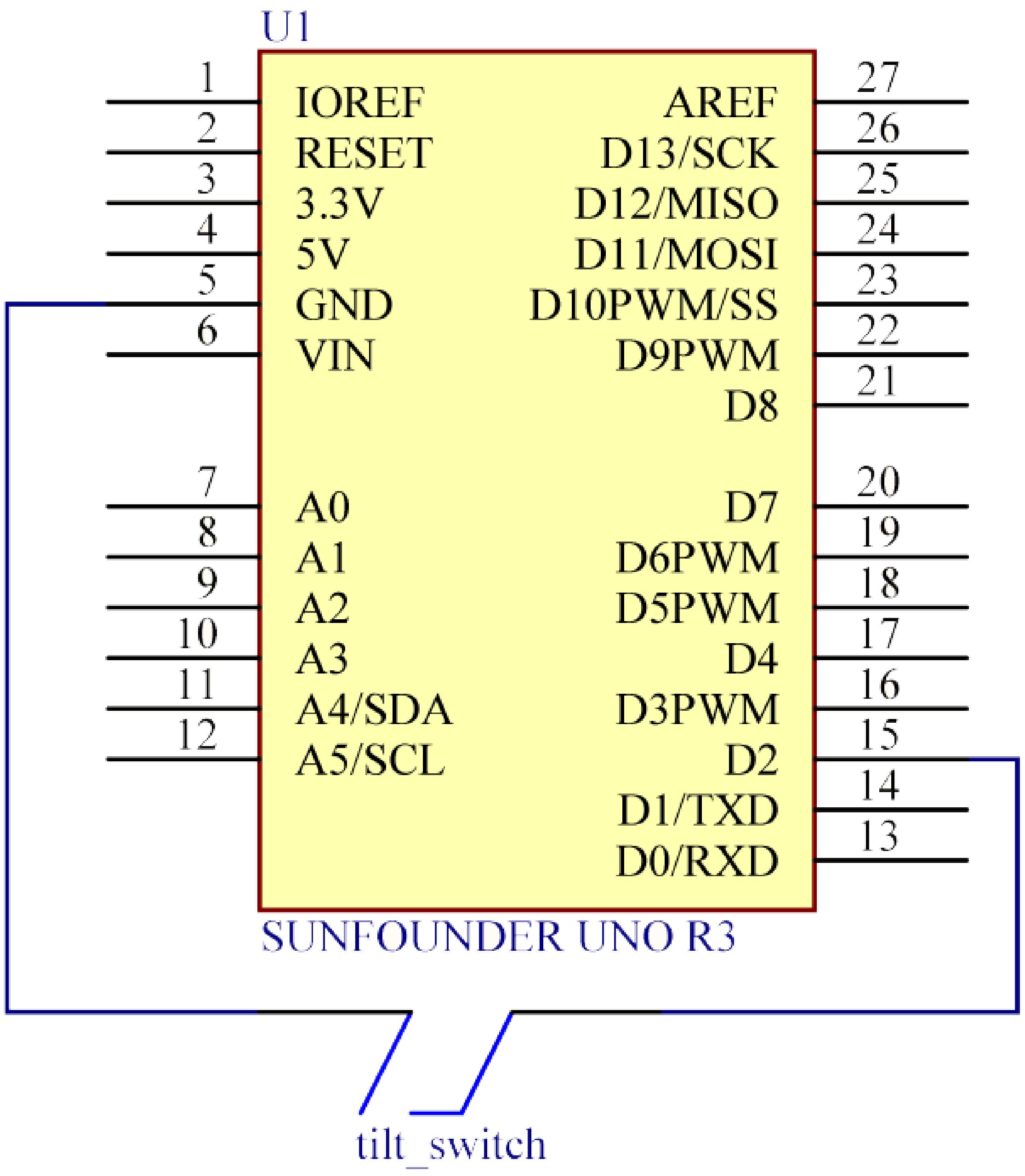
1 * Uno Board 	1*Tilt Switch
1 * USB Cable	2 * Dupont wires

Experimental Principle

The principle is very simple. When the switch is tilted in a certain angle, the ball inside rolls down and touches the two contacts connected to the pins outside, thus triggering circuits. Otherwise the ball will stay away from the contacts, thus breaking the circuits.

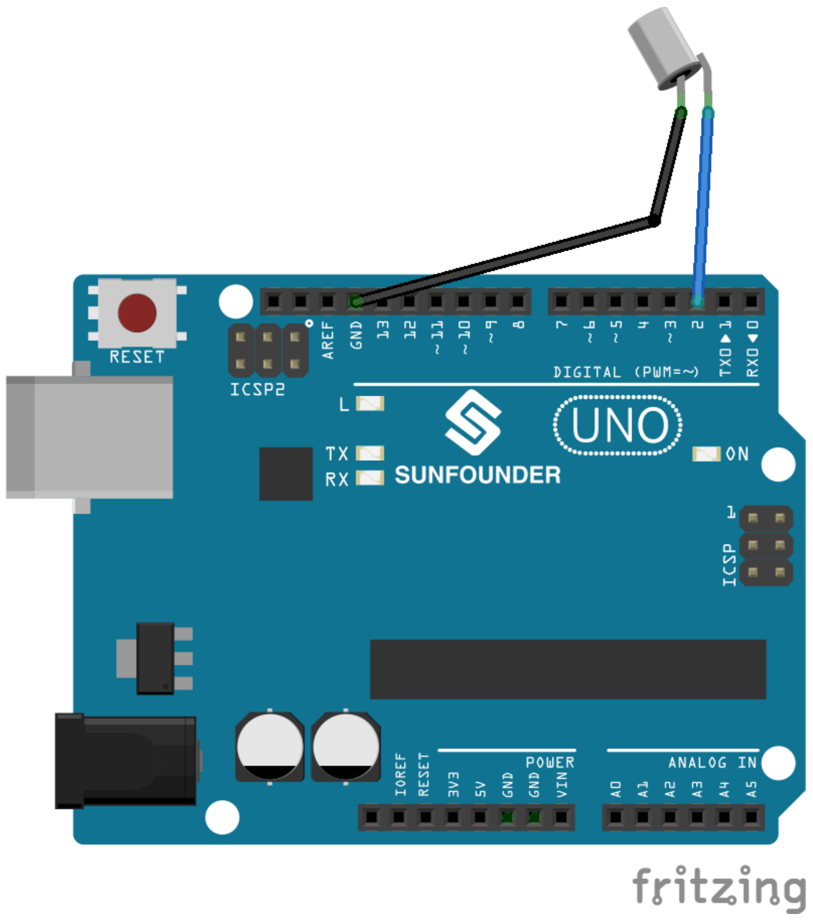


The schematic diagram:


U1
1
2
3
4
5
6
7
8
9
10
11
12
IOREF
RESET
3.3V
5V
GND
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
AREF
D13/SCK
D12/MISO
D11/MOSI
D10PWM/SS
D9PWM
D8
D7
D6PWM
D5PWM
D4
D3PWM
D2
D1/TXD
D0/RXD
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
SUNFOUNDER UNO R3
tilt_switch

Experimental Procedures

Step 1: Build the circuit

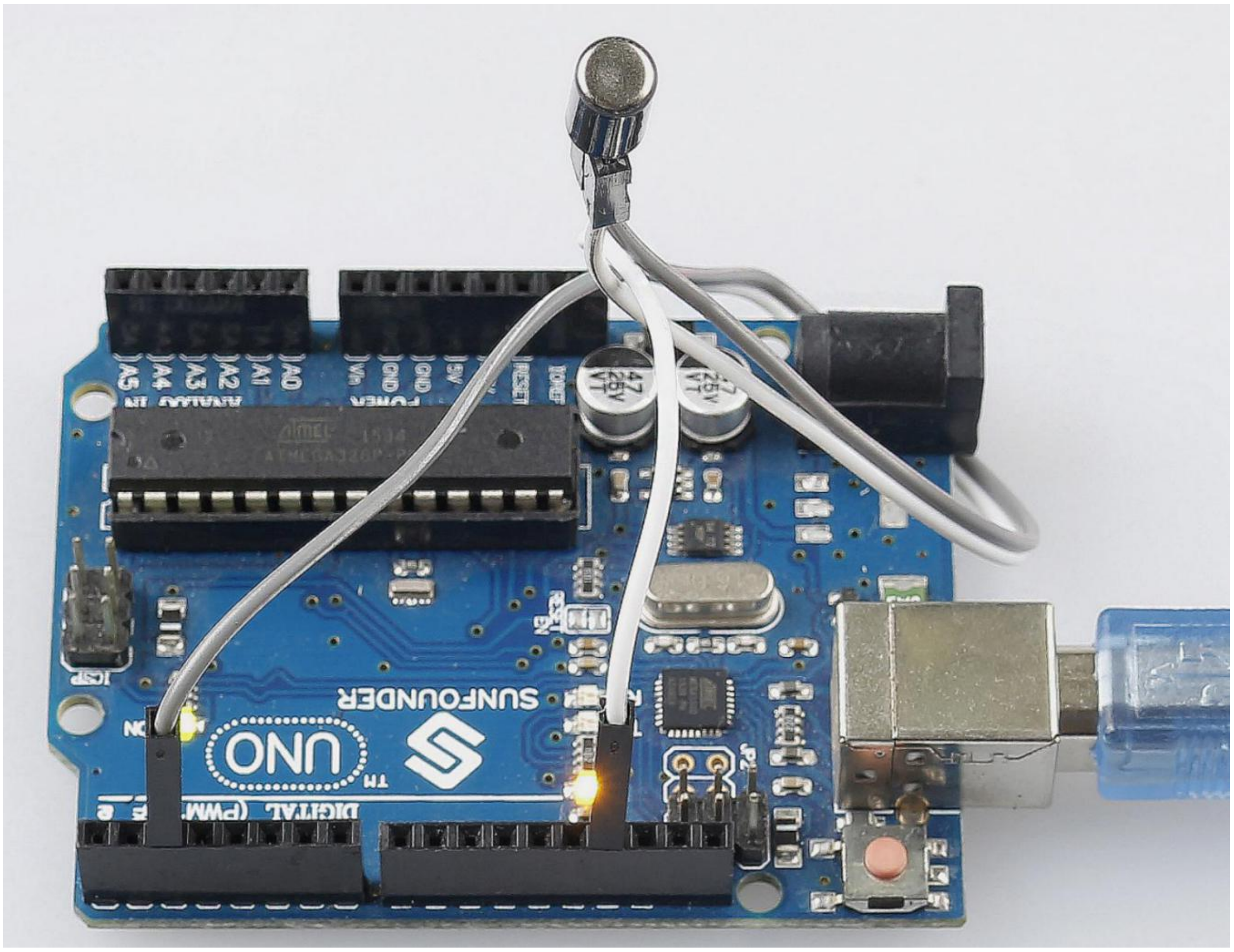


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, tilt the switch, and the LED attached to pin 13 on Uno board will light up.



Code

Code Analysis 7-1 Whole Code

```
const int ledPin = 13;      // the led attach to

void setup()

{

    pinMode(ledPin,OUTPUT); // initialize the ledPin as an output

    pinMode(2,INPUT);       // set pin2 as INPUT

    digitalWrite(2, HIGH);  // set pin2 as HIGH

}

/*******************************************/

void loop()
```

(continues on next page)

(continued from previous page)

```
{

    int digitalVal = digitalRead(2); // Read the value of pin2

    if(HIGH == digitalVal)           // if tilt switch is not breakover

    {

        digitalWrite(ledPin,LOW);    // turn the led off

    }

    else //if tilt switch breakover

    {

        digitalWrite(ledPin,HIGH);   // turn the led on

    }

}
```

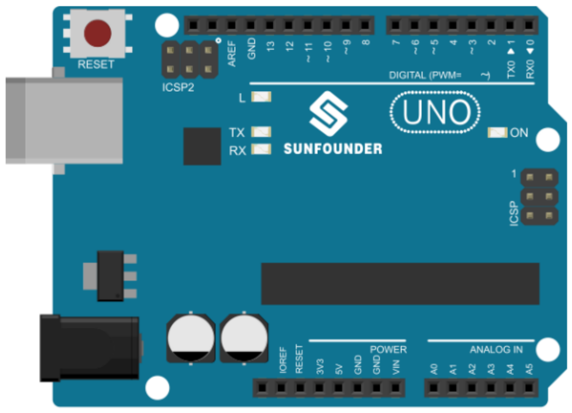
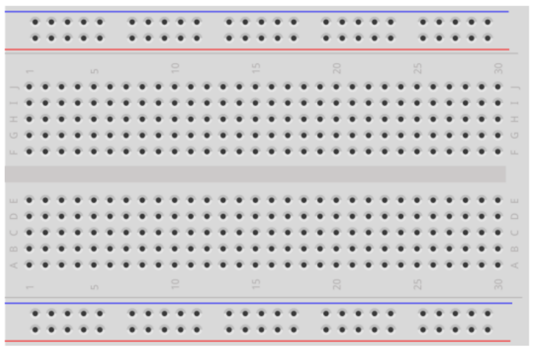
The whole code are very simple, one pin of the tilt switch is connected to pin2, another pin is connected to GND, when tilt the switch, the two pins of the switch will be connected to GND, then let the LED on the pin13 lights up.

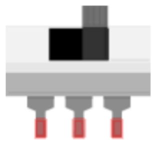
Lesson 8 Slide Switch

Introduction

In this lesson, we are going to use a slide switch to turn on/off an external LED. The slide switch is a device to connect or disconnect the circuit by sliding its handle. They are quite common in our surroundings. Now let's see how it works.

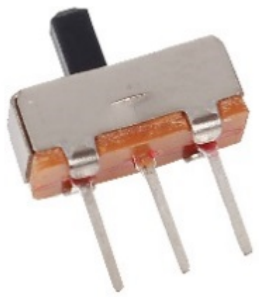
Components

1 * Uno Board	1 * Breadboard
	

1 * Resistor (220Ω)	1 * LED	1 *Slide Switch	1 * Capacitor 104
			
1*Resistor(10kΩ)			
1 * USB Cable		Several Jumper Wires	

Experimental Principle

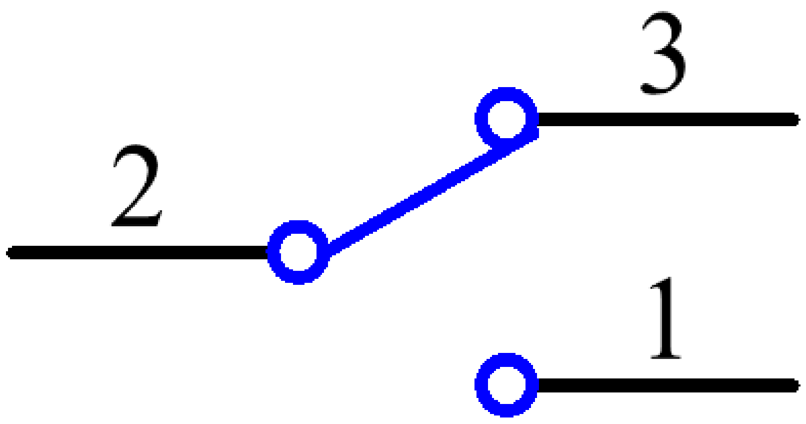
Slide Switch

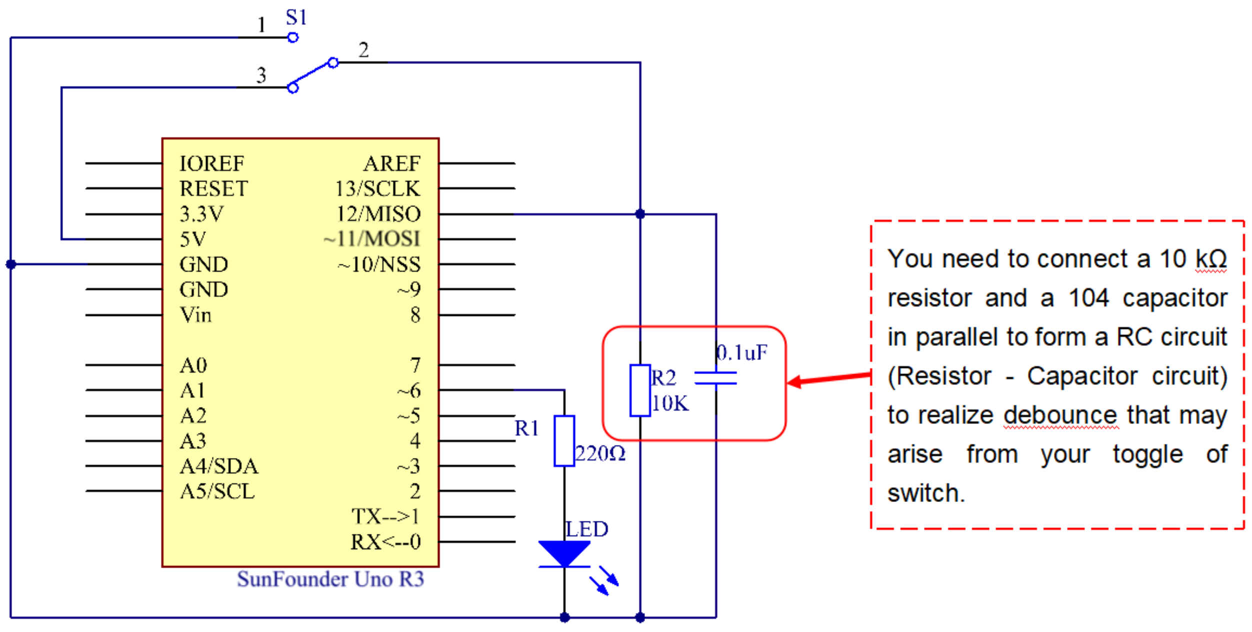


Just as its name suggests, slide switch is to connect or disconnect the circuit by sliding its switch handle so as to switch the circuit. The common types of slide switch include single pole double throw, single pole triple throw, double pole double throw, and double pole triple throw and so on. Generally, it is used in circuits with a low voltage and features flexibility and stabilization. Slide switches are commonly used in all kinds of instruments/meters equipment, electronic toys and other fields related.

How it works: The middle pin is fixed. When the handle is pushed to the left, the left two pins are connected; push it to the right, the two pins on the right connect, thus switching circuits.

See the circuit symbol for slide switch and 2 is the middle pin.





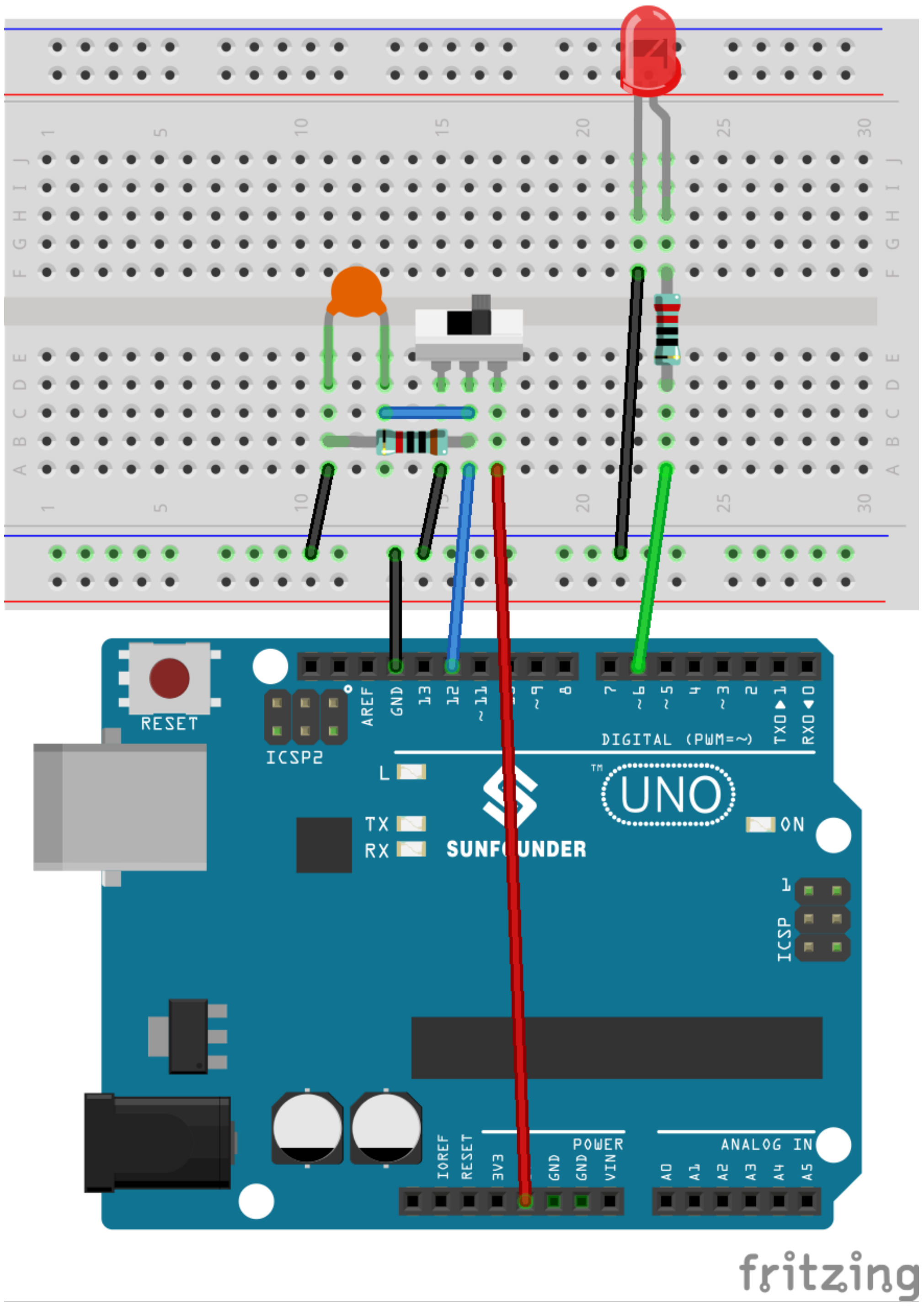
You need to connect a 10 kΩ resistor and a 104 capacitor in parallel to form a RC circuit (Resistor - Capacitor circuit) to realize debounce that may arise from your toggle of switch.

Principle:

Here we use a slide switch to control the on/off of an LED which is simple. Connect the middle pin of the switch to pin 12. Connect one pin at one end to VCC. After connecting a 10K resistor and a 104 capacitor, connect the last one pin to GND (to let the switch output stable level signal). Connect an LED to pin 6. Push the handle of the slide switch to the pin connected with pin 12 which is High level, we can light up the LED at pin 6 by programming.

Experimental Procedures

Step 1: Build the circuit

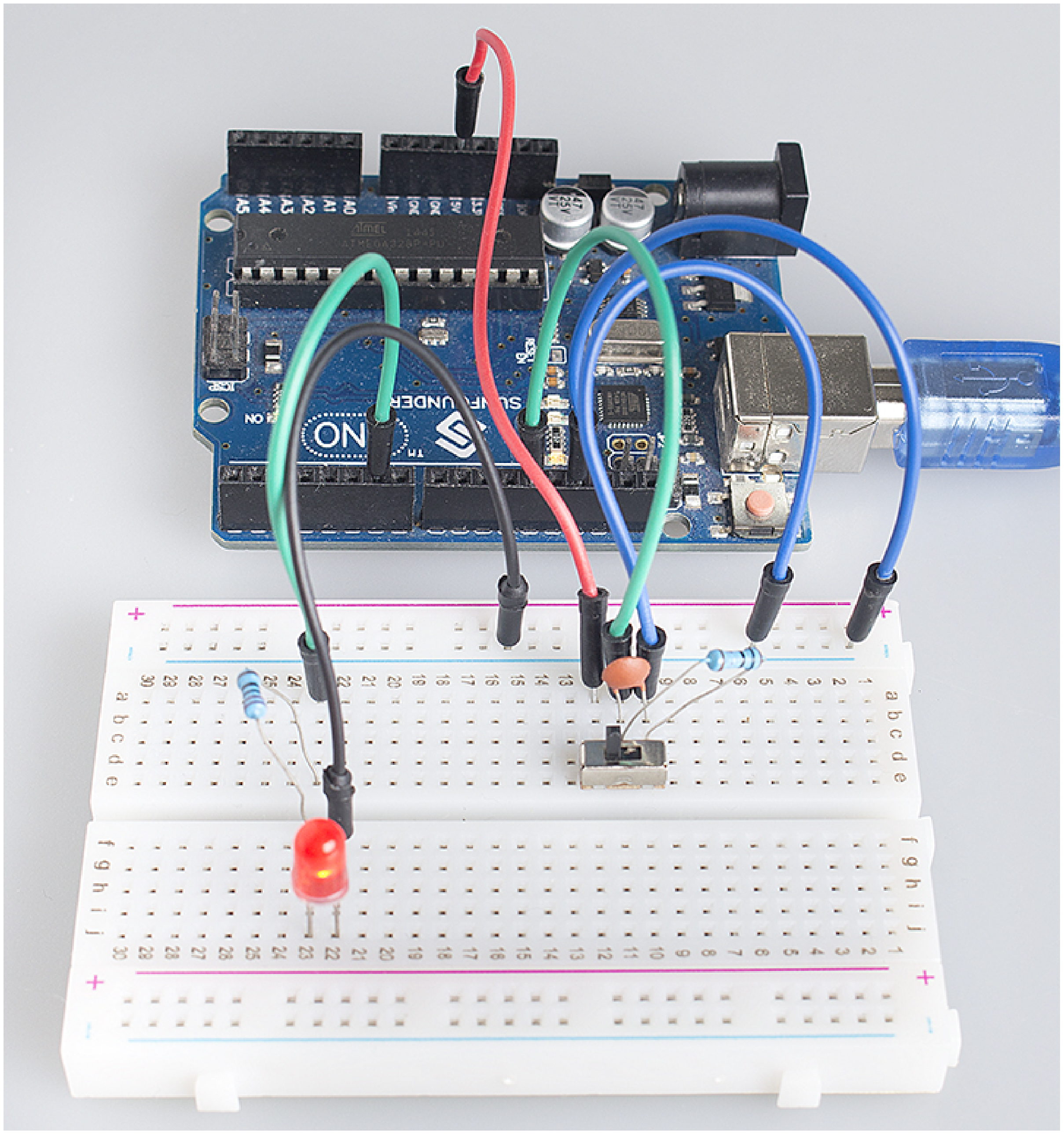


Step 2: Open the code file

Step 3: Select correct Board and Port

Step 4: Upload the sketch to the SunFounder Uno board

When you toggle the switch to pin12, the LED lights.



Code

Code Analysis 8-1 Read the switch state to turn on/off the LED

```
void loop()

{

    //read the state of the switch value

    switchState = digitalRead(switchPin);
```

(continues on next page)

(continued from previous page)

```
    if (switchState == HIGH )        //if it is,the state is HIGH

    {

        digitalWrite(ledPin, HIGH); //turn the led on

    }

    else

    {

        digitalWrite(ledPin, LOW);  //turn the led off

    }

}
```

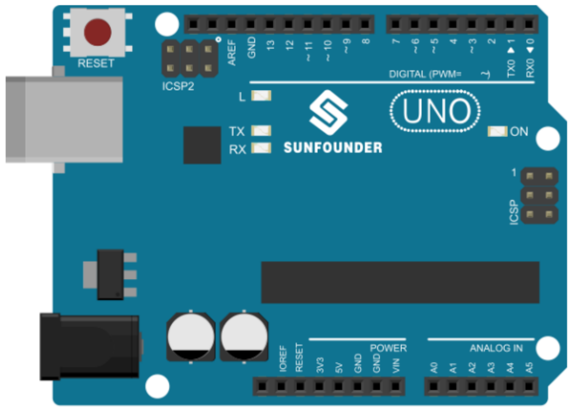
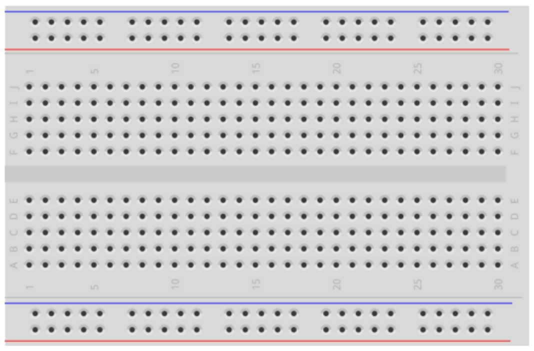
First, read the state of the *switchPin* and see whether you have moved the switch handle. If it has been pushed to pin 12, then the *switchState* is High level, so set *ledPin* as High level, which means to light up the LED; otherwise, to turn it off.

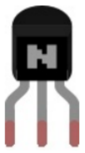
Lesson 9 Relay

Introduction

As we may know, relay is a device which is used to provide connection between two or more points or devices in response to the input signal applied. In other words, relays provide isolation between the controller and the device as devices may work on AC as well as on DC. However, they receive signals from a microcontroller which works on DC hence requiring a relay to bridge the gap. Relay is extremely useful when you need to control a large amount of current or voltage with small electrical signal.

Components

1 * Uno Board 	1 * Breadboard 

1*Resistor(1kΩ)	1 * Resistor (220Ω)	1 * Diode1N4007 (Rectifier)
1 * LED	1 * NPN S8050 	1* Relay

1 * USB Cable	Several Jumper Wires

Experimental Principle

Relay

There are 5 parts in every relay:

1. **Electromagnet** – It consists of an iron core wounded by coil of wires. When electricity is passed through, it becomes magnetic. Therefore, it is called electromagnet.

2. **Armature** – The movable magnetic strip is known as armature. When current flows through them, the coil is it energized thus producing a magnetic field which is used to make or break the normally open (N/O) or normally close (N/C) points. And the armature can be moved with direct current (DC) as well as alternating current (AC).

3. **Spring** – When no currents flow through the coil on the electromagnet, the spring pulls the armature away so the circuit cannot be completed.

