

# EE351实验报告

---

微机原理与嵌入式系统

韩骐骏

12212635

# 目录

---

1. EE351实验报告	3
1.1 实验报告列表	3
2. 实验报告列表	4
2.1 树莓派开发环境搭建	4
2.2 双色灯实验	8
2.3 轻触开关实验	10
2.4 PCF8691模数转换器实验	14
2.5 模拟温湿度传感器实验	18
2.6 Lab6实验报告：超声波传感器测距实验	21
2.7 蜂鸣器实验	24
2.8 PS2操纵杆实验	27
2.9 红外遥控实验	29
2.10 中断实验	31

# 1. EE351实验报告

---

此处是南方科技大学EE351“微机原理与嵌入式系统”24Fall课程实验报告的主页，你可以在这里找到所有的实验报告。

本课程所有实验在树莓派4B上进行，使用的操作系统为RaspberryPi OS-64-bit-desktop。

本实验用到的硬件设备包括： - 树莓派4B - PCF8591模数转换器 - 传感器模块（如温度传感器、超声波传感器等） - LED灯、蜂鸣器、电位器等 - PS2操纵杆、红外遥控器等 - 面包板、杜邦线等

本实验用到的软件工具包括： - [RPi.GPIO库](#) (Python) - [wiringPi库](#) (C/C++) - [python-smbus库](#) (I2C通信)

## 1.1 实验报告列表

---

- [实验一：树莓派开发环境搭建](#)
- [实验二：双色灯实验](#)
- [实验三：轻触开关实验](#)
- [实验四：PCF8691模数转换器实验](#)
- [实验五：模拟温湿度传感器实验](#)
- [实验六：超声波传感器实验](#)
- [实验七：蜂鸣器实验](#)
- [实验八：PS2操纵杆实验](#)
- [实验九：红外遥控实验](#)
- [实验十：中断实验](#)

作者: [韩骐骏](#)

## 2. 实验报告列表

### 2.1 树莓派开发环境搭建

#### Lab1实验报告：Raspberry Pi初体验与环境搭建实验

##### 一、实验介绍

本次实验将配置后续实验用到的软硬件环境，包括操作系统、网络、远程连接等。

##### 二、实验目标

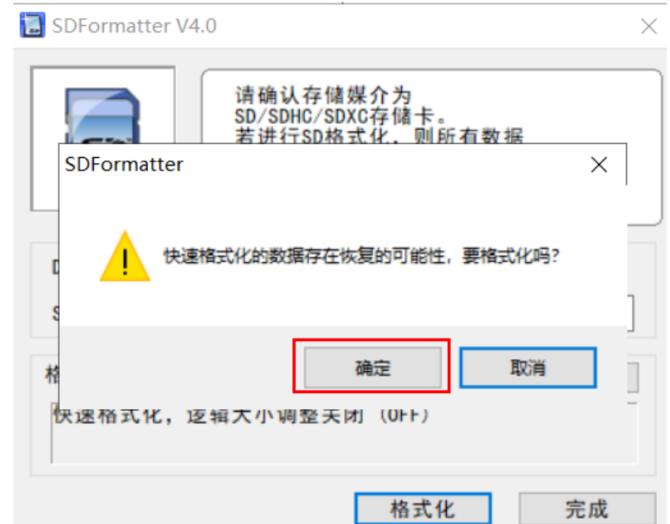
1. 熟悉Raspberry Pi硬件组成及其引脚布局。
2. 完成Raspberry Pi OS镜像的下载与烧录。
3. 配置Wi-Fi，确保能够访问互联网。

##### 三、实验步骤

###### (1) 硬件准备与检查

1. 确认所需材料：
2. Raspberry Pi 4 Model B
3. microSD卡
4. 电源适配器（最好使用官方提供的USB-C电源，否则可能出问题）
5. HDMI显示器及HDMI线缆
6. 键盘和鼠标
7. 检查硬件状态：
8. 插入microSD卡到Raspberry Pi的卡槽中。

• 如果是使用过的卡，先用[SDFormatter](#)工具格式化。



9. 连接显示器、键盘和鼠标（如果打算使用GUI）。
10. 将电源线插入Raspberry Pi，并确保另一端连接到合适的电源插座上。

###### (2) 操作系统安装

1. 下载Raspberry Pi Imager工具：

2. 访问Raspberry Pi官方网站下载Imager工具。

## Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Ubuntu for x86](#)

[Download for Windows](#)

[Download for macOS](#)

To install on **Raspberry Pi OS**, type

```
sudo apt install rpi-imager
```

in a Terminal window.



### 3. 选择并写入OS镜像：

4. 打开Raspberry Pi Imager，点击“CHOOSE OS”按钮，选择推荐的Raspberry Pi OS (32-bit)版本。
5. 点击“CHOOSE STORAGE”，挑选之前准备好的microSD卡作为存储介质。
6. 确认无误后，点击“WRITE”开始烧录过程。请耐心等待，直到提示写入完成。

### 7. 配置屏幕参数：

8. 在microSD卡的根目录下，找到 config.txt 文件，编辑并添加以下内容：

```
1 max_usb_current=1
2 hdmi_force_hotplug=1
3 config_hdmi_boost=7
4 hdmi_group=2
5 hdmi_mode=1
6 hdmi_mode=87
7 hdmi_drive=1
8 display_rotate=0
9 hdmi_cvt 1024 600 60 0 0 0 0
```

>在 hdmi\_cvt 1024 600 60 0 0 0 这里填入实际显示屏的分辨率，不同显示器分辨率不同。

9. 保存文件后，将microSD卡插回Raspberry Pi中。

### 10. 初次启动与初始化设置：

11. 将烧录好OS镜像的microSD卡重新插回Raspberry Pi后，给它通电。
12. 第一次启动时，根据屏幕上的指示进行语言、地区、时区等基本信息的配置。

(3) 网络配置

#### 1. 连接Wi-Fi：

2. 在命令行中输入 sudo raspi-config 打开配置菜单。
3. 选择“Network Options”，然后按照提示输入您的Wi-Fi SSID和密码。
4. 或者直接编辑 /etc/wpa\_supplicant/wpa\_supplicant.conf 文件添加Wi-Fi信息。
5. 验证网络连接：

6. 使用 `ping www.bing.com` 测试是否能成功访问外部网站。

(4) 配置开发环境

1. 更新软件包列表：

2. 执行 `sudo apt-get update` 刷新本地数据库以获取最新的软件包信息。

3. 升级已安装的软件包：

4. 使用 `sudo apt-get upgrade` 命令来更新所有现有的软件包至最新版本。

5. 安装额外的开发工具：

6. 安装Python相关工具：

```
1 sudo apt-get install python3-pip
2 pip3 install --upgrade pip
```

