

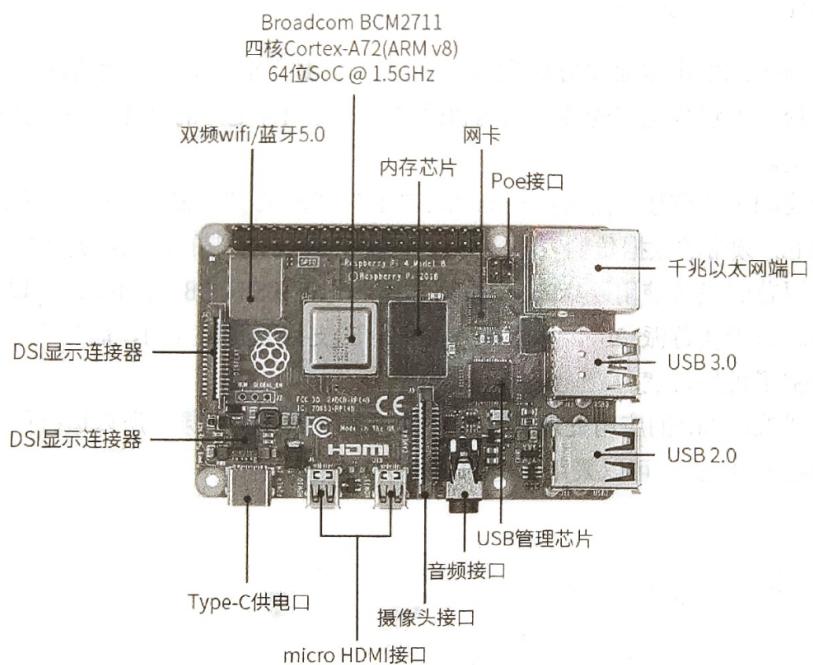
# EE351 微机原理与微系统 Mid Report

张皓东 12113010

## 实验1：树莓派简介与系统安装

### 树莓派简介

树莓派（Raspberry Pi）为本节课所有实验使用的主控板。它是一款小型、低成本的计算机开发板，旨在促进基础计算机科学和编程教育。树莓派由英国的树莓派基金会（Raspberry Pi Foundation）开发，并于2012年首次推出。它的设计灵感来自传统的个人计算机，但它更便宜、更小巧，可用于各种项目和应用。树莓派采用了不同型号，但通常包括基本的CPU、内存、GPIO（通用输入输出）引脚、USB端口、HDMI端口以及以太网接口。树莓派支持多种操作系统，包括Raspberry Pi OS（以前称为Raspbian）、Ubuntu等。用户可以选择并安装适合他们需求的操作系统。树莓派可以用于学习多种编程语言，如Python、C++等。它也支持各种开发工具和集成开发环境（IDE）。由于其小巧、低功耗和丰富的GPIO引脚，树莓派被广泛用于各种项目，如个人电脑、媒体中心、家庭自动化、物联网（IoT）设备、教育项目等。而且树莓派拥有庞大的全球社区，用户可以在社区中获得支持、分享项目经验和获取各种资源。其整体的实物图如下：



本节课实验主要使用了其GPIO口进行操作，其管脚示意图如下。同时可以使用T型转接板来方便GPIO的连接。目前，Raspberry Pi有三种引脚编号方法，分别是：根据引脚的物理位置编号（Header）、由C语言GPIO库wiringPi指定的编号（wiringPi Pin）、由BCM2837SOC指定的编号（BCM GPIO）。这里我们主要使用BCM编码。并且我们使用的T型扩展板采用的是BCM编码。需要注意的G27对应的是BCM编码的R1:21/R2:27管脚。

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



对于GPIO口的编程在本节课所有实验中都会被广泛使用，如果使用 Python 编程，可以使用 RPi.GPIO 提供的 API 对 GPIO 进行编程，该软件包提供了一个类来控制 Raspberry Pi 上的 GPIO。Raspberry Pi 的 RaspbianOS 镜像中默认安装 RPi.GPIO，因此可以直接使用它。如果需要安装 python-dev 包，则输入以下命令：

```
sudo apt-get install python-dev
```

## RaspberryPi 系统安装

首先，我们可以去官方下载安装文件：进入树莓派官网：Teach, Learn, and Make with Raspberry Pi，进入主页的 software，选择 Our software。在进入的页面选择 See all download options。选择 Raspberry Pi OS with desktop and recommended software。然后通过解压软件将下载好的 zip 文件解压后得到光盘映像文件。在 windows 上安装 Raspbian 系统时，我们可以使用 Win32DiskImager 软件进行安装，在电脑上插入装载了 SD 卡的读卡器，在界面中选择解压后的光盘映像文件与安装设备，选择写入即可。完成写入后，烧录好树莓派 Raspbian 系统的 TF 卡会被分成两个分区：一个FAT32 的 Boot 分区，和一个（或多个）Ext4 的 Linux 主分区，Windows 只能识别 Fat32 分区。找到 Micro SD 卡根目录下的 config.txt 文件并在文件末尾加入以下代码，保存并安全弹出 Micro SD 卡。

```
max_usb_current=1  
hdmi_force_hotplug=1  
config_hdmi_boost=7  
hdmi_group=2  
hdmi_mode=1  
hdmi_mode=87  
hdmi_drive=1  
display_rotate=0
```

系统安装成功后，可以外接屏幕和键盘对此进行编程，但有些麻烦，这时我们可以使用远程连接和控制的方法。

## SSH 与 VNC 远程连接

SSH 为 Secure Shell 的缩写，是一种网络协议，常用于计算机之间的登录、远程命令执行等。通过 SSH，可以使用终端远程连接 Raspberry Pi。具体步骤如下：点击左上角，依次选择 Preferences-Raspberry Pi Configuration-interfaces，将 SSH 和 VNC 都选择为 Enable，点击 OK。点击右上角 wifi 图标，连接 SUSTech-wifi。打开命令行，输入 ifconfig，查看 Raspberry Pi 在校园网局域网内的 ip 地址。SSH 客户端有很多，比如：putty、Mobaxterm、xshell、手机端的 juiceSSH 等。以 Putty 为例，putty 客户端可通过网站下载。确保电脑和 Raspberry 在同一局域网。点击左上 Session，在 Host Name (or IP address) 框中输入获得的 RaspberryPi ip 地址。点击 Open。也可以使用VScode中SSH插件对此进行操作，直接远程连接与控制。

另一种方式就是VNC连接，VNC 为 Virtual Network Computing 的缩写，通过 VNC 软件可以远程访问Raspberry Pi 的桌面。VNC-Viewer 软件可在其官网下载。具体连接方法如下所示：确保电脑和 Raspberry 在同一局域网。打开软件，在框内输入获取到的Raspberry Pi ip 地址，回车。输入 Raspberry Pi 的用户名和密码，选择 OK 进入桌面。这样就可以顺利远程连接了。接下来我们就可以对树莓派进行编程了。

# 实验2：双色LED颜色交替闪烁

## 实验目的

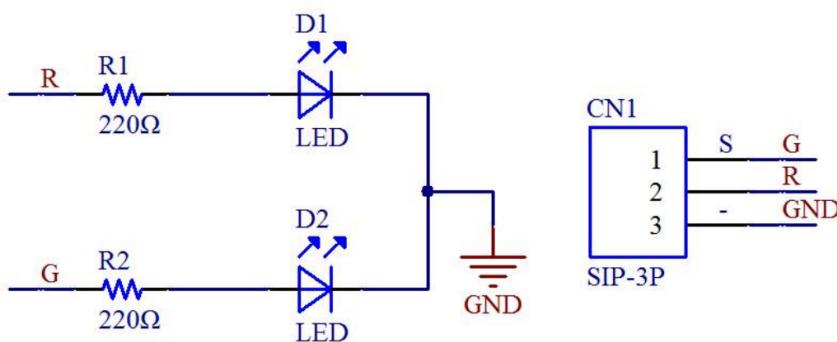
本实验将使用双色LED模块，通过树莓派编程控制使其可以红色和绿色交替闪烁。

## 实验原理与硬件连接

本实验所使用模块为双色LED灯，该模块只能显示两种颜色，一般为红色和绿色，总共存在三种状态：灭，颜色1亮，颜色2亮。该模块的引脚“-”需连接GND，引脚S和中间引脚分别控制着两种颜色。任一引脚接通高电平时，该引脚对应的颜色就会亮。如果两者同时接通高电平，则两种颜色将会同时显示。

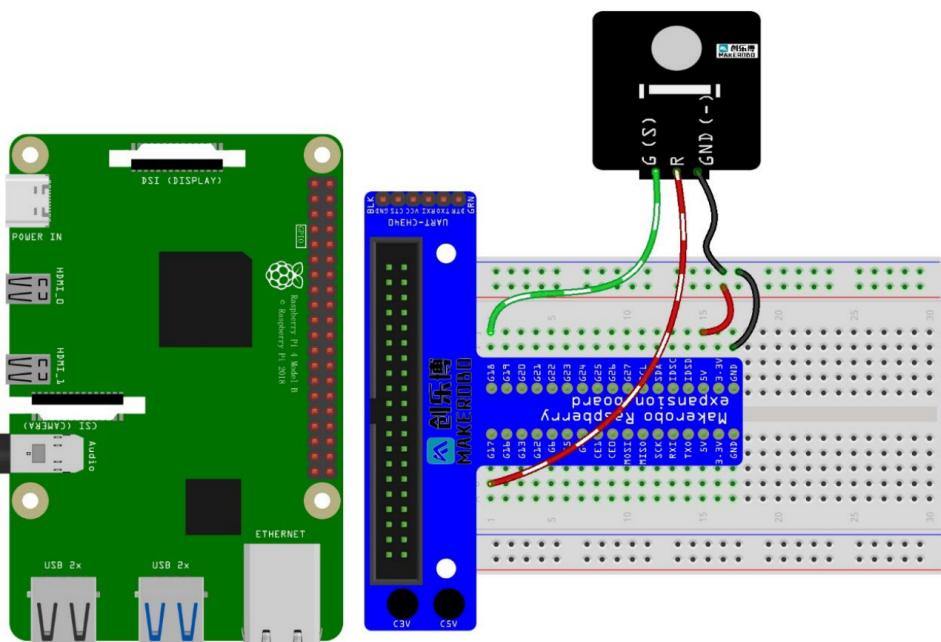


该模块对应的电路原理图为：



可见该模块为高电平触发，即引脚接通高电平，对应的发光二极管就会亮起，否则二极管保持灭的状态。

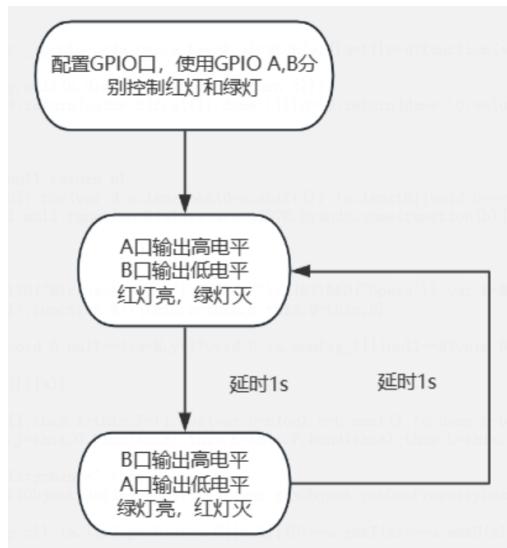
该实验的硬件连线图如下：



即将引脚 S (绿色) 和中间管脚 (红色) 分别连接到 Raspberry Pi 的 GPIO 接口20和21上，引脚一连接到 Raspberry Pi 的 GND 上，对 Raspberry Pi 进行编程控制相应GPIO输出高或者低电平，从而控制LED 的颜色变为红色或绿色。

## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

```
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(20, GPIO.OUT)
GPIO.setup(21, GPIO.OUT)

def led():
    while(True):
        GPIO.output(20, GPIO.LOW)
        GPIO.output(21, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(21,GPIO.LOW)
        GPIO.output(20,GPIO.HIGH)
        time.sleep(1)

if __name__ == '__main__':
    led()
```

即首先使用 GPIO.setmode(GPIO.BCM) 将GPIO的命名方式设置为BCM方式，然后使用 GPIO.setup(20, GPIO.OUT)和GPIO.setup(21, GPIO.OUT)将20和21端口配置为输出模式，然后 GPIO.output(20, GPIO.LOW) 和GPIO.output(21, GPIO.HIGH)分别将20口输出低电平，21口输出高电平，延时1s后，再翻转过来，从而实现LED灯颜色的交替变化。