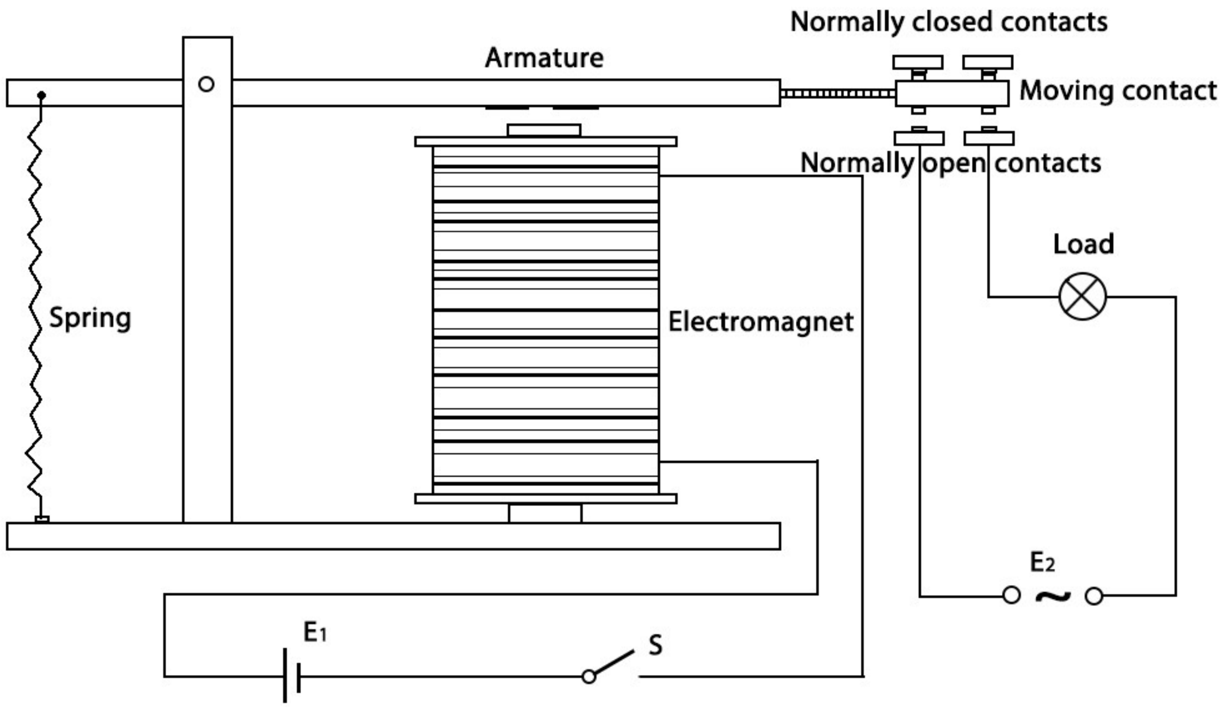
4. Set of electrical **contacts** – There are two contact points:

- Normally open – connected when the relay is activated, and disconnected when it is inactive.
- Normally close – not connected when the relay is activated, and connected when it is inactive.

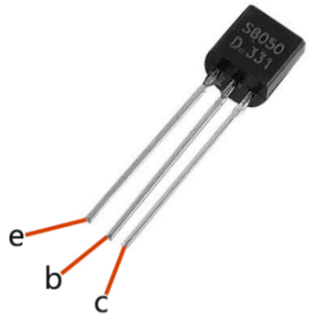
5. **Molded frame** – Relays are covered with plastic for protection.

Working of Relay

The working principle of relay is simple. When power is supplied to the relay, currents start flowing through the control coil; as a result, the electromagnet starts energizing. Then the armature is attracted to the coil, pulling down the moving contact together thus connecting with the normally open contacts. So the circuit with the load is energized. Then breaking the circuit would a similar case, as the moving contact will be pulled up to the normally closed contacts under the force of the spring. In this way, the switching on and off of the relay can control the state of a load circuit.

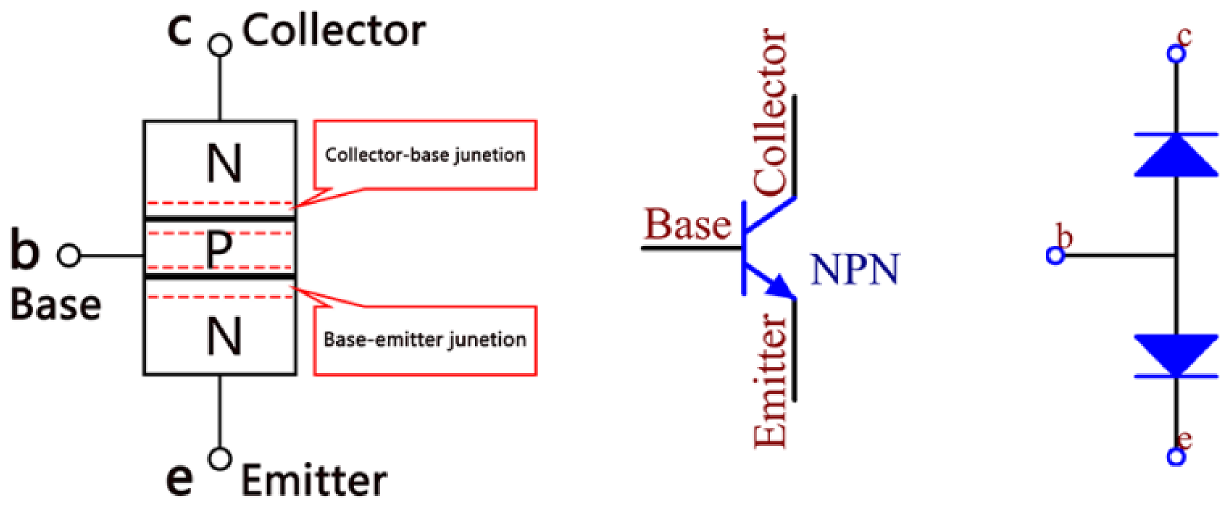


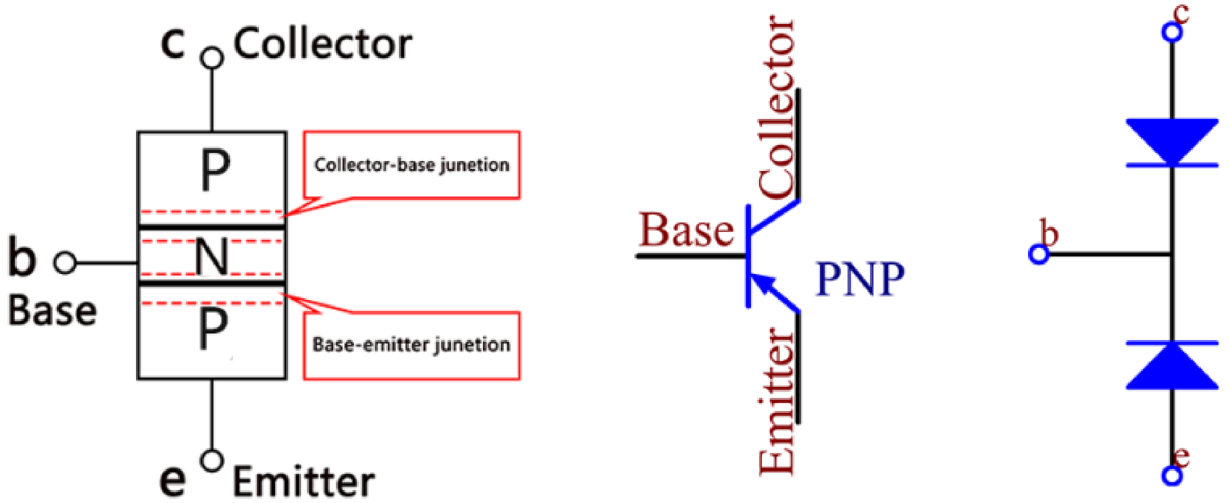
Transistor



Transistor is a semiconductor device that controls current by current. It functions by amplifying weak signal to larger amplitude signal and is also used for non-contact switch. A transistor is a three-layer structure composed of P-type and N-type semiconductors. They form the three regions internally. The thinner in the middle is the base region; the other two are both N-type or P-type ones – the smaller region with intense majority carriers is the emitter region, when the other one is the collector region. This composition enables the transistor to be an amplifier.

From these three regions, three poles are generated respectively, which are base (b), emitter (e), and collector (c). They form two P-N junctions, namely, the emitter junction and collection junction. The direction of the arrow in the transistor circuit symbol indicates that of the emitter junction. Based on the semiconductor type, transistors can be divided into two groups, the NPN and PNP ones. From the abbreviation, we can tell that the former is made of two N-type semiconductors and one P-type and that the latter is the opposite. See the figure below.





When a High level signal goes through an NPN transistor, it is energized. But a PNP one needs a Low level signal to manage it. Both types of transistor are frequently used for contactless switches, just like in this experiment.

Principle:

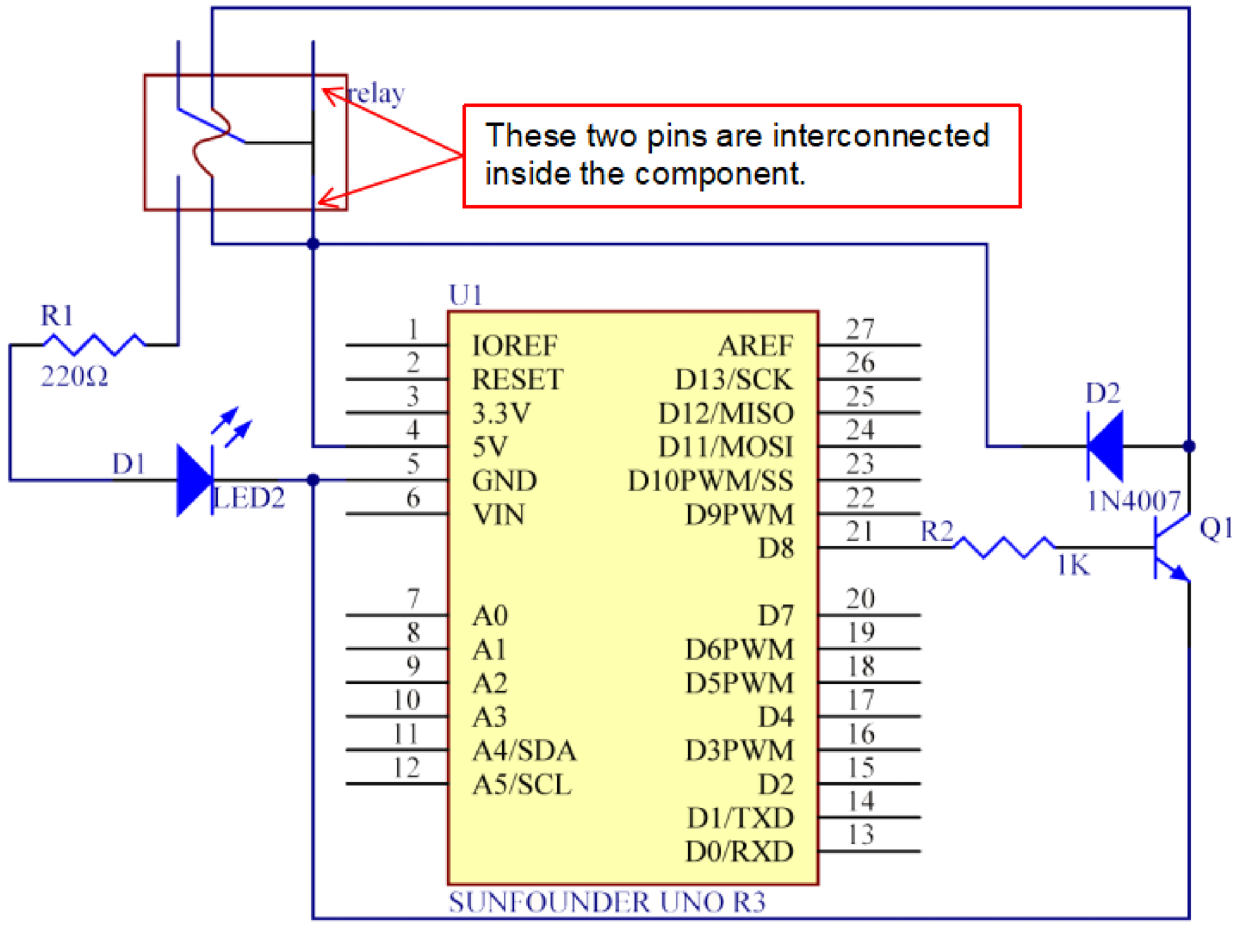
Connect a 1K resistor (for current limiting when the transistor is energized) to pin 8 of the SunFounder Uno board, then to an NPN transistor whose collector is connected to the coil of a relay and emitter to GND; connect the normally open contact of the relay to an LED and then GND. Therefore, when a High level signal is given to pin 8, the transistor is energized, thus making the coil of the relay conductive. Then its normally open contact is closed, and the LED will light up. When pin 8 is given a Low level, the LED will stay dim.

Function of the freewheeling diode:

When the voltage input changes from High (5V) to Low (0V), the transistor changes from saturation (three working conditions: amplification, saturation, and cut-off) to cut-off, the current in the coil suddenly has no way to flow through. At this moment, without the freewheeling diode, a counter-electromotive force (EMF) will be generated at the ends of the coil, with positive at the bottom and negative at the top, a voltage higher than 100V. This voltage plus that from the power at the transistor are big enough to burn it. Therefore, the freewheeling diode is extremely important in discharging this counter-EMF in the direction of the arrow in the figure above, so the voltage of the transistor to GND is no higher than +5V (+0.7V).

In this experiment, when the relay closes, the LED will light up; when the relay opens, the LED will go out.

The schematic diagram:



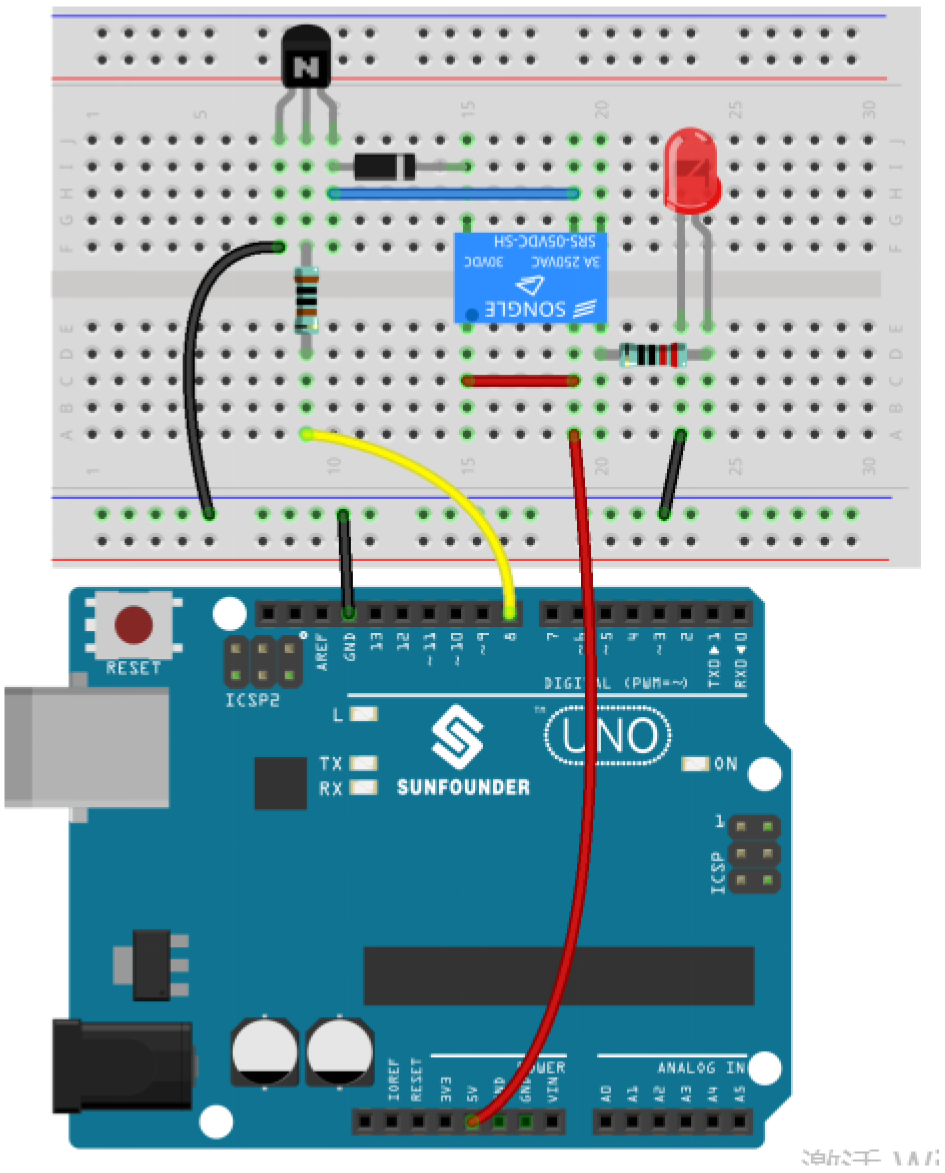
Experimental Procedures

Step 1: Build the circuit

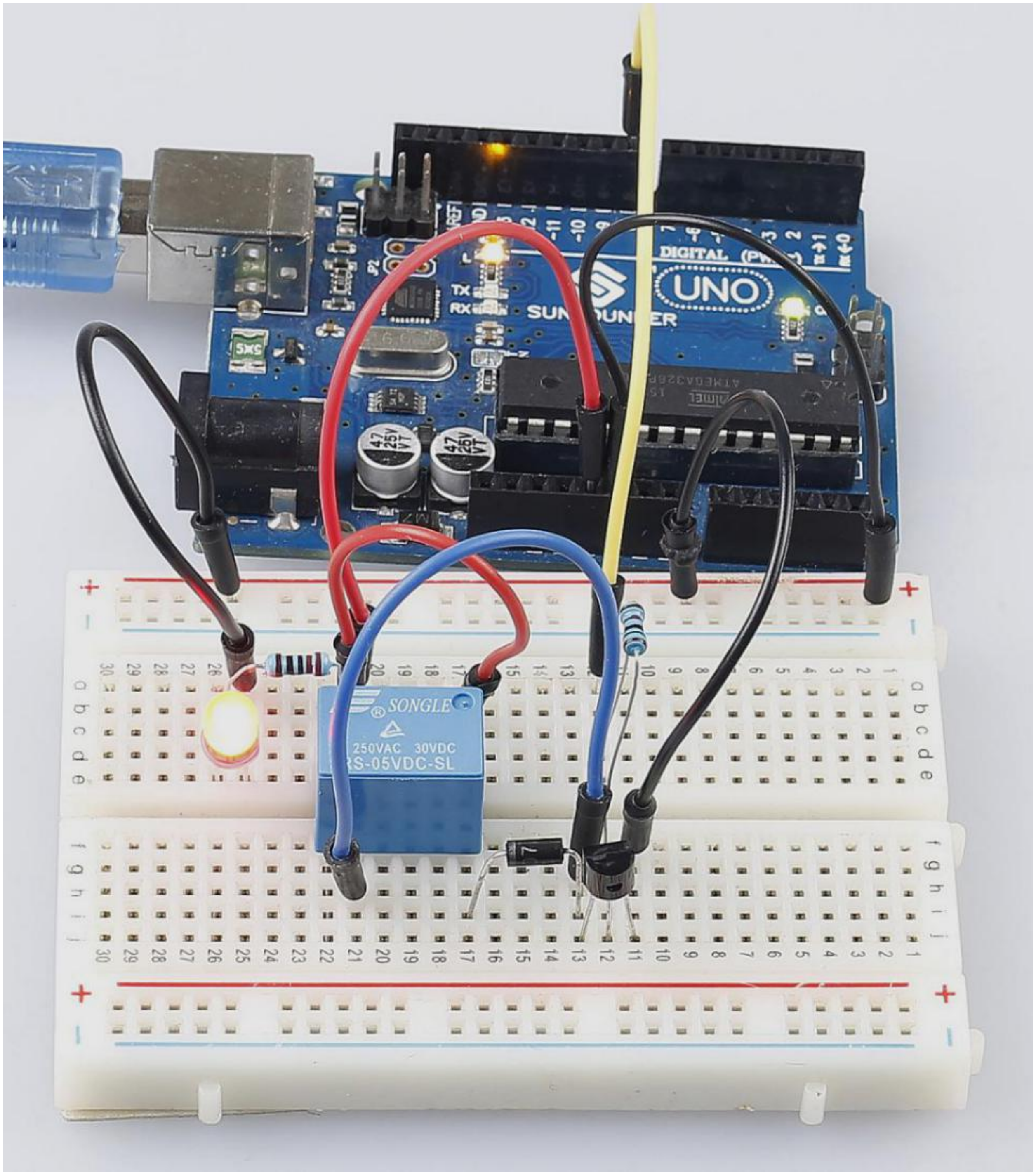
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, send a High level signal, and the relay will close and the LED will light up; send a low one, and it will open and the LED will go out. In addition, you can hear a tick-tock caused by breaking the normally close contact and closing the normally open one.


SONGLE
250VAC 30VDC
SRS-05VDC-SL
UNO
SUNFOUNDER
DIGITAL

Code

```
void loop()

{

    digitalWrite(relayPin, HIGH); //drive relay closure conduction

    delay(1000);                  //wait for a second

    digitalWrite(relayPin, LOW);  //drive the relay is closed off

    delay(1000);                  //wait for a second

}
```

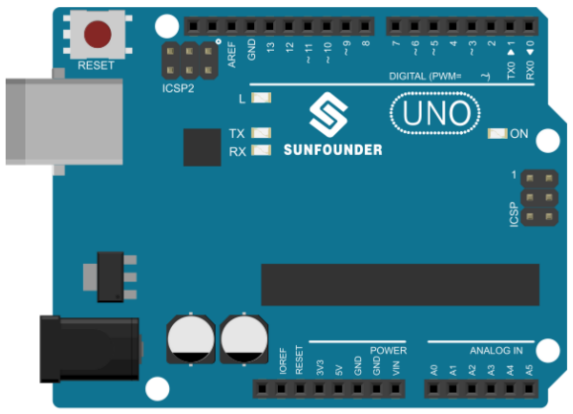
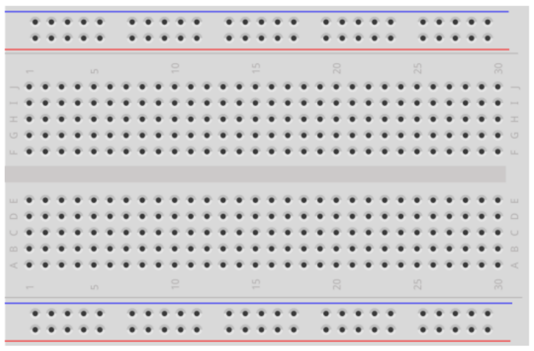
The code in this experiment is simple. First, set relayPin as HIGH level and the LED connected to the relay will light up. Then set relayPin as LOW level and the LED goes out.

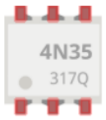
Lesson 10 4N35

Introduction

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

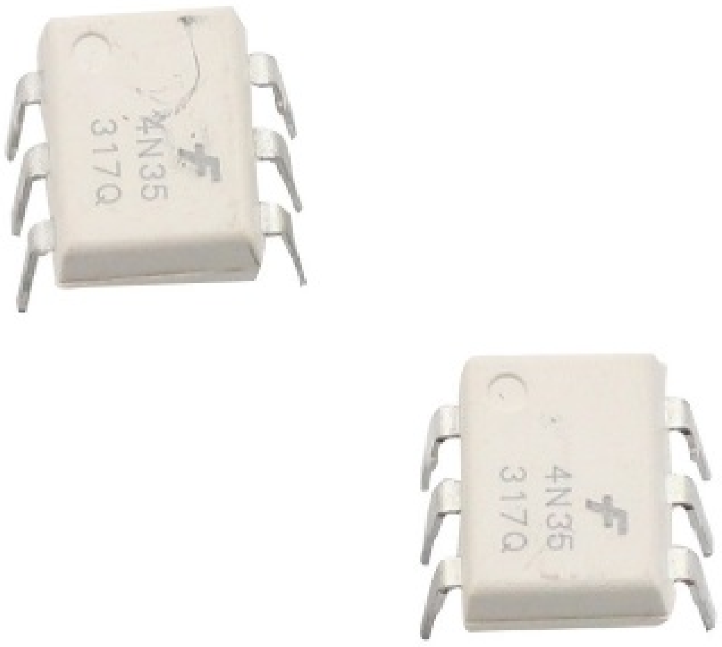
Components

1 * Uno Board 	1 * Breadboard 

1*4N35	1 * Resistor (220Ω)	1 * LED
		
1 * USB Cable	Several Jumper Wires	

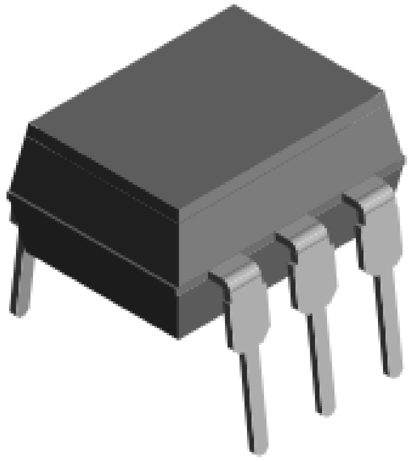
Experimental Principle

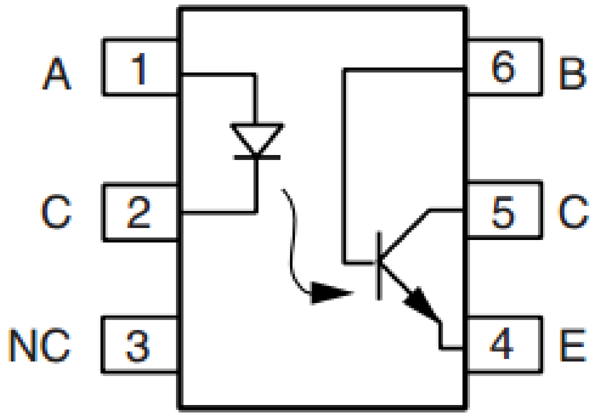
4N35



The 4N35 is an optocoupler for general purpose application. It consists of gallium arsenide infrared LED and a silicon NPN phototransistor.

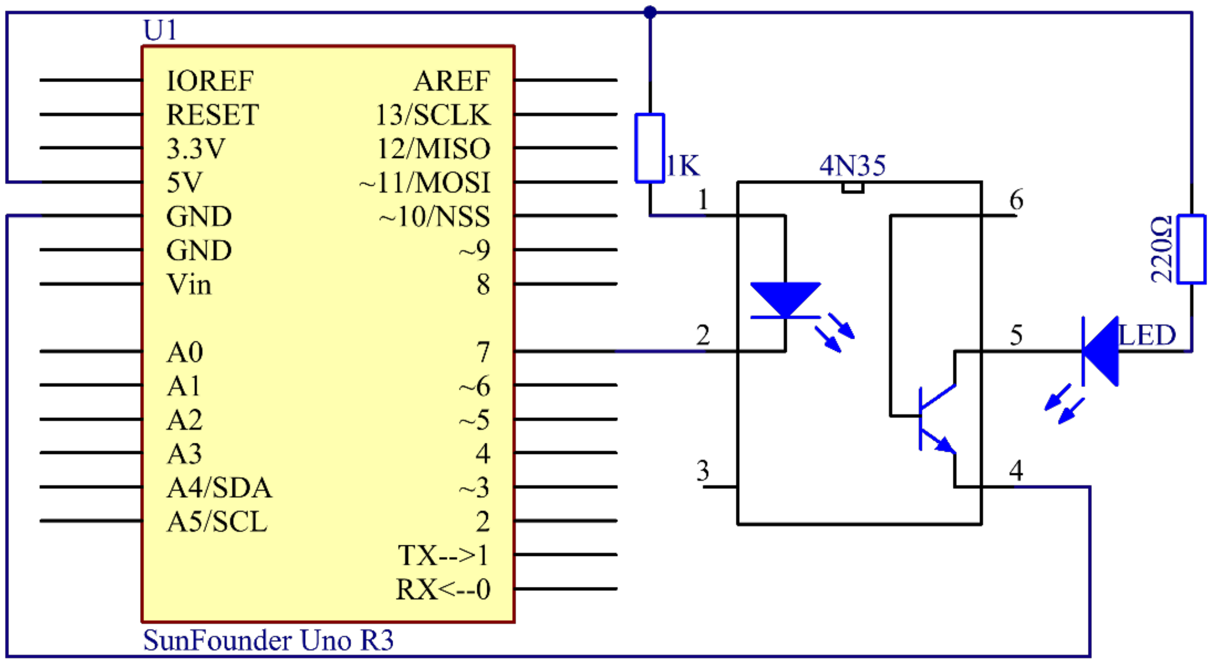
What an optocoupler does is to break the connection between signal source and signal receiver, so as to stop electrical interference. In other words, it is used to prevent interference from external electrical signals. 4N35 can be used in AV conversion audio circuits. Broadly it is widely used in electrical isolation for a general optocoupler.





See the internal structure of the 4N35 above. Pin 1 and 2 are connected to an infrared LED. When the LED is electrified, it'll emit infrared rays. To protect the LED from burning, usually a resistor (about 1K) is connected to pin 1. Then the NPN phototransistor is power on when receiving the rays. This can be done to control the load connected to the phototransistor. Even when the load short circuit occurs, it won't affect the control board, thus realizing good electrical isolation.