7. 安装Git用于版本控制：

```
1 sudo apt-get install git
```

8. 安装Vim编辑器： bash

```
sudo apt-get install vim
```

9. 安装远程连接工具：

10. 启动SSH服务：

```
1 sudo systemctl start ssh
```

11. 验证SSH服务是否正常运行：

```
1 sudo systemctl status ssh
```

12. 设置SSH服务开机自启动： bash

```
sudo systemctl enable ssh
```

13. 在本地通过vscode连接远程Raspberry Pi：

14. 安装Remote - SSH插件。

15. 在树莓派终端运行 `ifconfig` 命令查看IP地址。

16. 使用 `Ctrl+Shift+P` 打开命令面板，输入 `Remote-SSH: Connect to Host`，然后输入Raspberry Pi的IP地址和用户名。

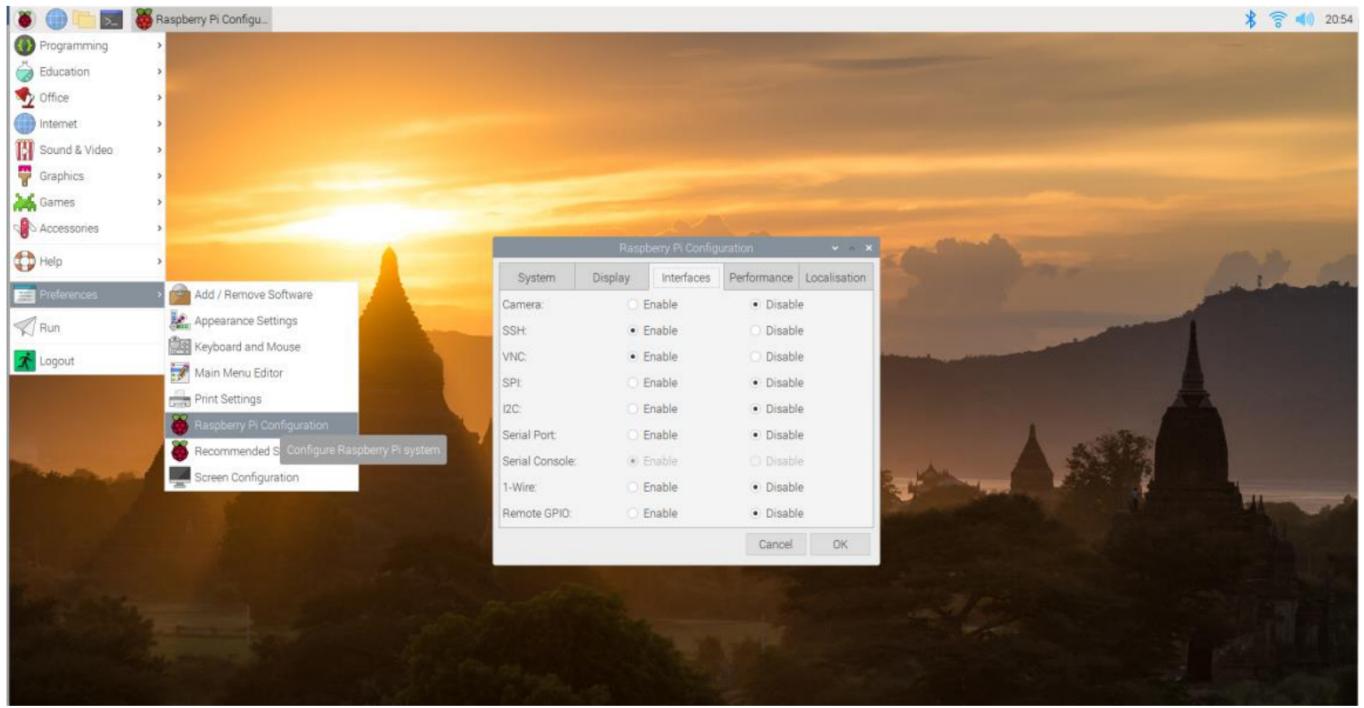
17. 输入密码后，即可通过VSCode连接到远程的Raspberry Pi。

18. 把 `ssh-rsa` 公钥添加到 `~/.ssh/authorized_keys` 文件中，实现无密码登录。

19. 远程可视化开发：

20. 在本地安装VNC Viewer。

## 21. 在树莓派上启用VNC Server：



22. 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。

23. 输入用户名和密码，即可远程连接到 Raspberry Pi 的桌面环境。

## 四、总结

本实验完成了Raspberry Pi的初步设置，包括操作系统的安装、网络的配置以及开发环境的搭建。

## 2.2 双色灯实验

---

### Lab2实验报告：学习知识准备与双色LED实验

#### 一、实验介绍

本实验旨在了解Raspberry Pi的IO接口及其引脚编号方式，并通过实际操作掌握使用wiringPi库和RPi.GPIO库控制硬件的方法，最终目标是实现一个简单的双色LED红绿交替闪烁效果。

#### 二、实验原理

##### 1. Raspberry Pi IO口：

2. Raspberry Pi拥有40个GPIO管脚，这些管脚可以通过不同的编号系统来引用，包括物理位置编号、wiringPi指定的编号以及BCM2837 SOC指定的编号。
3. 在本次实验（以及后续实验）中，我们使用BCM编码来连接和编程。

##### 4. wiringPi库：

5. wiringPi是一个用于C/C++语言的GPIO控制库，它简化了对Raspberry Pi GPIO的操作。安装此库后，可以方便地在命令行或程序中控制GPIO引脚。

##### 6. RPi.GPIO库：

7. RPi.GPIO是一个Python库，允许用户直接从Python代码中控制Raspberry Pi的GPIO。它是Raspbian操作系统的一部分，默认已安装，因此可以直接调用其API进行编程。

##### 8. Mu编辑器与Geany IDE：

9. Mu是一款适合初学者使用的Python编辑器，提供了基本的IDE功能如语法检查、运行和调试等。
10. Geany则是一款轻量级的跨平台IDE，支持多种编程语言，对于C/C++项目来说非常适合。
11. 后续经验表明，这两款IDE在树莓派下渲染效果不佳，推荐使用VSCode远程开发插件。

##### 12. 双色LED模块：

13. 双色LED通常指的是包含两个独立发光单元（红色和绿色）在一个封装内的LED。通过改变输入电压或电流的方向，可以使不同的颜色发光或者两者同时亮起形成黄色。

#### 三、实验步骤

##### 1. 硬件连线：

2. 将双色LED的S引脚（绿色）、中间引脚（红色）分别连接到Raspberry Pi的GPIO接口上，GND引脚连接到Raspberry Pi的地线。记住使用的GPIO引脚编号（本次实验使用的是BCM编号下的GPIO19, GPIO20, GND）。