演示效果视频见附件压缩包：“实验2：双色LED颜色交替闪烁”。

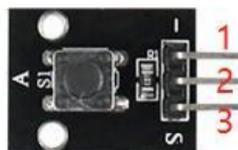
# 实验3：开关按键控制双色LED状态

## 实验目的

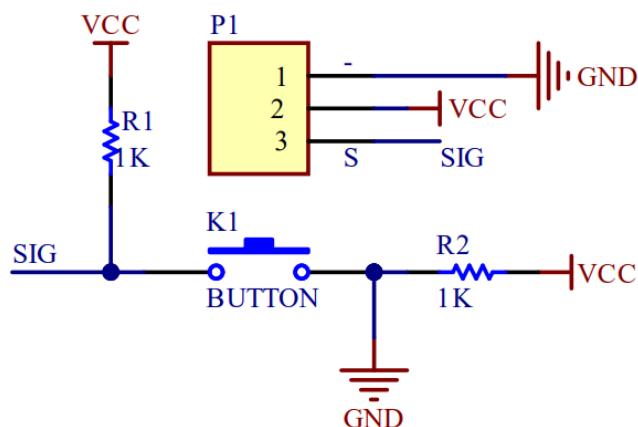
本实验将使用轻触开关模块和双色LED模块，实现通过轻触按键，使双色LED切换四种状态：红灯常亮、绿灯常亮、红灯闪烁、绿灯闪烁。

## 实验原理与硬件连接

本实验所用模块为轻触开关模块，该模块是最常见的开关模块，内部有一个轻触开关（按键开关）。“—”引脚接地，中间引脚接 VCC。按下按键时，S 脚输出为低电平；松开按键时，S 脚输出为高电平。

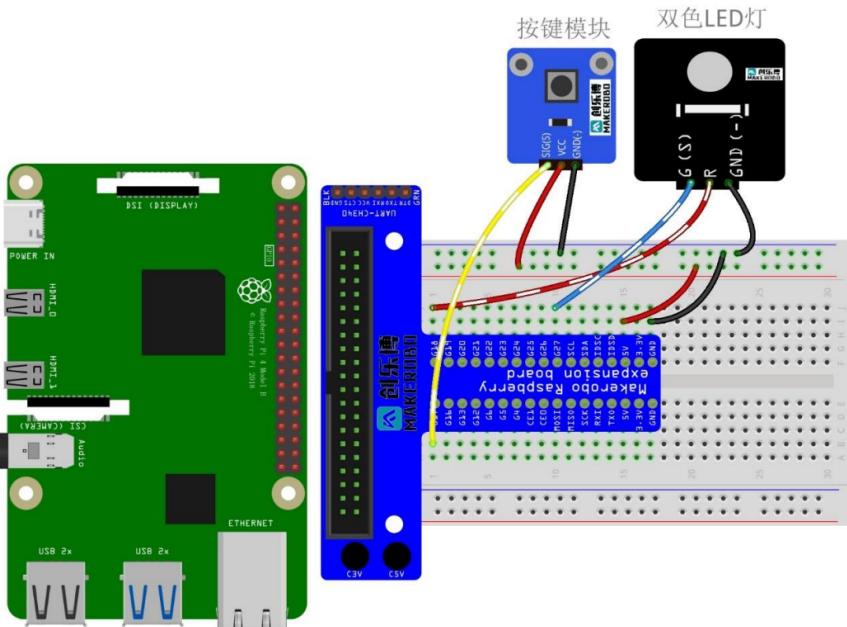


该模块的电路原理图如下：



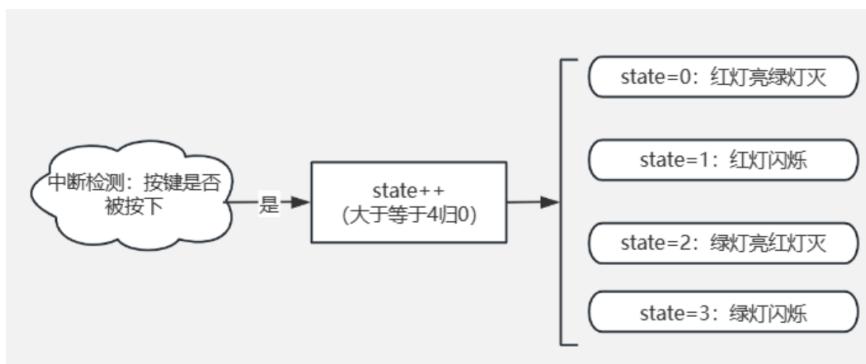
可见当按键被按下时，SIG被接地，S脚输出为低电平。当按键松开时，SIG通过一个电阻接VCC，输出为高电平。

本实验的硬件电路连接如下图所示，我们首先将按键模块的“—”引脚连接树莓派GND，中间引脚接5V，S引脚连接树莓的GPIO 17口，并将其配置为输入模式，可以通过此 GPIO 口检测轻触开关的 S 引脚。双色LED模块连接方式与实验二相同，即将引脚 S（绿色）和中间管脚（红色）分别连接到 Raspberry Pi 的 GPIO 接口27和18上，引脚一连接到 Raspberry Pi 的 GND。



## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

```

import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(27, GPIO.OUT)
GPIO.setup(17, GPIO.IN,GPIO.PUDUP)

def changeState():
    global state
    state = state+1
    if state >= 4:
        state = 0

def KEYInterrupt (key=17):
    changeState()

```

```

GPIO.add_event_detect(17, GPIO.FALLING, KEYInterrupt, 200)

def main():
    global state
    while(True):
        if state == 0:
            GPIO.output(18, GPIO.HIGH)
            GPIO.output(27, GPIO.LOW)
        elif state == 1:
            GPIO.output(27,GPIO.LOW)
            while(True):
                GPIO.output(18, GPIO.LOW)
                time.sleep(0.5)
                GPIO.output(18, GPIO.HIGH)
                time.sleep(0.5)
                if not state == 1:
                    break
        elif state == 2:
            GPIO.output(27, GPIO.HIGH)
            GPIO.output(18, GPIO.LOW)
        elif state == 3:
            GPIO.output(18, GPIO.LOW)
            while(True):
                GPIO.output(27, GPIO.LOW)
                time.sleep(0.5)
                GPIO.output(27 ,GPIO.HIGH)
                time.sleep(0.5)
                if not state == 3:
                    break

if __name__ == '__main__':
    global state
    state = 0
    main()

```

因为要在正常控制LED的同时还要检测按键是否被按下，我们在上述代码中使用了中断的方式，增加了一个线程来检测按键是否被按下，对应的代码如下：

```

GPIO.setup(17, GPIO.IN,GPIO.PUDUP)
def KEYInterrupt (key=17):
    changestate()
GPIO.add_event_detect(17, GPIO.FALLING, KEYInterrupt, 200)

```

如果按键被按下就调用changestate()函数，将全局变量state进行加一，因为共用4种状态，所以我们这里也设置了如果state大于等于4就将其重新置为0。接着在main函数里面我们通过if else的方式检测当下的state为多少，不同的state对应着不同的双色LED亮的方式。在闪烁过程中当state变化时，将while循环break出来。

## 如何消抖？

在上述实验中经常会出现按键按下会有抖动的现象，所以必须对按键按下进行消抖，这是为了在物理按键的信号中去除可能的噪声和抖动，确保在用户按下或释放按键时只触发一次相应的事件。通常有以下消抖的方法：

1. **软件延时消抖：**在按键按下时，等待一段短暂的时间，然后再检查按键的状态。如果在这段时间内按键保持按下状态，才认定为有效按下。这段时间通常称为“延时时间”或“消抖时间”。

```
def debounce(button_pin):  
    time.sleep(0.01)  # 等待10毫秒  
    if read_button_state(button_pin) == HIGH:  
        # 执行按键按下后的操作  
        print("Button pressed")  
  
def read_button_state(pin):  
    # 读取按键状态的代码  
    pass
```

2. **硬件滤波器：**在按键输入的物理电路中添加电容器或电感器，以平滑信号，减少抖动。这可以减少由于机械弹性或其他因素引起的电压抖动。
3. **使用中断：**使用硬件中断来捕获按键状态的变化。当按键状态发生变化时，中断被触发，可以在中断服务程序中进行相应的处理。这样可以确保及时响应按键状态的改变。

演示效果视频见附件压缩包：“实验3：开关按键控制LED状态”。实现了LED四种状态的切换。

## 实验4：PCF8591 模数转换器控制LED亮度

### 实验目的

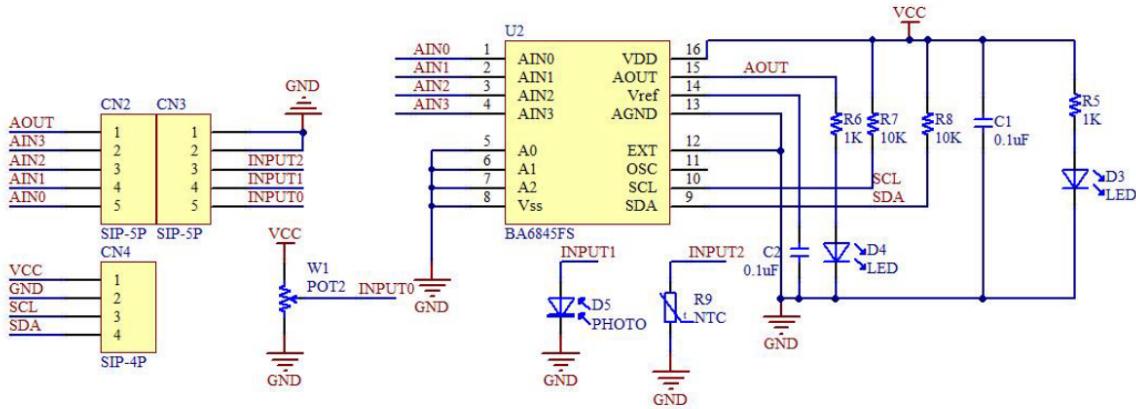
本实验我们将使用PCF8591 模数转换器模块，通过调节电位器，控制LED灯的亮度随着电位器调节而发生变化。

### 实验原理与硬件连接

本实验所使用的模块为PCF8591 模数转换器，该模块是单片机，单电源，低功耗 8 位 CMOS 数据采集设备，具有四个模拟输入，一个模拟输出和一个串行 I2C 总线接口。三个地址引脚 A0，A1 和 A2 用于对硬件地址进行编程，从而允许使用多达 8 个连接到 I2C 总线的设备，而无需额外的硬件。通过两行双向 I2C 总线串行传输与设备之间的地址，控制和数据。该设备的功能包括模拟输入多路复用，片上跟踪和保持功能，8 位模数转换和8位数模转换。最大转换率由 I2C 总线的最大速度决定。

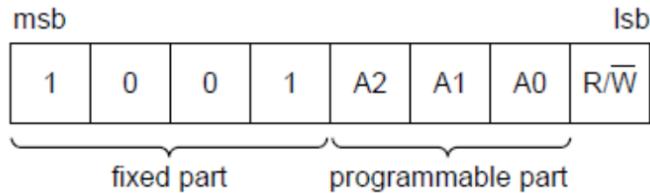


该模块的电路原理图如下：

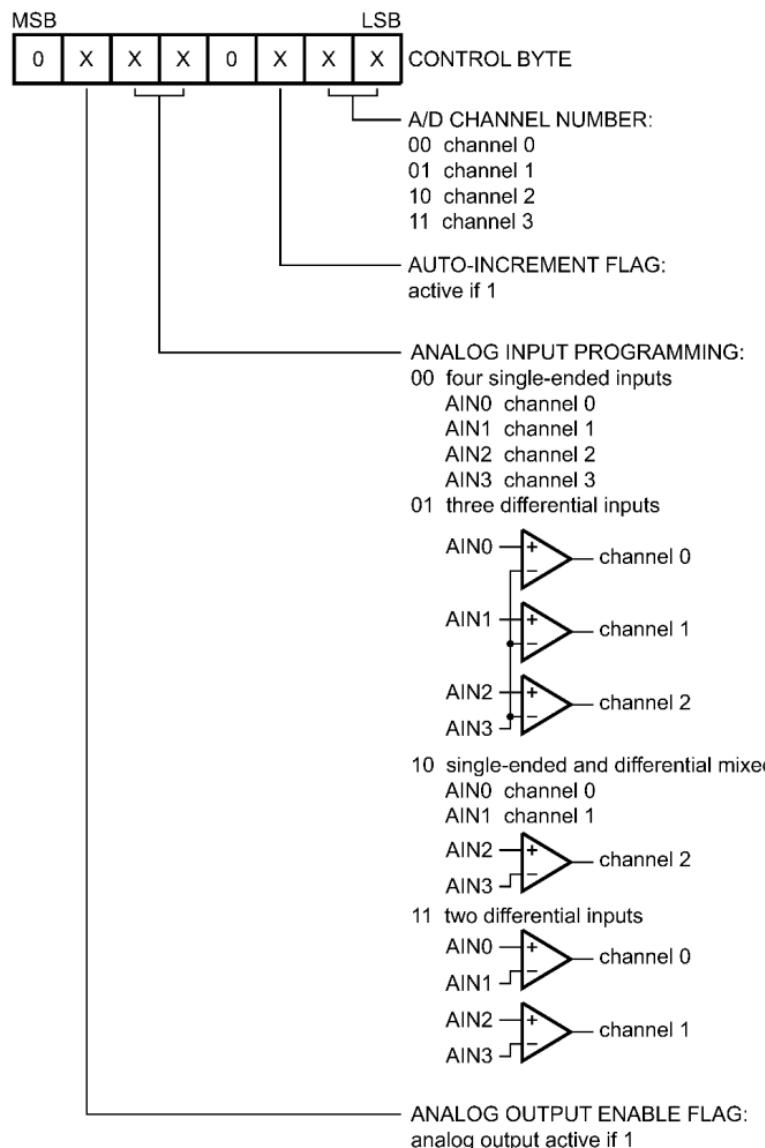


由上图可见该模块的输入端为AIN0-3，可外接电路，也可连接到模块本身的INPUT0-2，模块的INPUT0连接到一个可调电阻上，可以通过调整可调电阻的值来调节INPUT0的输出电压。模块的INPUT1连接到一个光敏电阻上，电压随光敏电阻的阻值发生变化。INPUT2连接到一个热敏电阻上，电压随着热敏电阻的阻值发生变化。AOUT为模块的模拟输出。