The schematic diagram:



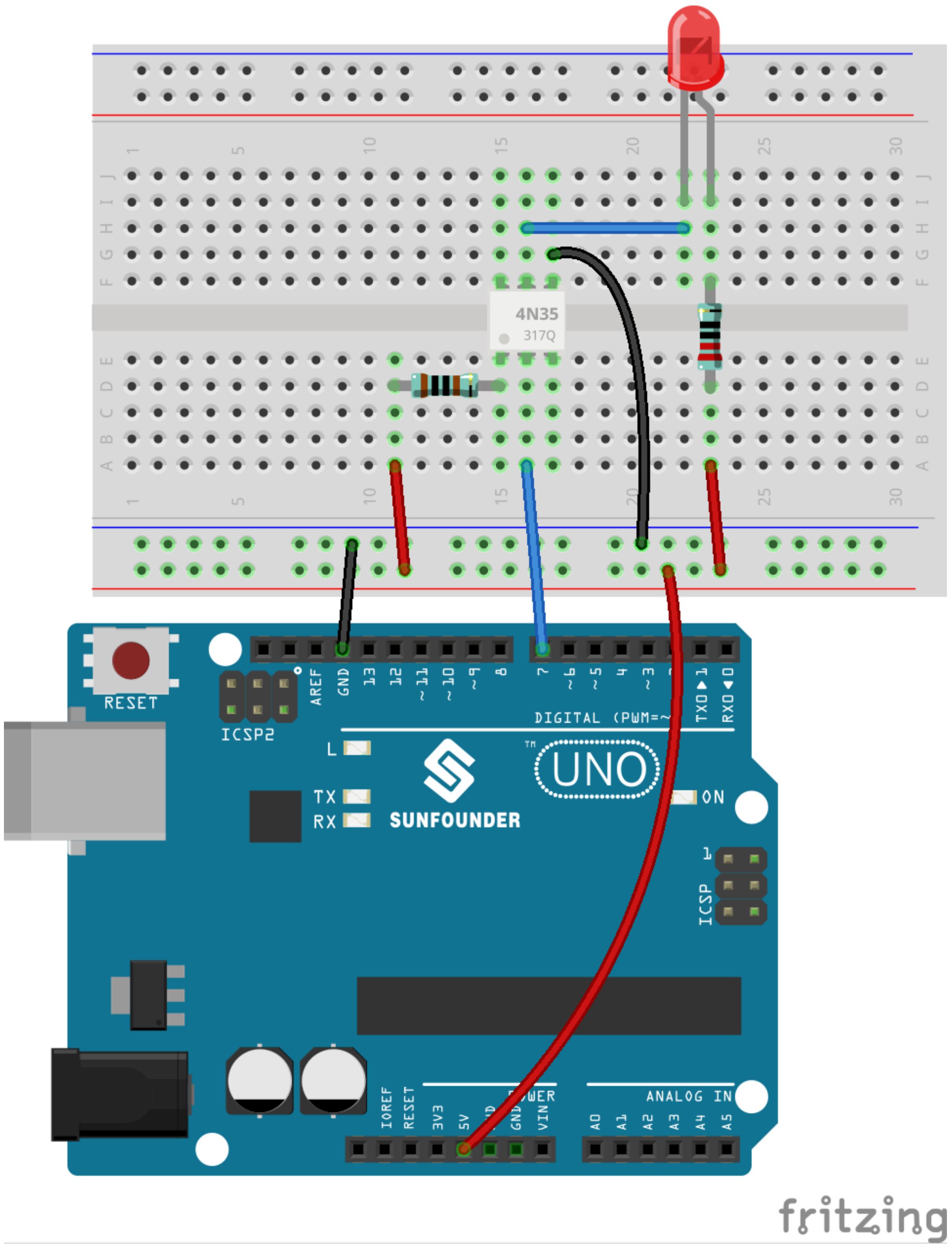
Principle:

In this experiment, use an LED as the load connected to the NPN phototransistor. Connect pin 2 of the 4N35 to pin 7 of the control board, and pin 1 to a 1K current limiting resistor and then to 5V. Connect pin 4 to GND of the Uno, and pin 5 to the cathode of the LED. Then hook the anode of the LED to 5V after connecting with a 220 Ohm resistor. When in program, a LOW level is given to pin 7, the infrared LED will emit infrared rays. Then the phototransistor receives infrared rays and gets electrified, and the LED cathode is LOW, thus

turning on the LED. Also you can control the LED by circuits only – connect pin 2 to ground and it will brighten.

Experimental Procedures

Step 1: Build the circuit (pay attention to the direction of the chip by the concave on it)

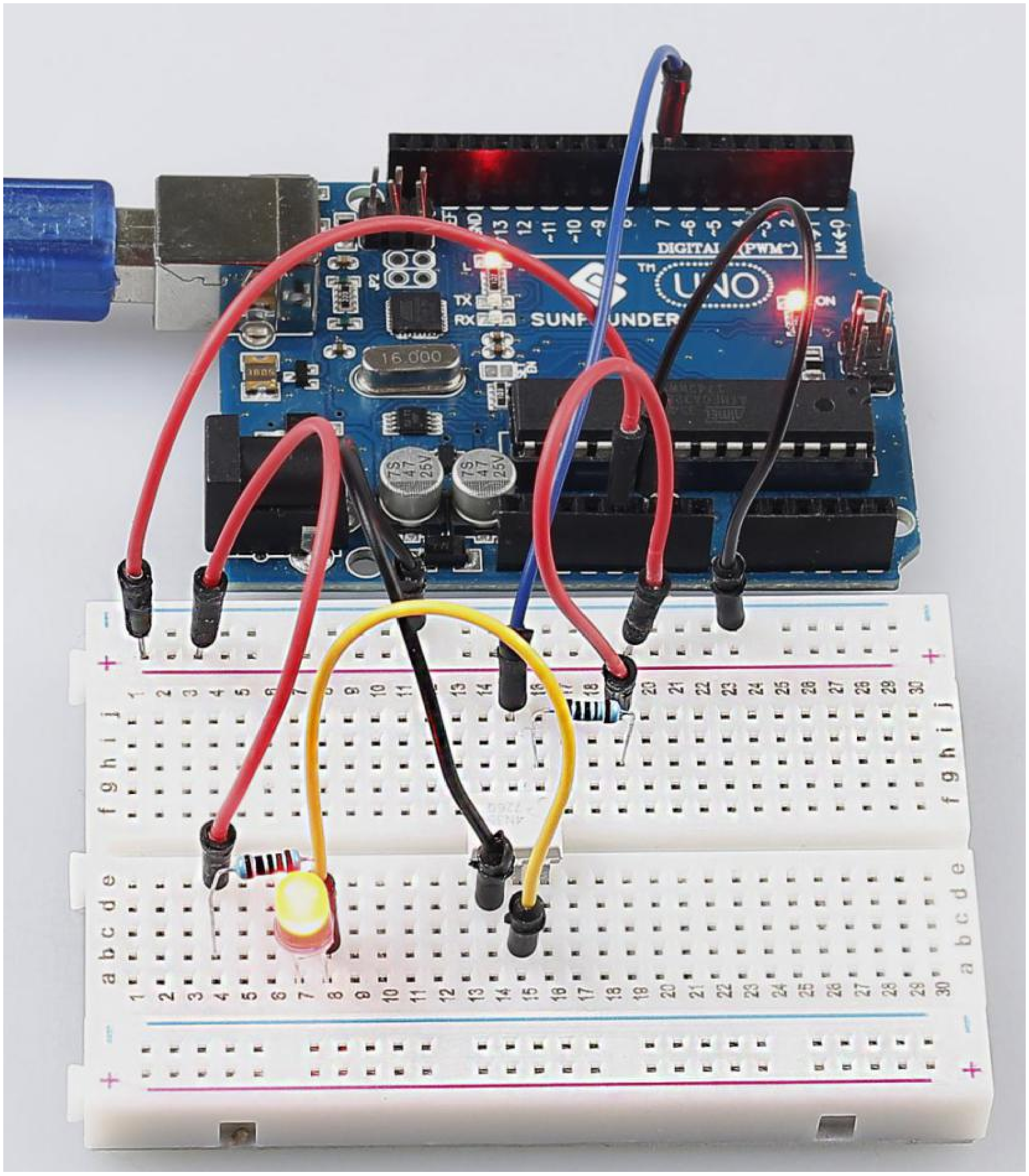


Step 2: Open the code file.

Step 3: Select correct Board and Port.

Step 4: Upload the sketch to the SunFounder Uno board.

You will see the LED blinks.



Exploration

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

Code

```
void loop()

{

    digitalWrite(OptoPin, LOW);
    //set the OptoPin as LOW level,then the led connected on the output of 4n35 will␣
↪be light

    delay(500);                   //delay 500ms

    digitalWrite(OptoPin, HIGH); //turn off the led

    delay(500);                   //delay 500ms

}
```

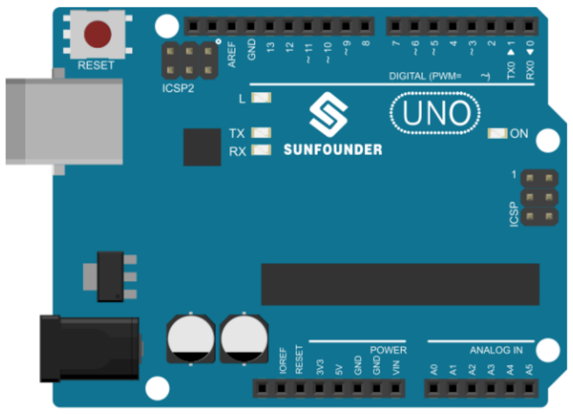
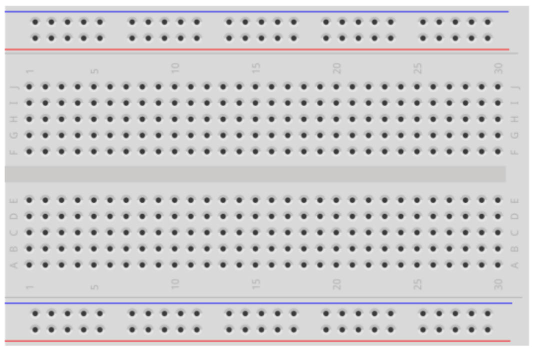
The code in this experiment is very easy to understand. Set pin 7 as Low level and the LED will light up; set it as High, and the LED goes out.

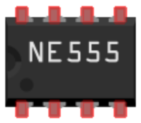
Lesson 11 NE555 Timer

Introduction

The NE555 Timer, a mixed circuit composed of analog and digital circuits, integrates analog and logical functions into an independent IC, thus tremendously expanding the applications of analog integrated circuits. It is widely used in various timers, pulse generators, and oscillators. In this experiment, the SunFounder Uno board is used to test the frequencies of square waves generated by the 555 oscillating circuit and show them on Serial Monitor.

Components

1 * Uno Board	1 * Breadboard
	

1 * Potentiometer (10KΩ)	2 * 104 ceramic capacitor	1 * NE555	1*Resistor(10kΩ)
			

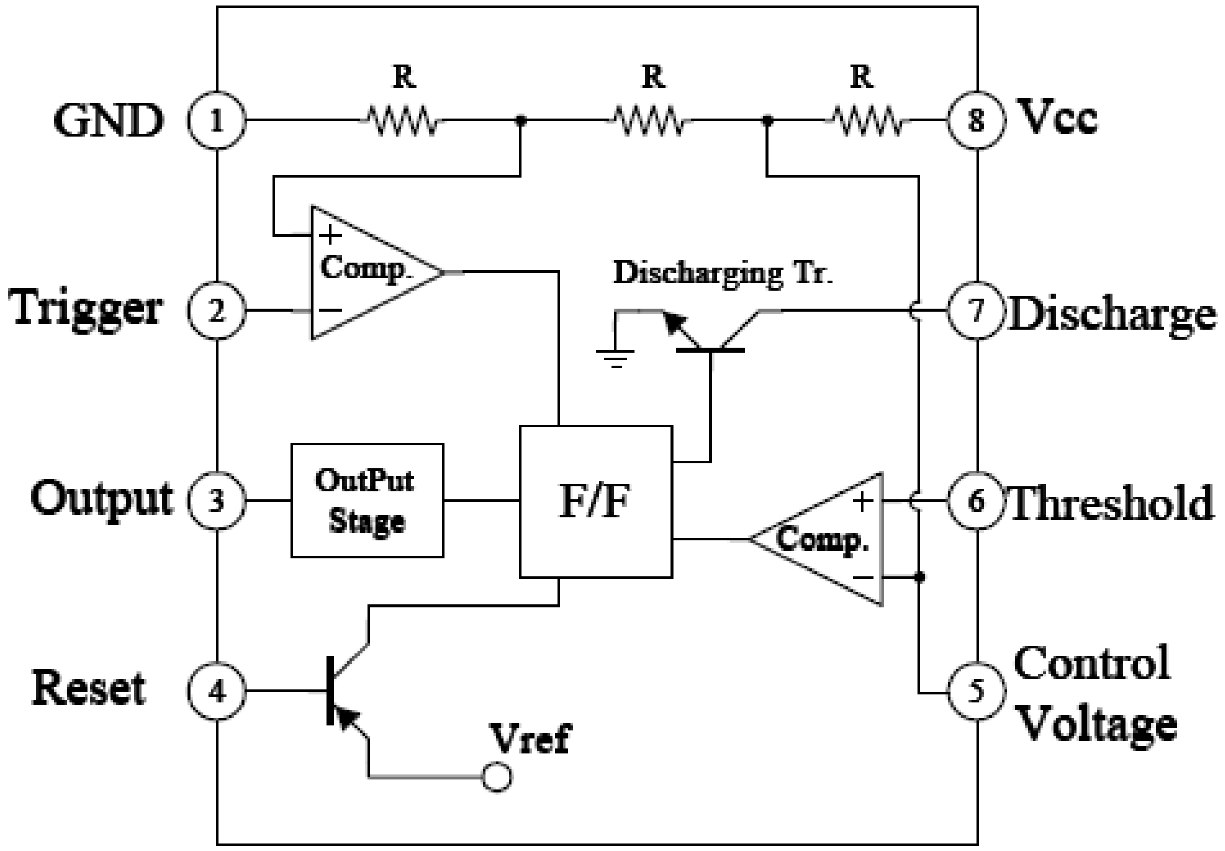
1 * USB Cable	Several Jumper Wires

Experimental Principle

555 IC

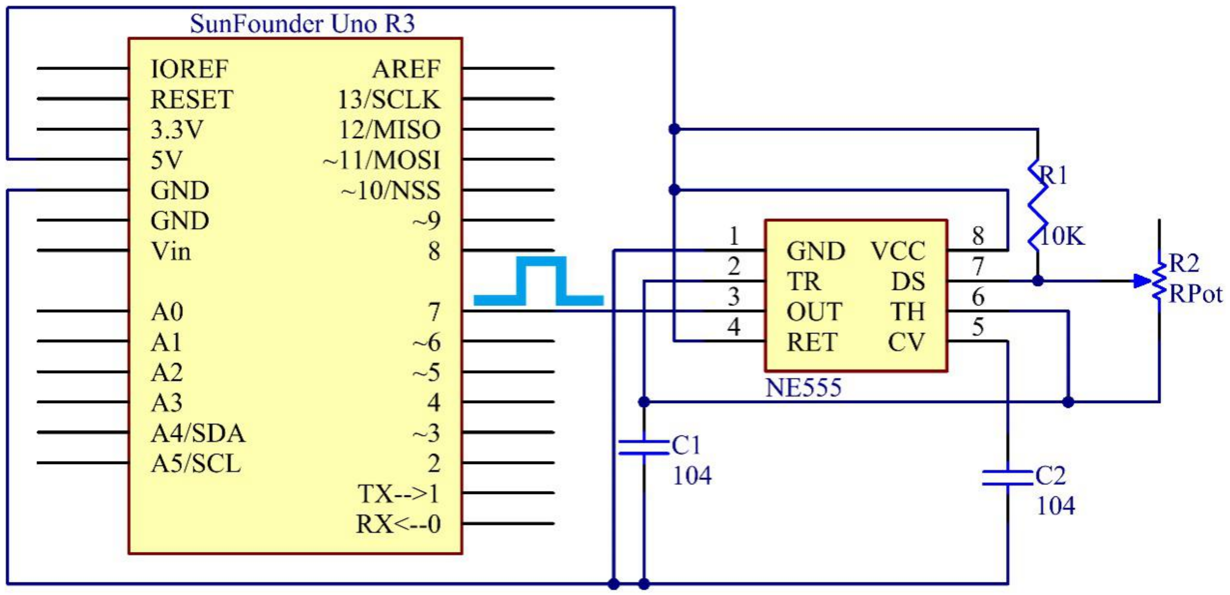
The 555 IC was originally used as a timer, hence the name 555 time base circuit. It is now widely used in various electronic products because of its reliability, convenience, and low price. The 555 is a complex hybrid circuit with dozens of components such as a divider, comparator, basic R-S trigger, discharge tube, and buffer.

Its pins and their functions:



As shown in the picture, the pins are set dual in-line with the 8-pin package.

- Pin 1 (**GND**): the ground
- Pin 2 (**TRIGGER**): when the voltage at the pin reduces to 1/3 of the VCC (or the threshold defined by the control board), the output terminal sends out a High level
- Pin 3 (**OUTPUT**): outputs High or Low, two states 0 and 1 decided by the input electrical level; maximum output current approx. 200mA at High
- Pin 4 (**RESET**): when a Low level is received at the pin, the timer will be reset and the output will return to Low level; usually connected to positive pole or neglected
- Pin 5 (**CONTROL VOLTAGE**): to control the threshold voltage of the chip (if it skips connection, by default, the threshold voltage is 1/3 VCC and 2/3 VCC)
- Pin 6 (**THRESHOLD**): when the voltage at the pin increases to 2/3 VCC (or the threshold defined by the control board), the output terminal sends out a High level
- Pin 7 (**DISCHARGE**): output synchronized with Pin 3, with the same logical level; but this pin does not output current, so pin 3 is the real High (or Low) when pin 7 is the virtual High (or Low); connected to the open collector (OC) inside to discharge the capacitor
- Pin 8 (**VCC**): positive terminal for the NE555 timer IC, ranging +4.5V to +16V
- The NE555 timer works under the monostable, astable and bistable modes. In this experiment, apply it under the astable mode, which means it works as an oscillator, as shown below:



Connect a resistor R1 between the VCC and the discharging pin DS, another resistor between pin DS and the trigger pin TR which is connected to the threshold pin TH and then to the capacitor C1. Connect the RET (pin 4) to VCC, CV (pin 5) to another capacitor C2 and then to the ground.

Working process:

The oscillator starts to shake once the circuit is power on. Upon the energizing, since the voltage at C1 cannot change abruptly, which means pin 2 is Low level initially, set the timer to 1, so pin 3 is High level. The capacitor C1 charges via R1 and R2, in a time span:

Tc=0.693(R1+R2)

When the voltage at C1 reaches the threshold 2/3Vcc, the timer is reset and pin 3 is Low level. Then C1 discharges via R2 till 2/3Vcc, in a time span:

Td=0.693(R2)

Then the capacitor is recharged and the output voltage flips again:

Duty cycle D=Tc/(Tc+Td) x 100%

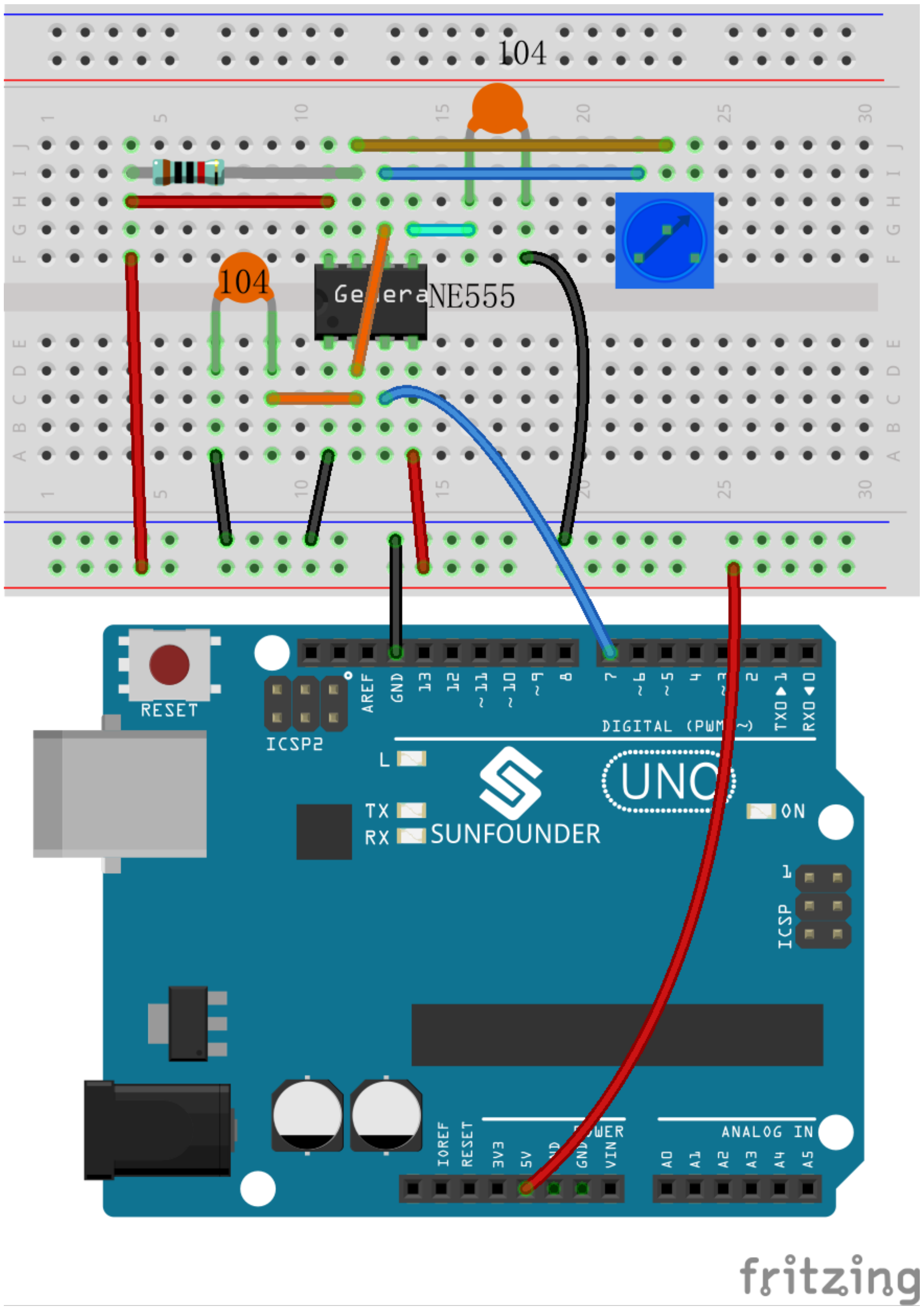
Since a potentiometer is used for resistor, we can output square wave signals with different duty cycles by adjusting its resistance. But R1 is a 10K resistor and R2 is 0k-10k, so the range of the ideal duty cycle is 66.7%-100%. If you want another else, you need to change the resistance of R1 and R2.

Dmin=(0.693(10K+0K))/(0.693(10K+0K)+0.693x0k) x100%=100%

Dmax=(0.693(10K+10K))/(0.693(10K+10K)+0.693x10k) x100%=66.7%

Experimental Procedures

Step 1: Build the circuit.

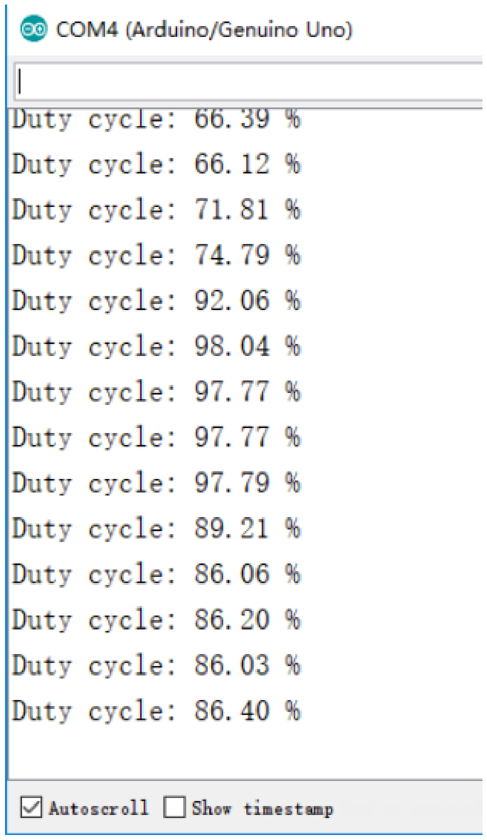

104
NE555
104
RESET
ICSP2
AREF
GND
DIGITAL (PWM ~)
UNO
SUNFOUNDER
TX
RX
L
ON
ICSP
IOREF
RESET
3V3
5V
GND
VIN
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5
fritzing

Step 2: Open the code file.

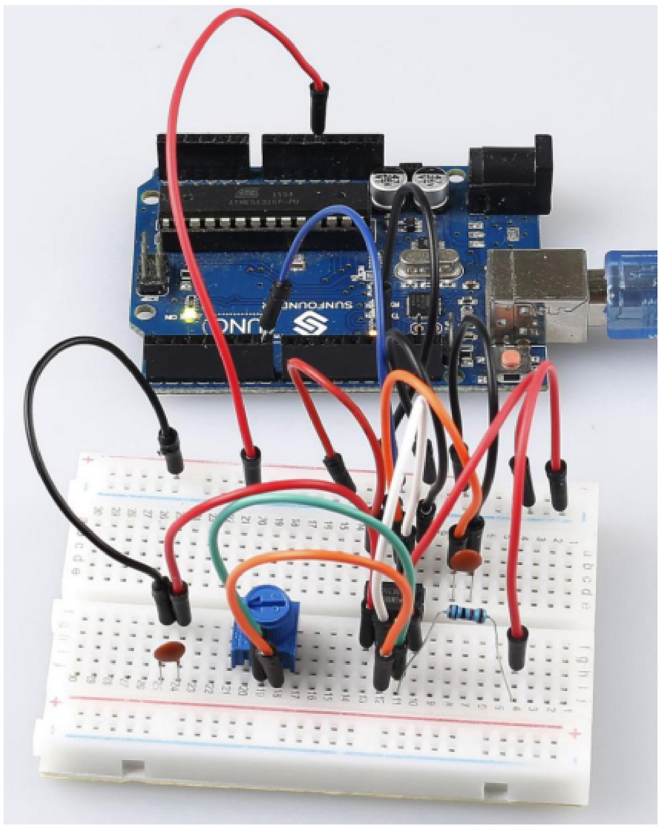
Step 3: Select correct Board and Port.

Step 4: Upload the sketch to the SunFounder Uno board.

After uploading, open the Serial Monitor and you will see the following window.



```
Duty cycle: 66.39 %
Duty cycle: 66.12 %
Duty cycle: 71.81 %
Duty cycle: 74.79 %
Duty cycle: 92.06 %
Duty cycle: 98.04 %
Duty cycle: 97.77 %
Duty cycle: 97.77 %
Duty cycle: 97.79 %
Duty cycle: 89.21 %
Duty cycle: 86.06 %
Duty cycle: 86.20 %
Duty cycle: 86.03 %
Duty cycle: 86.40 %
```



Code

Code Analysis 11-1 Calculate the duty cycle

```
void loop()

{

    duration1 = pulseIn(ne555, HIGH);   // Reads a pulse on ne555

    duration2 = pulseIn(ne555, LOW);    // Reads a pulse on ne555

    dc = float (duration1) / (duration1 + duration2) * 100;

    Serial.print("Duty cycle: ");

    Serial.print(dc);  // print the length of the pulse on the serial monitor
```

(continues on next page)

(continued from previous page)

```
    Serial.print(" %");

    Serial.println();  // print an blank on serial monitor

    delay(500);        // wait for 500 microseconds

}
```

Read a pulse waits for the ne555(pin 7) from HIGH to LOW firstly, then read a pulse waits for pin 7 from LOW to HIGH. So the range of the ideal duty cycle dc is float (duration1) / (duration1 + duration2) * 100; You can rotate the potentiometer and read the duty cycle from the serial monitor.

pulseIn()

[Advanced I/O]

Description

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go from LOW to HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds or gives up and returns 0 if no complete pulse was received within the timeout.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax

pulseIn(pin, value)

pulseIn(pin, value, timeout)

Parameters

pin: the number of the pin on which you want to read the pulse. (int)

value: type of pulse to read: either HIGH or LOW. (int)

timeout (optional): the number of microseconds to wait for the pulse to start; default is one second. (unsigned long)

Returns

the length of the pulse (in microseconds) or 0 if no pulse started before the timeout. (unsigned long)

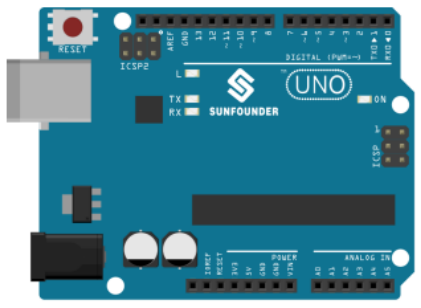
Lesson 12 Servo

Introduction

Servo is a type of geared motor that can only rotate 180 degrees. It is controlled by sending electrical pulses from your board. These pulses tell the servo what position it should move to.

A servo has three wires: the brown wire is GND, the red one is VCC, and the orange one is signal line.

Components

1 * Uno Board	1 * Servo	1 * USB Cable
		
	Several Jumper Wires(M to F)	

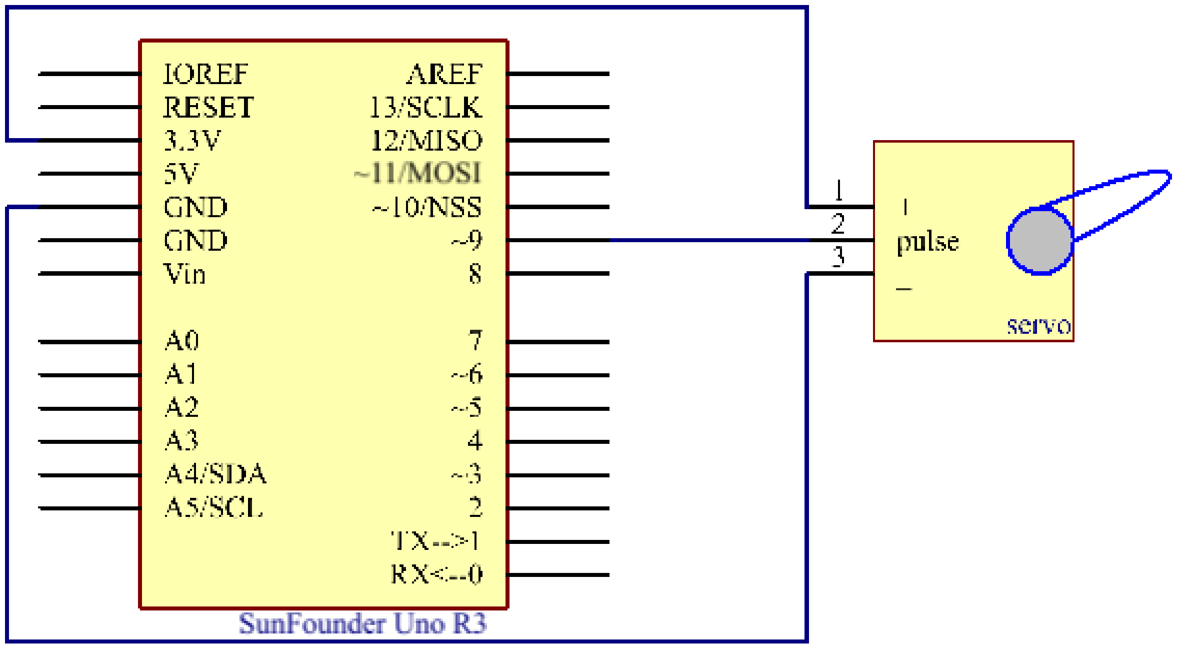
Experimental Principle

Servo

A servo is generally composed of the following parts: case, shaft, gear train, adjustable potentiometer, DC motor, and control circuit board.

