


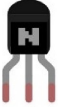




Chapter 6 Buzzer

In this chapter, we will learn a component that can sound, buzzer.

Project 6.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 BreadBoard x1		Jumper 		
NPN transistorx1 (S8050) 	Active buzzer x1 	Push button x1 	Resistor 1kΩ x1 	Resistor 10kΩ x2 

Component knowledge

Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock, alarm. Buzzer has active and passive type. Active buzzer has oscillator inside, and it will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.



Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2kHz, which means the passive buzzer is loudest when its resonant frequency is 2kHz.

Next, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzer is more complex than passive buzzer in manufacture. There are a lot of circuits and crystal oscillator elements inside active buzzer, which are covered with a waterproof coating, around its pins. Passive buzzer doesn't have these coatings. From the pins of passive buzzer, you can even see the circuit board, coils, and permanent magnets directly.

Transistor

Due to the current operating of buzzer is so large that GPIO of RPi output capability can not meet the requirement. A transistor of NPN type is needed here to amplify the current.

Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types shown below: PNP and NPN,

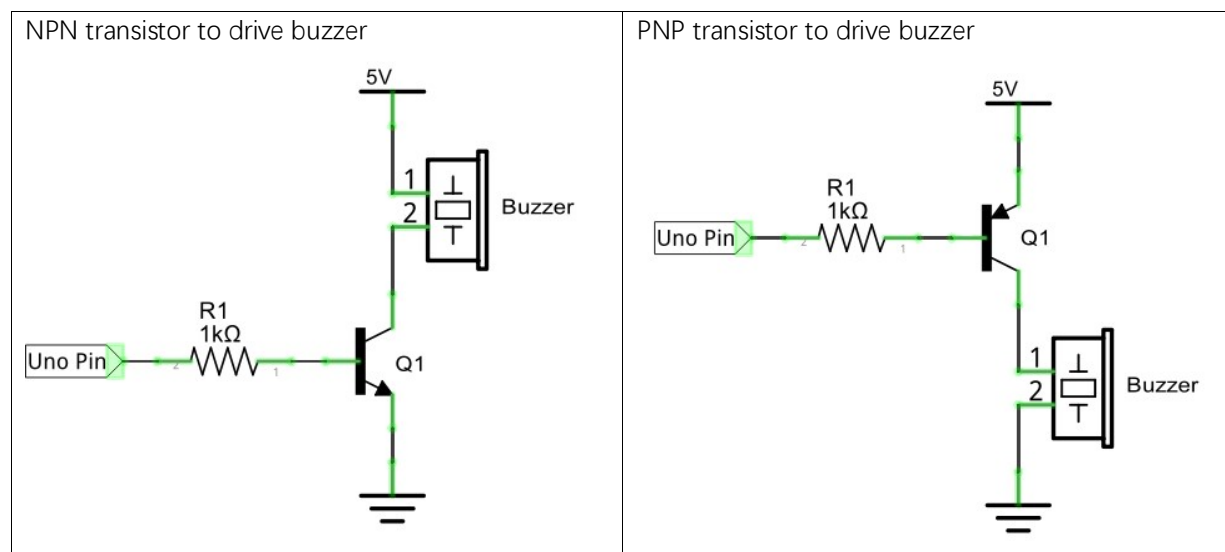


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

According to the transistor's characteristics, it is often used as a switch in digital circuits. For micro-controller's capacity of output current is very weak, we will use transistor to amplify current and drive large-current components.