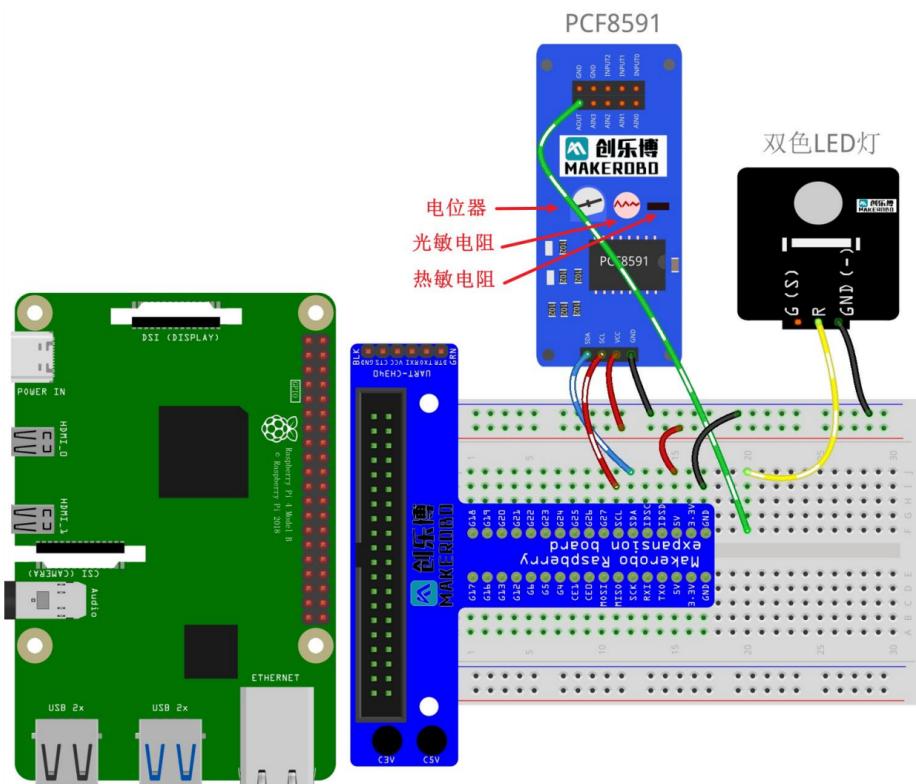
接下来介绍一下该模块的 I2C 通信方式。I2C 总线系统中的每个 PCF8591 设备通过向该设备发送有效地址而被激活。地址由固定部分和可编程部分组成。可编程部分由该模块的引脚 A0, A1, A2 来设置。地址字节的最后一个位是读/写位，它设置了下列数据传输的方向。



发送到 PCF8591 器件的第二个字节将被存储在其控制寄存器中，并且需要控制器件功能。使用 16 进制，只需要两位数作为参数控制该字节，其中每一位的控制作用总结如下。在本实验编程中需要注意配置其中的几个位置处的 bit。首先第二个 bit 用来控制是否使能输出，在本实验中需要将其设为 1，使其输出使能。第 3、4 个 bit 以及 7、8 个 bit 是用来控制其输入方式和 ADC 转化对应的 channel，这里我们使用 AIN0 作为输入，那我们将其配置为 00 和 00 即可。第 6 个 bit 是用来设置是否开启自动增加标志。如果设置了自动增加标志，则在每次 A/D 转换后，通道编号会自动递增。



电路连接图如下，首先连接好该模块与树莓派 I2C 通信电路，即PCF8591 模块的SDA连接树莓派的SDA，该模块的SCL连接到树莓派的SCL，其VCC连接到5V，GND接地。然后对于输入和输出端，在本实验中，我们使用AIN0(模拟输入 0)端口用于接收来自电位计模块的模拟信号，AOUT(模拟输出)用于将模拟信号输出到双色 LED 模块，以便改变 LED 的亮度。所以我们将AIN0连接到INPUT0，AOUT连接到双色LED灯的中间引脚，即控制红色灯亮度。



## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

```

import smbus
import time

bus = smbus.SMBus(1)

def read(chn):
    if chn == 0:
        bus.write_byte(0x48,0x40)
    if chn == 1:
        bus.write_byte(0x48,0x41)
    if chn == 2:
        bus.write_byte(0x48,0x42)
    if chn == 3:
        bus.write_byte(0x48,0x43)
    bus.read_byte(0x48)
    return bus.read_byte(0x48)

def write(val):
    temp = val
    temp = int(temp)

```

```
bus.write_byte_data(0x48, 0x40, temp)

if __name__ == "__main__":
    while True:
        tmp = read(0)
        write(tmp)
```

代码上通过bus.write\_byte() 函数对寄存器写入数据，使用bus.read\_byte() 函数读取设备的数据。其输入参数中的0x48为设备地址，用来选择IIC总线上的PCF8591器件。第二个参数就是我们上文讲到的第二个字节，根据上述我们的需求将其设置为0x40，来读取和写入AIN0通道的数据。然后就可以使用AOUT的输出电压来控制灯的亮度了。

演示效果视频见附件压缩包：“实验4：PCF8591 模数转换器控制LED亮度”。通过扭动可调电阻实现了调节LED的亮度。

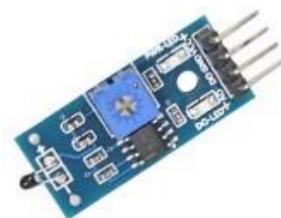
## 实验5：模拟温度传感器获取环境温度

### 实验目的

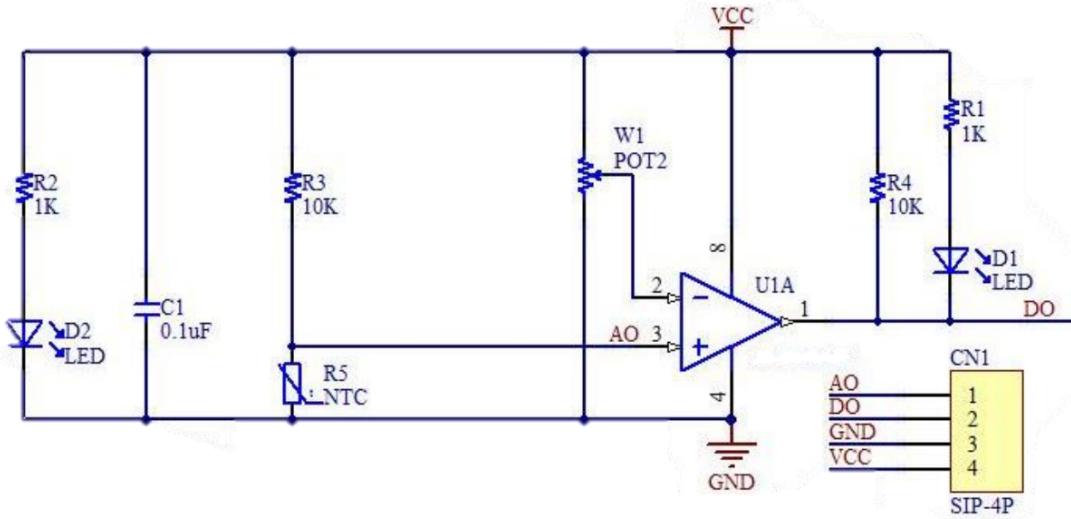
本实验将通过实验4中使用的PCF8591模块和由NTC 热敏电阻组成的温度传感器模块来获取当下环境的温度。

### 实验原理与硬件连接

本实验所使用的模块为温度传感器模块，它带有模拟和数字输出。该温度模块 使用 NTC (负温度系数) 热敏电阻来检测温度变化，其对温度感应非常灵敏。NTC 热敏电阻电路相对简单，价格低廉，组件精确，可以轻松获取项目的温度数据，因此广泛应用于各种温度的感测与补偿中。简而言之，NTC 热敏电阻将随 温度变化传递为电阻变化，利用这种特性，我们可以通过测量电阻网络 (例如分压器) 的电压来检测室内/环境温度。



该模块对应的电路原理图为：



由电路原理图可以看出温度传感器模块主要由一个 NTC 热敏电阻和一个  $10\text{k}\Omega$  电阻组成。热敏电阻的电阻值随环境温度变化而变化，从而 NTC 的分压也会随之发生变化，进而我们就可以使用 Steinhart-Hart 方程（见下）来求得热敏电阻的精确温度。其具体原理如下：首先通过 PCF8591 模块读取温度传感器模拟输出通过 A/D 转化后的数字值 Val，然后可以利用如下公式计算出热敏电阻的原始模拟电压值  $V_r$ ：

$$V_r = 5 * \text{float}(Val) / 255$$

再由上面的电路图可知，R3 与 R5 串联，所以电流值相等，R3 电阻阻值为  $10\text{k}\Omega$ ，即 10000，所以热敏电阻值为：

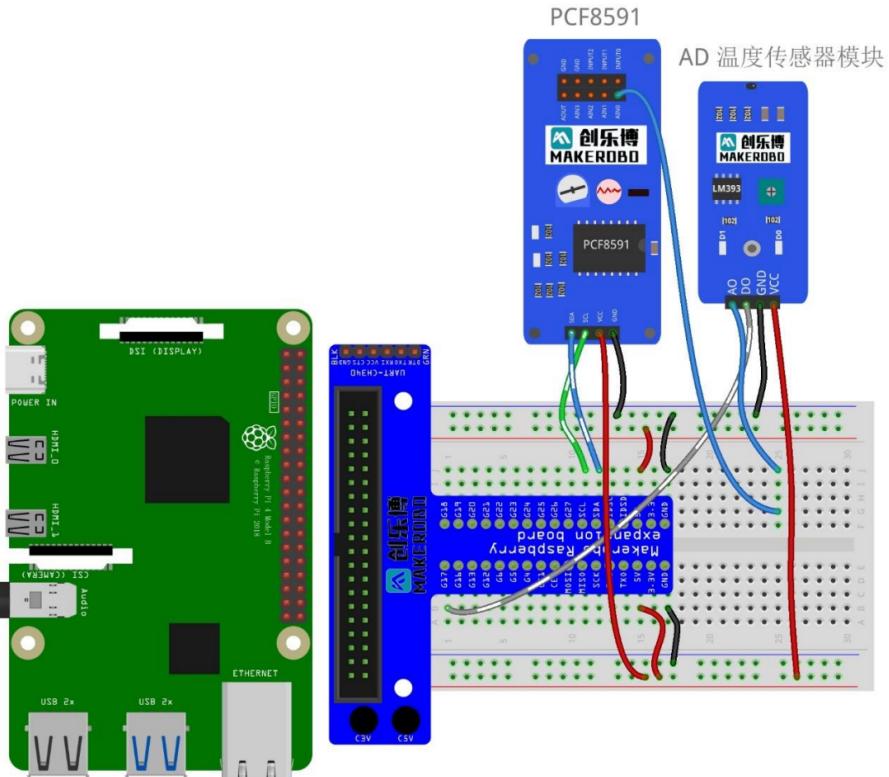
$$R_t = 10000 * V_r / (5 - V_r)$$

根据 Steinhart-Hart 方程，热敏电阻的阻值可以表示为：

$$R = R_0 e^{B(\frac{1}{T} - \frac{1}{T_0})}$$

其中  $T_0$  是标准温度值，即  $298.15\text{K}$  ( $25^\circ\text{C}$ )， $R_0$  是温度  $T_0$  温度下的阻值， $B$  是热敏电阻常数。在本次实验中  $R_0$  的值取为  $10\text{k}$ ， $B$  的值取为  $3950$ 。因此通过我们测出的电阻值就可以根据上式反算出当前环境温度。