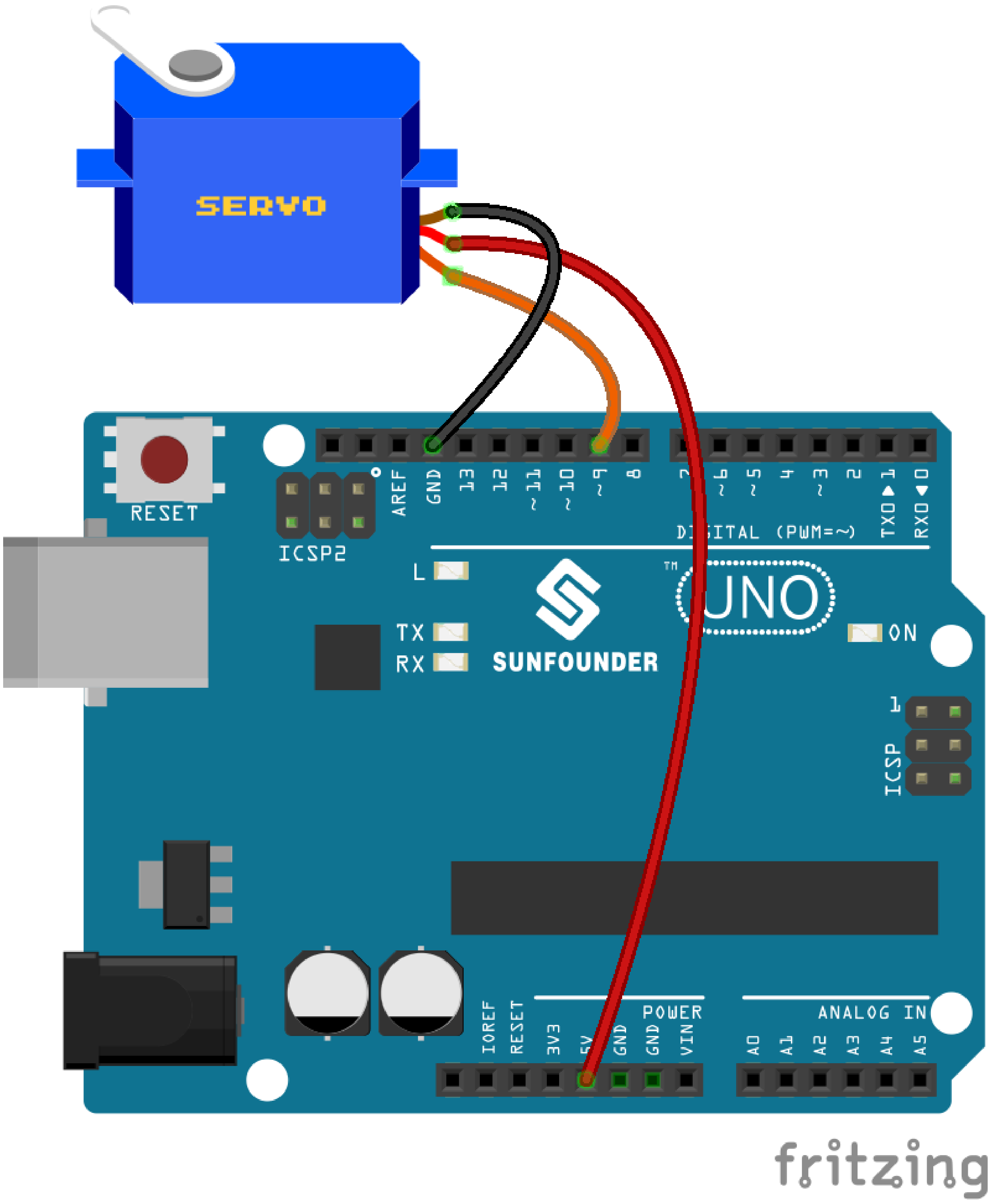
It works like this: The Uno board sends out PWM signals to the servo, and then the control circuit in the servo receives the signals through the signal pin and controls the motor inside to turn. As a result, the motor drives the gear chain and then motivates the shaft after deceleration. The shaft and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

The schematic diagram:



Experimental Procedures

Step 1: Build the circuit. (Brown to GND, Red to VCC, Orange to pin 9 of the control board)

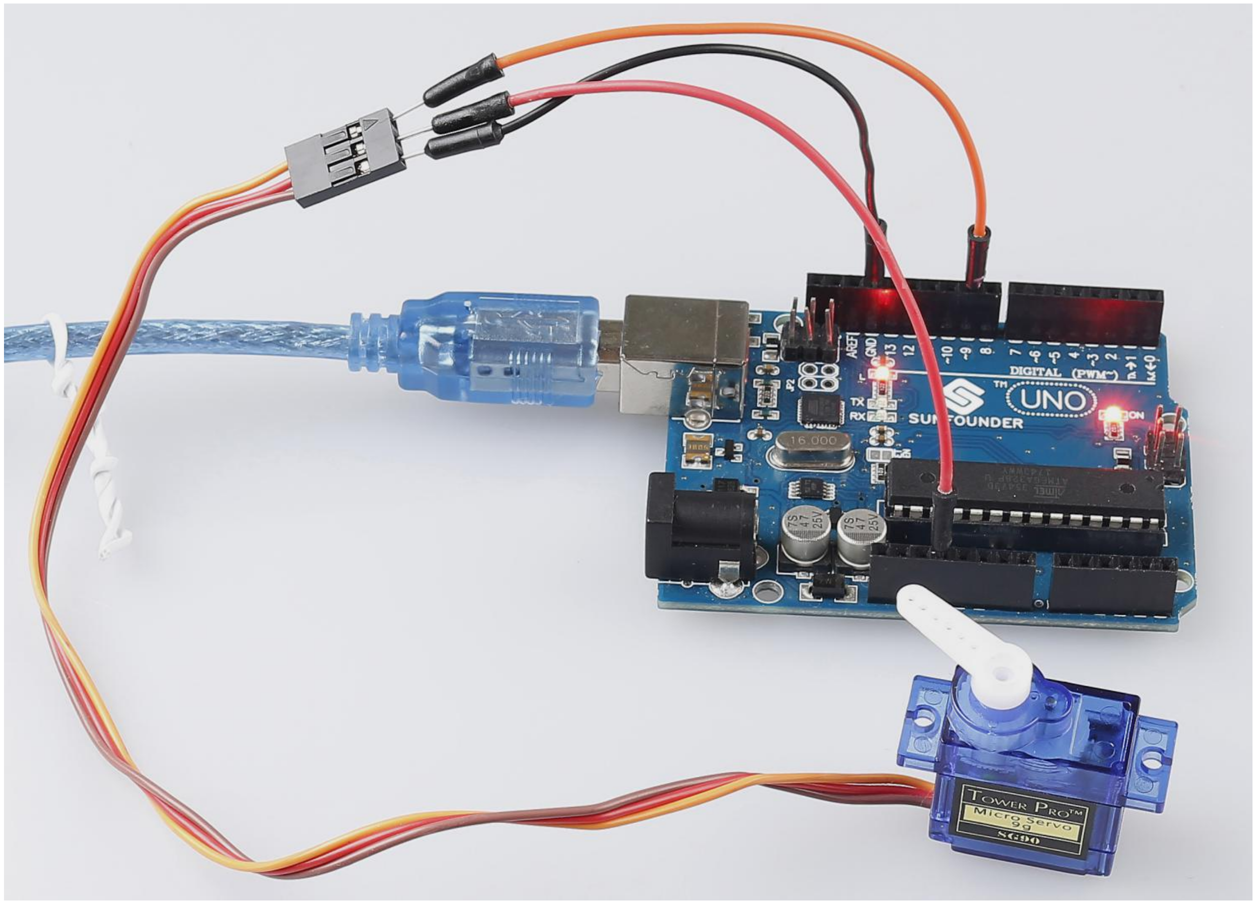


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Now, you can see the rocker arm of the servo rotate and stop at 90 degrees (15 degrees each time). And then it rotates in the opposite direction.



Code

Code Analysis 12-1 Include a library

```
#include <Servo.h>
Servo myservo;      //create servo object to control a servo
```

With the Servo.h file included, you can call the functions in this file later. Servo is a built-in library in the Arduino IDE. You can find the Servo folder under the installation path *C:\Program Files\Arduino\libraries*.

Code Analysis 12-2 Initialize the servo

```
{

    myservo.attach(9);   //attachs the servo on pin 9 to servo object

    myservo.write(0);    //back to 0 degrees

    delay(1000);         //wait for a second

}
```

myservo.attach(): Attach the Servo variable to a pin. Initialize the servo attach to pin9.

myservo.write(): Writes a value to the servo, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation. Here let the servo stay in the 0 angle firstly.

Code Analysis 12-3 Servo rotate

```
void loop()

{

    for (int i = 0; i <= 180; i++)

    {

        myservo.write(i);   //write the i angle to the servo

        delay(15);          //delay 15ms

    }

    for (int i = 180; i >= 0; i--)

    {

        myservo.write(i);   //write the i angle to the servo

        delay(15);          //delay 15ms

    }

}
```

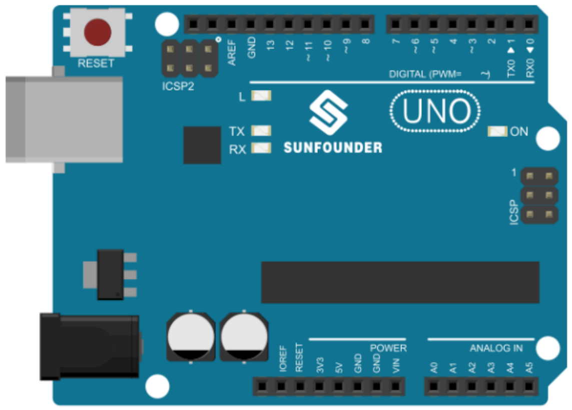
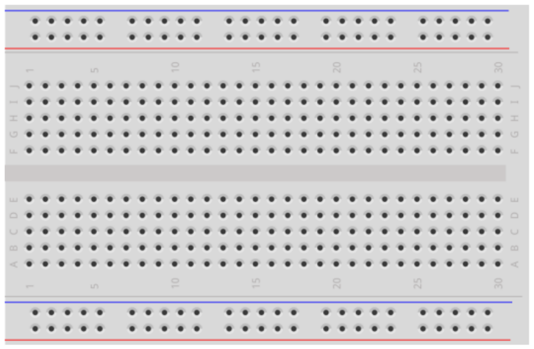
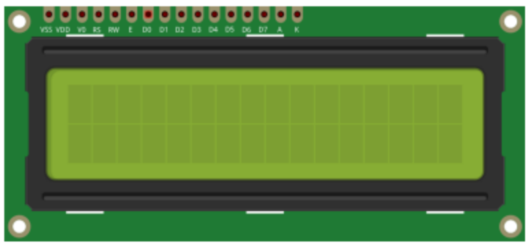
Use 2 for() statement to write 0 - 180 to the servo, so that you can see the servo rotate from 0 to 180 angle, then turn back to 0.

Lesson 13 LCD1602

Introduction

In this lesson, we will learn how to use an LCD1602 to display characters and strings. LCD1602, or 1602 character-type liquid crystal display, is a kind of dot matrix module to show letters, numbers, and characters and so on. It's composed of 5x7 or 5x11 dot matrix positions; each position can display one character. There's a dot pitch between two characters and a space between lines, thus separating characters and lines. The number 1602 means on the display, 2 rows can be showed and 16 characters in each. Now let's check more details!

Components

1 * Uno Board 	1 * Breadboard 
1 * Potentiometer (10kΩ)	1 * LCD1602 
1 * USB Cable	Several Jumper Wires

Experimental Principle

Generally, LCD1602 has parallel ports, that is, it would control several pins at the same time. LCD1602 can be categorized into eight-port and four-port connections. If the eight-port connection is used, then all the digital ports of the Uno board are almost completely occupied. If you want to connect more sensors, there will be no ports available. Therefore, the four-port connection is used here for better application.

Pins of LCD1602 and their functions

VSS: connected to ground

VDD: connected to a +5V power supply

VO: to adjust the contrast

RS: A register select pin that controls where in the LCD's memory you are writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

R/W: A Read/Write pin to select between reading and writing mode

E: An enabling pin that reads the information when High level (1) is received. The instructions are run when the signal changes from High level to Low level.

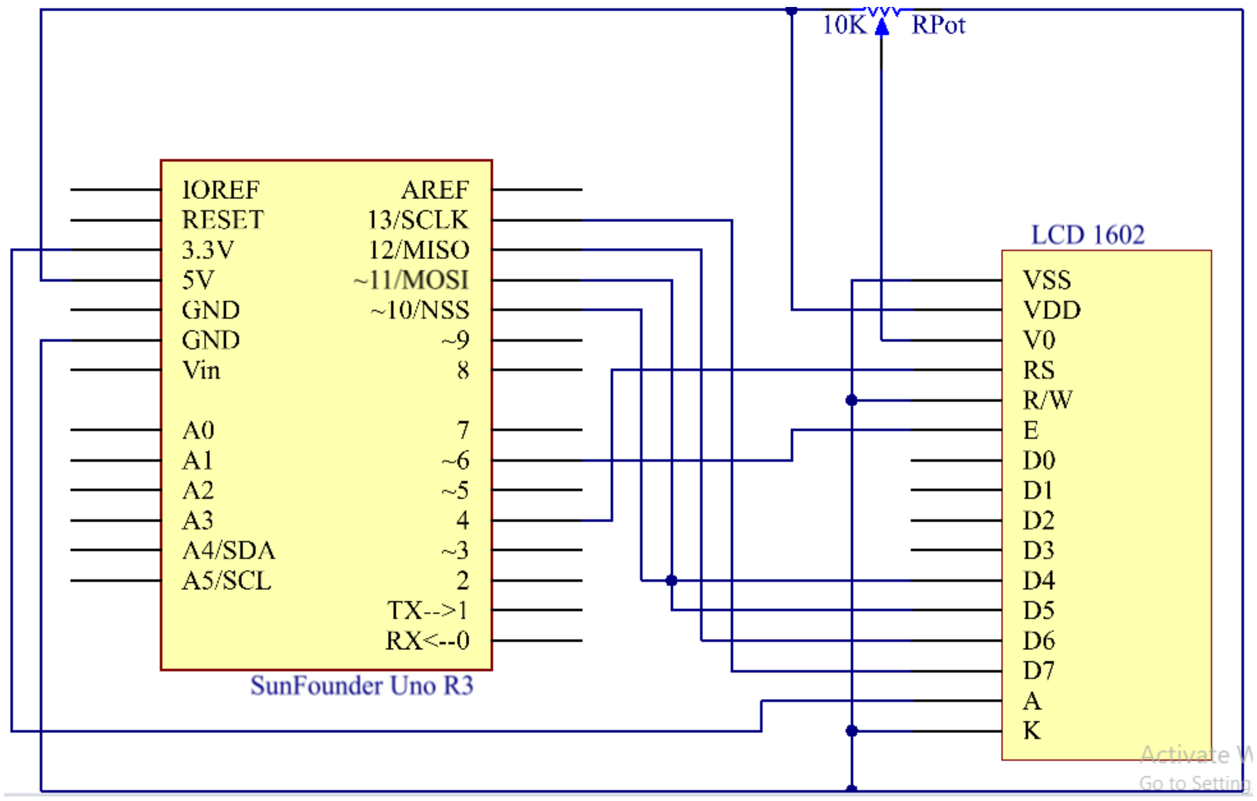
D0-D7: to read and write data

A and K: Pins that control the LCD backlight. Connect K to GND and A to 3.3v. Open the backlight and you will see clear characters in a comparatively dark environment.

Principle:

Connect K to GND and A to 3.3 V, and then the backlight of the LCD1602 will be turned on. Connect VSS to GND and the LCD1602 to the power source. Connect VO to the middle pin of the potentiometer - with it you can adjust the contrast of the screen display. Connect RS to D4 and R/W pin to GND, which means then you can write characters to the LCD1602. Connect E to pin6 and the characters displayed on the LCD1602 are controlled by D4-D7. For programming, it is optimized by calling function libraries.

The schematic diagram:



Experimental Procedures

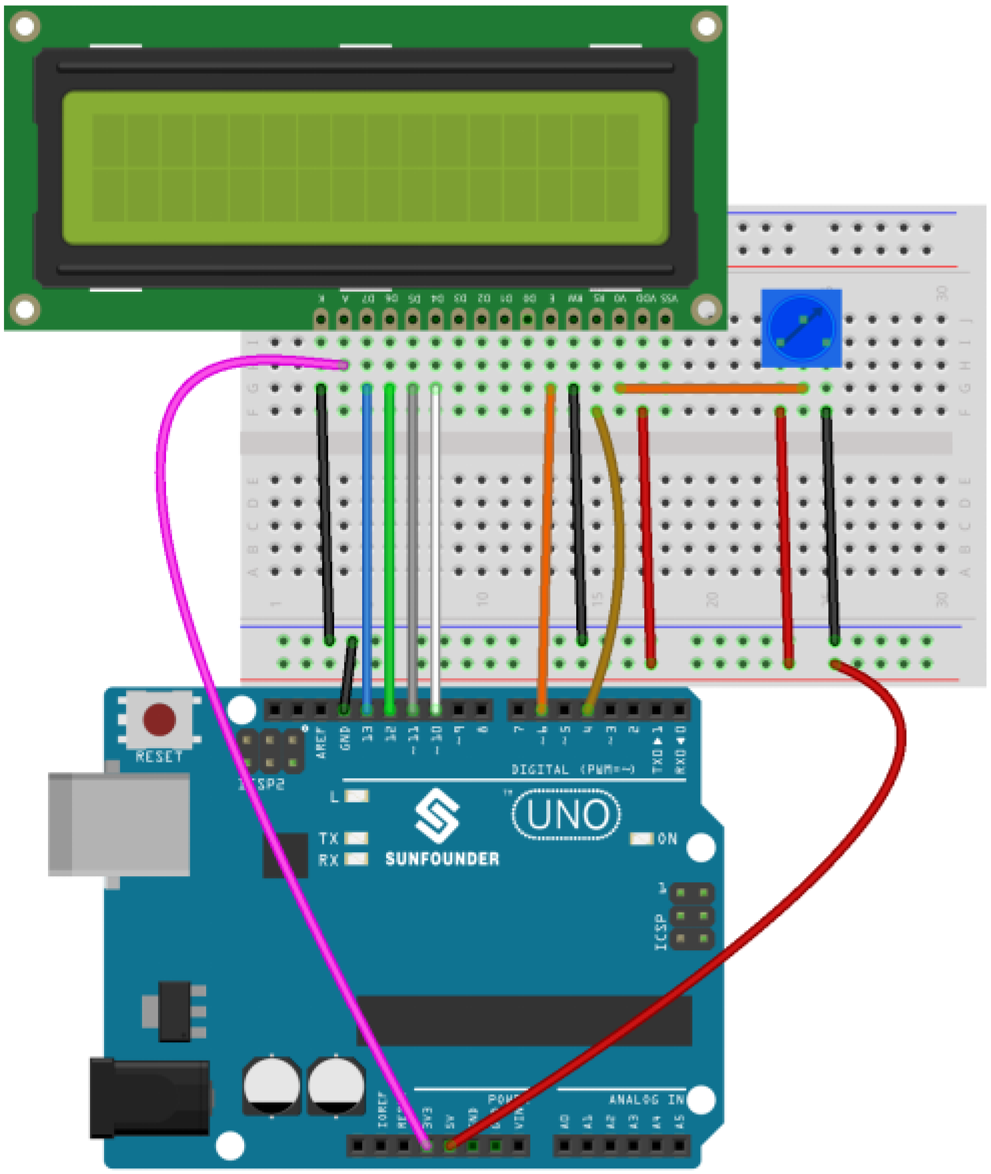
Step 1: Build the circuit (make sure the pins are connected correctly. Otherwise, characters will not be displayed properly):

Step 2: Open the code file.

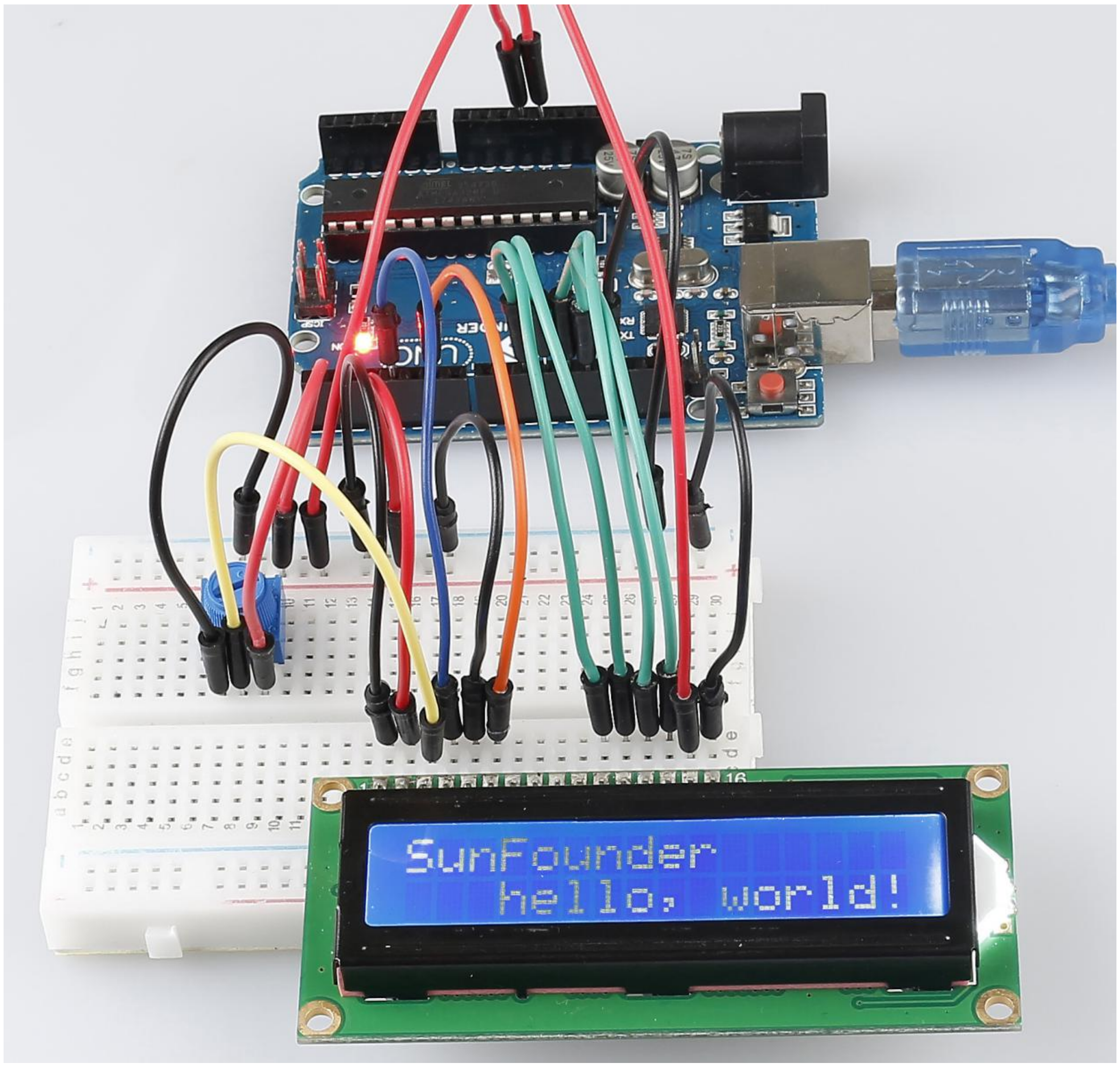
Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

Note: you may need to adjust the potentiometer until the LCD1602 can display clearly.



You should now see the characters “**SunFounder**” and “**hello, world**” rolling on the LCD.



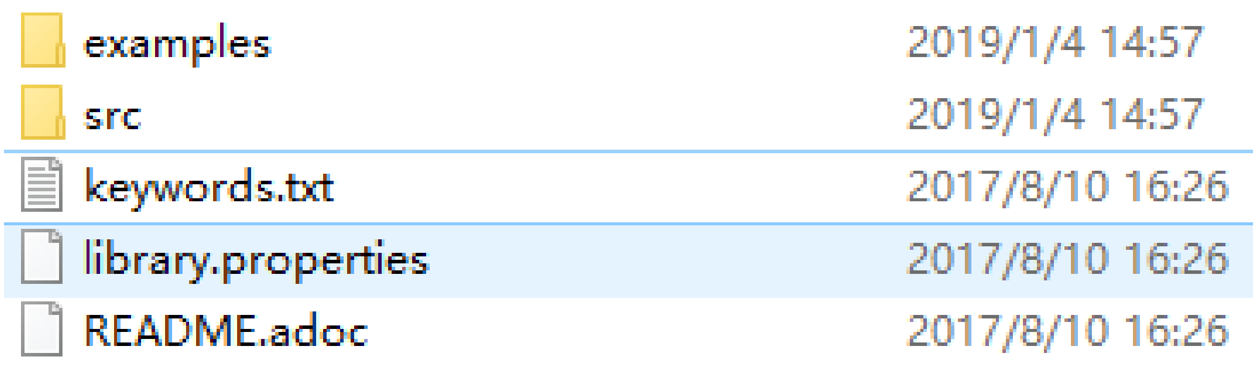
Code

Code Analysis 13-1 Include a library

```
#include <LiquidCrystal.h> // include the library code
```

With the *LiquidCrystal.h* file included, you can call the functions in this file later.

LiquidCrystal is a built-in library in the Arduino IDE. You can find the LiquidCrystal folder under the installation path *C:\Program Files\Arduino\libraries*.



```
examples            2019/1/4 14:57
src                 2019/1/4 14:57
keywords.txt        2017/8/10 16:26
library.properties  2017/8/10 16:26
README.adoc         2017/8/10 16:26
```

There is an example in the *examples* folder. The src folder contains the major part of the library: *LiquidCrystal.cpp* (execution file, with function implementation, variable definition, etc.) and LiquidCrystal.h (header file, including function statement, Macro definition, struct definition, etc.). If you want to explore how a function is implemented, you can look up in the file *LiquidCrystal.cpp*.

Code Analysis 13-2 Displayed characters

```
char array1[]=" SunFounder ";   // the string to print on the LCD

char array2[]="hello, world! ";   // the string to print on the LCD
```

These are two character type arrays: *arry1[]* and *array2[]*. The contents in the quotation marks ”xxx” are their elements, including 26 characters in total (spaces counted). *array1[0]* stands for the first element in the array, which is a space, and *array1[1]* means the second element *S* and so on. So *array1[25]* is the last element (here it’s also a space).

Code Analysis 13-3 Define the pins of LCD1602

```
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);
```

Define a variable *lcd* of LiquidCrystal type. Here use *lcd* to represent *LiquidCrystal* in the following code.

The basic format of the *LiquidCrysral()* function is: LiquidCrystal (rs, enable, d4, d5, d6, d7). You can check the *LiquidCrystal.cpp* file for details.

So this line defines that pin RS is connected to pin 4, the enable pin to pin 6, and d4-d7 to pin10-13 respectively.

Code Analysis 13-4 Initialize the LCD

```
lcd.begin(16, 2);
// set up the LCD's number of columns and rows: begin(col,row) is to set the display␣
↪of LCD. Here set as 16 x 2.
```

Code Analysis 13-5 Set the cursor position of LCD

```
lcd.setCursor(15,0); // set the cursor to column 15, line 0
```

setCursor(col,row) sets the position of the cursor which is where the characters start to show. Here set it as 15col, 0 row.