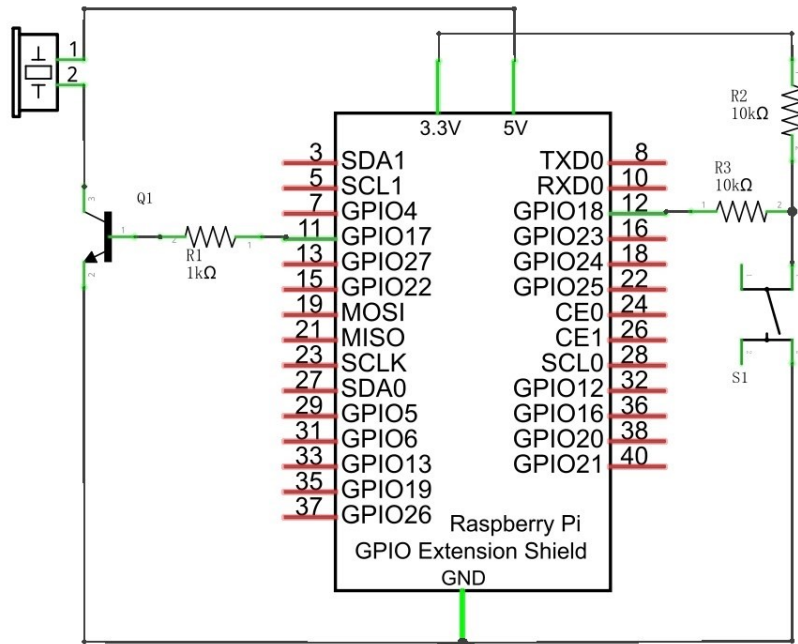
When use NPN transistor to drive buzzer, we often adopt the following method. If GPIO outputs high level, current will flow through R1, the transistor gets conducted, and the buzzer make a sound. If GPIO outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When use PNP transistor to drive buzzer, we often adopt the following method. If GPIO outputs low level, current will flow through R1, the transistor gets conducted, buzzer make a sound. If GPIO outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

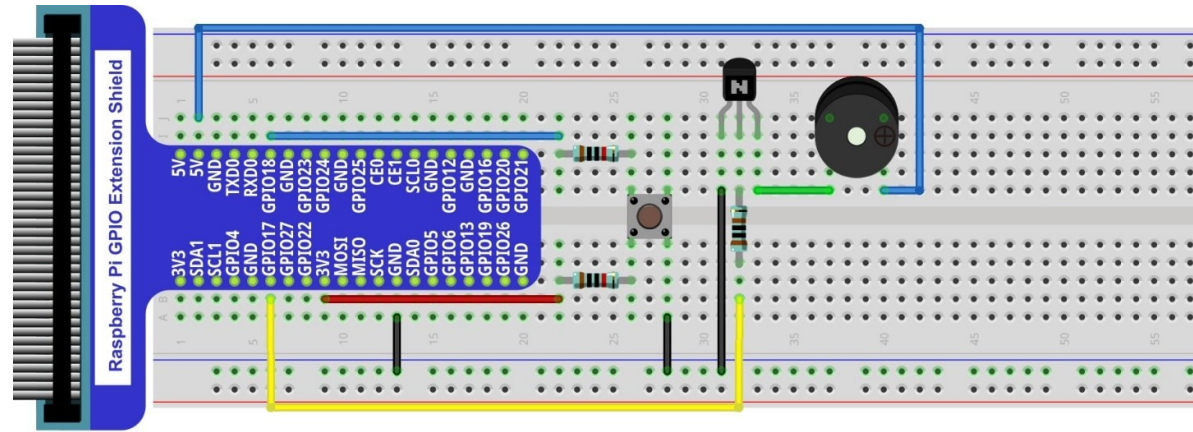


Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Note: in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.

Code

In this project, buzzer is controlled by the button. When the button is pressed, the buzzer sounds. And when the button is released, the buzzer stops sounding. In the logic, it is the same to using button to control LED.

C Code 6.1.1 Doorbell

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

1. Use cd command to enter 06.1.1_Doorbell directory of C code.

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/06.1.1_Doorbell
```

2. Use following command to compile "Doorbell.c" and generate executable file "Doorbell".

