4.Active_Buzzer

Introduction

In this lesson, we will learn how to drive an active buzzer to beep with a PNP transistor.

Hardware Required

- ✓ 1 * Raspberry Pi
- ✓ 1 * T-Extension Board
- ✓ 1 * Active Buzzer
- ✓ 1 * 40-pin Cable
- ✓ 1 * S8050 PNP Transistor
- ✓ Several Jumper Wires
- ✓ 1 * Breadboard
- ✓ 1 * Resistor(1kΩ)

Principle

Active Buzzer

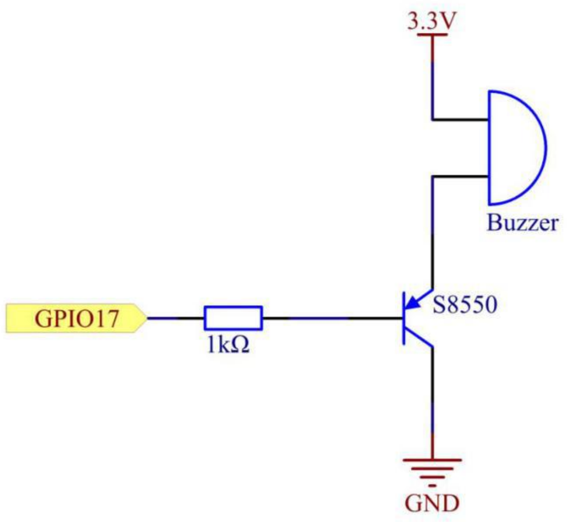
An active buzzer will generate a tone using an internal oscillator, so all that is needed is a DC voltage. A passive buzzer requires an AC signal to make a sound. It is like an electromagnetic speaker, where a changing input signal produces the sound, rather than producing a tone automatically. To identify them, if you apply a DC voltage to them and it buzzes, it's an active.
As far as commands go if you want to control the pitch, you would need a passive buzzer. PWM on the Arduino can be used to control the pitch and the volume at the same time (which may or may not be what you want). If you wanted to change just volume or just pitch I suppose some external circuitry would be required to change the amplitude without changing the voltage, and vice versa.

Schematic Diagram

In this experiment, an active buzzer, a PNP transistor and a 1k resistor are used between the base of the transistor and GPIO to protect the transistor. When the

GPIO17 of Raspberry Pi output is supplied with low level (0V) by programming, the transistor will conduct because of current saturation and the buzzer will make sounds. But when high level is supplied to the IO of Raspberry Pi, the transistor will be cut off and the buzzer will not make sounds.

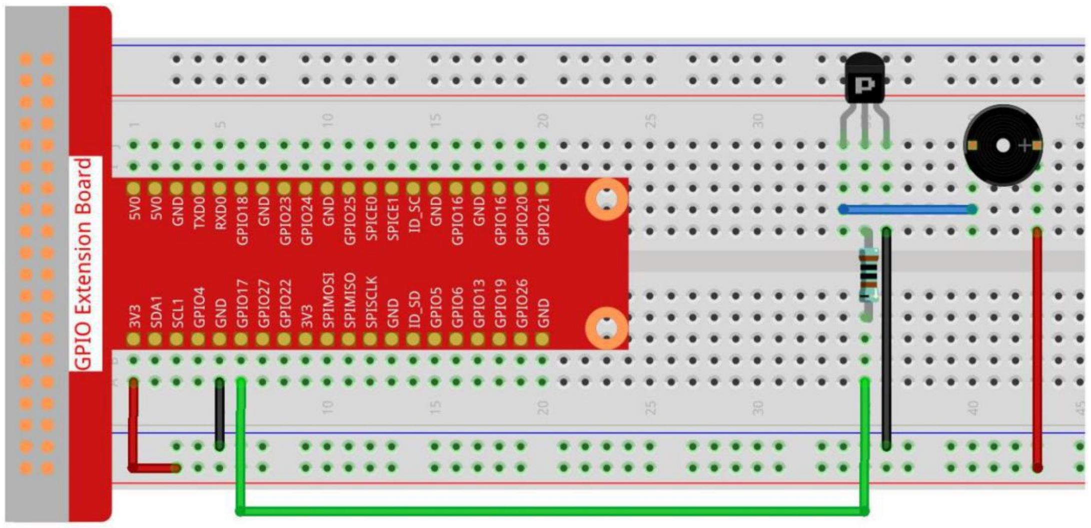
T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17



Experimental Procedures

Step 1: Build the circuit.

(Pay attention to poles of the buzzer: The one with + label is the positive pole and the other is the negative.)



For C Language Users

Step 2: Open the code file.

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/4.Active_Buzzer
```

Step 3: Compile the code.

```
gcc 4.Active_Buzzer.c -o Active_Buzzer.out -lwiringPi
```

Step 4: Run the executable file above.

```
sudo ./Active_Buzzer.out
```

The code run, the buzzer beeps.

Code

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

#define BeepPin 0
int main(void){
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }

    pinMode(BeepPin, OUTPUT);    //set GPIO0 output
    while(1){
        //beep on
        printf("Buzzer on\n");
        digitalWrite(BeepPin, LOW);
        delay(500);
        printf("Buzzer off\n");
        //beep off
        digitalWrite(BeepPin, HIGH);
        delay(500);
    }
    return 0;
}
```

Code Explanation

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

We use an active buzzer in this experiment, so it will make sound automatically when connecting to the direct current. This sketch is to set the I/O port as low level (0V), thus to manage the transistor and make the buzzer beep.

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

To set the I/O port as high level(3.3V), thus the transistor is not energized and the buzzer doesn't beep.

For Python Language Users

Step 2: Open the code file.

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/Python
```

Step 3: Run.

```
sudo python3 4.Active_Buzzer.py
```

The code run, the buzzer beeps.

Code

The code here is for Python3, if you need for Python2, please open the code with the suffix py2 in the attachment.

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

import RPi.GPIO as GPIO
import time

# Set #17 as buzzer pin
BeepPin = 17
```

```
def setup():
    # Set the GPIO modes to BCM Numbering
    GPIO.setmode(GPIO.BCM)
    # Set LedPin's mode to output,
    # and initial level to High(3.3v)
    GPIO.setup(BeepPin, GPIO.OUT, initial=GPIO.HIGH)

def main():
    while True:
        # Buzzer on (Beep)
        print ('Buzzer On')
        GPIO.output(BeepPin, GPIO.LOW)
        time.sleep(3)
        # Buzzer off
        print ('Buzzer Off')
        GPIO.output(BeepPin, GPIO.HIGH)
        time.sleep(3)

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

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

Code Explanation

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

Set the buzzer pin as low level to make the buzzer beep.

```
        time.sleep(3)
```

Wait for 3 second. Change the switching frequency by changing this parameter.

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

Note: Not the sound frequency. Active Buzzer cannot change sound frequency.

close the buzzer.

Phenomenon Picture