##### 3. 编写并上传代码：

4. 使用Mu编辑器创建一个新的Python脚本文件，编写一段代码来控制双色LED的红绿交替闪烁。代码应该设置好相应的GPIO模式（输入/输出），然后按照设定的时间间隔切换LED的状态。
5. 如果使用C/C++，则可以在Geany中新建一个源文件，同样需要配置GPIO模式，并且要记得在编译时链接wiringPi库。

##### 6. 运行测试：

7. 执行编写的Python脚本或编译后的C/C++程序，观察双色LED是否能够按照预期的顺序红绿交替闪烁。
8. 除了代码逻辑，一定要检查硬件连接是否正确无误！！！（比如是否插紧T型板、是否接错了引脚等）

##### 9. 清理工作：

10. 实验结束后，记得关闭所有运行中的进程，否则LED灯可能会一直亮着。

#### 四、PYTHON代码

下面提供了本实验用到的Python代码，用于实现双色LED的红绿交替闪烁：

```
1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pins for the LED (BCM numbering)
5 RED_PIN = 19 # Red part of the dual-color LED
6 GREEN_PIN = 20 # Green part of the dual-color LED
7
8 # Setup GPIO mode and pin directions
9 GPIO.setmode(GPIO.BCM)
10 GPIO.setup(RED_PIN, GPIO.OUT)
11 GPIO.setup(GREEN_PIN, GPIO.OUT)
12
13 try:
14     while True:
15         # Turn on red LED
16         GPIO.output(RED_PIN, GPIO.HIGH)
17         GPIO.output(GREEN_PIN, GPIO.LOW)
18         print("Red LED is ON")
19         time.sleep(1) # Wait for 1 second
20
21         # Turn on green LED
22         GPIO.output(RED_PIN, GPIO.LOW)
23         GPIO.output(GREEN_PIN, GPIO.HIGH)
24         print("Green LED is ON")
25         time.sleep(1) # Wait for 1 second
26
27 except KeyboardInterrupt:
28     print("Program stopped by user")
29
30 finally:
31     # Clean up GPIO settings before exiting
32     GPIO.cleanup()
```