Code Analysis 13-6 LCD displays the elements inside array1[]and array2[]

```
for (int positionCounter1 = 0; positionCounter1 < 26; positionCounter1++)

{

    lcd.scrollDisplayLeft();              // Scrolls the contents of the display one␣
↪space to the left.

    lcd.print(array1[positionCounter1]); // Print a message to the LCD.

    delay(tim);                           // wait for 250 microseconds

}
```

When positionCounter1=0, which accords with positionCounter1<26, positionCounter1 adds 1. Move one bit to the left through lcd.scrollDisplayLeft(). Make the LCD display array1[0] by lcd.print(array1[positionCounter1]) and delay for tim ms (250 ms). After 26 loops, all the elements in array1[] have been displayed.

```
lcd.clear(); // Clears the LCD screen.
```

Clear the screen with lcd.clear() so it won't influence the display next time.

```
lcd.setCursor(15,1);
// set the cursor to column 15, line 1
// Set the cursor at Col. 15 Line 1, where the characters will start to show.

for (int positionCounter2 = 0; positionCounter2 < 26; positionCounter2++)
{

    lcd.scrollDisplayLeft();              // Scrolls the contents of the display one␣
↪space to the left.

    lcd.print(array2[positionCounter2]); // Print a message to the LCD.

    delay(tim);                           // wait for 250 microseconds

}
```

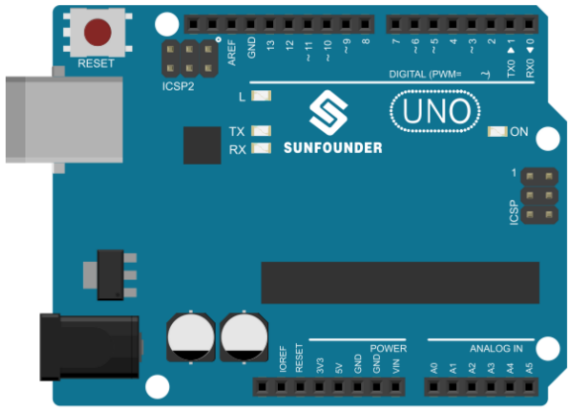
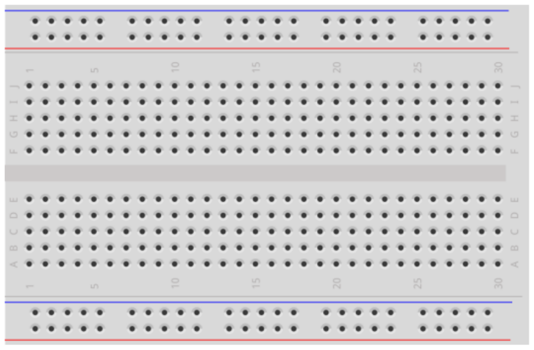
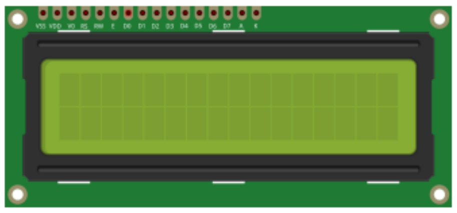
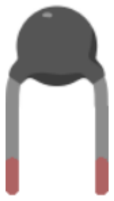
Similarly, the code is to display the elements in *array2[]* on the LCD. Therefore, you will see "SunFounder" scroll in the top line of the LCD, move left until it disappears. And then in the bottom line, "hello, world ! " appears, scrolls to the left until it disappears.

Lesson 14 Thermistor

Introduction

We've learnt many devices so far. To make more things, you need to have a good command of more knowledge. Today we're going to meet a thermistor. It is similar to photoresistor in being able to change their resistance based on the outer change. Different from photoresistor, resistance of thermistor varies significantly with temperature in the outer environment.

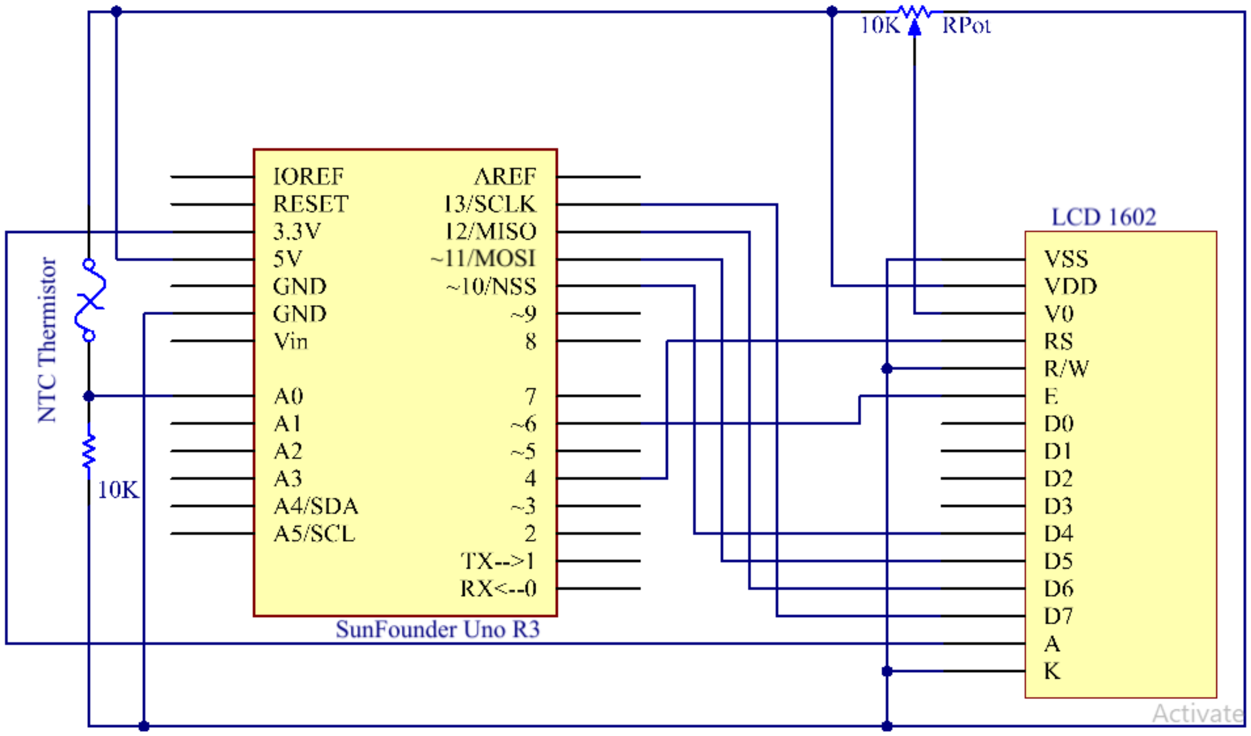
Components

1 * Uno Board 	1 * Breadboard 
1 * LCD1602 	1 * Thermistor 
1 * Resister (10KΩ)	1 * Potentiometer (10KΩ)
1 * USB Cable	Several Jumper Wires

Experimental Principle

Thermistor is a sensitive element, it has two types: Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC), also NTC and PTC. Its resistance varies significantly with temperature. The resistance of PTC thermistor increases with higher temperature when that of NTC, decreases. In this experiment we use an NTC one.

The schematic diagram:



The principle is that the resistance of the NTC thermistor changes with the temperature difference in the outer environment. It detects the real-time temperature of the environment. When the temperature gets higher, the resistance of the thermistor decreases and the voltage of pin A0 increases accordingly. The voltage data then is converted to digital quantities by the A/D adapter. The temperature in Celsius and Fahrenheit then is output via programming and then displayed on LCD1602.

In this experiment a thermistor and a 10k pull-up resistor are used. Each thermistor has a normal resistance. Here it is 10k ohm, which is measured under 25 degree Celsius.

Here is the relation between the resistance and temperature change:

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

RT: resistance of the NTC thermistor when the temperature is T_K.

RN: resistance of the NTC thermistor under the rated temperature which is T_N.

TK is a Kelvin temperature and the unit is K.

TN is a rated Kelvin temperature; the unit is K, also.

And, beta, here is the material constant of NTC thermistor, also called heat sensitivity index.

exp is short for exponential, an exponential with the base number e, which is a natural number and equals 2.7 approximately.

Note that this relation is an empirical formula. It is accurate only when the temperature and resistance are within the effective range.

Since T_K =T+273, T is Celsius temperature, the relation between resistance and temperature change can be transformed into this:

R =R_o $\exp^{B[1/(T+273) - 1/(To+273)]}$

B, short for beta, is a constant. Here it is 4090. R_o is 10k ohms and T_o is 25 degrees Celsius. The data can be found in the datasheet of thermistor. Again, the above relation can be transformed into one to evaluate temperature:

T= B/[ln(R/ 10) + (B/ 298)] – 273 (So ln here means natural logarithm, a logarithm to the base e)

If we use a resistor with fixed resistance as 10k ohms, we can calculate the voltage of the analog input pin A0 with this formula:

V =10k x 5/(R+10K)

So, this relation can be formed:

R = (5 x 10k /V) - 10k

The voltage of A0 is transformed via A/D adaptor into a digital number a.

a=V x (1024/5)

V=a/205

Then replace V in the relation *R* = (5 x *10k /V)* - 10k with the expression, and we can get this: R=1025 x 10k/a-10k.

Finally replace R in the formula here T= B/[ln(R/ 10) + (B/ 298)] – 273, which is formed just now. Then we at last get the relation for temperature as this:

T =B/[ln{[1025 X 10/a) - 10]/10} (B/298)] – 273

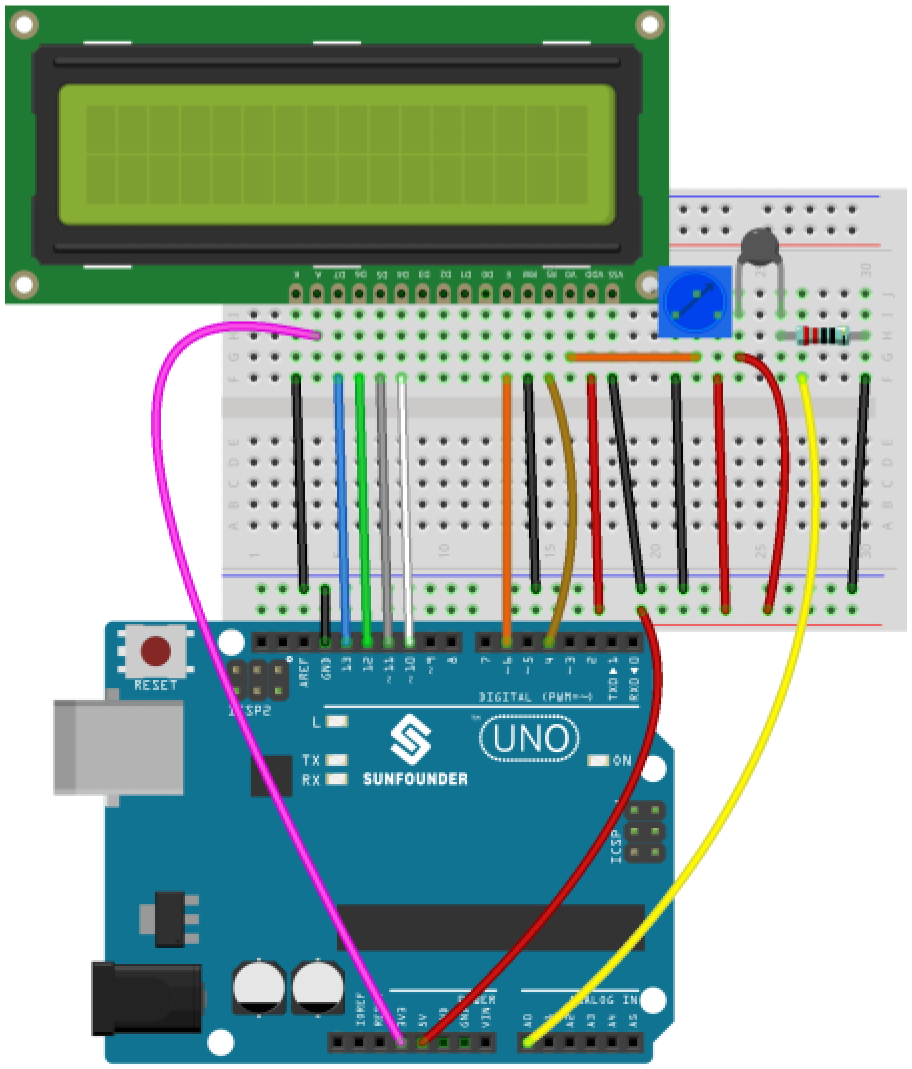
Experimental Procedures

Step 1: Build the circuit

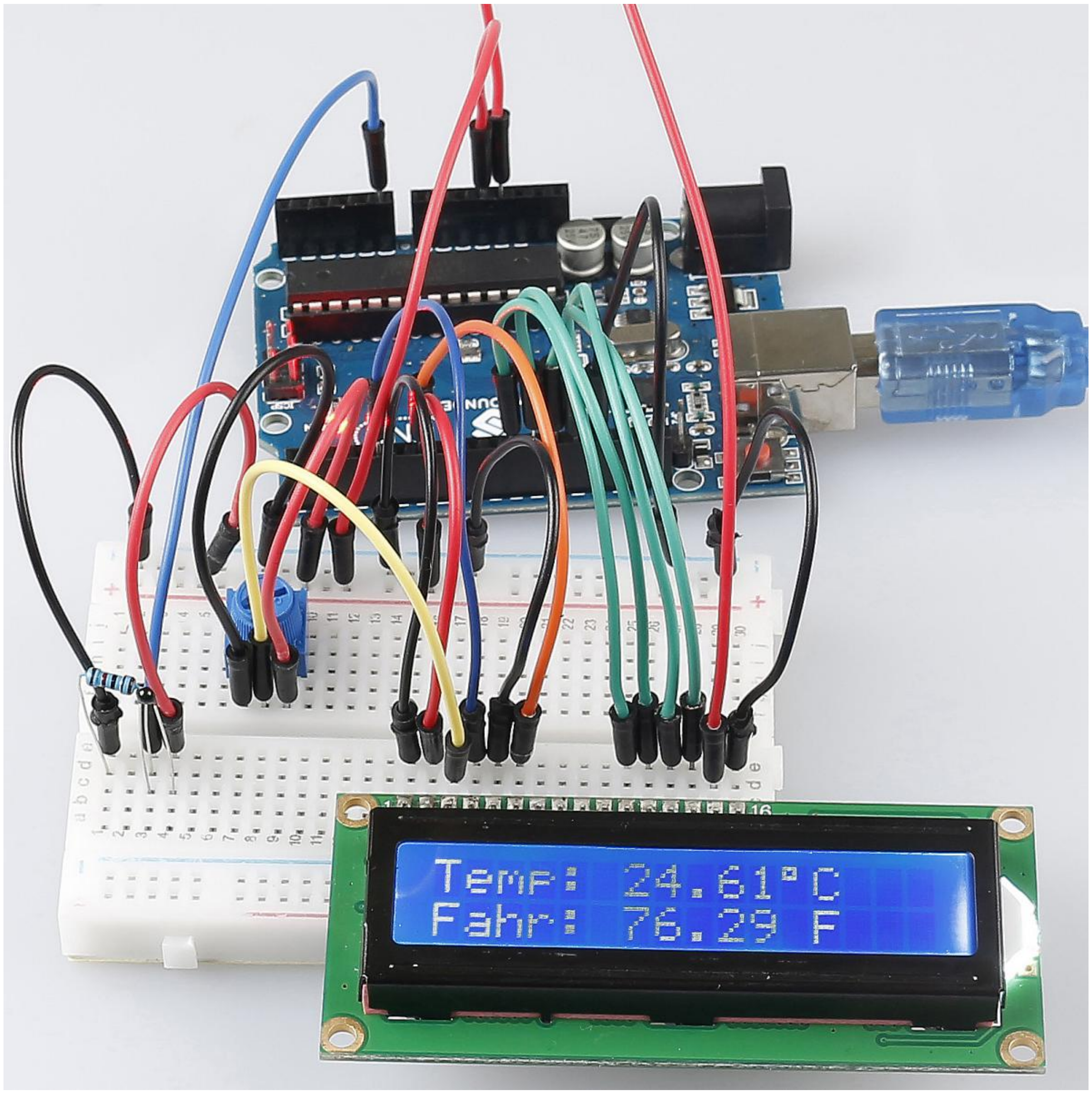
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, you can see the current temperature displayed both in Celsius and Fahrenheit degrees on the LCD1602.



Code

Code Analysis 14-1 Set the variables

```
#define analogPin A0   // the thermistor attach to

#define beta 3950      // the beta of the thermistor

#define resistance 10  // the value of the pull-up resistor
```

Define the beta coefficient as 3950, which is described in the datasheet of thermistor.

Code Analysis 14-2 Get the temperature

```
long a = analogRead(analogPin);
// Read the resistance value of the thermistor to a via the signal from the analog␣
↪pin.
// Here use a long type to make the value of a to be a long integer.

float tempC = beta / (log((1025.0 * 10 / a - 10) / 10) + beta / 298.0) - 273.0;
// The formula here is to calculate the temperature in Celsius, which we deduced␣
↪previously.

float tempF = 1.8 * tempC + 32.0;
// define the temperature in Fahrenheit. As we know Fahrenheit equals to 1.8 *␣
↪Celsius + 32.
```

Code Analysis 14-3 Display the temperature on LCD1602

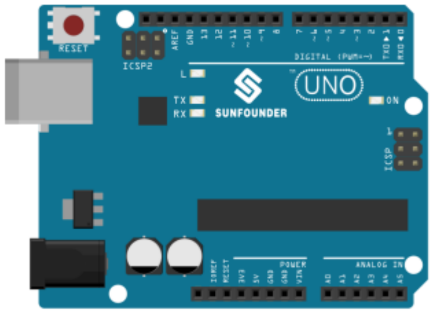
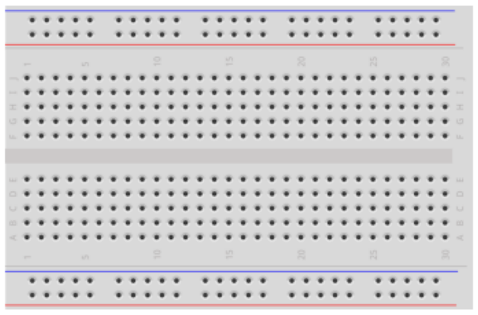
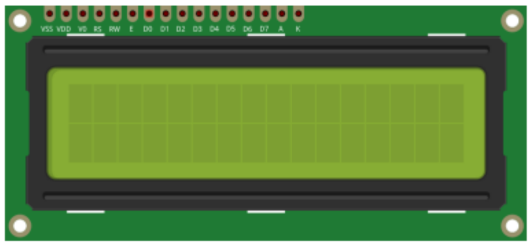
```
lcd.setCursor(0, 0);   // set the cursor to column 0, line 0

lcd.print("Temp: ");   // Print a message of "Temp: "to the LCD.

lcd.print(tempC);      // Print the tempC value on display.

lcd.print(char(223));  // print the unit" ° "

lcd.print("C");

// (note: line 1 is the second row, since counting begins with 0):

lcd.setCursor(0, 1);   // set the cursor to column 0, line 1

lcd.print("Fahr: ");

lcd.print(tempF);    // Print a Fahrenheit temperature to the LCD.

lcd.print(" F");       // Print the unit of the Fahrenheit temperature to the LCD.

delay(200);           // wait for 100 milliseconds
```

Lesson 15 Voltmeter

Introduction

In this lesson, we will use two potentiometers and an LCD1602 to make a DIY voltmeter.

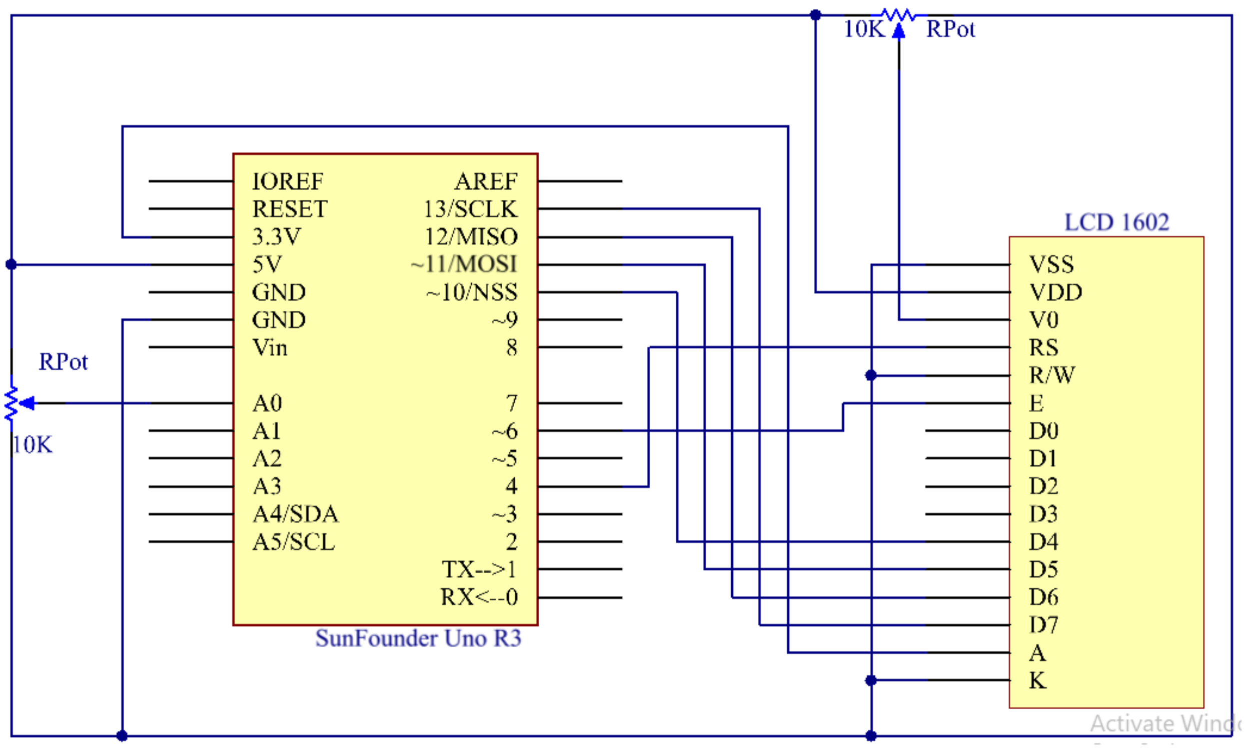
Components

1 * Uno Board 	1 * Breadboard 
1 * USB Cable	Several Jumper Wires
2 * Potentiometer (10kΩ)	1 * LCD1602 

Experimental Principle

Here one potentiometer is used to adjust the contrast of the LCD1602 and the other to divide voltage. When you adjust the potentiometer connected to pin A0 of the SunFounder Uno board, the resistance of the potentiometer will change and the voltage at pin A0 will change accordingly. This voltage change is converted into digital values by A/D converter on the SunFounder Uno board. We can see this change on the serial monitor. Then convert the digital values into voltage with the following formula: the voltage equals the digital value divides by 1024 and then multiplies by 5.0. Finally, display the voltage on the LCD1602.

The schematic diagram:



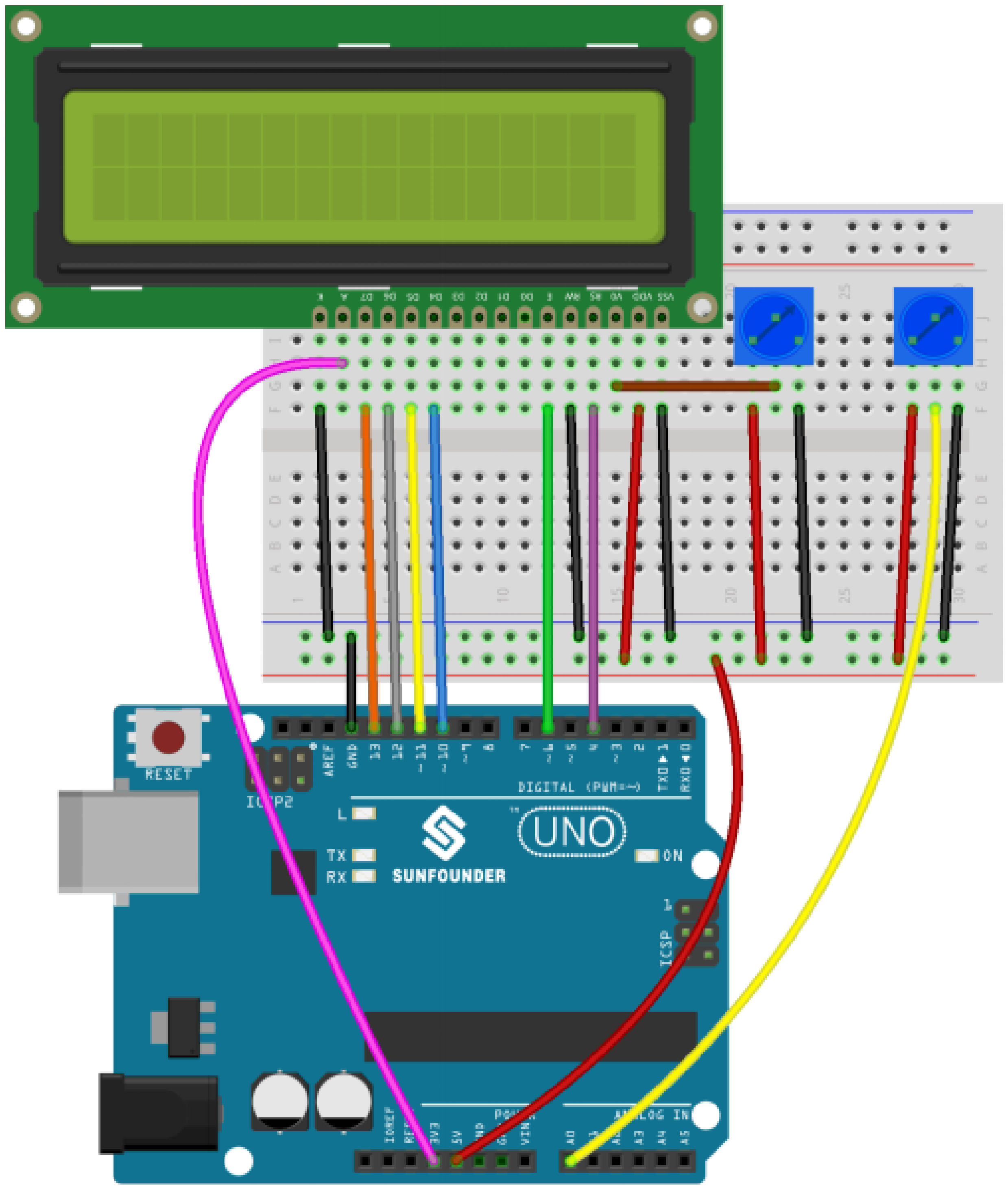
Experimental Procedures

Step 1: Build the circuit

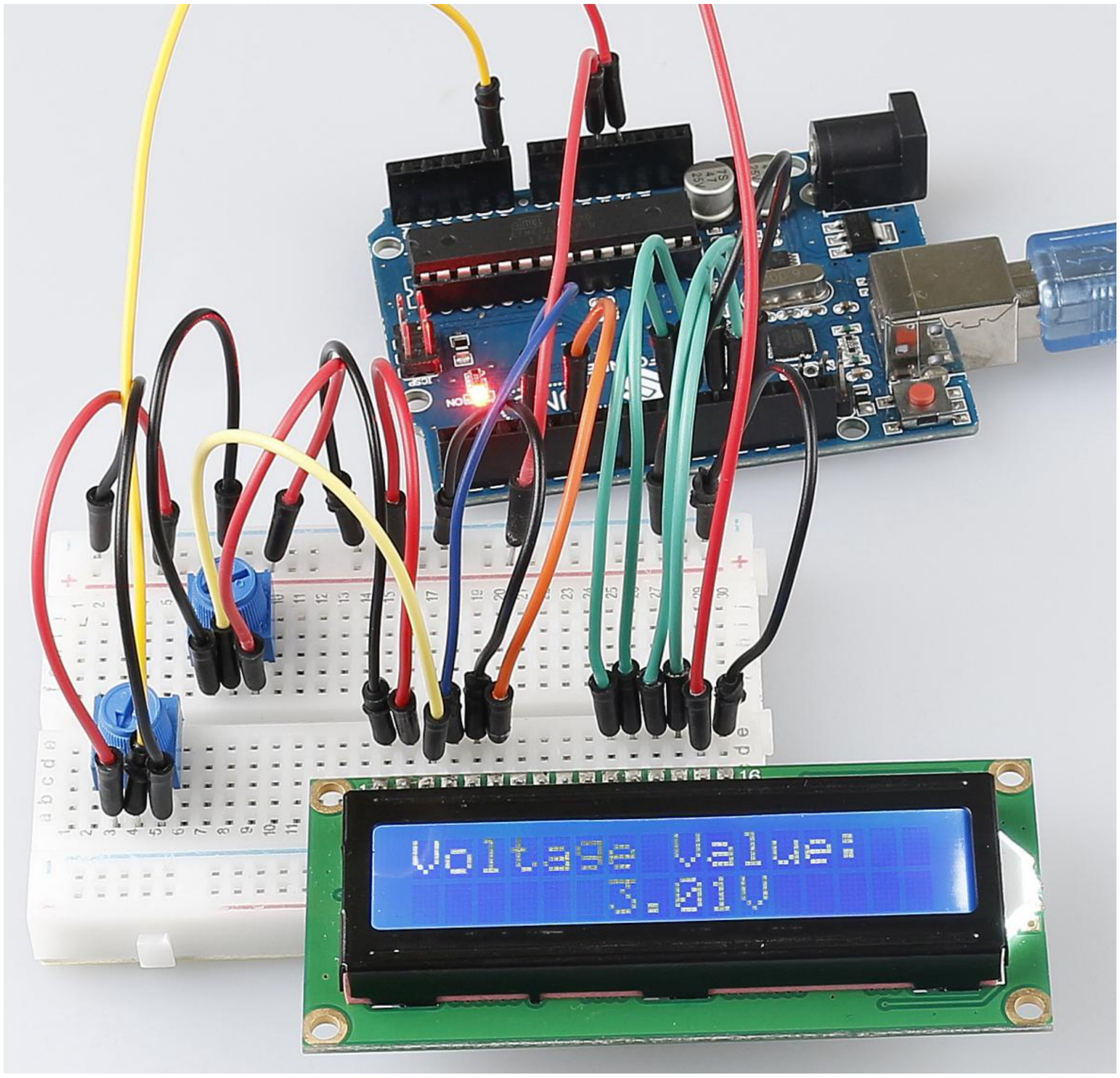
Step 2: Open the code file

Step 3: Select the correct **Board** and **Port**

Step 4: Upload the sketch to the SunFounder Uno board



Now, adjust the potentiometer connected to pin A0, and you will see the voltage displayed on the LCD1602 varies accordingly.



Code

Code Analysis 15-1 Define the pins of LCD1602 andpotentiometer

```
#include <LiquidCrystal.h>

/****************************************************/

float analogIn = 0;                     // store the analog value of A0

LiquidCrystal lcd(4, 6, 10, 11, 12, 13); // lcd(RS,E,D4,D5,D6.D7)

float vol = 0;                          // store the voltage
```

Call the LiquidCrystal library and define the pins of LCD1602 connect to 4,6 and 10 to 13 of Uno board.