该实验的硬件连线图如下：



PCF8591模块与树莓派之间按实验4所说的根据I2C通信方式连接。温度传感器模块的A0引脚连接PCF8591的AIN0，将采集的模拟信号输入到PCF8591转化为数字信号，在通过I2C通信的方式将此数据传递给树莓派进行处理。

## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

PCF8591的相关代码与实验4中的代码相同，包装成PCF8591.py函数：

```

**PCF8591.py**

import smbus
import time

bus = smbus.SMBus(1)

def setup(Addr):
    global address
    address = Addr

def read(chn): #channel

```

```

if chn == 0:
    bus.write_byte(address,0x40)
if chn == 1:
    bus.write_byte(address,0x41)
if chn == 2:
    bus.write_byte(address,0x42)
if chn == 3:
    bus.write_byte(address,0x43)
bus.read_byte(address)
return bus.read_byte(address)

def write(val):
    temp = val
    temp = int(temp)
    bus.write_byte_data(address, 0x40, temp)

if __name__ == "__main__":
    setup(0x48)
    while True:
        print('电位计 AIN0 = ', read(0))
        print('光敏电阻 AIN1 = ', read(1))
        print('热敏电阻 AIN2 = ', read(2))
        tmp = read(0)
        write(tmp)
        # time.sleep(2)

```

有关温度传感器的代码，包装成函数TemperatureSensor.py：

```

**TemperatureSensor.py**

import PCF8591 as ADC
import RPi.GPIO as GPIO
import time
import math

DO = 17
GPIO.setmode(GPIO.BCM)

def setup():
    ADC.setup(0x48)
    GPIO.setup(DO, GPIO.IN)

def loop():
    while True:
        analogval = ADC.read(0)
        print(analogval)
        Vr = 5 * float(analogval) / 255
        Rt = 10000 * Vr / (5 - Vr)
        temp = 1/(((math.log(Rt / 10000)) / 3950) + (1 / (273.15+25)))

```

```

temp = temp - 273.15
print('The temperature is {}'.format(temp))
time.sleep(0.1)

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

```

其代码实现与前文中的实验原理部分的流程相同，即通过ADC.read(0)读取传感器模拟输出通过PCF8591后A/D转化后的数字值。然后利用上面的值计算热敏电阻的原始模拟电压值 $V_r$ ：

$V_r = 5 * float(analogVal)/255$ ，然后计算电阻： $Rt = 10000 * Vr/(5 - Vr)$ ，然后根据Steinhart-Hart 方程算出温度值得到temp：

$temp = 1/(((math.log(Rt/10000))/3950) + (1/(273.15 + 25)))$ ,  $temp = temp - 273.15$ ，这样就可以得到最终环境的温度。

演示效果视频见附件压缩包：“实验5：模拟温度传感器获取环境温度”。将温度传感器模块靠在树莓派芯片附近，温度在屏幕打印出来，大约40度左右，符合实验要求。

## 实验6：超声波传感器测量手机长度

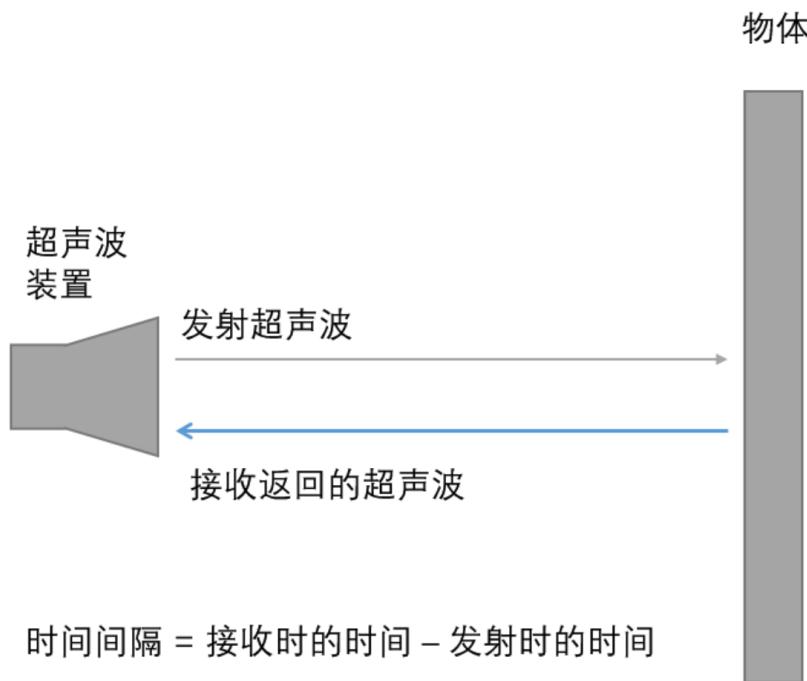
### 实验目的

本实验将使用超声波传感器实现测距，并使用该模块实现测量手机的长度。

### 实验原理与硬件连接

超声波是指频率大于20 kHz的在弹性介质中产生的机械震荡波，其具有指向性强、能量消耗缓慢、传播距离相对较远等特点，因此常被用于非接触测距。声音是由振动产生的，能够产生超声波的装置就是超声波传感器，习惯上称为超声换能器，或者超声探头。超声波探头主要由压电晶片组成，既可以发射超声波，也可以接收超声波。常用的是压电式超声波发生器，是利用压电晶体的谐振来工作的。超声波传感器探头内部有两个压电晶片和一个共振板。当它的两极外加脉冲信号，其频率等于压电晶片的固有振荡频率时，压电晶片将会发生共振，并带动共振板振动，便产生超声波。反之，如果两电极间未外加电压，当共振板接收到超声波时，将压迫压电晶片作振动，将机械能转换为电信号，这时它就成为超声波接收器了。超声波传感器就是利用压电效应的原理将电能和超声波相互转化，即在发射超声波的时候，将电能转换成超声波发射出去；而在接收时，则将超声振动转换成电信号。

超声波测距的原理如下图所示，即其通过自发自收超声波信号，通过计算时间间隔可以计算出超声波发射装置与反射超声波的物体的距离。

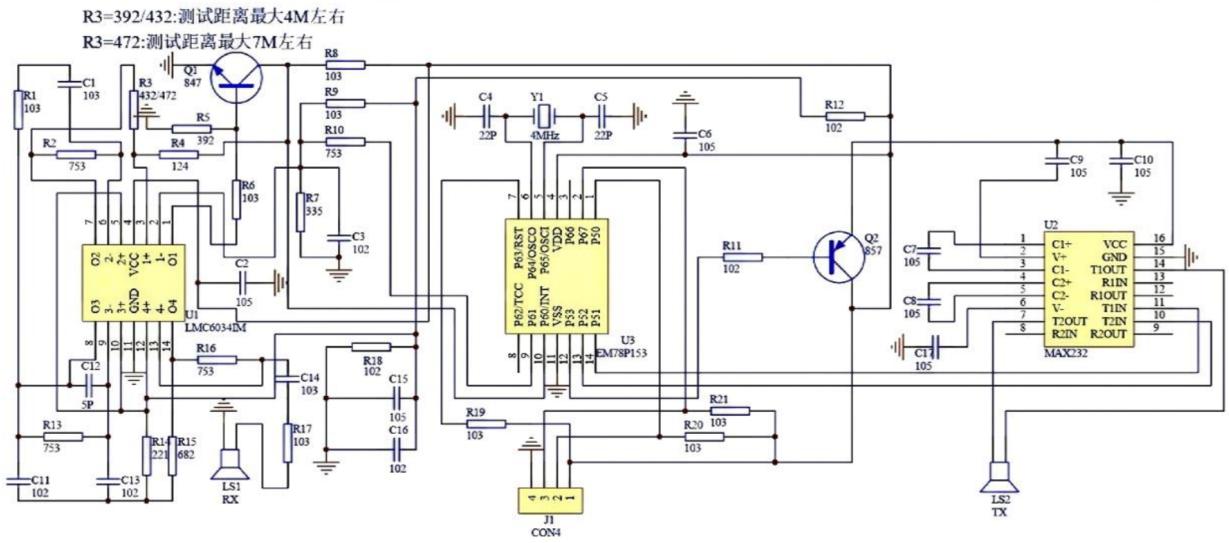


本实验所使用的超声波测距模块主要是由两个通用的压电陶瓷超声传感器，并加外围信号处理电路构成的。超声传感器中的一个用作发射器，将电信号转换为 40KHz 超声波脉冲信号；另一个用作接收器，监听发射的脉冲。超声波距离传感器体积小，易于在项目中使用，可以提供 2cm 至 400cm 左右的非接触距离检测，精度为 3mm。由于它的工作电压为 5 伏，因此可以直接连接到 Raspberry 或任何其他 5V 逻辑微控制器。

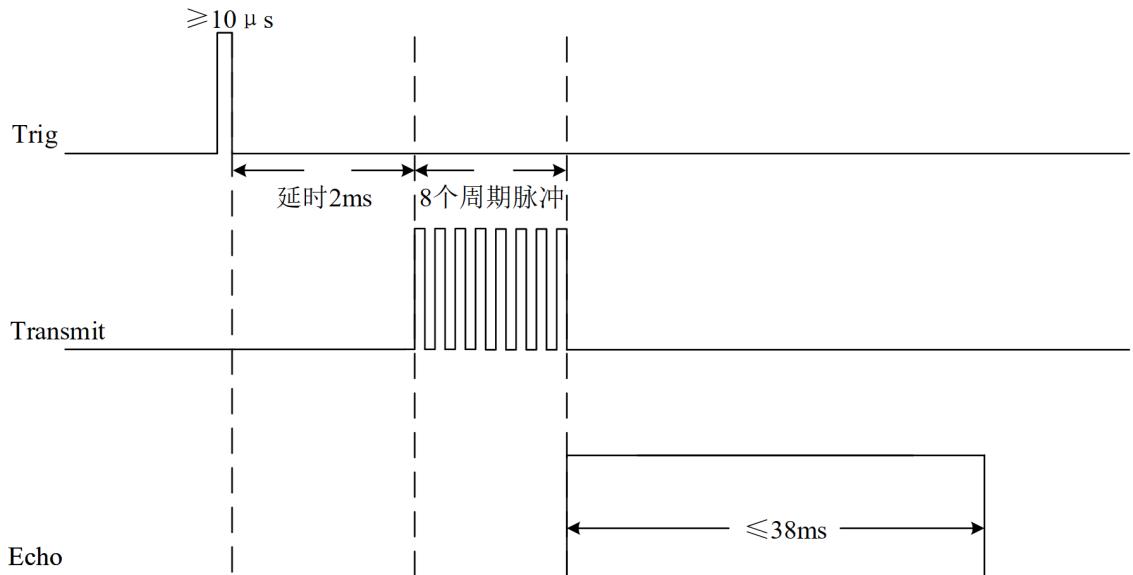
模块的示意图如下，其共有4个引脚：1. VCC：超声波模块电源脚，接 5V 电源。2. Trig：超声波发送引脚。3. Echo：超声波接收检测引脚。4. GND：超声波模块接地引脚。



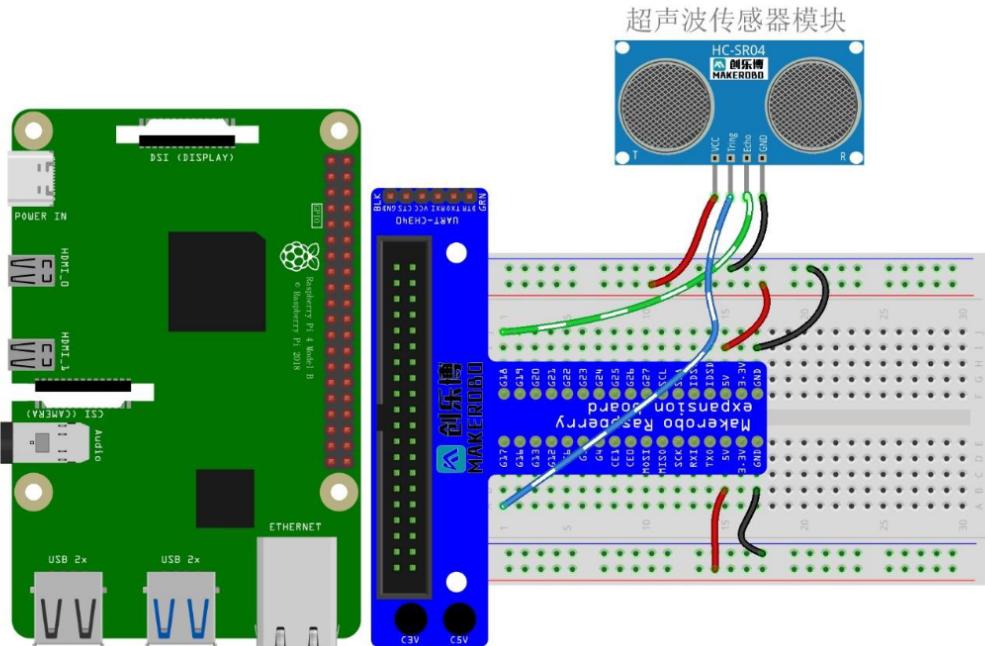
该模块对应的电路原理图为：



其工作原理如下：首先控制器给 Trig 引脚至少  $10\mu s$ (10 微秒)的高电平信号，触发传感器开始工作。超声波传感器经过一定的延时 (2ms) 后，发送 8 个 40KHz 的超声波脉冲信号。同时，Echo 引脚变为高电平。当检测到有回声信号返回或大于 38ms 时，Echo 引脚变为低电平。所以测试距离= (高电平时间\*声速) /2。值得注意的是 Echo 引脚变为高电平时为 5V，而树莓派 GPIO 输入一般不能超过 3.3V，故应使用分压器测量。但由于本次实验 Echo 引脚高电平时间非常短，故可不使用分压。