这段代码将使双色LED按照红-绿-红-绿的顺序交替闪烁，每次持续1秒钟。

## 2.3 轻触开关实验

### Lab3实验报告：轻触开关实验

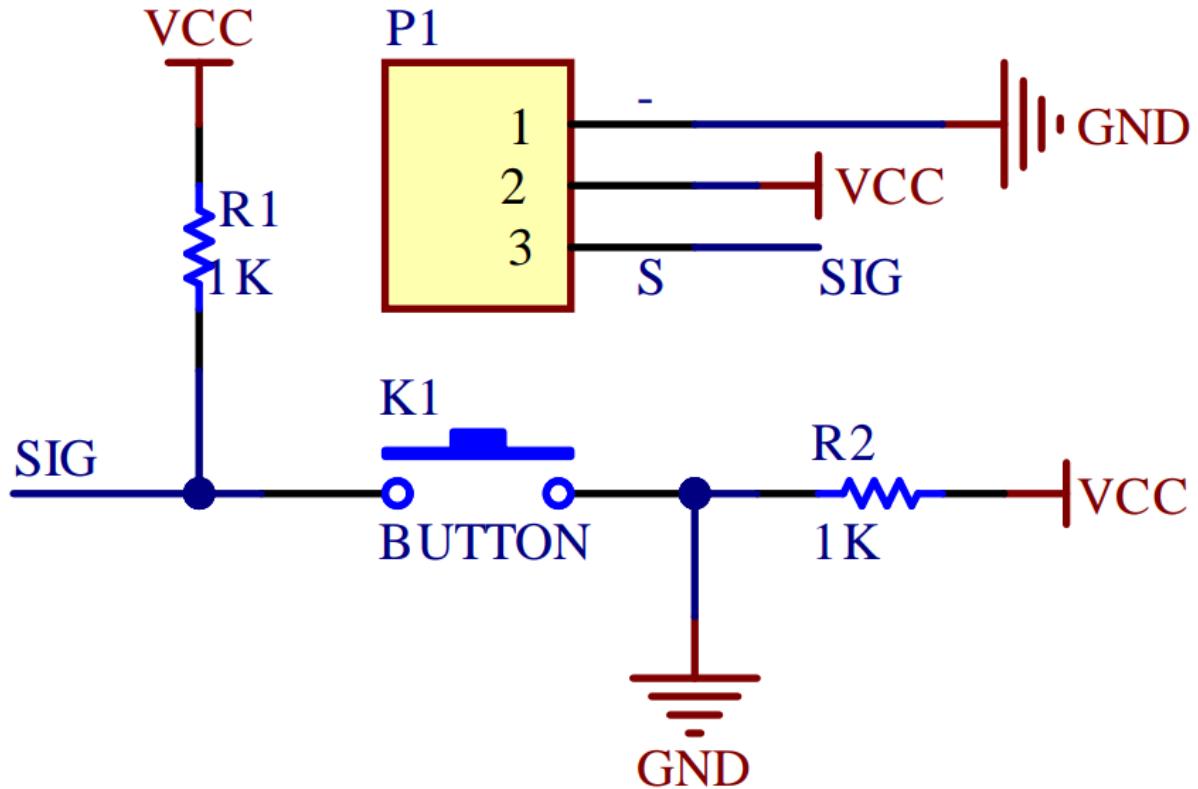
#### 一、实验介绍

轻触开关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

块的原理图为：

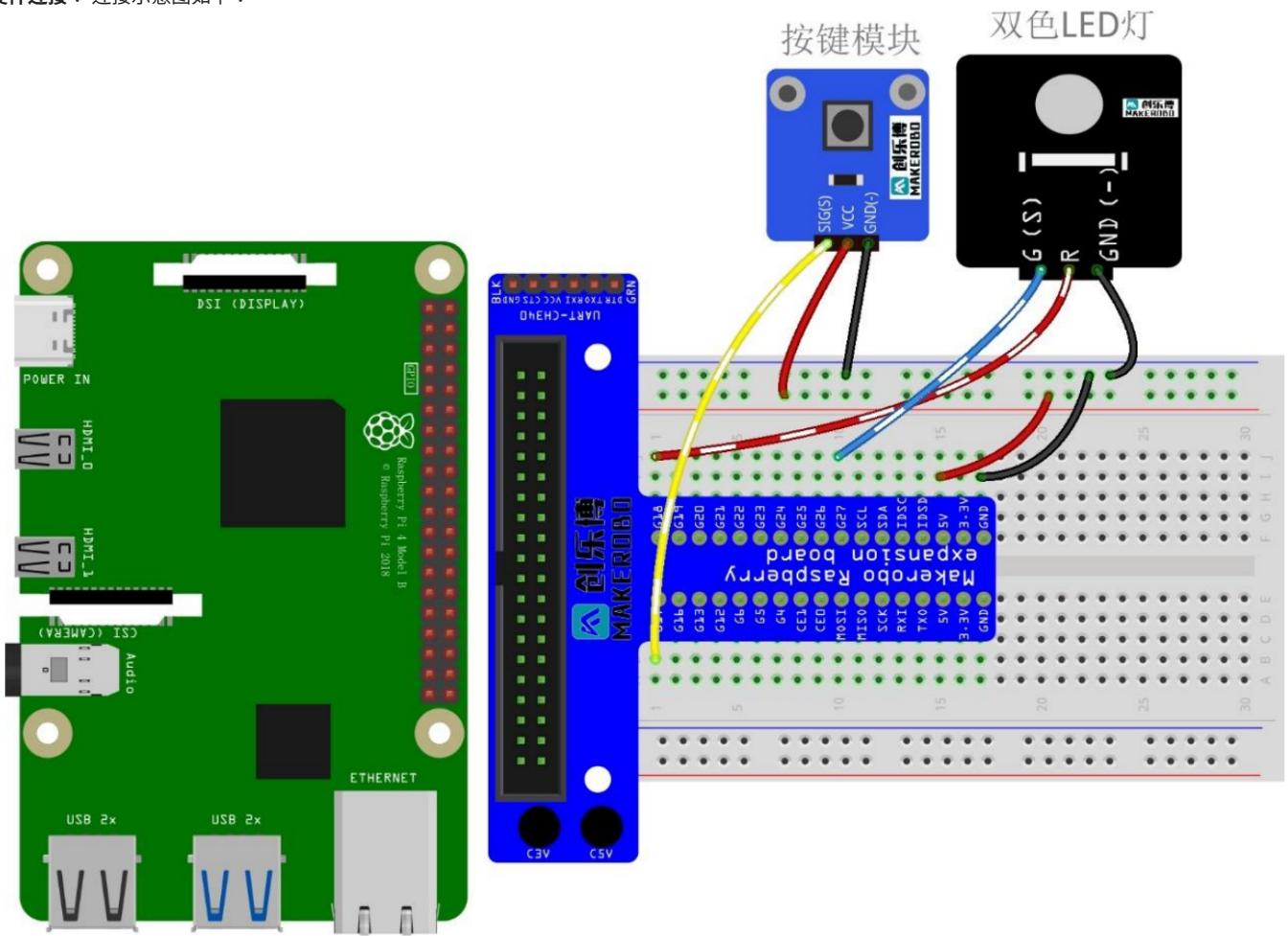


## 二、实验原理

在本实验中，使用轻触开关作为树莓派的输入设备。将树莓派某 GPIO 口设 置为输入模式，通过此 GPIO 口检测轻触开关的 S 引脚。当检测到 S 引脚为低电平时，表示按键被按下，检测到 S 引脚为高电平时，表示按键松开。通过两种不同颜色的 LED 指示按键的状态。即当按键按下时，一种颜色 LED 亮；按键松开时，另一种 LED 亮。通过延时函数能有效地减少按键抖动带来的误触发问题。**延时函数**： - 在Python中，`time.sleep()` 函数可以让程序暂停执行一段时间（秒），这对于控制LED的闪烁频率或者按键的检测非常有用。

## 三、实验步骤

1. 硬件连接：连接示意图如下：



2. 编写代码：使用Mu编辑器或VSCode等工具，编写Python代码来控制双色LED和轻触开关。代码逻辑如下：

3. 设置GPIO引脚的模式（输入/输出）。
4. 不断检测轻触开关的状态，根据按键的状态控制LED的亮灭。

5. 为了防止按键抖动，可以在检测到按键状态变化后加入一个短的延时。

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pins for the LED (BCM numbering)
5 RED_PIN = 19 # Red part of the dual-color LED
6 GREEN_PIN = 20 # Green part of the dual-color LED
7 SWITCH_PIN = 21 # GPIO pin for the tactile switch
8
9 # Setup GPIO mode and pin directions
10 GPIO.setmode(GPIO.BCM)
11 GPIO.setup(RED_PIN, GPIO.OUT)
12 GPIO.setup(GREEN_PIN, GPIO.OUT)
13 GPIO.setup(SWITCH_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
14
15 def switch_with_delay(pin, delay=0.1):
16     state = GPIO.input(pin)
17     time.sleep(delay)
18     return state == GPIO.input(pin)
19
20 try:
21     while True:
22         if switch_with_delay(SWITCH_PIN):
23             GPIO.output(RED_PIN, GPIO.HIGH)
24             GPIO.output(GREEN_PIN, GPIO.LOW)
25         else:
26             GPIO.output(RED_PIN, GPIO.LOW)
27             GPIO.output(GREEN_PIN, GPIO.HIGH)
28
29 except KeyboardInterrupt:
30     print("Exiting...")
31 finally:
32     GPIO.cleanup()

```

## 1. 实验拓展

通过开关和 LED 及相应的编程，实现以下功能：1. 按一下按键，LED 红灯亮起；2. 再次按一下按键，LED 红灯闪烁；3. 再次按一下按键，LED 绿灯亮起；4. 再次按一下按键，LED 绿灯闪烁；再次按下按键红灯亮起……如此循环。

需要注意消除好按键抖动。

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pins for the LED (BCM numbering)
5 RED_PIN = 19 # Red part of the dual-color LED
6 GREEN_PIN = 20 # Green part of the dual-color LED
7 SWITCH_PIN = 21 # GPIO pin for the tactile switch
8
9 # Setup GPIO mode and pin directions
10 GPIO.setmode(GPIO.BCM)
11 GPIO.setup(RED_PIN, GPIO.OUT)
12 GPIO.setup(GREEN_PIN, GPIO.OUT)
13 GPIO.setup(SWITCH_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
14
15 def switch_with_delay(pin, delay=0.1):
16     state = GPIO.input(pin)
17     time.sleep(delay)
18     return state == GPIO.input(pin)
19
20 def toggle_led(pin):
21     GPIO.output(pin, not GPIO.input(pin))
22
23 try:
24     while True:
25         if switch_with_delay(SWITCH_PIN):
26             toggle_led(RED_PIN)
27             time.sleep(0.5)
28             toggle_led(RED_PIN)
29         else:
30             toggle_led(GREEN_PIN)
31             time.sleep(0.5)
32             toggle_led(GREEN_PIN)
33
34 except KeyboardInterrupt:
35     print("Exiting...")
36 finally:
37     GPIO.cleanup()