Assign the value of A0 to analogIn.

Code Analysis 15-2 Initialize the LCD1602 and serial monitor

```
void setup()

{

    Serial.begin(9600);           // Initialize the serial monitor

    lcd.begin(16, 2);             // set the position of the characters on the LCD as␣
↪Line 2, Column 16

    lcd.print("Voltage Value:"); // print "Voltage Value:"

}
```

Initialize the baud rate of serial monitor to 9600bps and set the position of the characters on the LCD as Line 2, Column 16. Print "Voltage Value: " on the LCD1602.

Code Analysis 15-3 Read the analog of A0 and convert to voltage

```
void loop()

{

    analogIn = analogRead(A0); // Read the value of the potentiometer to val

    vol = analogIn/1024*5.0;      // Convert the data to the corresponding voltage␣
↪value in a math way

    Serial.print(vol);            // Print the number of val on the serial monitor

    Serial.println("V");        // print the unit as V, short for voltage on the␣
↪serial monitor

    lcd.setCursor(6,1);       // Place the cursor at Line 1, Column 6. From here the␣
↪characters are to be displayed

    lcd.print(vol);              // Print the number of val on the LCD

    lcd.print("V");             // Then print the unit as V, short for voltage on the␣
↪LCD

    delay(200);                 // Wait for 200ms

}
```

The analog value of A0 is: Analog value=5/VA0 * 1024, so VA0= Analog value/1024 * 5, if you connect the potentiometer to 3.3v, then modify 5V to 3.3V.

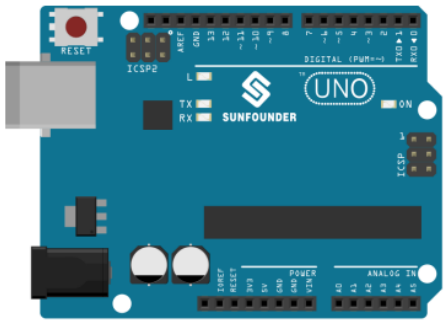
Print the voltage to serial monitor or the LCD1602.

Lesson 16 Automatically Tracking Light Source

Introduction

In this lesson, we will make some interesting creations – use a servo motor, a photoresistor and a pull-down resistor to assemble an automatically tracking light source system.

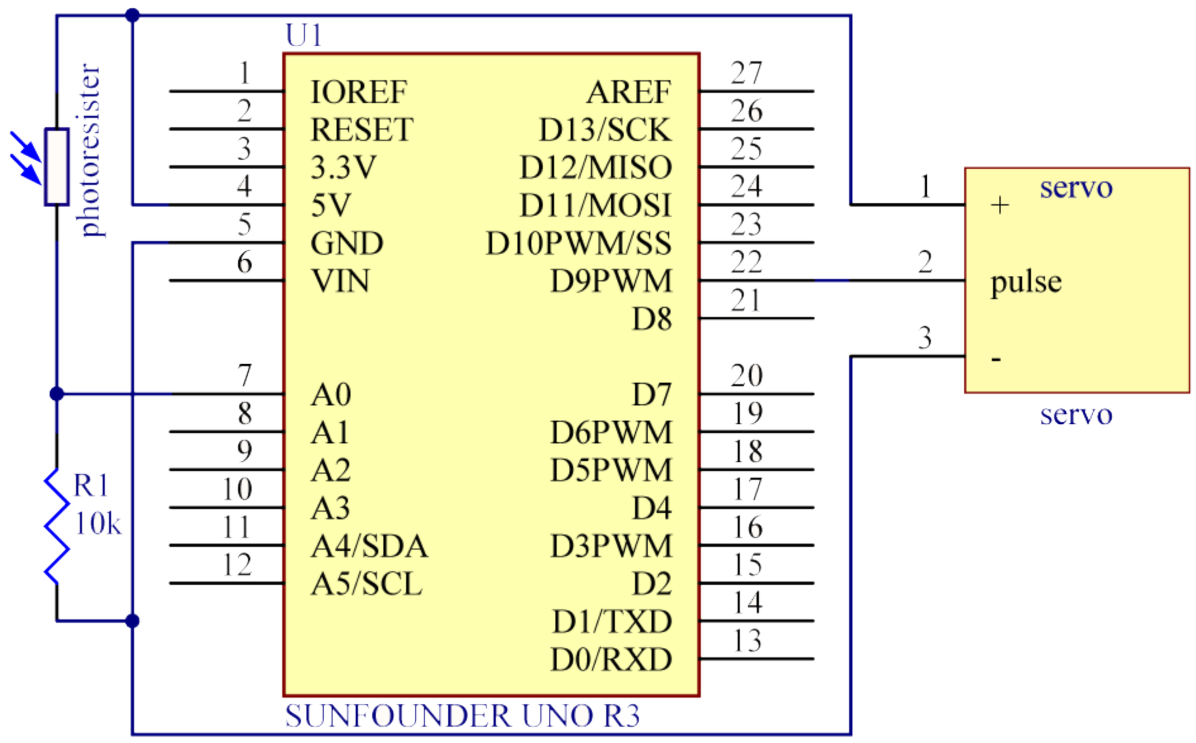
Components

1* Uno Board 	1 * Photo resistor 1 * Resistor (10KΩ)	1 * Servo
1 * USB Cable	Several Jumper Wires	

Experimental Principle

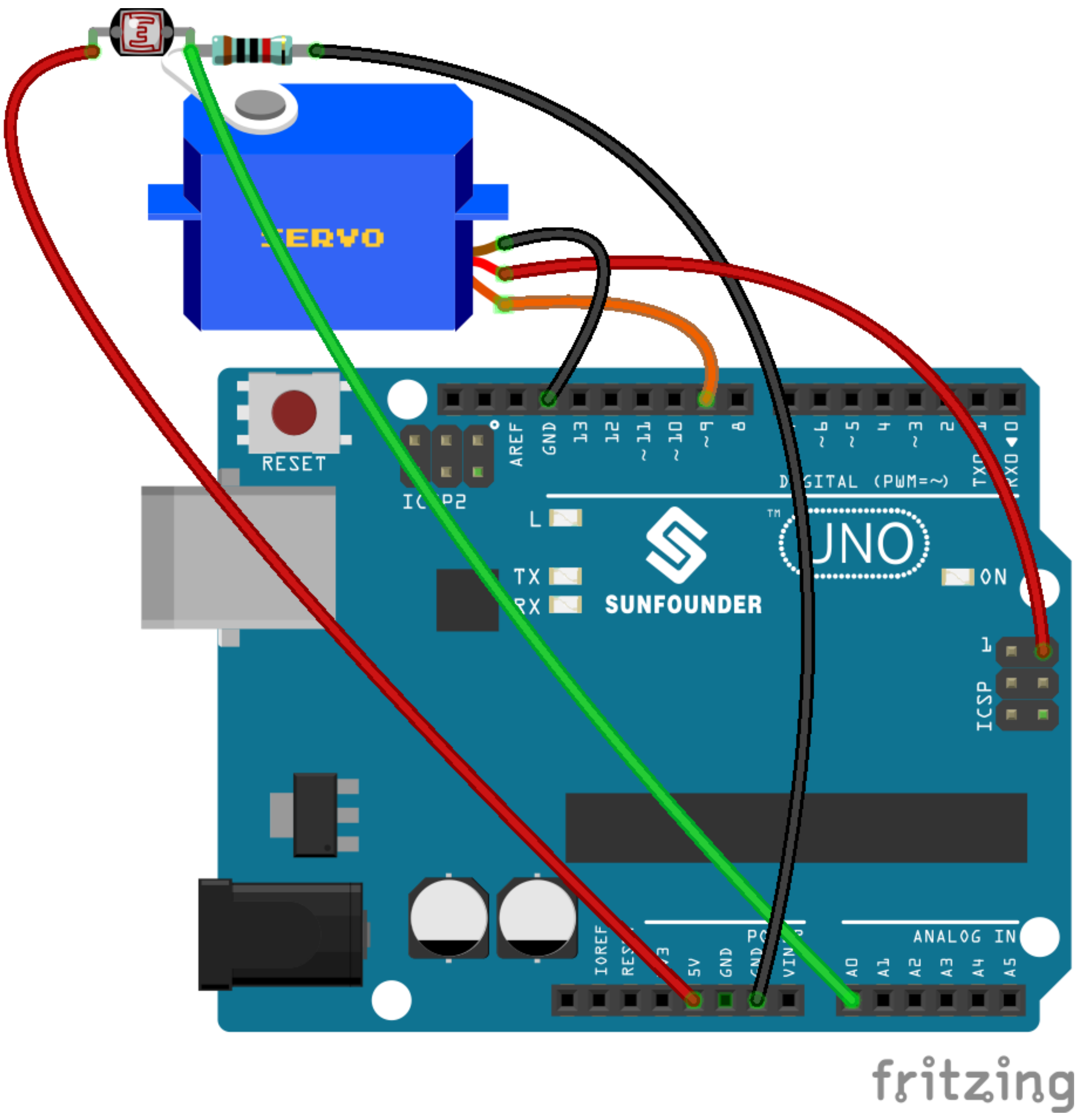
The rocker arm of the servo and the bundled photoresistor sway together to scan and "look" for light source within 180 degrees and record the location of light source when finding one. Then they stop swaying just at the direction of the light source.

The schematic diagram:



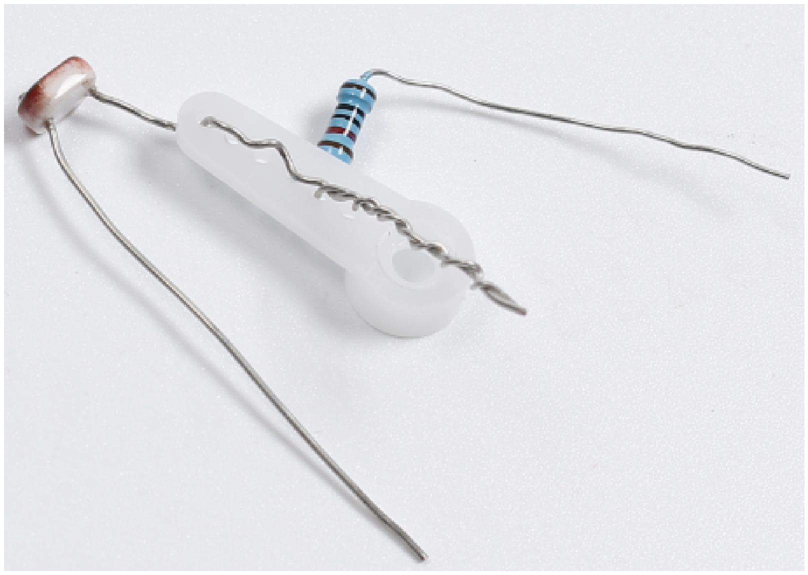
Experimental Procedures

Step 1: Build the circuit



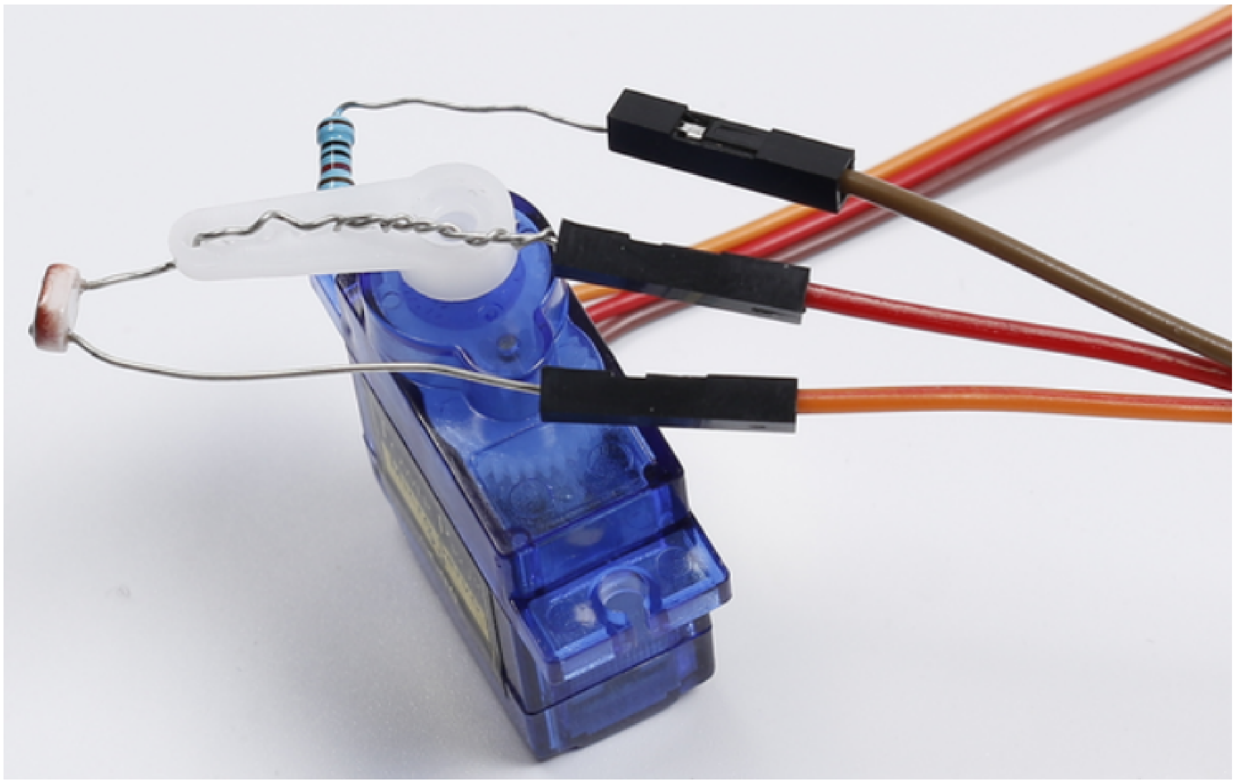
Note: you need to bind one end of the resistor and photoresistor to the rocker arm of the servo (cross the pin through the holes of the arm).

1) Insert one pin of the photoresistor and 10 resistor through the holes on the rocker arm. Pay attention here to tightly winding them because you need to make sure they are connected in the circuit.

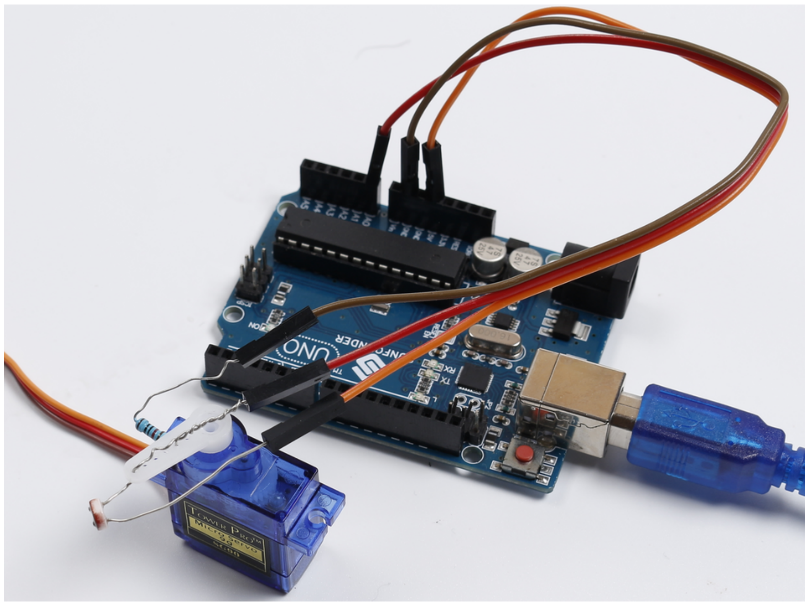


2) Plug in the rock arm to the servo and use 3 jumper wires to hook up the 3 pins.

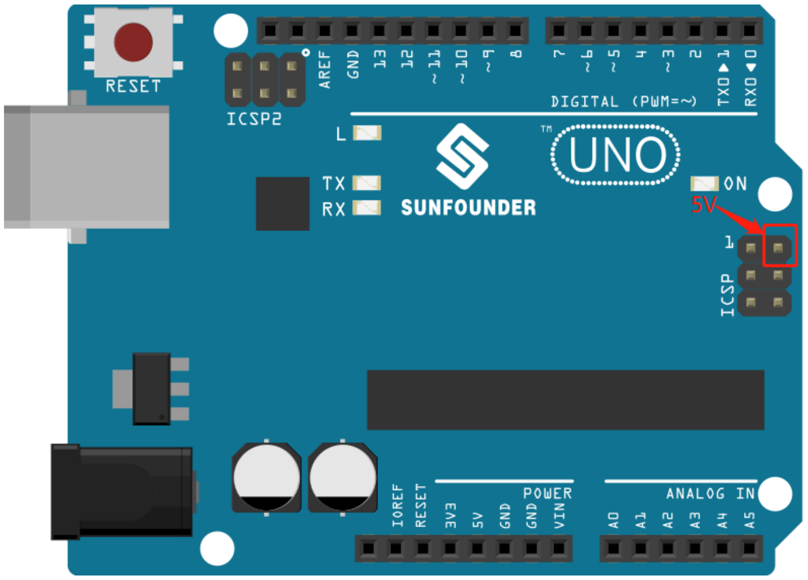
pay attention to plug the pin tightly in case of disentanglement.



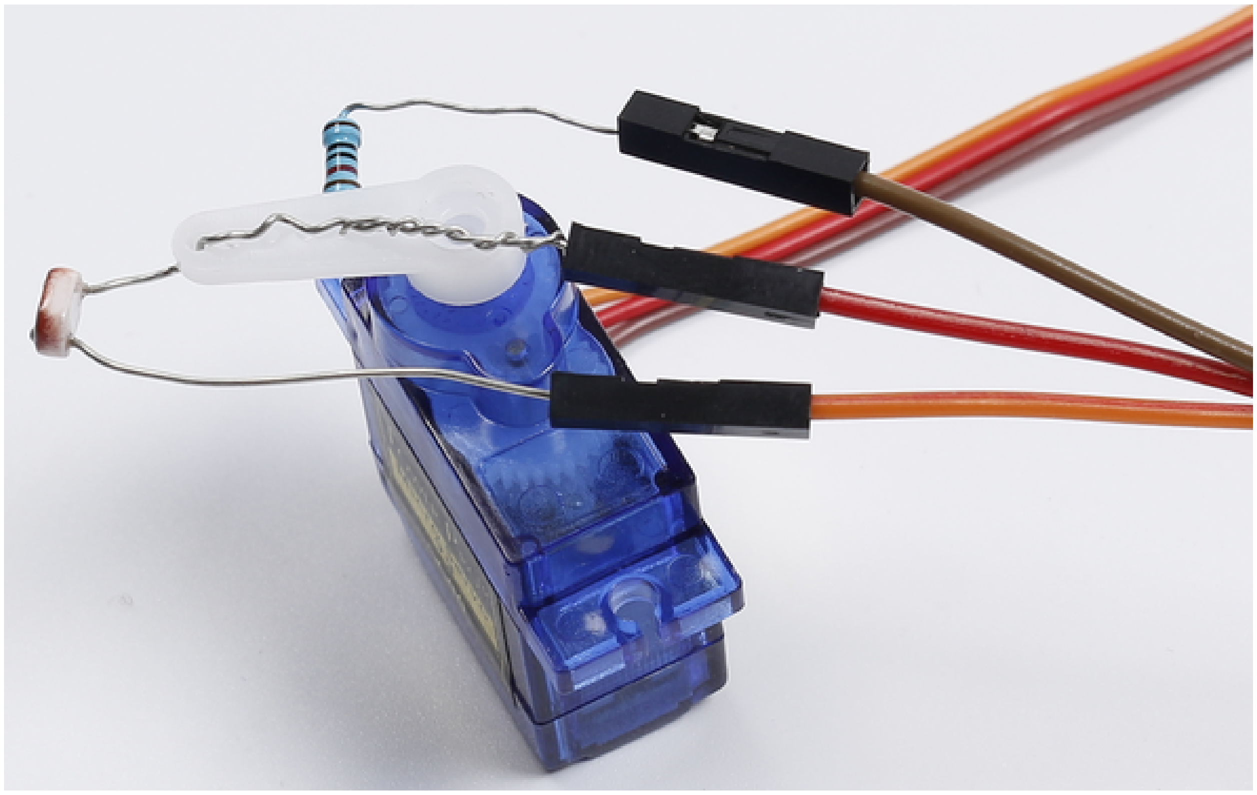
3) Hook up the middle pin to pin A0 of the Uno board, another pin of the 10k resistor to GND, photoresistor to 5V.



4) Connect the brown wire of servo to GND and red to 5v. Since the 5v usually used is occupied already, you need to connect the other 5v as the following picture shows.



5) Next connect the orange wire to pin 9 of the SunFounder Uno board. OK now the circuit is completed! Connect the Uno board to your computer with a USB cable.



Step 2: Open the code file

Step 3: Select the correct **Board** and **Port**

Step 4: Upload the sketch to the SunFounder Uno board

Now, shine a flashlight onto the photoresistor. Then you will see the rocker arm of the servo and the photoresistor rotate and finally stop at the direction of light source.

Code

Code Analysis 16-1 Initialize and define variables

```
#include <Servo.h>

const int photocellPin = A0; //The photoresistor is connected to A0

/**************************************************/

Servo myservo;//create servo object to control a servo

int outputValue = 0; //Save the value read from A0

int angle[] = {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150,␣
↪160, 170, 180};
//Define the angle of servo

int maxVal = 0; //Record the maximum number
```

(continues on next page)

(continued from previous page)

```
int maxPos = 0; //Record the angle of the servo when the read the maximum number of␣
↪photoresistor.
```

Define an integer array angle[], which contains 19 elements from 0 to 18, representing 0 to 180 which indicates the degree of servo rotation. For example, angle[0] means 0 degree, angle[1] is 10 degrees, and so forth.

Code Analysis 16-2 Servo rock arm stop at the direction of light source

```
void loop()

{

    for(int i = 0; i < 19; i ++)

    {

        myservo.write(angle[i]);
        //write the angle from the angle[i] array to servo.
        //When i=0, angle[0]=0, i=1, angle[1]=10, and so on.

        outputValue = analogRead(photocellPin); //read the value of A0

        Serial.println(outputValue); //print it

        if(outputValue > maxVal)
        //if the current value of A0 is greater than previous

        {

            maxVal = outputValue; // write down the value

            maxPos =i; // write down the angle

        }

        delay(200); // delay 200ms

    }

    myservo.write(angle[ maxPos]);
    // write the angle to servo which A0 has greatest value

    delay(1000); //delay 1s

}
```

Set the servo to rotate from 0 to 180, and the angle is defined in the angle [] array. Since the photoresistor is wound with the servo rock arm, the resistance of the photoresistor changes with different light intensities each time the servo is rotated, so the analog value of A0 is changed at the same time.

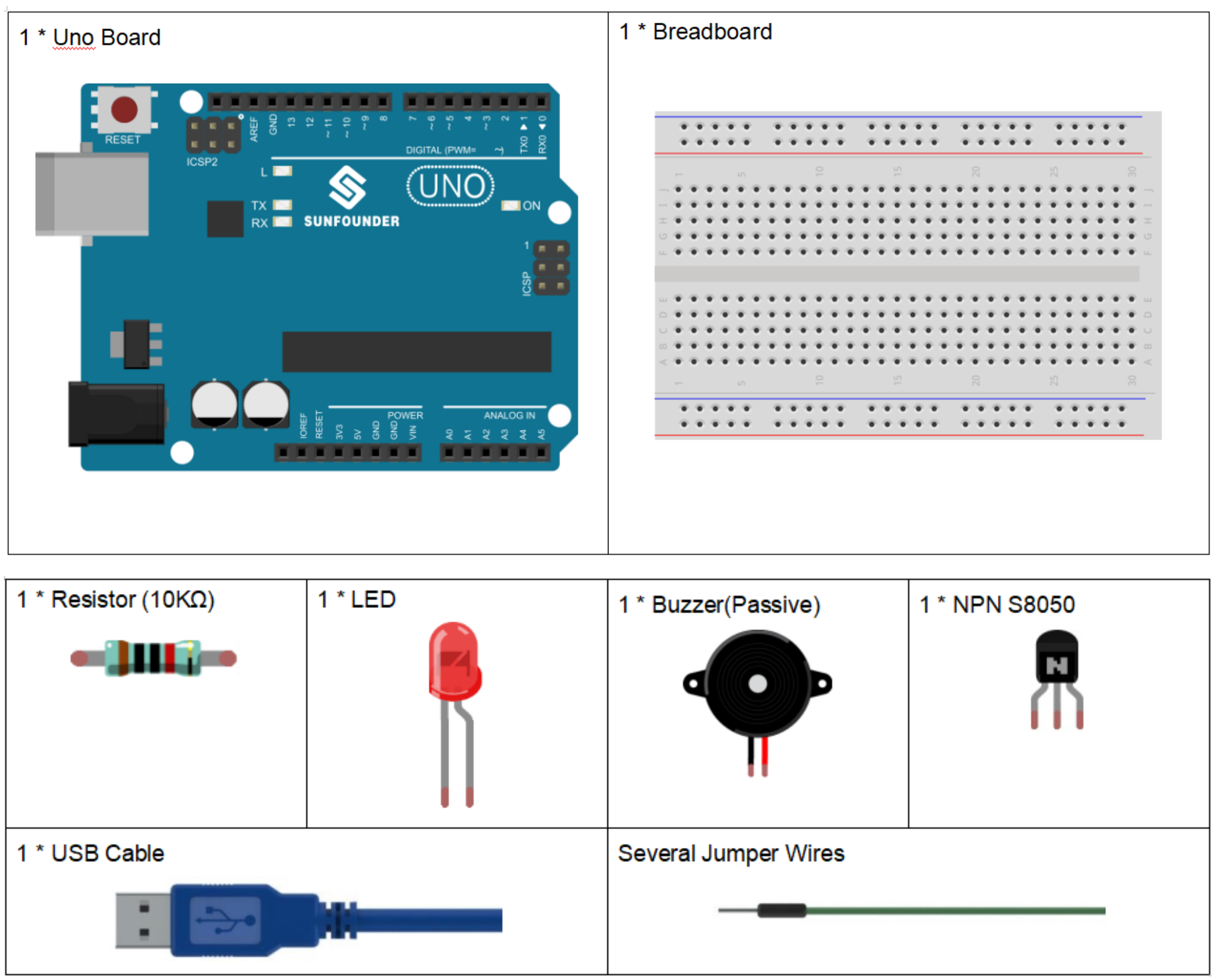
By comparing the value of A0 with the previously recorded maximum value, record the maximum A0 value and current angle of the servo. Finally let the servo turn to this angle.

Lesson 17 Light Alarm

Introduction

This experiment is a very interesting one – a DIY phototransistor. DIY phototransistors use the glow effect and photoelectric effect of LEDs. That is, LEDs will generate weak currents when some light is shined on it. And we use a transistor to amplify the currents generated, so the SunFounder Uno board can detect them.

Components



1 * Uno Board	1 * Breadboard

1 * Resistor (10KΩ)	1 * LED	1 * Buzzer(Passive)	1 * NPN S8050

1 * USB Cable	Several Jumper Wires

Experimental Principle

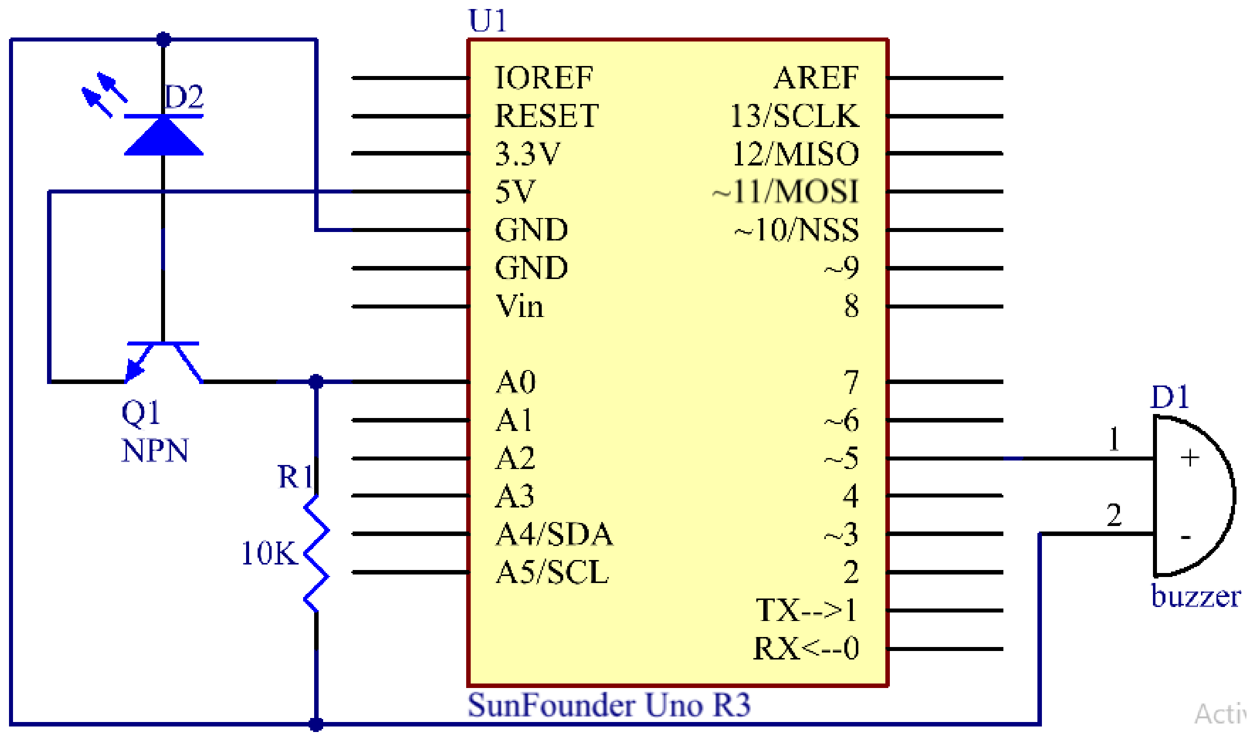
LEDs not only have a glow effect, but also a photoelectric effect. They will generate weak currents when exposed to light waves.