该实验的硬件连线图如下：



超声波的VCC接5V，Trig引脚接树莓派GPIO口17，Echo引脚接树莓派GPIO口18，GND接地。将17配置为输出口，用来产生高电平，以发射超声波信号。将18引脚配置为输入口，来检测是否收到了反射的超声波信号。

## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

```

import RPi.GPIO as GPIO
import time

TRIG = 17
ECHO = 18

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(TRIG, GPIO.OUT, initial = GPIO.LOW)
    GPIO.setup(ECHO, GPIO.IN)

def distance():
    GPIO.output(TRIG, 1) #给Trig一个10us以上的高电平
    time.sleep(0.00001)
    GPIO.output(TRIG, 0)
  
```

```

while GPIO.input(ECHO) == 0: #捕捉 echo 端输出上升沿
    pass
time1 = time.time()

while GPIO.input(ECHO) == 1: #捕捉 echo 端输出下降沿
    pass
time2 = time.time()

duration = time2 - time1
return duration * 340 / 2 * 100
#超声波传播速度为340m/s,换算成cm

def loop():
    while True:
        dis = distance()
        print(f'{dis} cm')
        time.sleep(0.3)

def destroy():
    GPIO.cleanup()

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

```

实现过程与上述介绍的原理一致，先使用GPIO.output(TRIG, 1) 配置高电平以发射一个超声波信号，用time.sleep(0.00001)控制其时间长度为10us，然后不断循环用GPIO.input(ECHO)捕捉echo引脚是否出现高电平，如果出现记录下时间，然后继续捕捉其出现低电平，记录下此时时间，用两个时间相减即可得到高电平的持续时间，利用公式就可以计算出对应的距离：duration \* 340 / 2 \* 100。

我们可以用此代码来测手机的长度，即把手机一端顶住墙面，另一端放置超声波传感器，此时测得的距离就是手机的长度，首先我们先测量出手机的真实长度以进行判断超声波测距效果，可见实际长度大约为15.5cm



真实测量的演示效果视频见附件压缩包：“实验6：超声波传感器测量手机长度”。其最终结果大约为15.7cm，由0.2cm误差，满足精度要求。

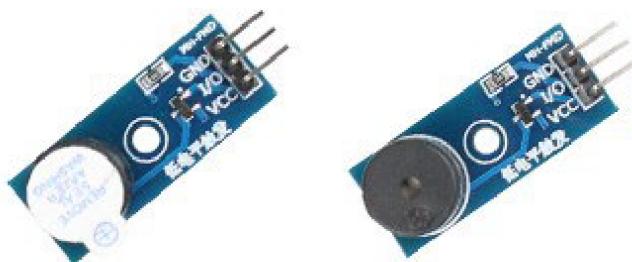
## 实验7：有源蜂鸣器报警与无源蜂鸣器播放音乐

### 实验目的

本实验使用树莓派控制有源和无源两种蜂鸣器，并使用无源蜂鸣器播放一段选定的音乐。

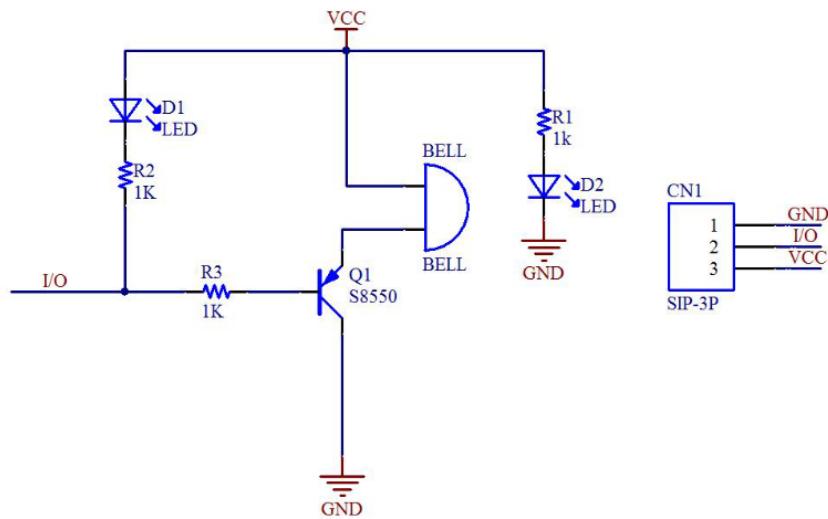
### 实验原理与硬件连接

蜂鸣器属于声音模块，一般可以分为有源蜂鸣器和无源蜂鸣器。有源和无源是指内部是否有震荡源。有源蜂鸣器内置振荡器，没有频率变化，直接接上合适的直流电源即可发声，常用于发出单一的提示性报警声音；无源蜂鸣器由于内部没有震荡源，所以其驱动方式为脉冲频率调制（Pulse-Frequency Modulation, PFM），可以通过调控脉冲频率发出不同频率的声音信号。所以可以实现控制脉冲频率来播放一段音乐。



有源蜂鸣器（左）和无源蜂鸣器（右）

有源蜂鸣器内部有一个简单的振荡电路，能将恒定的直流电转化成一定频率的脉冲信号，程序控制方便但频率固定，单片机一个低（高）电平就可以让其发出声音，实验所使用的有源蜂鸣器为低电平触发。其原理图如下：

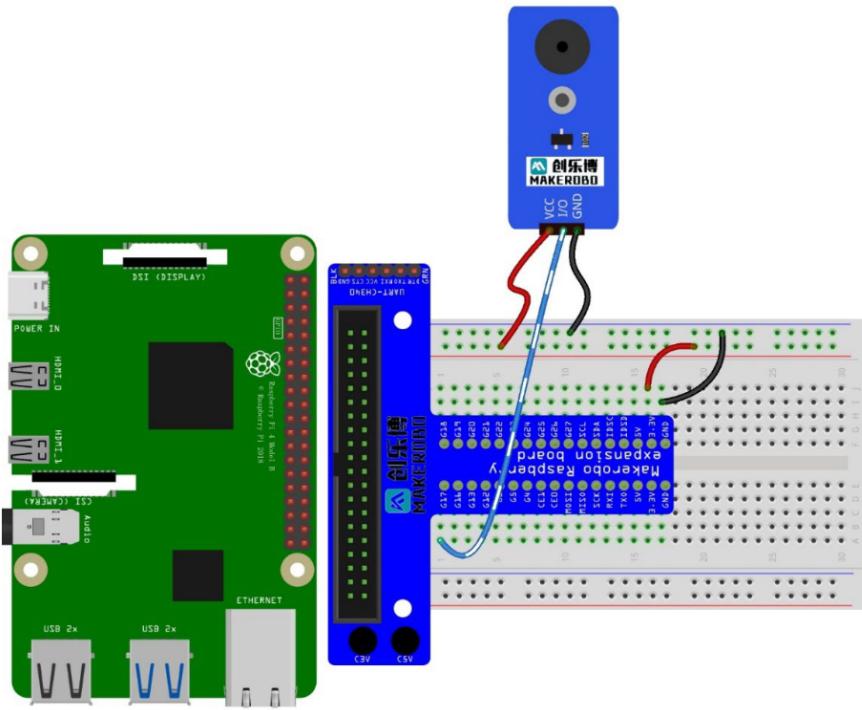


而无源蜂鸣器内部没有驱动电路，需要接在特定的音频输出电路中才能发出声音。如果直接给直流信号，则无声音，一般需要用 2K-5KHz 的方波去驱动它。由于声音频率可控，可以发出“do re mi fa so la xi”的声效。本实验中无源蜂鸣器采用 PFM 驱动方式。PFM 是一种仅使用两个电平（1 和 0）表示模拟信号的调制方法，在改变方波频率的同时固定方波脉冲的宽度（一般高电平宽度占空比（Duty）为 50%）。在开关电源领域，PFM 模式是在驱动轻负载时提高开关降压 DC-DC 转换器效率的常用技术。PFM 模式通过降低轻载时转换器的开关频率，明显减小开关管的开关损耗，进而提高轻载效率。与 PFM 类似的另一种广泛使用的调制方式是脉冲宽度调制（Pulse-Width Modulation，PWM）。不同的是 PWM 脉冲的频率固定，但方波脉冲的宽度会依信号的大小而改变。PWM 由于其脉冲频率固定，实现简单等特点在控制和通信等领域应用非常广泛，比如电机转速控制、屏幕 PWM 调光等。总而言之，可以通过输入不同频率的方波信号来驱动无源滤波器，从而控制其发声的音调。

该实验的硬件连线图如下：

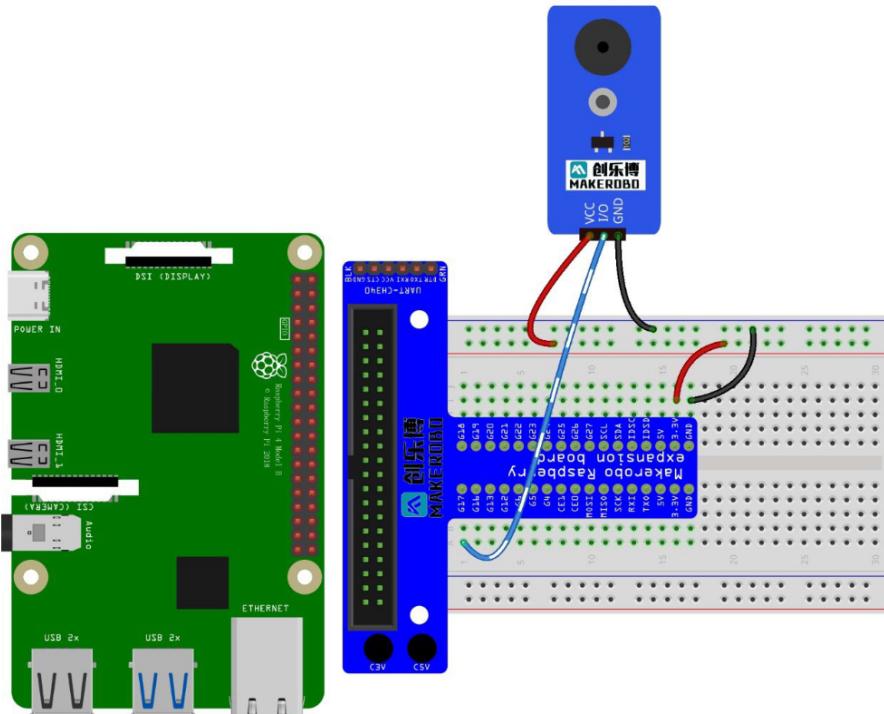
有源蜂鸣器：

有源蜂鸣器



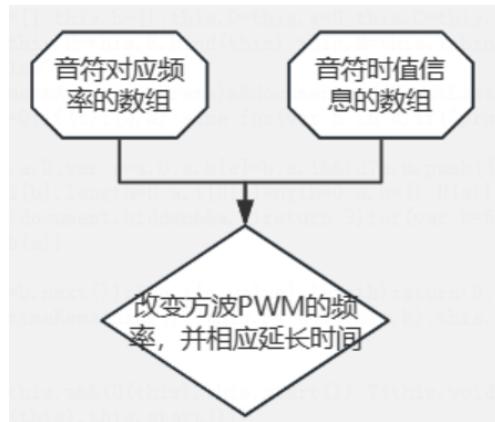
无源蜂鸣器：VCC连接3.3V，GND接地，蜂鸣器I/O口接GPIO17，用来输出不同频率的方波信号以产生不同声音。