```

## 2.4 PCF8691模数转换器实验

### Lab5实验报告：PCF8591模数转换器实验

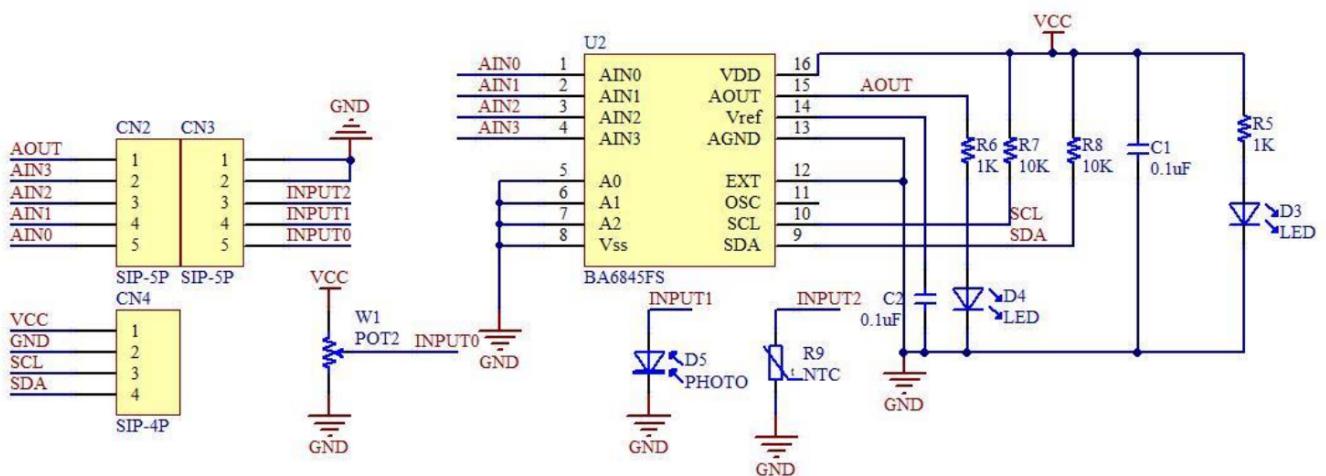
#### 一、实验介绍

PCF8591 是一款单芯片，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(I<sup>2</sup>C)总线接口。三个地址引脚(A\_0, A\_1 和 A\_2)用于对硬件地址进行编程，从而允许使用多达 8 个连接到(I<sup>2</sup>C)总线的设备，而无需额外的硬件。通过两行双向(I<sup>2</sup>C)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(I<sup>2</sup>C)总线的最大速度决定。本次实验目标为：通过控制 PCF8591，将 LED 灯点亮。

#### 二、实验原理

##### 1. PCF8591特性：

2. PCF8591是一款单芯片、低功耗的CMOS数据采集设备，它包含模拟输入多路复用、片上跟踪保持功能、8位A/D转换和8位D/A转换。
3. 设备通过I<sup>2</sup>C总线接口与主控制器通信，默认地址为0x48，但可以通过设置地址引脚A0, A1, 和A2改变其硬件地址，最多允许连接8个相同类型的从设备到同一(I<sup>2</sup>C)总线上。
4. 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。控制寄存器的高半字节用于使能模拟输出，并将模拟输入编程为 单端或差分输入。下半字节选择由上半字节定义的一个模拟输入通道。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。
5. 在本实验中，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。该模块的原理图如下所示：



需要注意的是，除了电位器，PCF8591 模块还带有光电二极管和负温度系数（NTC）热敏电阻，原理图如下所示。当外部光强或温度变化时，光敏或热敏电阻的阻值也会发生变化，通过采集 INPUT1 和 INPUT2 的电压值，可以实现光强和温度感知。

##### 6. I<sup>2</sup>C总线通信：

7. (I<sup>2</sup>C)是一种简单的两线式串行通信标准，由SDA（数据线）和SCL（时钟线）组成。在本实验中，Raspberry Pi作为主设备，负责发送命令给PCF8591并接收来自它的响应。

##### 8. 模拟信号采集与处理：

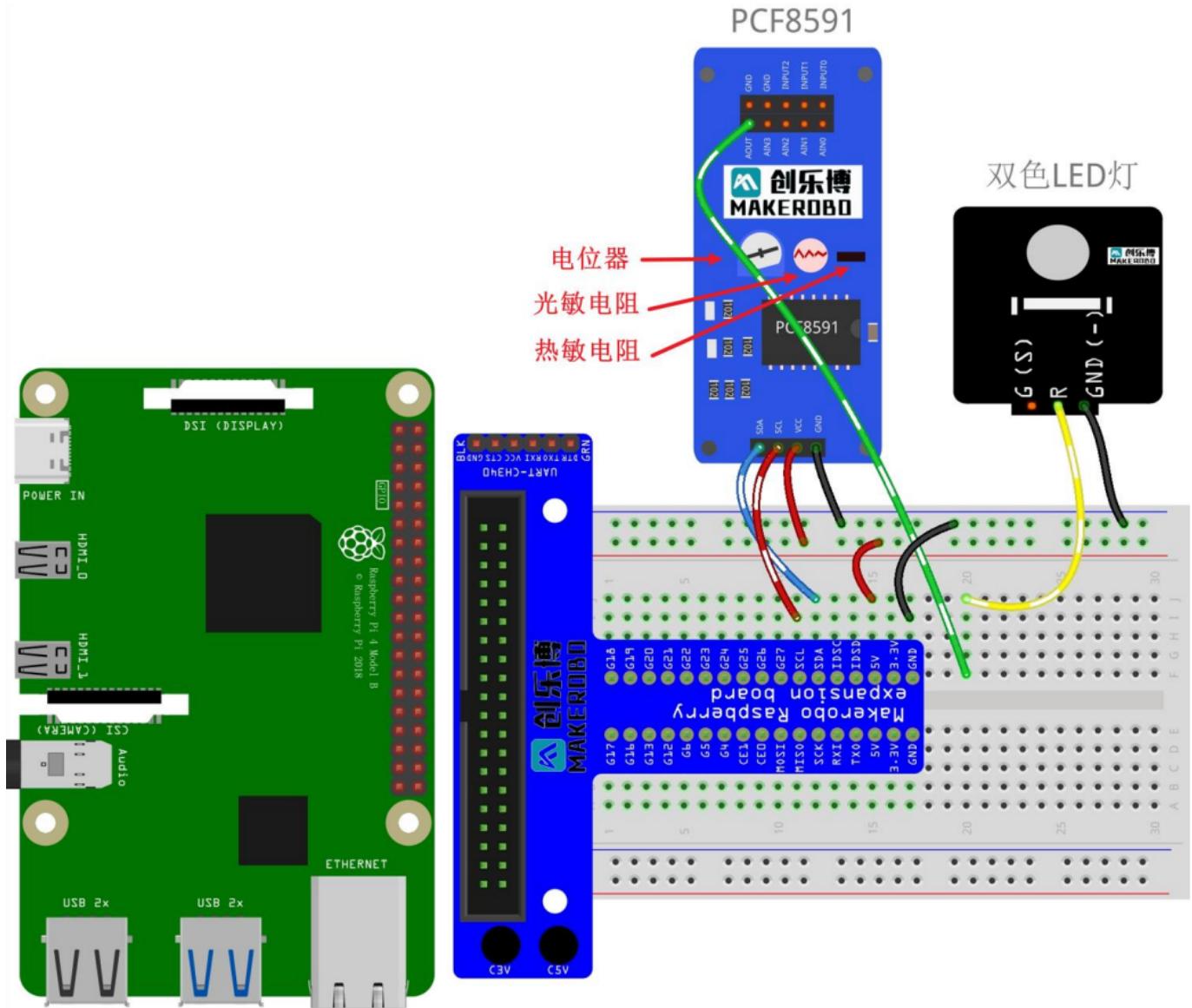
9. 在这个实验里，AIN0端口被用来接收来自电位计模块的模拟信号，而AOUT端口则输出模拟电压以驱动双色LED模块，从而改变LED的亮度。
10. 当外部条件发生变化时（例如光照强度或温度变化），相应的传感器（如光电二极管或NTC热敏电阻）的阻值也会随之变化，通过测量这些元件两端的电压，我们可以得知环境的变化情况。