NPN consists of a layer of P-doped semiconductor (the "base") between two N-doped layers (see the picture above). A small current entering the base is amplified to produce a large collector and emitter current. That is, when there is a positive potential difference measured from the emitter of a NPN transistor to its base (i.e., when the base is high relative to the emitter) as well as positive potential difference measured from the base to the collector, the transistor becomes active. In this "on" state, current flows between the collector and emitter of the transistor.

A 10k pull-down resistor is attached to the transistor output stage in order to avoid analog port suspending to interfere with signals and cause misjudgment.

When the LED exposed to light waves, the LED generates weak currents, then the NPN transistor becomes active. Then read the analog value of A0, when A0>0, then set pin 5(buzzer) to high level.

The schematic diagram:



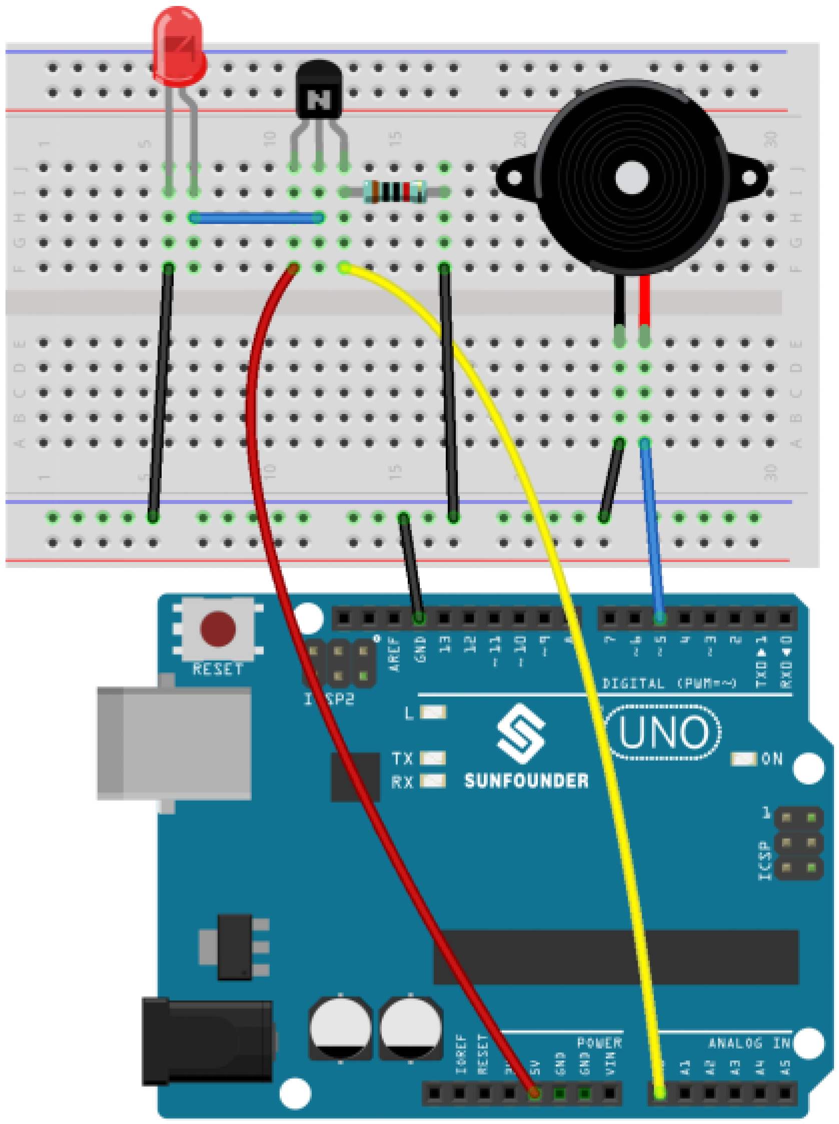
Experimental Procedures

Step 1: Build the circuit.

Step 2: Open the code file

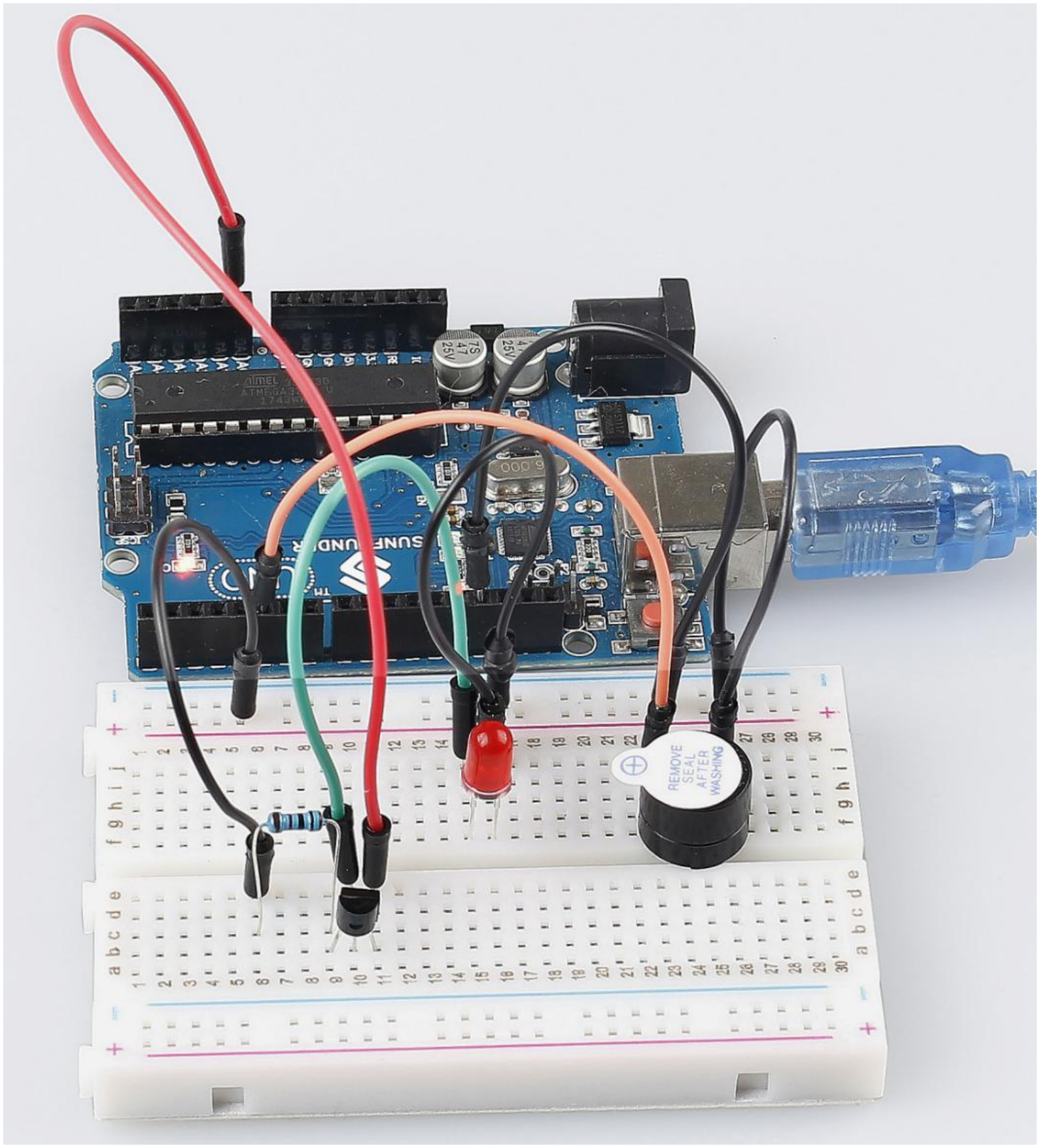
Step 3: Select the correct Board and Port

Step 4: Upload the sketch to the board



Now, you can hear the buzzer beep when shining a flashlight on the LED.

Note: You need to do this experiment in a dark environment, or the lights you give need to be much stronger than ambient light.



Code

Code Analysis 17-1 Whole Code

```
void setup()

{

    Serial.begin(9600);    // start serial port at 9600 bps:

}

void loop()

{

    int n=analogRead(A0); //read the value from analog pin AO

    Serial.println(n);

    if(n>0) //If there is a voltage

    {

        pinMode(5,OUTPUT); //set the digital pin 5 as an output

        tone(5,10000);
        //Generates a square wave of the frequency of 10000 Hz (and 50% duty cycle)␣
↪on pin 5

        pinMode(5,INPUT); //set the pin 5 as an input

    }

}
```

tone()

Description

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

Syntax

tone(pin, frequency)

tone(pin, frequency, duration)

Parameters

pin: the pin on which to generate the tone

frequency: the frequency of the tone in hertz - unsigned int

duration: the duration of the tone in milliseconds (optional) - unsigned long

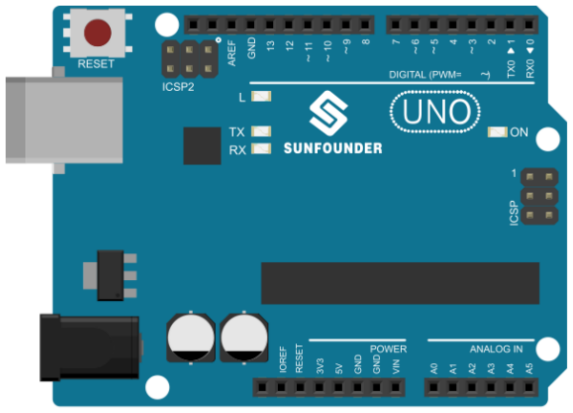
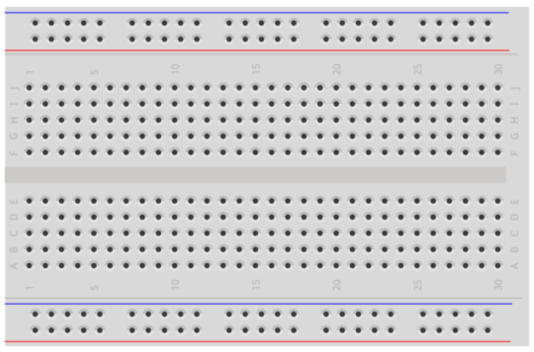
Lesson 18 Answer Machine

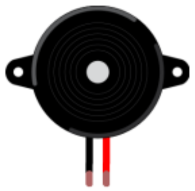
Introduction

In quiz shows, especially entertainment activities (e.g. competitive answering activities), organizers often apply a buzzer system in order to accurately, fairly and visually determine the seat number of a responder.

Now the system can illustrate the accuracy and equity of the judgment by data, which improves the entertainment. At the same time, it is more fair and just. In this lesson, we will use some buttons, buzzers, and LEDs to make a quiz buzzer system.

Components

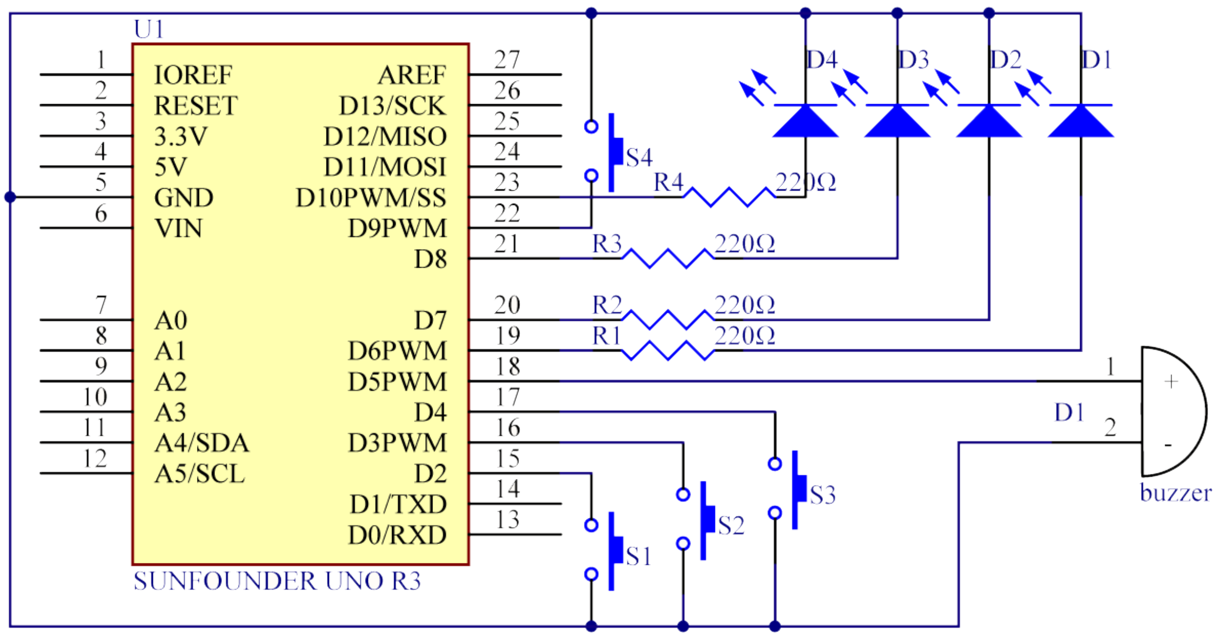
1 * Uno Board  RESET ICSP2 AREF GND DIGITAL (PWM~) L TX RX UNO SUNFOUNDER ON ICSP POWER ANALOG IN	1 * Breadboard 

1 * Red LED	1 * Yellow LED	1 * Green LED	1 * Blue LED
1 * Active Buzzer 	4 * Button 	4 * Resistor (220Ω) 	
1 * USB Cable 	Several Jumper Wires 		

Experimental Principle

Button 1, 2 and 3 are answer buttons, and button 4 is the reset button. If button 1 is pressed first, the buzzer will beep, the corresponding LED will light up and all the other LEDs will go out. If you want to start another round, press button 4 to reset.

The schematic diagram:



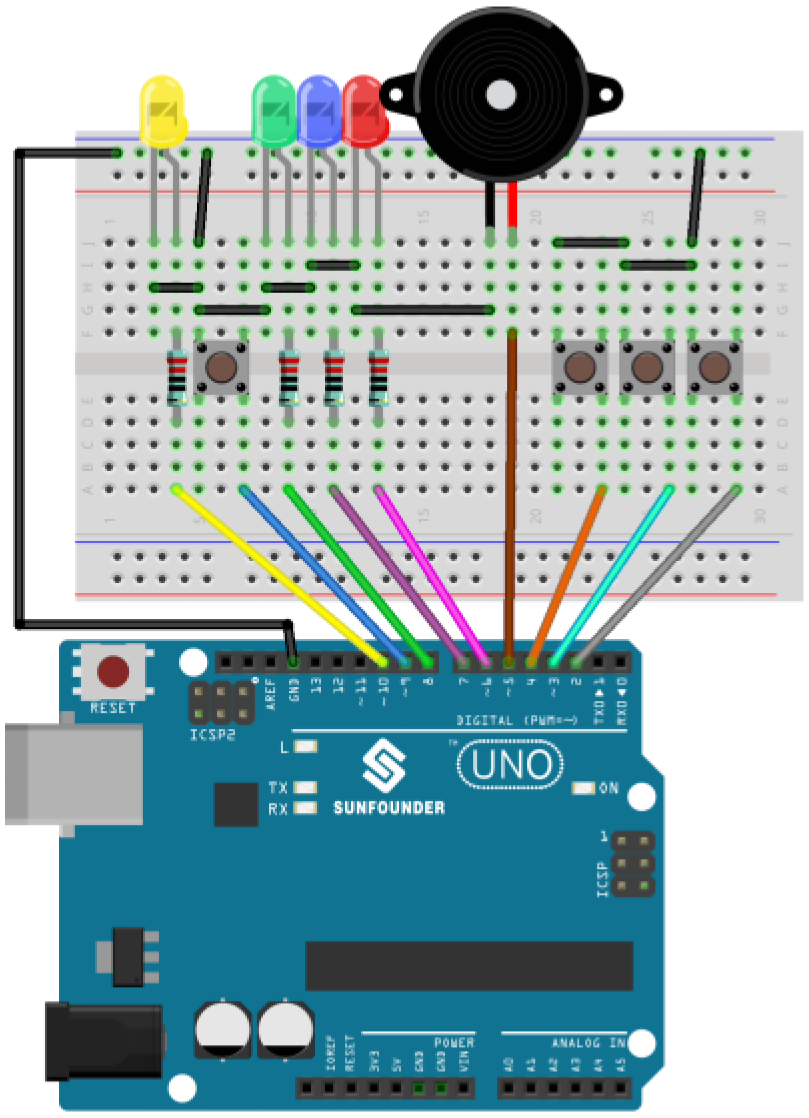
Experimental Procedures

Step 1: Build the circuit

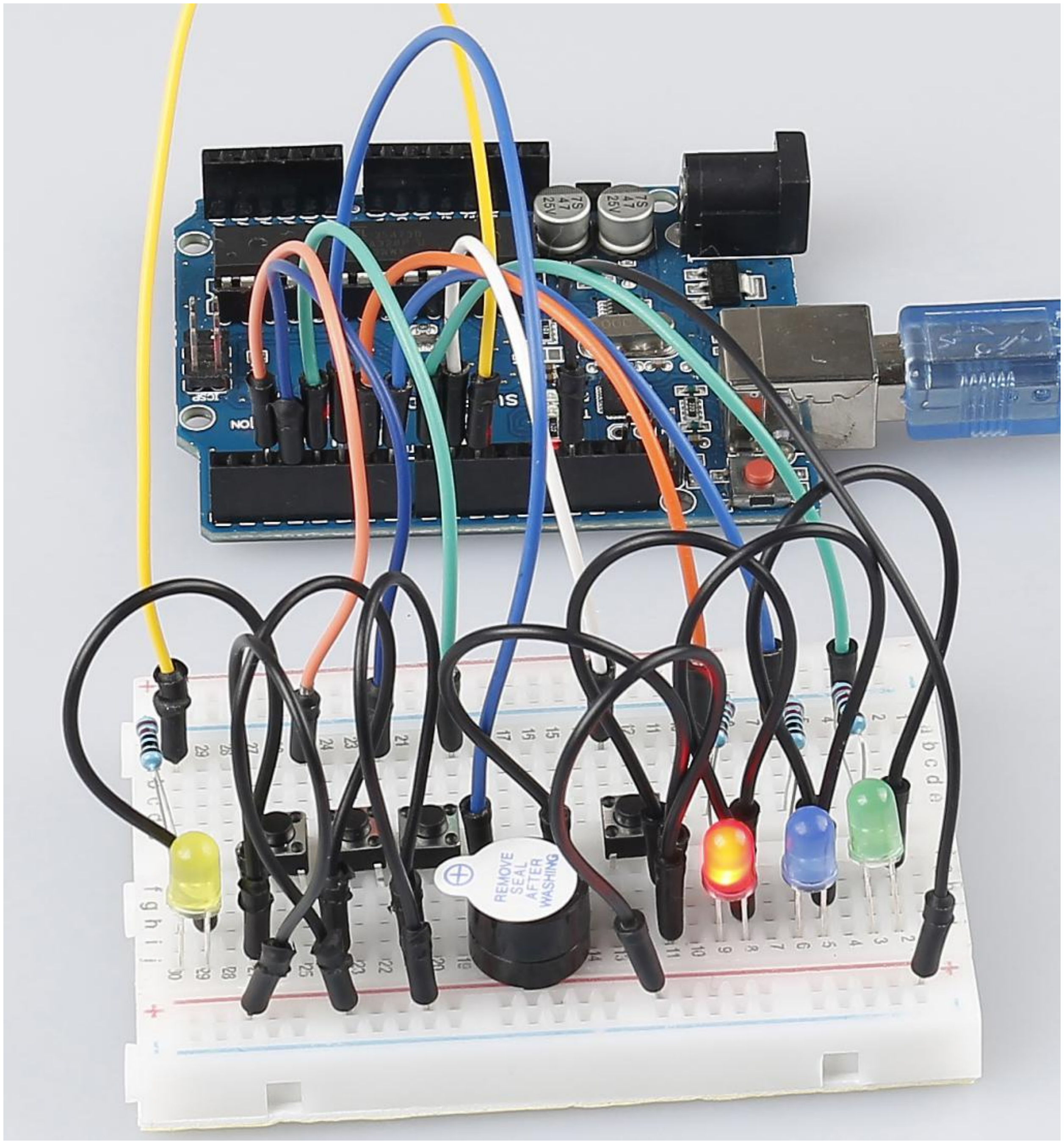
Step 2: Open the code file.

Step 3: Select the Board and Port.

Step 4: Upload the sketch to the board.



Now, first press button 4 to start. If you press button 1 first, you will see the corresponding LED light up and the buzzer will beep. Then press button 4 again to reset before you press other buttons.


REMOVE SEAL AFTER WASHING

Code

The code for this experiment may be a bit long. But the syntax issimple. Let's see. **Workflow:** Read the state of button 4, if button 4 is pressed, theLED on pin 10 is illuminated while reading the state of the remaining buttons. If one of the buttons is detected to be pressed, the buzzer beeps and lights the corresponding LED until button 4 is pressed again.

Code Analysis 18-1 loop() function

```
b4State = digitalRead(button4); // read the value of button4 to see if it was pressed.

Serial.println(b4State); // print it value.

//when button4 pressed

if(b4State == 0) // if the button4 is pressed, the b4State=0

{

    if(b4State == 0) /*confirm that the button4 is pressed. One pin of the
    button is connected to pin 9, the other pin is connected to GND, and
    when the button is pressed, pin 9 is pulled low.*/

    {

        flag = 1; // if so,flag is 1

        digitalWrite(LED4, HIGH); // turn the reset LED on

        delay(200); // delay 200ms

    }

}

if(1 == flag)

{

    // read the state of other buttons

    b1State = digitalRead(button1);

    b2State = digitalRead(button2);

    b3State = digitalRead(button3);

    // If the button1 press the first

    if(b1State == 0) // if button1 is pressed

    {

        flag = 0; // flag equals to 0

        digitalWrite(LED4, LOW);
```

(continues on next page)

(continued from previous page)

```
        Alarm(); // buzzer sound

        digitalWrite(LED1,HIGH); // turn the LED1 on only

        digitalWrite(LED2,LOW);

        digitalWrite(LED3,LOW);

        while(digitalRead(button4)); // detect the button4,if pressed,out of the␣
↪while loop

    }

    ...
```

Use the same way to detect the button2 and button3, if one of the buttons is detected to be pressed, the buzzer beeps and lights the corresponding LED until button 4 is pressed again.

Code Analysis 18-2 Alarm() function

```
void Alarm()

{

    for(int i=0;i<100;i++)
    {

        digitalWrite(buzzerPin,HIGH); // the buzzer sound

        delay(2); // delay 2ms

        digitalWrite(buzzerPin,LOW); // without sound

        delay(2); // when delay time changed,the frequency changed

    }

}
```

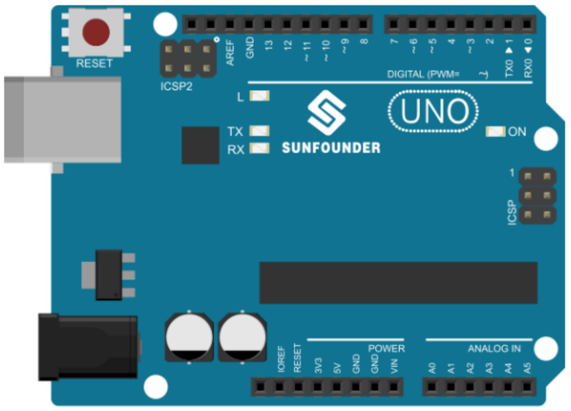
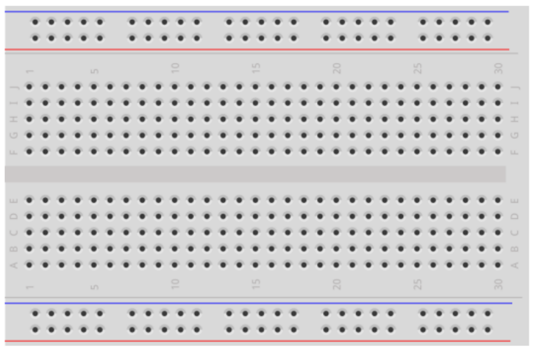
The alarm() function is to set the buzzer to beep. You can change the frequency and time of the buzzer sound.

Lesson 19 Controlling Voice by Light

Introduction

Previously we have learnt how to use a photoresistor. In this lesson, let's get further - control a buzzer to beep in different frequencies by the photoresistor.

Components

1 * Uno Board  RESET ICSP2 AREF GND DIGITAL (PWM= ~) L TX RX SUNFOUNDER UNO ON ICSP POWER ANALOG IN IOREF RESET 3V3 5V GND GND VIN A0 A1 A2 A3 A4 A5	1 * Breadboard 

1 * Active Buzzer	1 * Photoresistor	1 * Resistor (10KΩ)

1 * USB Cable	Several Jumper Wires

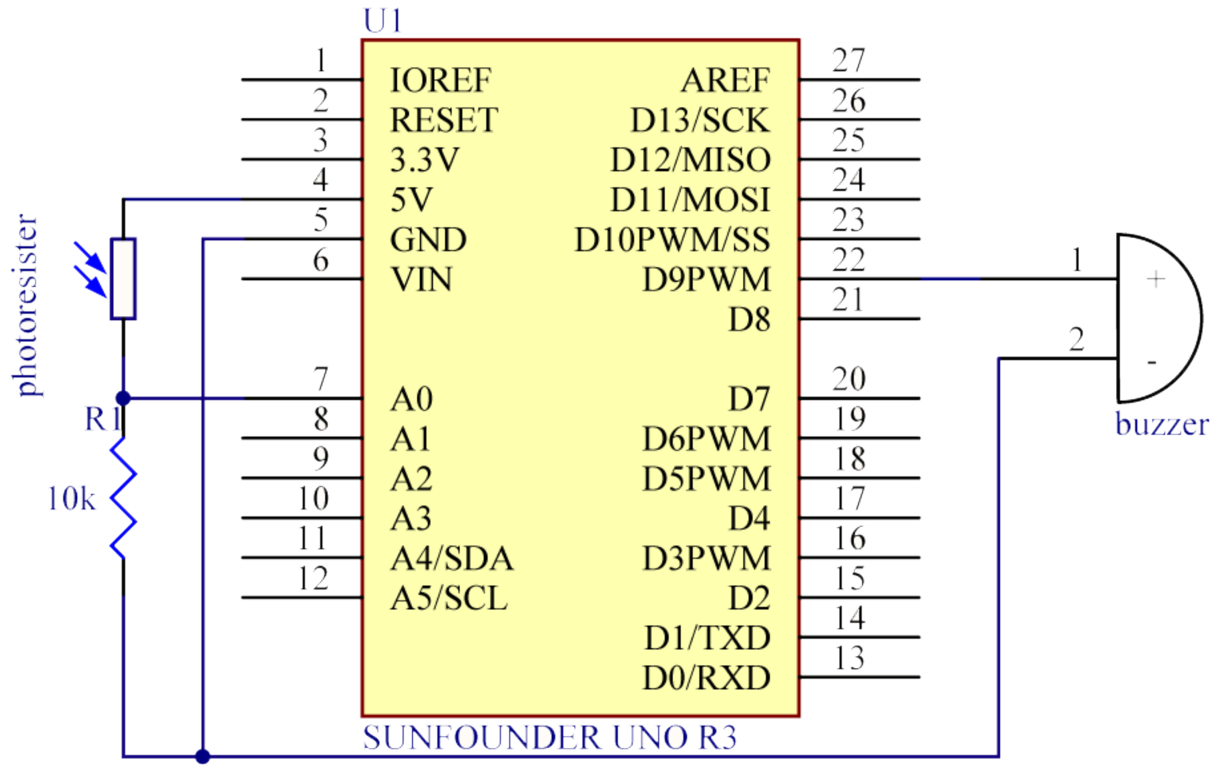
Experiment Principle

When you shine some light on the photoresistor, if the incident light gets stronger, the resistance of the photoresistor will decrease; if the incident light becomes weaker, the resistance will increase. We can apply this principle to change the voltage distribution in the circuit.

In this experiment, the output of the photoresistor is sent to pin A0 on the SunFounder Uno board and then processed by the ADC on the board to output a digital signal. We use this digital signal as the parameter of the delay() function in the sketch to make the buzzer beep.

When the incident light is strong, the output value gets greater, thus the buzzer will beep slowly; when incident light is weak, the output value is smaller, thus the buzzer will beep sharply.