```
gcc Doorbell.c -o Doorbell -lwiringPi
```

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

After the program is executed, press the button, then buzzer sounds. And when the button is release, the buzzer will stop sounding.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define buzzeRPin 0    //define the buzzeRPin
5  #define buttonPin 1    //define the buttonPin
6
7  int main(void)
8  {
9      if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
10         printf("setup wiringPi failed !");
11         return 1;
12     }
13
14     pinMode(buzzeRPin, OUTPUT);
15     pinMode(buttonPin, INPUT);
16
17     pullUpDnControl(buttonPin, PUD_UP); //pull up to high level
18     while(1){
19
20         if(digitalRead(buttonPin) == LOW){ //button has pressed down
21             digitalWrite(buzzeRPin, HIGH); //buzzer on
22             printf("buzzer on...\n");
23         }
24         else { //button has released
25             digitalWrite(buzzeRPin, LOW); //buzzer off
26             printf("...buzzer off\n");
27         }
28     }
```

```

29
30     return 0;
31 }

```

The code is exactly the same to using button to control LED logically. You can try to use the PNP transistor to achieve the function of his circuit once again.

Python Code 6.1.1 Doorbell

First observe the project result, then analyze the code.

1. Use cd command to enter 06.1.1_Doorbell directory of Python code.

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/06.1.1_Doorbell
```

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

```
python Doorbell.py
```

After the program is executed, press the button, then buzzer sounds. And when the button is released, the buzzer will stop sounding.

The following is the program code:

```

1  import RPi.GPIO as GPIO
2
3  buzzerPin = 11    # define the buzzerPin
4  buttonPin = 12    # define the buttonPin
5
6  def setup():
7      print ('Program is starting...')
8      GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
9      GPIO.setup(buzzerPin, GPIO.OUT)  # Set buzzerPin's mode is output
10     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set buttonPin's mode is
11     input, and pull up to high level(3.3V)
12
13     def loop():
14         while True:
15             if GPIO.input(buttonPin)==GPIO.LOW:
16                 GPIO.output(buzzerPin,GPIO.HIGH)
17                 print ('buzzer on ...')
18             else :
19                 GPIO.output(buzzerPin,GPIO.LOW)
20                 print ('buzzer off ...')
21
22     def destroy():
23         GPIO.output(buzzerPin, GPIO.LOW)    # buzzer off
24         GPIO.cleanup()                      # Release resource
25
26     if __name__ == '__main__':              # Program start from here
27         setup()
28         try:
29             loop()
30

```

```

31     except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the subprogram destroy() will
32     be executed.
33         destroy()

```

The code is exactly the same to using button to control LED logically. You can try to use the PNP transistor to achieve the function of his circuit once again.

Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.

Component list and the circuit part is the similar to last section. In the Doorbell circuit only the active buzzer needs to be replaced with a passive buzzer.

Code

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. In the logic, it is the same to using button to control LED. In the control method, passive buzzer requires PWM of certain frequency to sound.

C Code 6.2.1 Alertor

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

1. Use cd command to enter 06.2.1_Alertor directory of C code.

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/06.2.1_Alertor
```

2. Use following command to compile "Alertor.c" and generate executable file "Alertor". "-lm" and "-lpthread" compiler options are needed to add here.

```
gcc Alertor.c -o Alertor -lwiringPi -lm -lpthread
```

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

After the program is executed, press the button, then buzzer sounds. And when the button is release, the buzzer will stop sounding.

The following is the program code:

```

1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <softTone.h>
4  #include <math.h>
5  #define buzzerPin    0    //define the buzzerPin
6  #define buttonPin    1    //define the buttonPin
7  void alertor(int pin) {
8      int x;
9      double sinVal, toneVal;
10     for(x=0;x<360;x++){ // The frequency is based on the sine curve.
11         sinVal = sin(x * (M_PI / 180));
12         toneVal = 2000 + sinVal * 500;
13         softToneWrite(pin, toneVal);

```

```

14     delay(1);
15 }
16 }
17 void stopAlertor(int pin) {
18     softToneWrite(pin, 0);
19 }
20 int main(void)
21 {
22     if(wiringPiSetup() == -1){ //when initialize wiring failed, print message to screen
23         printf("Setup wiringPi failed !");
24         return 1;
25     }
26     pinMode(buzzerPin, OUTPUT);
27     pinMode(buttonPin, INPUT);
28     softToneCreate(buzzerPin);
29     pullUpDnControl(buttonPin, PUD_UP); //pull up to high level
30     while(1){
31         if(digitalRead(buttonPin) == LOW){ //button has pressed down
32             alertor(buzzerPin); //buzzer on
33             printf("alertor on...\n");
34         }
35         else { //button has released
36             stopAlertor(buzzerPin); //buzzer off
37             printf("...buzzer off\n");
38         }
39     }
40     return 0;
41 }