#### 三、实验步骤

##### 1. 硬件连接：

2. 连接Raspberry Pi、T型转接板和PCF8591模块之间的SDA、SCL、VCC和GND引脚。

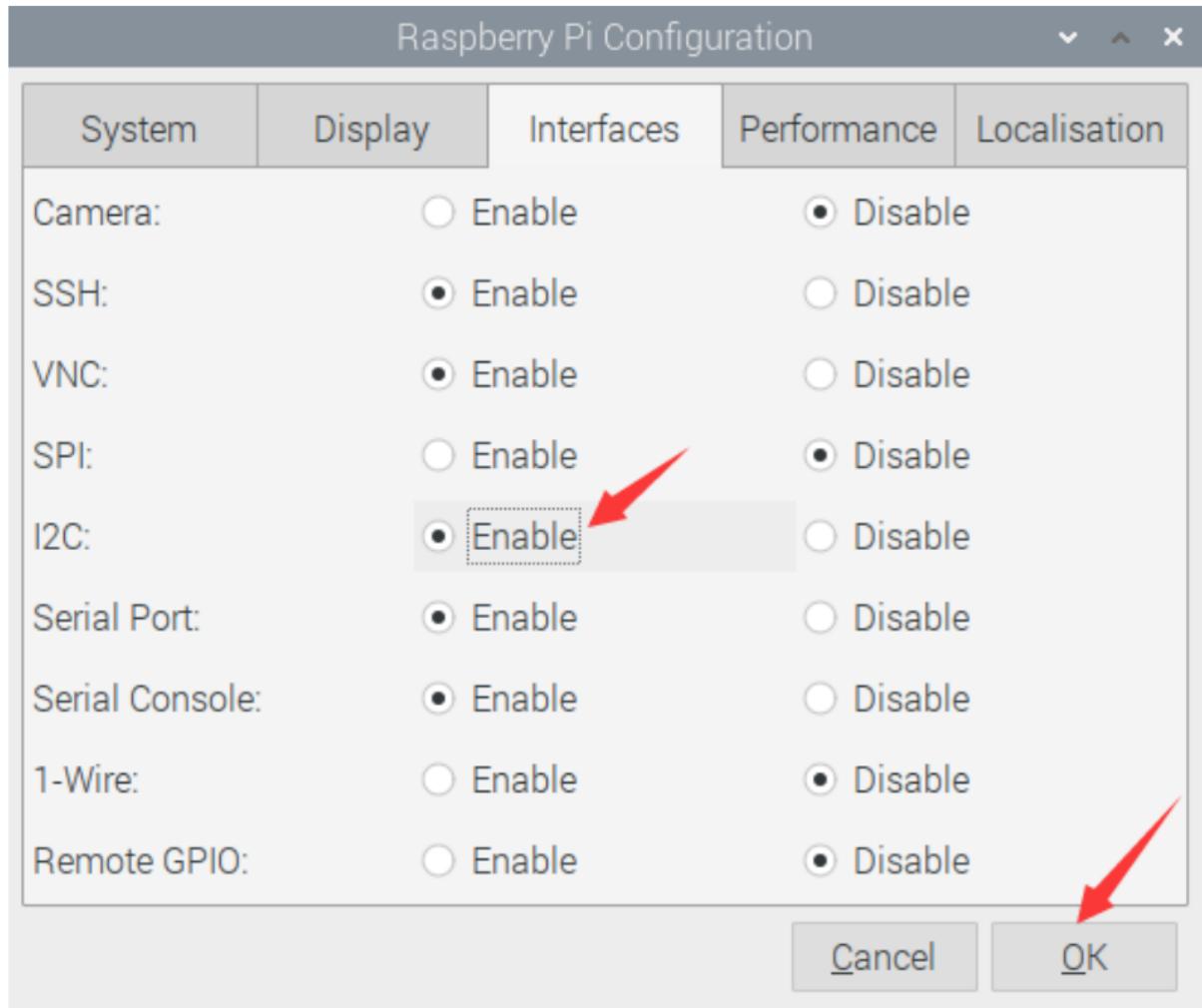
3. 将双色LED的中间引脚（红色）连接到PCF8591的AOUT引脚，GND引脚接地。



4. 配置I2C总线：

5. 点击Raspberry Pi桌面环境中的开始菜单，选择Preferences -> Raspberry Pi Configuration。

6. 进入Interfaces标签页，开启I2C选项，点击OK保存更改并重启系统。



7. 查看设备地址：

8. 在终端中输入 `sudo i2cdetect -y 0` 命令，查看I2C总线上所有设备的地址。

9. 如果一切正常，应该能看到PCF8591的地址（默认为0x48）显示在对应的位置上。

10. 编写代码：

11. 使用Python语言编写程序，首先需要安装 `smbus` 库，它可以方便地操作I2C设备。

12. 导入必要的库后，创建一个SMBus实例并与PCF8591建立连接，读取AIN0上的模拟值并根据该值调整AOUT输出，进而控制LED亮度。

```
1 import smbus
2 import time
3
4 # Define the I2C address of the PCF8591 and control bits
5 address = 0x48 # Default address for PCF8591
6 control_bit = 0x40 # Command to start conversion on channel 0 (AIN0)
7
8 # Initialize the SMBus library
9 bus = smbus.SMBus(1) # Use I2C bus 1
10
11 try:
12     while True:
13         # Write the control byte to initiate an A/D conversion on channel 0
14         bus.write_byte(address, control_bit)
15
16         # Read back the converted value from the PCF8591
17         analog_value = bus.read_byte(address)
18
19         # Print out the raw analog value
20         print("Analog Value:", analog_value)
21
22         # Map the analog value to a range suitable for controlling LED brightness
23         led_brightness = int((analog_value / 255.0) * 100)
24
25         print("LED Brightness (%):", led_brightness)
26
27         time.sleep(0.1) # Small delay between readings
28
29 except KeyboardInterrupt:
30     print("Exiting...")
```