无源蜂鸣器



## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

有源蜂鸣器：

```

import RPi.GPIO as GPIO
import time

Buzzer = 11

def setup(pin):
    global BuzzerPin
    BuzzerPin = pin
    GPIO.setmode(GPIO.BARD)
    GPIO.setup(BuzzerPin, GPIO.OUT)
    GPIO.output(BuzzerPin, GPIO.HIGH)

def on():
    GPIO.output(BuzzerPin, GPIO.LOW)
    #低电平是响
def off():
    GPIO.output(BuzzerPin, GPIO.HIGH)
    #高电平是停止响
def beep(x):
    on()
    time.sleep(x)
    off()
    time.sleep(x)

def loop():
    while True:
        beep(0.1)

def destroy():
    GPIO.output(BuzzerPin, GPIO.HIGH)
    GPIO.cleanup()

if __name__ == '__main__':
    setup(Buzzer)
    try:

```

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

以上有源蜂鸣器发声的代码非常简单，配置好GPIO口后，输出低电平时蜂鸣器就开始响，输出高电平蜂鸣器就停止响，因此不断交替输出高低电平就可以实现蜂鸣器交替响的效果。

无源蜂鸣器：

```
import RPi.GPIO as GPIO
import time

Buzzer = 11

song_1 = [
    493.883301256124, 554.365261953744, 587.329535834815,
    554.365261953744, 587.329535834815, 739.988845423269,
    554.365261953744, 369.994422711634, 369.994422711634,
    493.883301256124, 440, 493.883301256124, 587.329535834815,
    440, 369.994422711634, 369.994422711634, 391.995435981749,
    369.994422711634, 391.995435981749, 587.329535834815,
    369.994422711634, 587.329535834815, 587.329535834815,
    587.329535834815, 554.365261953744, 415.304697579945,
    415.304697579945, 554.365261953744, 554.365261953744,
    493.883301256124, 554.365261953744, 587.329535834815,
    554.365261953744, 587.329535834815, 739.988845423269,
    554.365261953744, 369.994422711634, 369.994422711634,
    493.883301256124, 440, 493.883301256124, 587.329535834815
]

beat_1 = [
    0.5, 0.5, 1.5, 0.5, 1, 1, 3, 1, 1, 1.5, 0.5, 1, 1, 3, 0.5, 0.5,
    1.5, 0.5, 0.5, 1.5, 2.5, 0.5, 0.5, 0.5, 1.5, 0.5, 1, 1, 3, 0.5, 0.5,
    1, 1, 1.5, 0.5, 0.5, 1, 1, 3, 0.5, 0.5
]

def setup():
    GPIO.setmode(GPIO.BRD)
    GPIO.setup(Buzzer, GPIO.OUT)
    global Buzz
    Buzz = GPIO.PWM(Buzzer, 440)
    Buzz.start(99.9)

def loop():
    while True:
        for i in range(0, len(song_1)):
            Buzz.ChangeFrequency(song_1[i])
            time.sleep(beat_1[i] * 0.5)
        time.sleep(3)
```

```

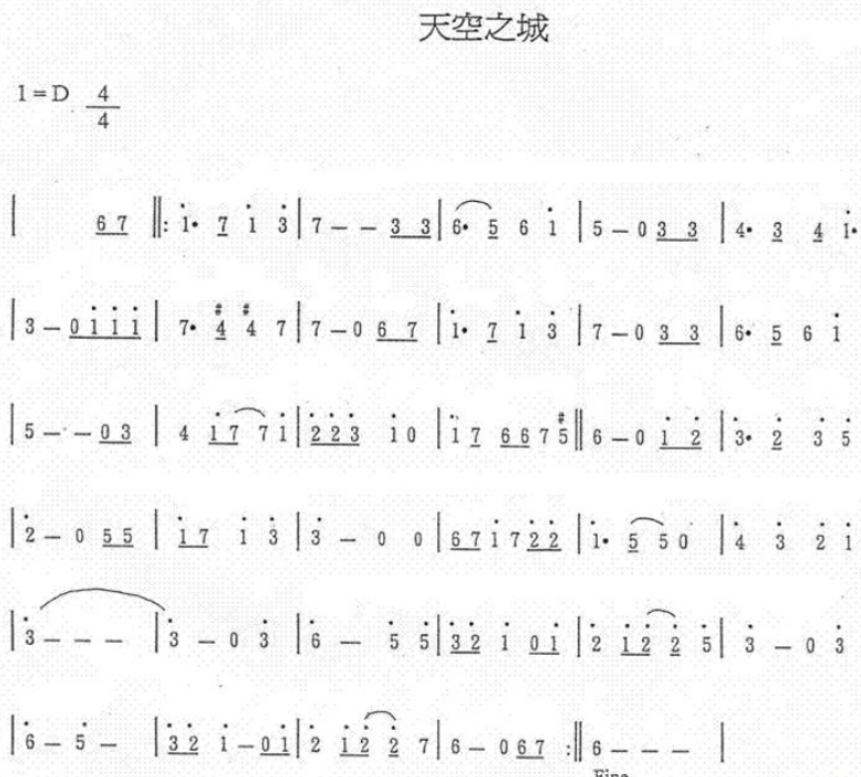
def destroy():
    Buzz.stop()
    GPIO.output(Buzzer, 1)
    GPIO.cleanup()

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

```

无源蜂鸣器即通过配置输出不同频率的PWM波来实现发不同音调的声音。可以通过Buzz = GPIO.PWM(Buzzer, 440), Buzz.start(99.9) 来声明好PWM波的初始频率和占空比，然后使用函数Buzz.ChangeFrequency() 指定固定的频率，然后使用函数 time.sleep()来延时，调整上一个音符的延续时长。将不同的频率信息和每一个频率声音对应的持续时长写成两个数组，就形成了一个简单的乐谱，对数组进行遍历，每个频率和时长都赋值给上述函数，连起来就可以形成该乐谱对应的一段简单的音乐了。

此处我们选择的音乐是“天空之城”，该数组song\_1对应的频率信息就是根据它的乐谱生成出来的，其乐谱为：



为了根据乐谱生成频率信号，我们写了一段MATLAB代码来实现这个过程，如下：

```

function freq = tone2frequency(tone, scale, noctave, rising)

% 编码说明：
% 'tone' 表示音乐谱中的数字符号，表示每个音符的音高，范围从1到7。

```

```
% 'scale' 表示音乐中的调号，输入为C、D、E、F、G、A或B。  
% 'noctave' 表示每个音高的八度数，数值范围为整数。0表示中音八度，正值表示较高的八度，负值表示较低的八度。  
% 'rising' 表示是否有升降号。1 表示升号，-1 表示降号，0 表示没有升号或降号。  
% 理解该代码需要一定的乐理知识，需知道不同调号对应的频率关系，并利用十二平均律对应的关系可以求出每个音调的频率
```

```
c frequency = 261.5;  
index = [2^(2/12) , 2^(2/12) , 2^(1/12) , 2^(2/12) , 2^(2/12) ,  
2^(2/12) , 2^(1/12) ];  
f = c frequency;  
if scale == "C"  
    f scale = f ;  
    int scale = 1;  
elseif scale == "D"  
    for i = 2:2  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;  
    int scale = 2;  
elseif scale == "E"  
    for i = 2:3  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;  
    int scale = 3;  
elseif scale == "F"  
    for i = 2:4  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;  
    int scale = 4;  
elseif scale == "G"  
    for i = 2:5  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;  
    int scale = 5;  
elseif scale == "A"  
    for i = 2:6  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;  
    int scale = 6;  
elseif scale == "B"  
    for i = 2:7  
        f = f*index( mod(i-2, 7)+1);  
    end  
    f scale = f ;
```

```

int scale = 7;
end
if tone>=2
    for i = 2:tone
        f scale = f scale *index( mod(i+( int scale -3), 7)+1);
    end
end
if tone == 0
    freq = 0;
else
    freq = f scale *2^noctave*2^( rising /12) ;
end
end

```

其基本原理就是根据乐理相关知识，将不同的调号以及不同的音调（包括数字简谱中的数字对应的音调和是否升降八度，是否升调降调）。比如对于该首天空之城的音乐，我们调号选为D，并将数字简谱中的音调信息依次输入到函数中去，就可以将其转化为频率了。对于每个音符长短，我们也可以根据其乐谱手动编写出数组beat\_1。

演示效果视频见附件压缩包：“实验7：有源蜂鸣器报警与无源蜂鸣器播放音乐”。

## 实验8：PS2 操纵杆控制LED灯颜色与亮度变化

### 实验目的

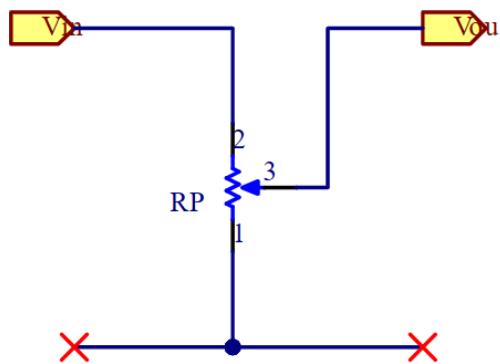
本实验将使用 PS2 操纵杆控制不同的 LED 以及其亮度变化，实现向左推动操纵杆，绿色灯逐渐变亮，向右推动操纵杆，红色灯变亮。

### 实验原理与硬件连接

PS2 操纵杆模块类似于手柄中的模拟游戏杆，是一种输入设备，其在许多项目中得到应用。它是通过以 90 度角安装两个电位计来制成的。电位计连接到以弹簧为中心的短杆上。PS2 操纵杆有两个模拟输出(对应 X 和 Y 坐标)和一个数字输出，表示是否在 Z 轴上按下。处于静止位置时，其在 X 和 Y 方向产生约 2.5V 的输出，移动操纵杆将导致输出在 0v 到 5V 之间变化，具体取决于其方向。按下按钮时，其 SW 引脚输出为低电平。其内部结构实际上就是两个 X, Y 方向上的滑动变阻器。当 VCC 连接 5V 电压时，X, Y 方向电压常态时为 2.5V，最大值 5V，最小值 0V，用 PCF8591 模数转换模块的两个通道分别检测电压值的变化就可以知道摇杆指向的位置了。

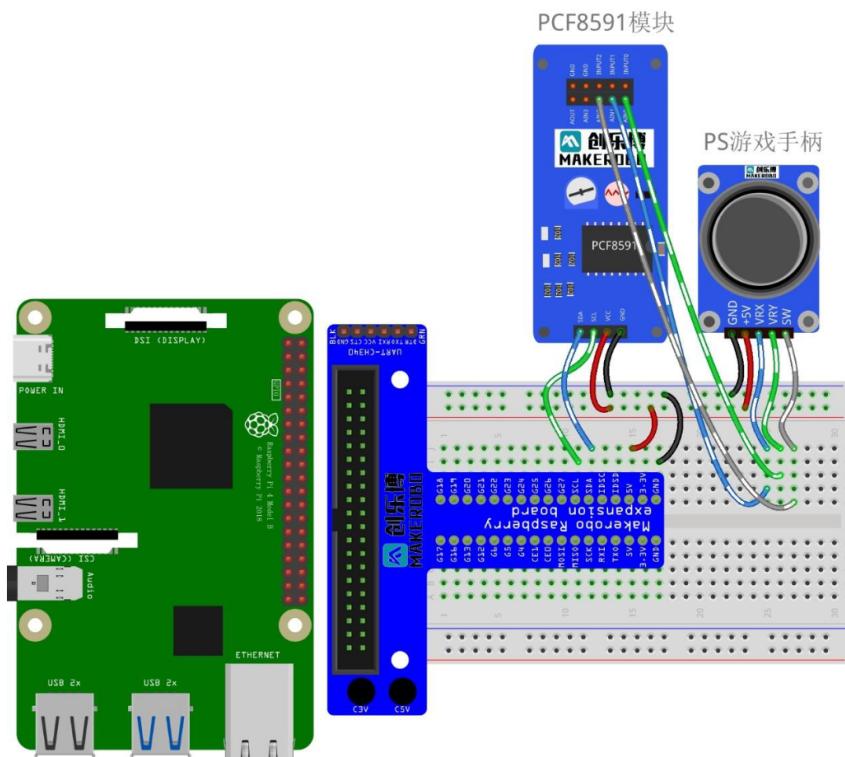


该模块对应的电路原理图为：



在本实验中，我们将引脚 X 和 Y 连接到 A/D 转换器PCF8591的模拟输入端口，以便将模拟量转换为数字量。然后在 Raspberry Pi 上编程以检测操纵杆的移动方向。会在其静止位置读取大约 128 的值(由于弹簧和机构的微小误差而引起的细微变化)。移动操纵杆时，应该看到该值从 0 变为到 255，具体取决于其位置，我们就可以利用该值来控制灯的亮度了。由于PCF8591只有一个输出端口AOUT，所以只能控制一个方向灯的亮度变化，另一方向无法控制其亮度，只可以做到有该方向移动的时候另一种颜色灯变亮，而不产生亮度变化。