The schematic diagram:



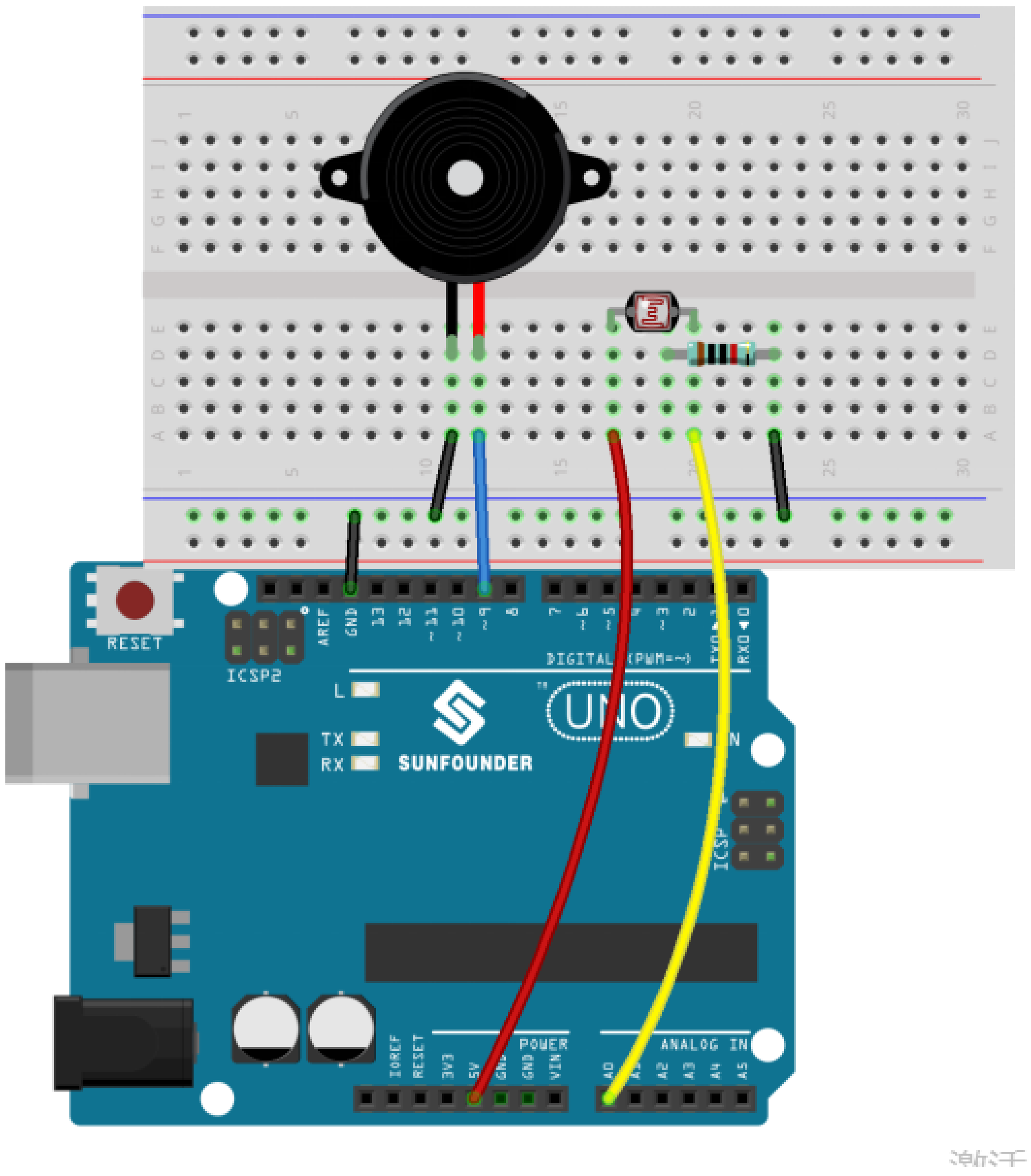
Experiment Procedures

Step 1: Build the circuit

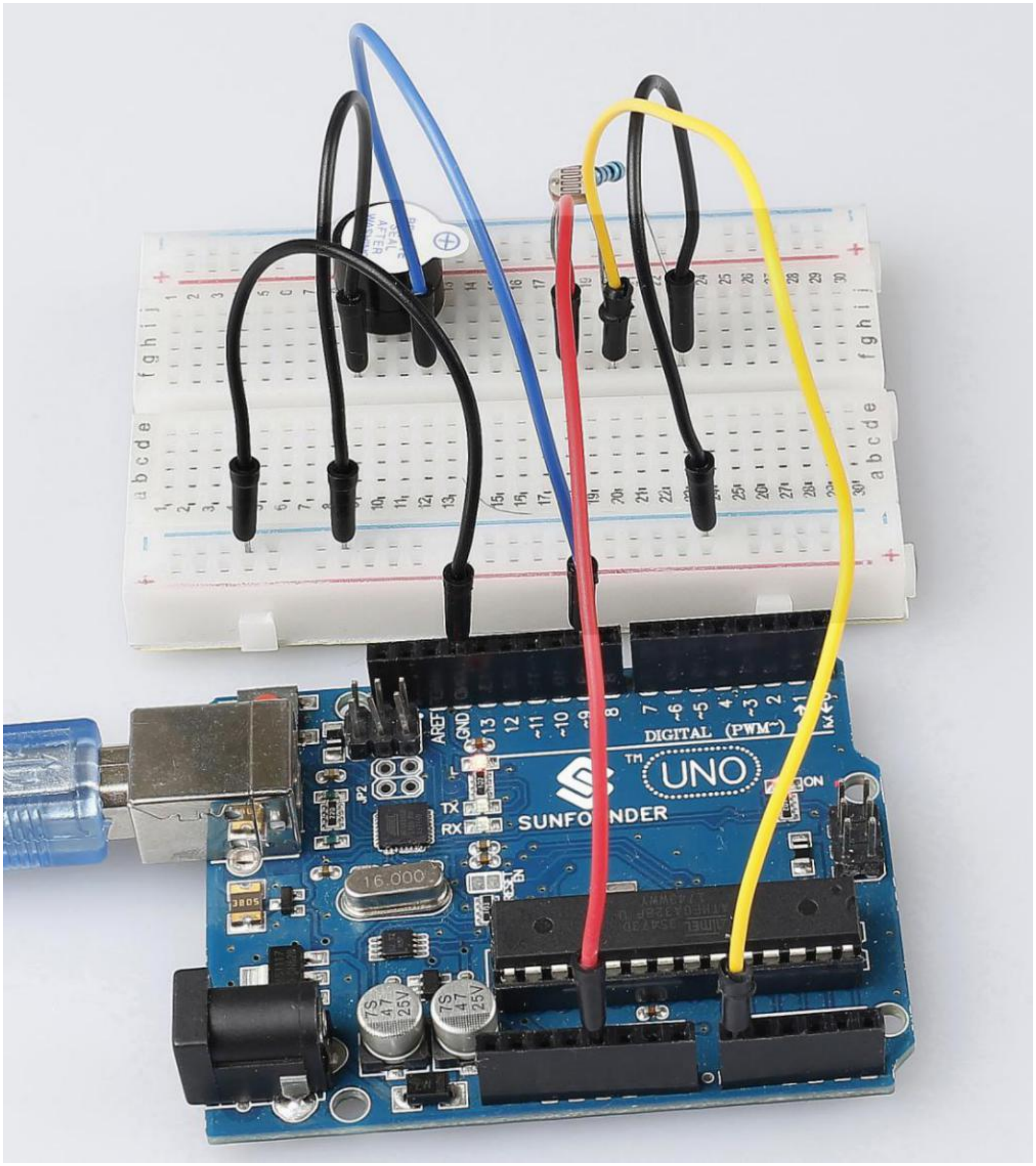
Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.



Now, if you place the photoresistor in a dark environment, the buzzer will beep sharply; if you shine a flashlight on the photoresistor, the buzzer beeping will slow down.



Code

Code Analysis 19-1 Set the array elements

```
void loop()

{

    sensorValue = analogRead(photocellPin); //read the value of A0

    digitalWrite(buzzerPin, HIGH);

    delay(sensorValue); //wait for a while,and the delay time depend on the␣
↪sensorValue

    digitalWrite(buzzerPin, LOW);

    delay(sensorValue);

}
```

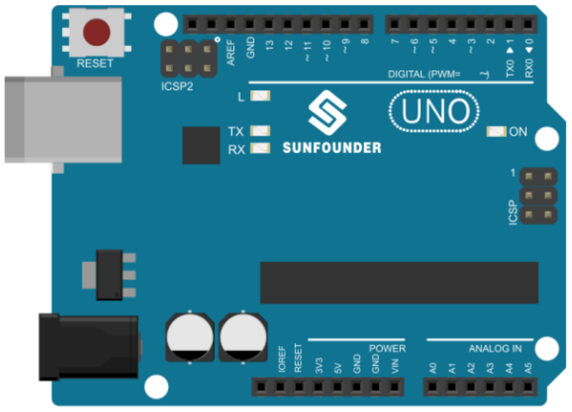
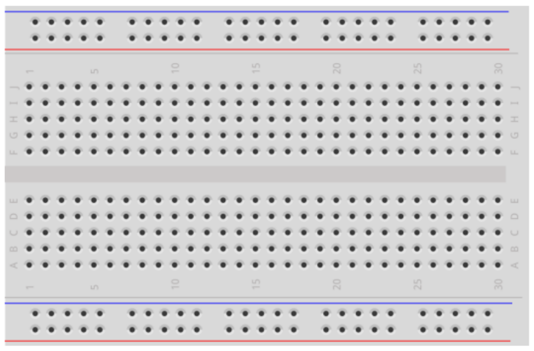
The value of the photoresistor is read, and when the incident light is strong, the output value becomes large. Then set the buzzer to high level to make it beep, delay the **sensorvalue** ms, then turn off the buzzer and also delay the **sensorvalue** ms. So you can see that if you put the photoresistor in a dark environment, the buzzer will make a sharp humming sound; if you illuminate the flashlight on the photoresistor, the buzzer will beep.

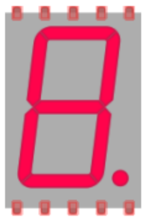
Lesson 20 74HC595

Introduction

Generally, there are two ways to drive a single 7-segment display. One way is to connect its 8 pins directly to eight ports on the Uno board. Or you can connect the 74HC595 to three ports of the Uno board and then the 7- segment display to the 74HC595. In this experiment, we will use the latter. By this way, we can save five ports – considering the Uno board's limited ports, this is very important. Now let's get started!

Components

1 * Uno Board	1 * Breadboard
	

1 * 7-segment display	1 * 74HC595	8 * Resistor (220Ω)
		

1 * USB Cable	Several Jumper Wires

Experimental Principle

7-Segment Display

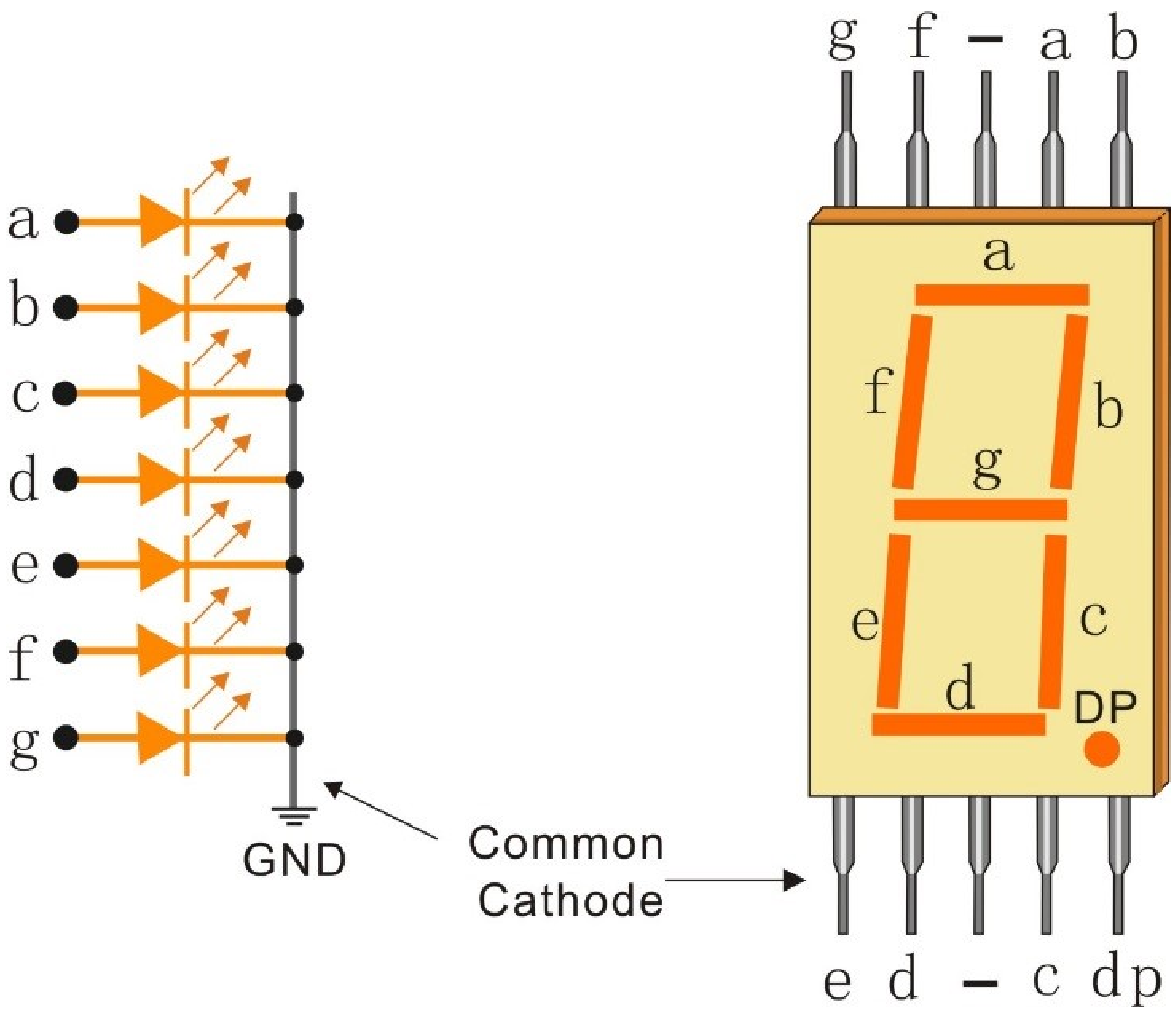
A 7-segment display is an 8-shaped component which packages 7 LEDs. Each LED is called a segment – when energized, one segment forms part of a numeral (both decimal and hexadecimal) to be displayed. An additional 8th LED is sometimes used within the same package thus allowing the indication of a decimal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

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

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

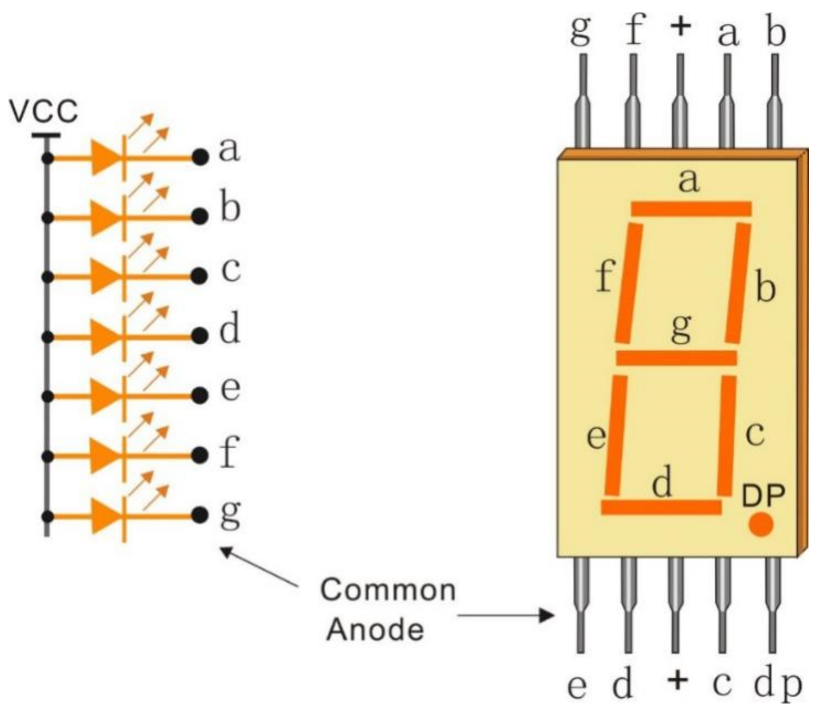
Common Cathode 7-Segment Display

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



Common Anode 7-Segment Display

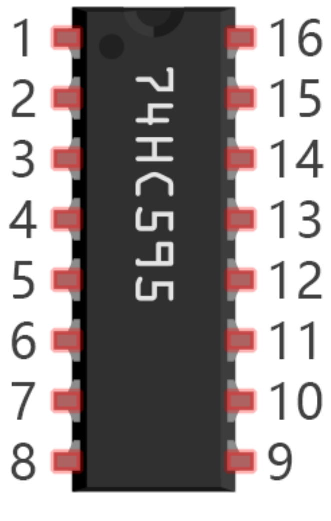
In a common anode display, the anodes of all the LED segments are connected to the logic "1". Then an individual segment (a-g) is energized by a ground, logic "0" or "LOW" signal via a current limiting resistor to the cathode of the segment.

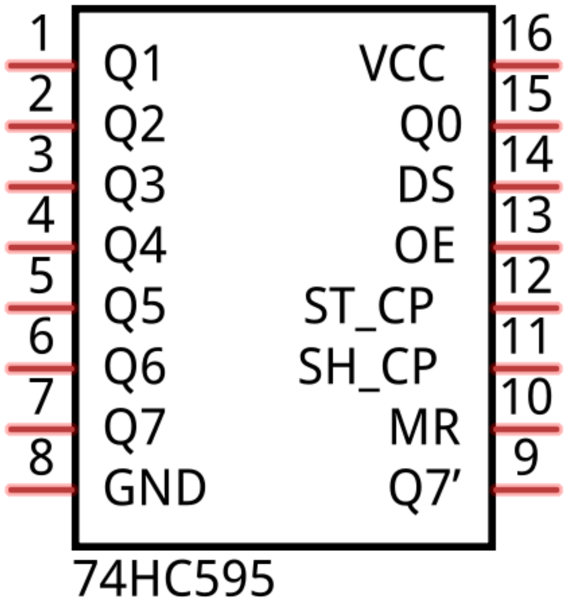


74HC595

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

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





Pins of 74HC595 and their functions:

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

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

MR: Reset pin, active at low level; here it is directly connected to 5V.

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

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

OE: Output enable pin, active at low level. Here connected to GND.

DS: Serial data input pin

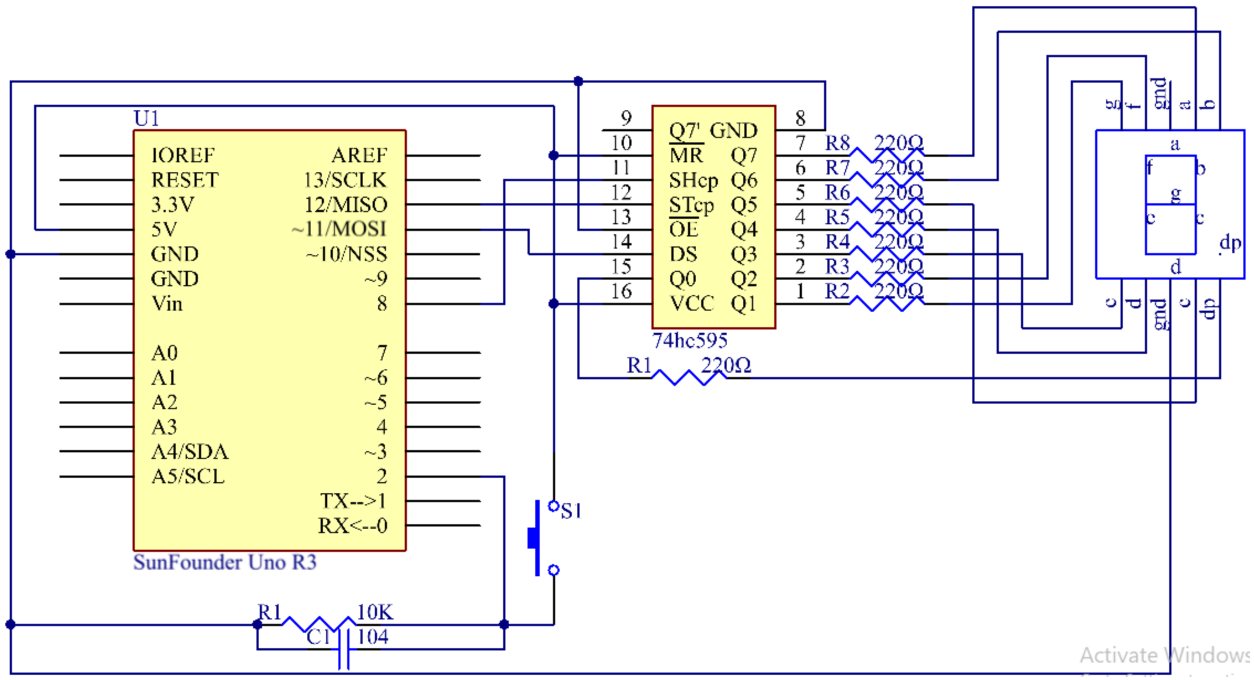
VCC: Positive supply voltage

GND: Ground

Principle:

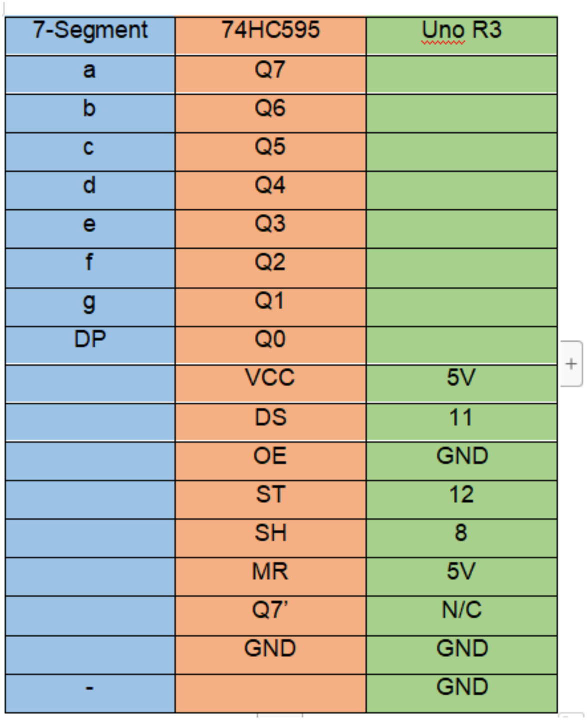
In the experiment MR (pin10) is connected to 5V (HIGH Level) and OE (pin 13) to GND (LOW Level). Therefore, the data is input into the rising edge of SHcp and enters the memory register through the rising edge. We use the shiftout() function to output a 8-bit data to the shift register through DS. In the rising edge of the SHcp, the data in the shift register moves successively one bit in one time, i.e. data in Q1 moves to Q2, and so forth. In the rising edge of STcp, data in the shift register moves into the memory register. All data will be moved to the memory register after 8 times. Then the data in the memory register is output to the bus (Q0-Q7). So the 16 characters are displayed in the 7-segment in turn.

The schematic diagram:

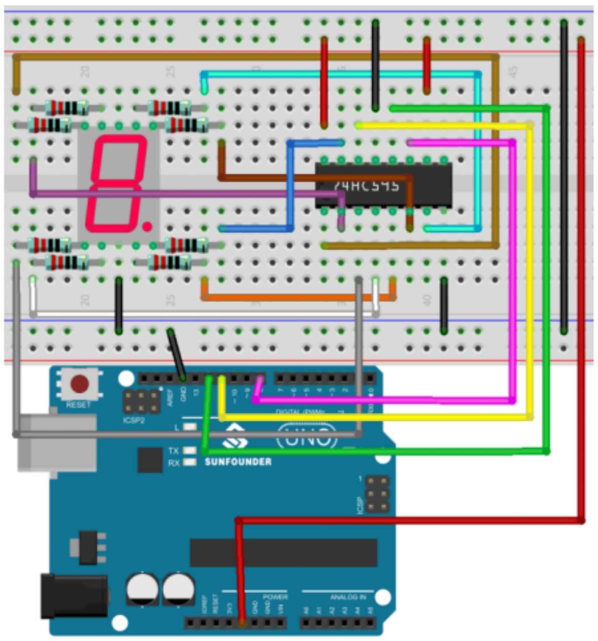


Experimental Procedures

Step 1: Build the circuit (pay attention to the direction of the chip by the concave on it)



7-Segment	74HC595	Uno R3
a	Q7	
b	Q6	
c	Q5	
d	Q4	
e	Q3	
f	Q2	
g	Q1	
DP	Q0	
	VCC	5V
	DS	11
	OE	GND
	ST	12
	SH	8
	MR	5V
	Q7’	N/C
	GND	GND
-		GND

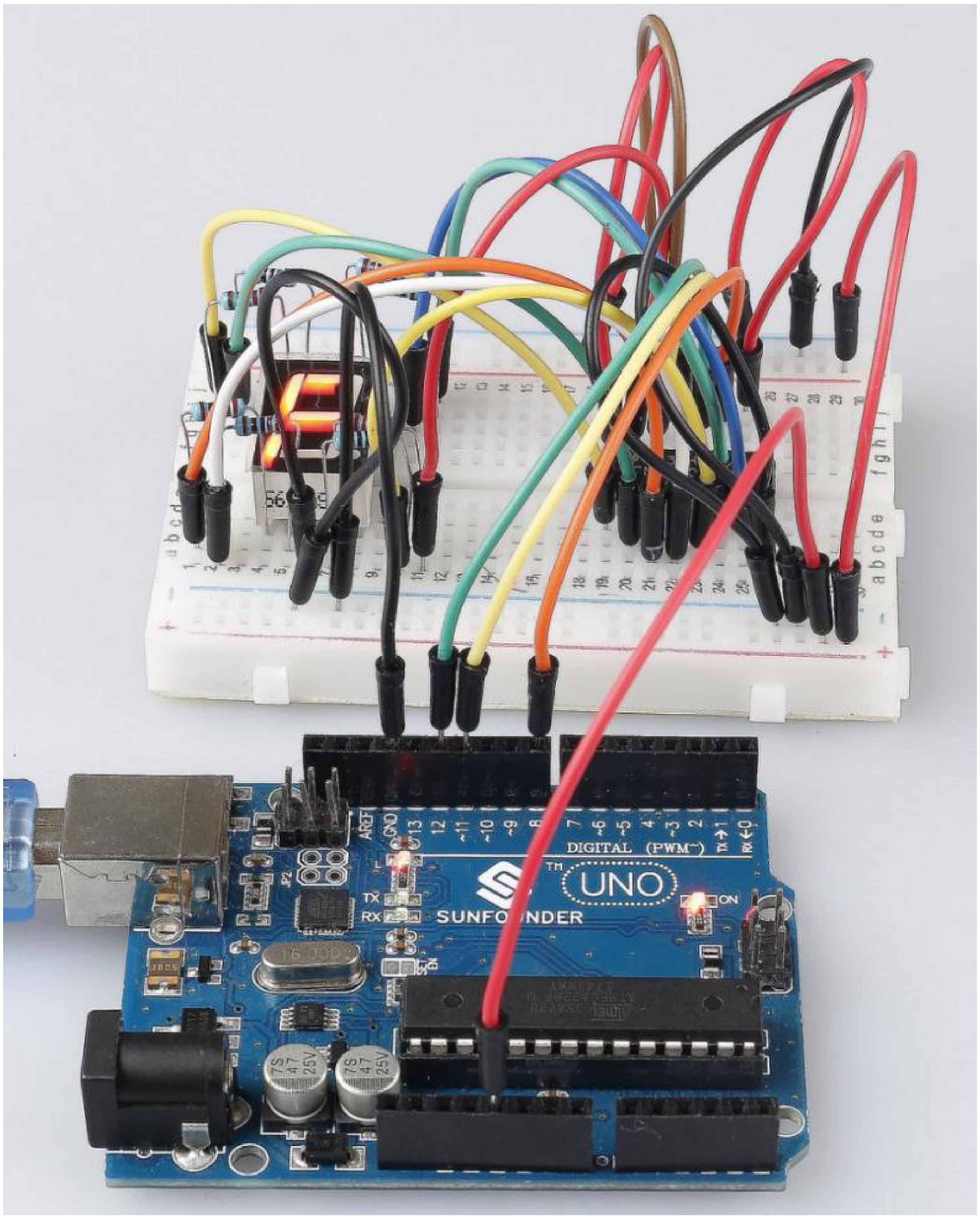


Step 2: Open the code file.

Step 3: Select the **Board** and **Port.**

Step 4: Upload the sketch to the board.

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


AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
UNO
SUNFOUNDER
ON
TX
RX
L

Code

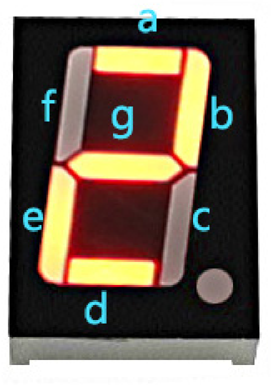
Code Analysis 20-1 Set the array elements

```
int datArray[16] = {252, 96, 218, 242, 102, 182, 190, 224, 254, 246, 238, 62, 156,␣
↪122, 158, 142};
```

This array stores the data of the 16 characters from 0 to F. 218 stands for 2, which you can calculate by yourself. To display 2, the segment f and c of the 7-segment display must be low level (dim).

Since the segment f and c are connected to Q2 and Q5 of the 74HC595, set both Q0, Q2 and Q5 (the dot) as low level and leave the rest pins as high level. Therefore, the values of Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 are 1 1 0 1 1 0 1 0.

Change the binary numbers into decimal ones: $1x2^7+1x2^6+0x2^5+1x2^4+1x2^3+0x2^2+1x2^1+0x2^0=218$.



So that's the value for the number **2** to be displayed. You can calculate other characters similarly.

Code Analysis 20-2 Display 0-F in the 7-segment display

```
for(int num = 0; num < 16; num++)

{

    digitalWrite(STcp,LOW); // ground ST_CP and hold low for as long as you are␣
↪transmitting

    shiftOut(DS,SHcp,MSBFIRST,datArray[num]);

    // return the latch pin high to signal chip that it

    // no longer needs to listen for information

    digitalWrite(STcp,HIGH); // pull the ST_CPST_CP to save the data

    delay(1000); // wait for a second

}
```

Set *STcp* as low level first and then high level. It will generate a rising edge pulse of STcp.

shiftOut() is used to shift out a byte of data one bit at a time, which means to shift a byte of data in *dataArray[num]* to the shifting register with the DS pin. *MSBFIRST* means to move from high bits.

After *digitalWrite(STcp,HIGH)* is run, the STcp will be at the rising edge. At this time, the data in the shift register will be moved to the memory register.

A byte of data will be transferred into the memory register after 8 times. Then the data of memory register is output to the bus (Q0-Q7). You will see a character is displayed on the 7-segment. Then delay for 1000ms. After that line, go back to *for()*. The loop repeats until all the characters are displayed in the 7-segment display one by one after 16 times.

1.4 FAQ

1.4.1 C code is not working?

- Check your wiring for problems.
- Check if the code is reporting errors, if so, refer to: Check and Install the WiringPi.
- Has the code been compiled before running.
- If all the above 3 conditions are OK, it may be that your wiringPi version (2.50) is not compatible with your Raspberry Pi 4B and above, refer to Check and Install the WiringPi to manually upgrade it to version 2.52.

1.5 Thank You

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

CHAPTER

TWO

COPYRIGHT NOTICE