## 2.5 模拟温湿度传感器实验

### Lab5实验报告：模拟温度传感器实验

#### 一、实验介绍

温度感测模块提供易于使用的传感器，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。本次实验的任务为：获取当前环境的温度值。

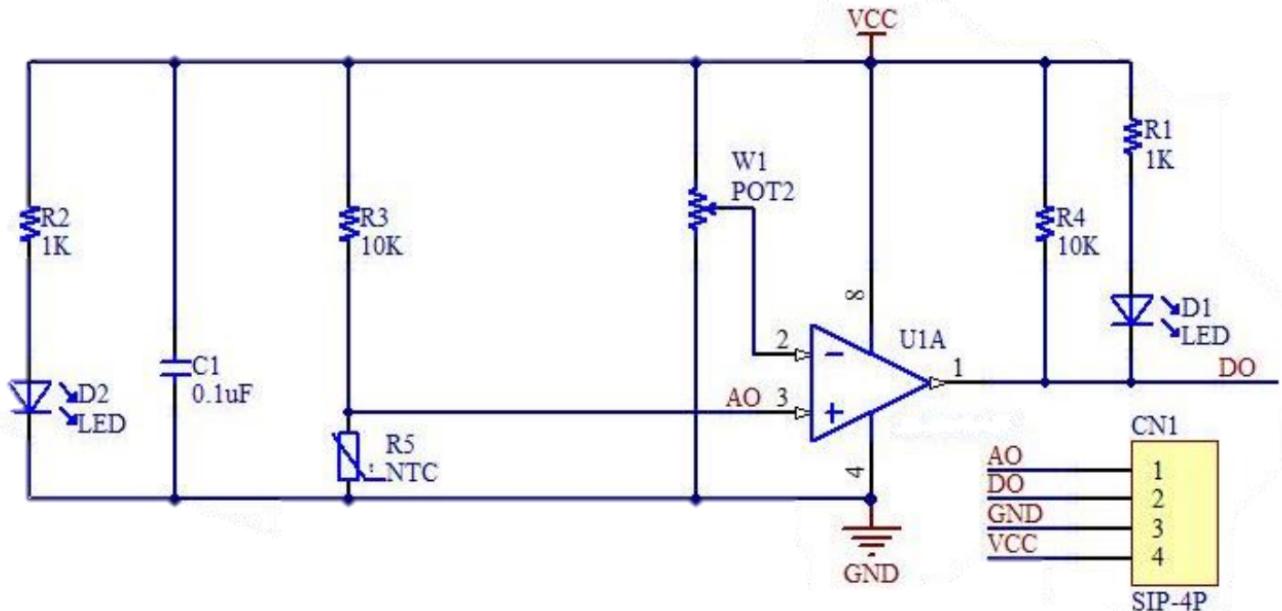
#### 二、实验原理

##### 1. NTC热敏电阻特性：

- 在本实验中，我们将使用Steinhart-Hart方程来计算热敏电阻的精确温度，这是一个用于描述热敏电阻电阻-温度特性的经验公式，即Steinhart-Hart方程。
- Steinhart-Hart方程表达式为  $\frac{1}{T} = A + B \ln(R) + C(\ln(R))^3$ ，其中  $T$  是以开尔文为单位的绝对温度，  $R$  是热敏电阻在给定温度下的电阻值，而  $A$ ,  $B$ ,  $C$  则是取决于具体型号的常数参数。对于本次实验，假设  $R_0$  为  $10k\Omega$ ,  $B$  值为  $3950K$ 。