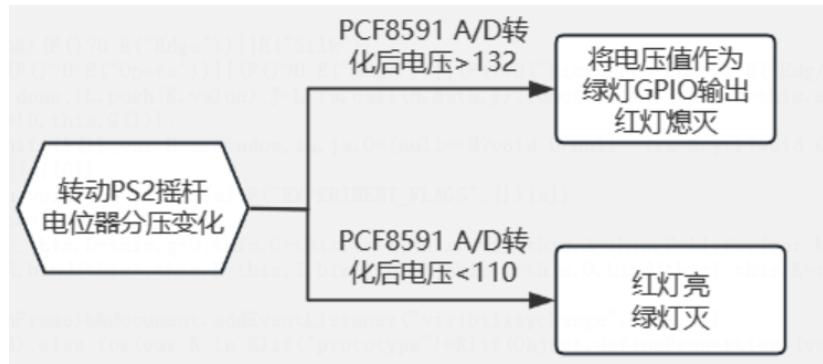
该实验的硬件连线图如下：



同样地，PCF8591模块与树莓派之间按实验4所说的根据I2C通信方式连接。PS2模块的VRY接PCF8591的AIN0，VRX接PCF8591的AIN1，SW接AIN2，以使得PCF8591检测三个方向的值的输出和变化，VCC接5V，GND接地。然后将LED绿色灯引脚接PCF8591的输出引脚AOUT，红色灯对应的引脚接到GPIO13。

## 实验流程与代码实现

该实验的流程或编程思路的流程图如下：



实现代码如下：

PCF8591的相关代码与实验4中的代码相同，包装成PCF8591.py函数：

```

**PCF8591.py**

import smbus
import time

bus = smbus.SMBus(1)

def setup(Addr):
    global address
    address = Addr

def read(chn): #channel
    if chn == 0:
        bus.write_byte(address, 0x40)
    if chn == 1:
        bus.write_byte(address, 0x41)
    if chn == 2:
        bus.write_byte(address, 0x42)
    if chn == 3:
        bus.write_byte(address, 0x43)
    bus.read_byte(address)
    return bus.read_byte(address)

def write(val):
    temp = val
    temp = int(temp)
    bus.write_byte_data(address, 0x40, temp)

if __name__ == "__main__":
    setup(0x48)
    while True:
        print('电位计 AIN0 = ', read(0))
        print('光敏电阻 AIN1 = ', read(1))
        print('热敏电阻 AIN2 = ', read(2))
        tmp = read(0)

```

```
    write(tmp)
    # time.sleep(2)
```

PS2操纵杆控制灯颜色、亮度变化代码：

```
import PCF8591 as ADC
import RPi.GPIO as GPIO
import time

def setup():
    ADC.setup(0x48)          # Setup PCF8591
    global state
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(13, GPIO.OUT)

def direction():      #获取操纵杆方向结果
    state = ['home', 'up', 'down', 'left', 'right', 'Button pressed']
    i = 0
    intensity = 0
    if ADC.read(0) > 132:
        intensity = ADC.read(0)
        ADC.write(intensity*2)
        GPIO.output(13,GPIO.LOW)
    if ADC.read(0) < 110:
        GPIO.output(13,GPIO.HIGH)
        ADC.write(0)
    if ADC.read(0) <= 2:
        i = 1      #up
    if ADC.read(0) >= 253:
        i = 2      #down
    if ADC.read(1) <= 2:
        i = 3      #left
    if ADC.read(1) >= 253:
        i = 4      #right
    if i == 0:
        GPIO.output(13,GPIO.LOW)
        ADC.write(0)
    return state[i]

def loop():
    status = ''
    while True:
        tmp = direction()
        if tmp != None and tmp != status:
            status = tmp

def destroy():
    pass
```

```

if __name__ == '__main__':
    global status
    global tmp
    setup()
    try:
        loop()
    except KeyboardInterrupt:
        destroy()

```

该代码通过使用PCF8591读取PS2传来的电压值，进行A/D转化后，如果ADC.read(0) > 132，说明PS2向左扭动，这时让绿色灯亮度随着PS2向左扭动程度而增大，所以将其读取的值按比例写入到输出引脚：intensity = ADC.read(0)，ADC.write(intensity\*2)，同时红色灯熄灭：GPIO.output(13,GPIO.LOW)。当检测到PS2向右扭动时，即ADC.read(0) < 110，直接GPIO.output(13,GPIO.HIGH)将红色点亮，同时绿色灯熄灭。上述代码也实现了PS2不同方向扭动时，将对应的扭动方向打印出来。

演示效果视频见附件压缩包：“实验8：PS2 操纵杆控制LED灯颜色与亮度变化”。

## 实验9：红外遥控模块按键读取

### 实验目的

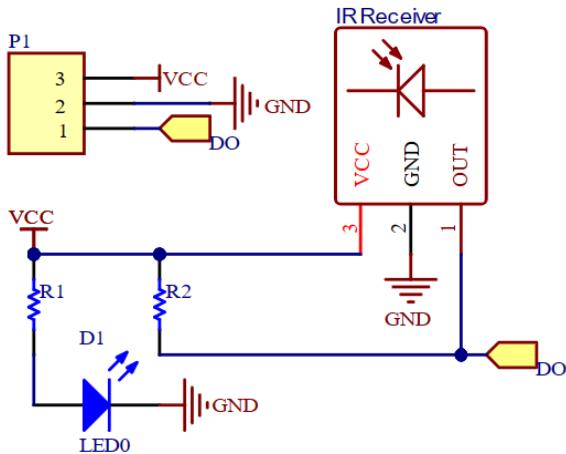
本实验将使用红外遥控器模块，通过按下不同按钮产生不同的红外信号，并利用树莓派识别到哪个按钮被按下，并将它们转换为按钮值，打印出来。

### 实验原理与硬件连接

遥控器接收头是用于接收遥控器所发射信号进而读取按键信息而执行操作的一种光电信号转换器件。遥控器接收头是利用最新的 IC 技术开发和设计出来的小型红外控系统接收器。在支架上装着 PIN 二极管和前置放大器，环氧树脂包装成一个红外过滤器，解调输出信号可以由微处理器解码，一般三条腿的红外线遥控接收头是接收、放大、解调一体头，接收头出的是解调后的数据信号，Raspberry pi 里面需要相应的读取程序。红外通信是利用红外技术实现两点间的近距离保密通信和信息转发。它一般由红外发射和接收系统两部分组成。发射系统对一个红外辐射源进行调制后发射红外信号，而接收系统用光学装置和红外探测器进行接收，就构成红外通信系统。



该模块对应的电路原理图为：



该实验的硬件为：D0接GPIO17，VCC接5V，GND接地。

在本实验中，我们使用 lirc 库读取遥控器按钮返回的红外信号，并将它们转换为按钮值。首先我们打开命令终端，输入以下指令，安装 lirc：

```
sudo apt-get update
sudo apt-get install lirc
```

然后修改配置文件，在命令终端输入 `sudo nano /boot/config.txt` 找到

```
#dtoverlay= gpio-ir,gpio_pin=17
#dtoverlay= gpio-ir-tx,gpio_pin=18
```

删除这两句前面的 # 符号。这里其中第一句是红外接收引脚，第二句是红外发送引脚，采用的是BCM编号模式。

接着设置驱动文件，在命令终端输入

```
sudo nano /etc/lirc/lirc_options.conf
```

把

```
driver = devinput
device = auto
```

修改为

```
driver = default
device = /dev/lirc1
```

之后重启树莓派

```
sudo reboot
```

然后测试红外信号，把红外接收模块接好，注意模块的信号输出端是接在BCM17号上。然后在命令终端输入

```
sudo service lircd stop  
mode2 -d /dev/lirc1
```

此时用遥控器对着红外接收器，随便按几个按钮。会打印出若干信息。

然后，我们可以查看可用遥控键名，在命令终端输入，并记录下输出的信息。

```
irrecord -l
```

最后我们开始录制按键信息，就如同手机指纹密码要录指纹一样：

首先输入

```
irrecord -d /dev/lirc1 ~/lircd.conf
```

屏幕出现

```
Press RETURN to continue.
```

按一下回车。等待屏幕出现

```
Enter name of remote (only ascii, no spaces) :
```

然后输入文件名，文件名只能纯英文字符并且不能有空格。回车后屏幕出现

```
Press RETURN now to start recording.
```

再次按下回车后，树莓派便开始记录按键。这时需要轮流随机按遥控器上的按键，每按下一个按键屏幕就会出现一个点。一直重复随机按下遥控器的上按键，直到屏幕出现

```
Please enter the name for the next button (press <ENTER> to finish recording)
```

然后需要为遥控器上的按键配置名字，名字只能从之前查看可用按键名词里面取，例如输入 KEY\_1 然后按下回车屏幕出现

```
Now hold down button "KEY_1".
```

按下用遥控器的数字 1 键，然后屏幕出现

```
Please enter the name for the next button (press <ENTER> to finish recording)
```

输入 KEY\_2，回车，等待屏幕出现

```
Now hold down button "KEY_2".
```

然后按下遥控器的数字 2，以此重复按下所有的键

最后需要重复的按遥控器上的同一个按键。注意不是长按！直到屏幕出现

```
Successfully written config file myir.lircd.conf
```

然后复制文件到Lirc目录下

```
sudo cp myir.lircd.conf /etc/lirc/lircd.conf.d
```

然后重命名文件，重命名devinput.lircd.conf需要改名为devinput.lircd.dist

```
cd /etc/lirc/lircd.conf.d  
sudo mv devinput.lircd.conf devinput.lircd.dist
```

测试录制后的按键输出

```
sudo service lircd restart  
sudo lircd --nodaemon --device /dev/lirc1 --driver default  
sudo irw
```

最后我们可以关联我们的python 程序以实现按不同按键，打印出不同的按键名

修改文件名

```
cd /etc/lirc  
sudo mv irexec.lircrc lircrc
```

配置lircrc 文件

```
sudo nano lircrc
```

其中prog = test.py 即为关联的程序名称，button = KEY\_1 即刚刚记录的按键名，Config = echo "KEY\_1" 即传递给程序的消息，可以设置为按键名。

需关联的python代码如下：

```
import lirc  
  
def pasreset(data): #解析按键  
    if data == 'echo "KEY_1"':  
        print("1 按下") #遥控器按下1:  
    elif data == 'echo "KEY_2"':  
        print("2 按下") #遥控器按下2:  
    elif data == 'echo "KEY_3"':  
        print("3 按下") #遥控器按下3:  
  
    with lirc.LircdConnection("test.py",) as conn:  
        while True:  
            string = conn.readline()  
            pasreset(string)  
            #print("收到:",end = '')  
            #print(type(string))
```

这时按不同的按键，屏幕就会打印出相关的信息了。

演示效果视频见附件压缩包：“实验9：红外遥控模块按键读取”。

**拓展功能：LCD1602模块显示系统状态**

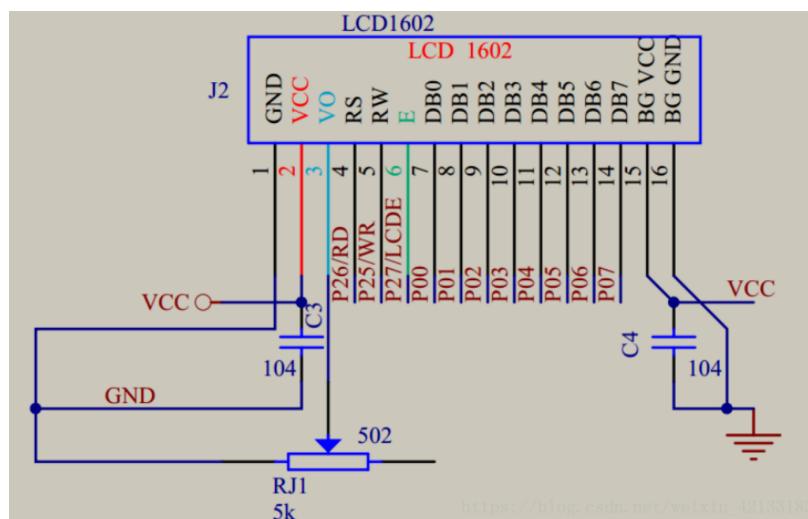
## 实验介绍和原理

除了上面模块和功能，我们也可以使用树莓派实现一些其他较为有用的功能。在之前实验中为了进行远程连接和控制，需要获取树莓派的ip地址，要么在树莓派连显示屏，要么通过其他各种方式抓包获得树莓派此时的ip地址，都显得十分麻烦。如果我们可以外接一个小显示屏，写一段程序将其ip打印在显示屏上，如果让这段程序开机自动运行，就可以获得树莓派的ip了，十分容易和便捷。在这里我们使用了LCD1602模块作为液晶显示屏来显示ip地址，还可以显示系统当前时间，以及任何想要查看的任何其他系统状态等。所使用到的模块为LCD1602：