```

The code is the same to the active buzzer logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so you need to create a software PWM pin through `softToneCreate(buzzerPin)`. Here `softTone` is dedicated to generate square wave with variable frequency and duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

```
softToneCreate(buzzerPin);
```

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

```

void alertor(int pin) {
    int x;
    double sinVal, toneVal;
    for(x=0; x<360; x++){ //The frequency is based on the sine curve.
        sinVal = sin(x * (M_PI / 180));
        toneVal = 2000 + sinVal * 500;
        softToneWrite(pin, toneVal);
    }
}

```



```

        delay(1);
    }
}

```

If you want to close the buzzer, just set PWM frequency of the buzzer pin to 0.

```

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

```

The related functions of softTone is described as follows:

```
int softToneCreate (int pin);
```

This creates a software controlled tone pin.

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

This updates the tone frequency value on the given pin.

For more details about softTone, please refer to <http://wiringpi.com/reference/software-tone-library/>

Python Code 6.2.1 Alertor

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

1. Use cd command to enter 06.2.1_Alertor directory of Python code.

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/06.2.1_Alertor
```

2. Use the python command to execute the Python code "Alertor.py".

```
python Alertor.py
```

After the program is executed, press the button, then the buzzer sounds. When the button is released, the buzzer will stop sounding.

The following is the program code:

```

1  import RPi.GPIO as GPIO
2  import time
3  import math
4
5  buzzerPin = 11    # define the buzzerPin
6  buttonPin = 12    # define the buttonPin
7
8  def setup():
9      global p
10     print ('Program is starting...')
11     GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
12     GPIO.setup(buzzerPin, GPIO.OUT)  # Set buzzerPin's mode is output
13     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)  # Set buttonPin's mode is
14     input, and pull up to high level(3.3V)
15     p = GPIO.PWM(buzzerPin, 1)
16     p.start(0);
17
18  def loop():
19     while True:
20         if GPIO.input(buttonPin)==GPIO.LOW:
21             alertor()
22             print ('buzzer on ...')

```

```

23         else :
24             stopAlertor()
25             print ('buzzer off ...')
26     def alertor():
27         p.start(50)
28         for x in range(0,361):      #frequency of the alarm along the sine wave change
29             sinVal = math.sin(x * (math.pi / 180.0))      #calculate the sine value
30             toneVal = 2000 + sinVal * 500      #Add to the resonant frequency with a Weighted
31             p.ChangeFrequency(toneVal)      #output PWM
32             time.sleep(0.001)
33
34     def stopAlertor():
35         p.stop()
36     def destroy():
37         GPIO.output(buzzerPin, GPIO.LOW)      # buzzer off
38         GPIO.cleanup()      # Release resource
39     if __name__ == '__main__':      # Program start from here
40         setup()
41         try:
42             loop()
43         except KeyboardInterrupt:      # When 'Ctrl+C' is pressed, the subprogram destroy() will
44             be executed.
45             destroy()

```

The code is the same to the active buzzer logically, but the way to control the buzzer is different. Passive buzzer requires PWM of certain frequency to control, so you need to create a software PWM pin through `softToneCreate` (`buzzerPin`). The way to create PWM is also introduced before in the sections about `BreathingLED` and `RGBLED`.

```

def setup():
    global p
    print ('Program is starting...')
    GPIO.setmode(GPIO.BOARD)      # Numbers GPIOs by physical location
    GPIO.setup(buzzerPin, GPIO.OUT)      # Set buzzerPin's mode is output
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)      # Set buttonPin's mode is
input, and pull up to high level(3.3V)
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);

```

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

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

When the button is released, the buzzer will be closed.

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