它是一块液晶显示屏，与很多很多液晶显示屏一样有一大缺点，就是当它们连接到控制器时，需要占用大量的IO口，但是一般的控制器没有那么多的外部端口，也限制了控制器的其他功能。因此，具有使用I2C组件的LCD1602可以解决该问题，LCD1602是一种只用来显示字母、数字、符号等的点阵型液晶模块。我们可以使用了一款通过I2C总线扩展IO的芯片，PCF8574，与LCD1602连接，这样它与树莓派可以通过I2C进行通信，而不需要占用过多的IO口。字符串型液晶显示模块是由字符型液晶显示屏LCD、控制驱动主电路HD44780/KS0066及其扩展驱动电路HD44100或与其兼容的IC，少量阻、容元件结构件等装配在PCB板上而成。

其原理图如下：



背后的原理较为复杂，这里只是简单的介绍和不做详细展开。

LCD（液晶显示器）的工作原理基于液晶材料的光学性质。液晶分子在电场的作用下可以改变其方向，从而调节光的透過程度。液晶显示器通常包含两个玻璃基板，之间夹有液晶层和电极。通过控制电极上的电压，可以使液晶分子排列成不同的方式，从而调节透光程度，实现显示效果。其中的"1602"表示这个LCD模块有16列和2行，即每行可以显示16个字符。LCD1602模块通常使用HD44780或类似的控制器芯片。这个控制器负责接收来自主控制器（这里使用树莓派）的指令和数据，并将它们转换成LCD上的字符显示。与主控制器的连接通常包括数据线和命令线。数据线用于传输要显示的字符的ASCII码，而命令线用于发送LCD的控制指令，例如清屏、设置光标位置等。

在液晶屏上显示字符涉及到液晶显示模块内部的存储器机制，其中LCD1602模块包含三种存储器：CGROM、CGRAM、DDRAM。其中，DDRAM（Display Data RAM）充当显示数据的暂存区，总共包含80个字节的空间，相当于PC机的显存。将待显示的字符代码存储在DDRAM中，即将需要显示的字符代码送入显存，这样就能在屏幕上显示所需的字符。每个LCD1602模块都具有80个字节的DDRAM，但由于LCD1602的显示能力有限，每次只能显示16个字符。因此，即使将字符代码写入DDRAM，也并不是每个字符都能在屏幕上显示出来。实现在液晶屏上显示字符的关键是向DDRAM中写入字符的字符代码。例如，如果要在屏幕上显示字符"A"，就需要将字符代码为41H的字符写入DDRAM的00H地址处。LCD1602模块还包含固化的字模存储器，即CGROM和CGRAM。HD44780控制器内置了192个常用字符的字模，存储在字符产生器CGROM中。此外，还有8个用户可自定义的字符产生RAM，称为CGRAM，共有8个地址用于自定义字符或图形。如果要在屏幕上显示CGROM中已有的字符，只需在DDRAM中写入相应的字符代码即可。如果想显示CGROM中不存在的字符，例如美元符号，首先需要在CGRAM中定义该字符，然后在DDRAM中写入自定义字符的字符代码。这样，通过合理地使用这三种存储器，可以在LCD1602上实现显示所需的字符和符号。

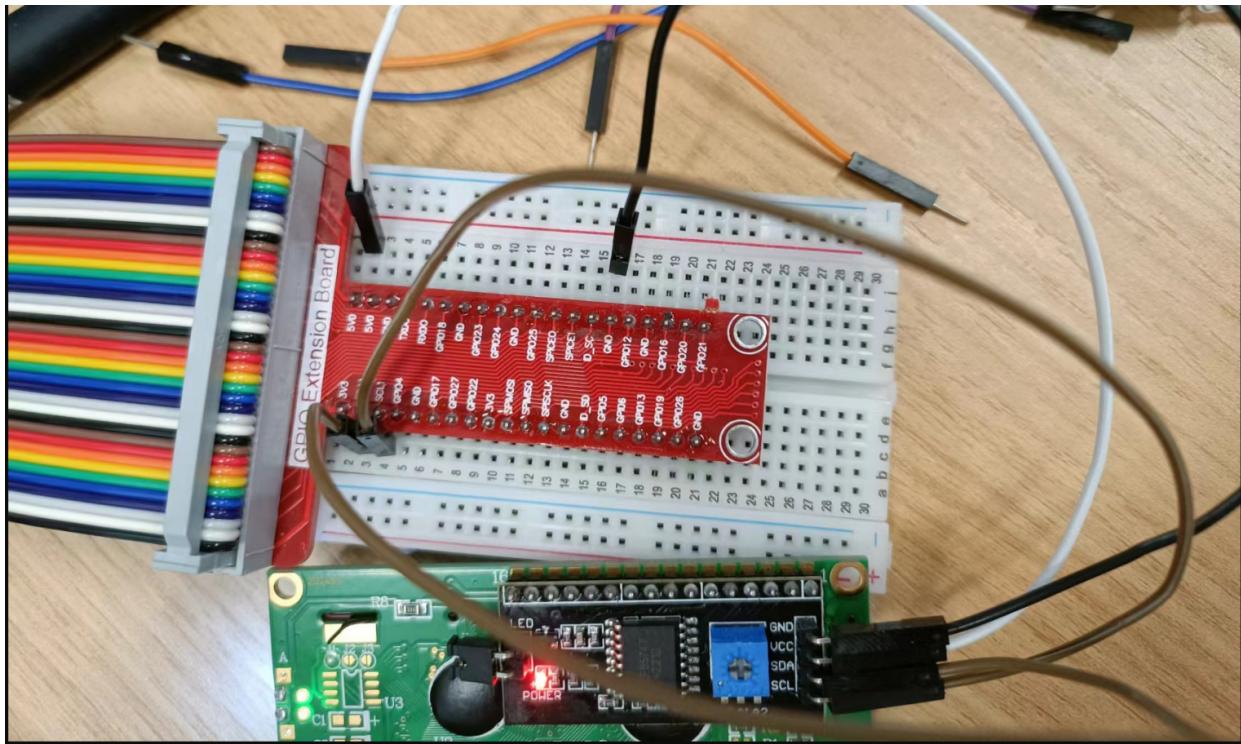
对LCD1602的基本操作涉及到模块内部的两个8位寄存器：指令寄存器（IR）和数据寄存器（DR）。用户可以通过RS和R/W输入信号的组合选择指定的寄存器，进行相应的操作。LCD1602的基本操作有以下：

1. 读状态：输入RS=0, RW=1, E=高脉冲。输出：D0—D7为状态字。
2. 读数据：输入RS=1, RW=1, E=高脉冲。输出：D0—D7为数据。
3. 写命令：输入RS=0, RW=0, E=高脉冲。输出：无。（写完置E=高脉冲）
4. 写数据：输入RS=1, RW=0, E=高脉冲。输出：无。

通过查找数据手册相关内容，包括地址和命令方式就可以实现对其编程操作。

## 实验步骤

首先连接电路，连接图如下：



首先，我们需要查询LCD1602的地址。得出地址为0x27：

```
● ullr@raspberrypi:~ $ ls /dev/i2c-*
/dev/i2c-1  /dev/i2c-20  /dev/i2c-21
● ullr@raspberrypi:~ $ sudo i2cdetect -y 1
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          - - - - - - - - - - - - - - - - - - - -
10:          - - - - - - - - - - - - - - - - - - - -
20:          - - - - - - - - 27 - - - - - - - - - - - -
30:          - - - - - - - - - - - - - - - - - - - -
40:          - - - - - - - - - - - - - - - - - - - -
50:          - - - - - - - - - - - - - - - - - - - -
60:          - - - - - - - - - - - - - - - - - - - -
70:          - - - - - - - - - - - - - - - - - - - -
```

然后可以编写函数LCD1602.py，用来驱动LCD1602，如下：

```
**LCD1602.py**
import time
import smbus

BUS = smbus.SMBus(1)

def write_word(addr, data):
    global BLEN
    temp = data
    if BLEN == 1:
        temp |= 0x08
    else:
        temp &= 0xF7
    BUS.write_byte(addr, temp)

def send_command(comm):
    buf = comm & 0xF0
    buf |= 0x04
```

```
write_word(LCD_ADDR ,buf)
time.sleep(0.002)
buf &= 0xFB
write_word(LCD_ADDR ,buf)

buf = (comm & 0x0F) << 4
buf |= 0x04
write_word(LCD_ADDR ,buf)
time.sleep(0.002)
buf &= 0xFB
write_word(LCD_ADDR ,buf)

def send_data(data):
    buf = data & 0xF0
    buf |= 0x05
    write_word(LCD_ADDR ,buf)
    time.sleep(0.002)
    buf &= 0xFB
    write_word(LCD_ADDR ,buf)

    # Send bit3-0 secondly
    buf = (data & 0x0F) << 4
    buf |= 0x05
    write_word(LCD_ADDR ,buf)
    time.sleep(0.002)
    buf &= 0xFB
    write_word(LCD_ADDR ,buf)

def init(addr, b1):
    global LCD_ADDR
    global BLEN
    LCD_ADDR = addr
    BLEN = b1
    try:
        send_command(0x33)
        time.sleep(0.005)
        send_command(0x32)
        time.sleep(0.005)
        send_command(0x28)
        time.sleep(0.005)
        send_command(0x0C)
        time.sleep(0.005)
        send_command(0x01)
        time.sleep(0.005)
        send_command(0x06)
        BUS.write_byte(LCD_ADDR, 0x08)
    except:
        return False
    else:
```

```

        return True

def clear():
    send_command(0x01)

def write(x, y, str):
    if x < 0:
        x = 0
    if x > 15:
        x = 15
    if y < 0:
        y = 0
    if y > 1:
        y = 1
    addr = 0x80 + 0x40 * y + x
    send_command(addr)

    for chr in str:
        send_data(ord(chr))

```

上述代码实现了通过使用smbus模块来与树莓派的I2C总线通信，实现对LCD1602模块的控制。其中函数 `write_word(addr, data)` 用来向指定地址写入数据，其中 `BLEN` 表示总线位数。函数 `send_command(comm)` 用来向LCD1602发送指令。函数 `send_data(data)` 用来向LCD1602发送数据。函数 `init(addr, b1)` 用来LCD1602的初始化函数，设置LCD1602的基本参数，如显示模式、光标属性等。函数 `clear()` 用来清除LCD1602屏幕上的内容。函数 `write(x, y, str)` 用来在LCD1602指定位置写入字符串，其中 `(x, y)` 表示字符的横纵坐标，`str` 是要显示的字符串，用于在 LCD1602 的指定位置 `(x, y)` 写入字符串 `str`。首先，它通过一系列判断确保横纵坐标在合法范围内，然后计算出地址 `addr`，通过 `send_command` 函数设置显示位置。之后，遍历字符串，通过 `send_data` 函数发送字符的 ASCII 码。

然后编写显示代码，调用上面的函数，显示ip地址和当前的时间（年月日）：

```

import LCD1602
import time
import subprocess
import re
import time
from datetime import datetime

def get_wlan0_ip_address():
    try:
        result = subprocess.check_output(["ifconfig",
"wlan0"]).decode("utf-8")
        match = re.search(r"inet\s+([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)", result)
        if match:
            ip_address = match.group(1)

```

```

        return ip_address
    else:
        return None
except Exception as e:
    print(f"Error: {e}")
    return None

def get_current_time():
    try:
        current_time = datetime.now()
        return current_time
    except Exception as e:
        print(f"Error: {e}")
        return None

def setup():
    LCD1602.init(0x27, 1)
    time.sleep(2)

def loop():
    ip_address = get_wlan0_ip_address()
    current_time = get_current_time()
    if current_time:
        formatted_date = current_time.strftime("%Y-%m-%d")
        time_string = "time: " + str(formatted_date)
        LCD1602.write(0, 0, time_string)
    else:
        LCD1602.write(0, 0, "Unable to retrieve the time")

    if ip_address:
        info = "IP:" + ip_address
        LCD1602.write(1, 1, info)
    else:
        LCD1602.write(1, 1, "Unable to retrieve wlan0 IP address")

def destroy():
    LCD1602.clear()

if __name__ == "__main__":
    try:
        setup()
        while True:
            loop()
    except KeyboardInterrupt:
        destroy()

```

该代码通过get\_wlan0\_ip\_address()函数获取ip地址，实现方式是首先执行ifconfig命令并捕获输出：

```
result = subprocess.check_output(["ifconfig", "wlan0"]).decode("utf-8")
```

然后使用正则表达式从输出中提取IP地址，即可得到ip。

```
match = re.search(r"inet\s+([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)", result)
```

在get\_current\_time()函数中获取当前时间，只显示年月日，并转化为可以打印的字符串形式。  
然后使用LCD1602.write(0, 0, "string") 和 LCD1602.write(1, 1, "string") 函数分布在第一行和  
第二行在显示屏打印出时间和ip地址。

演示效果视频见附件压缩包：“拓展功能：LCD1602模块显示系统状态”。