##### 4. 电路：

- 温度传感器模块由一个NTC热敏电阻和一个固定电阻组成分压电路。当环境温度发生变化时，热敏电阻的阻值也会随之改变，从而影响分压点处的电压输出。



- 通过连接到PCF8591的模拟输入端口AIN0，我们可以采集这个电压信号，并将其转换为数字形式以便后续分析。

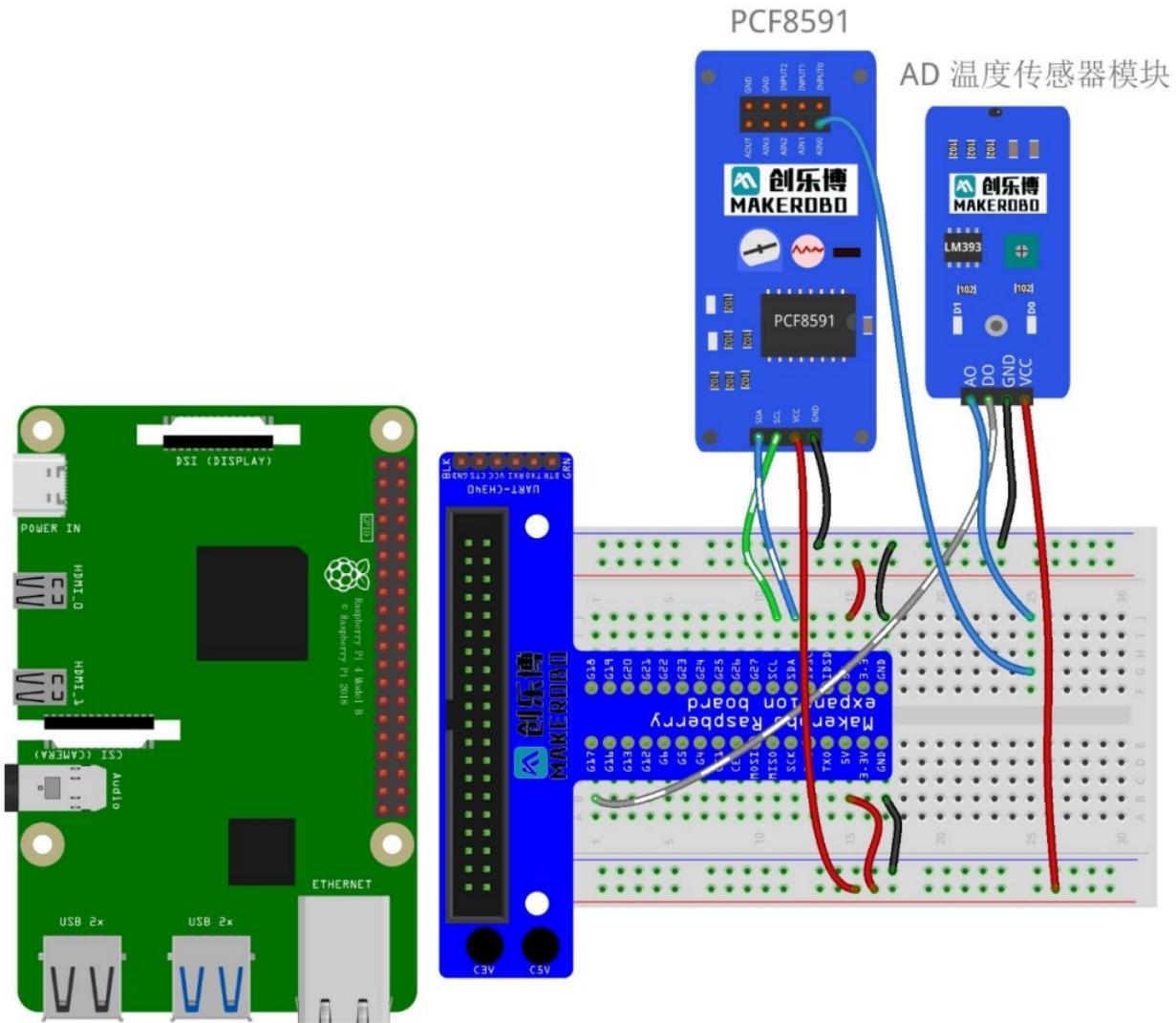
##### 7. 数据处理：

- 首先从PCF8591读取经过A/D转换后的数值，然后根据已知条件（如供电电压5V，ADC分辨率为8位即0-255对应0-5V）计算出对应的模拟电压。
- 接着利用分压比公式计算得到热敏电阻的实际阻值，再代入Steinhart-Hart方程求解温度T。

#### 三、实验步骤

##### 1. 硬件连接：

2. 连接Raspberry Pi、T型转接板和PCF8591模块之间的SDA、SCL、VCC和GND引脚。



3. 将模拟温度传感器的AO引脚连接到PCF8591模块的AIN0，DO引脚可以留空或接地，VCC引脚接5V电源，GND引脚接地。

4. 配置I2C总线：见[Lab4实验报告](#)中的第2步。

5. 编写代码

6. 导入必要的库。

## 7. 创建一个SMBus实例并与PCF8591建立连接，读取AIN0上的模拟值并根据该值计算温度。

```

1 import smbus
2 import math
3 import time
4
5 # Define the I2C address of the PCF8591 and control bits
6 address = 0x48 # Default address for PCF8591
7 control_bit = 0x40 # Command to start conversion on channel 0 (AIN0)
8
9 # Constants for the thermistor calculation
10 R0 = 10000 # Resistance at 25°C in ohms
11 B = 3950 # Thermistor constant in Kelvin
12 T0 = 298.15 # Standard temperature in Kelvin (25°C)
13 Vcc = 5.0 # Supply voltage in volts
14
15 # Initialize the SMBus library
16 bus = smbus.SMBus(1) # Use I2C bus 1
17
18 def read_temperature():
19     try:
20         # Write the control byte to initiate an A/D conversion on channel 0
21         bus.write_byte(address, control_bit)
22
23         # Read back the converted value from the PCF8591
24         analog_value = bus.read_byte(address)
25
26         # Calculate the analog voltage
27         Vr = (analog_value / 255.0) * Vcc
28
29         # Calculate the resistance of the thermistor
30         Rt = R0 * Vr / (Vcc - Vr)
31
32         # Apply the Steinhart-Hart equation to calculate temperature
33         temp_kelvin = 1 / (math.log(Rt / R0) / B + 1 / T0)
34         temp_celsius = temp_kelvin - 273.15
35
36         return round(temp_celsius, 2)
37
38     except Exception as e:
39         print("Error reading temperature:", str(e))
40         return None
41
42     try:
43         while True:
44             temperature = read_temperature()
45             if temperature is not None:
46                 print(f"Temperature: {temperature}°C")
47             else:
48                 print("Failed to read temperature.")
49
50             time.sleep(1) # Small delay between readings
51
52     except KeyboardInterrupt:
53         pass # Allow the program to exit cleanly with Ctrl+C

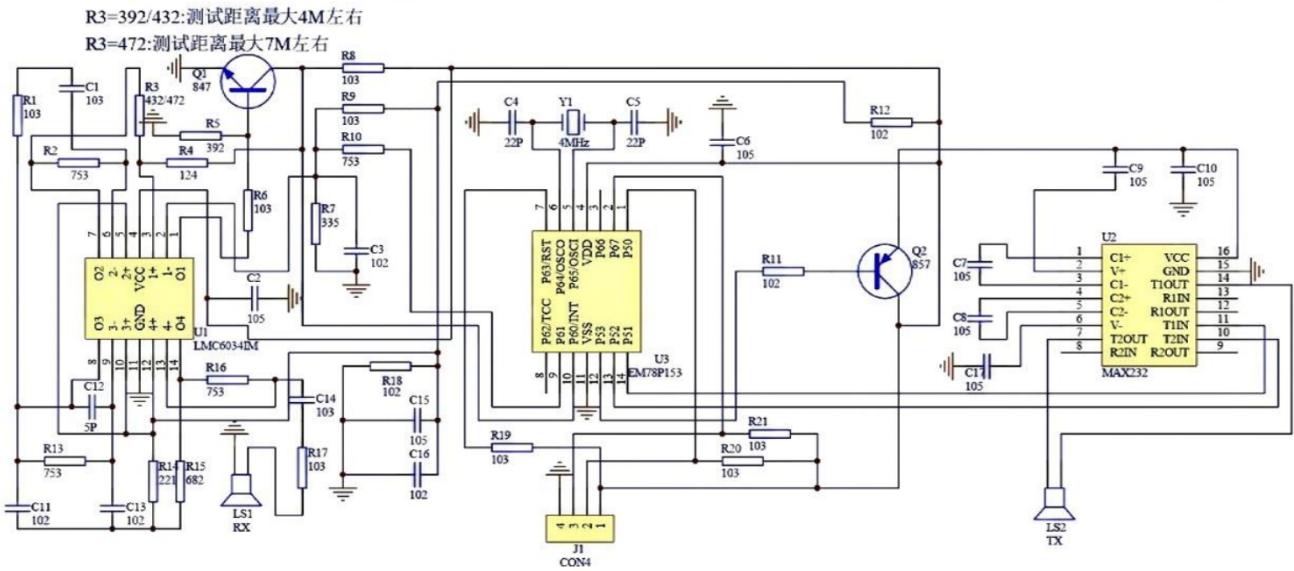
```

## 2.6 Lab6实验报告：超声波传感器测距实验

### 一、实验介绍

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。该传感器有 4 个引脚：- **VCC**：5V 电源供电；- **Trig**：触发引脚，用于启动超声波发射；- **Echo**：回波引脚，表示是否检测到返回的超声波；- **GND**：接地。

该模块的原理图如下：



### 二、实验原理

#### 1. 超声波传感器工作流程：

2. 超声波传感器包括一个发射器和一个接收器。当触发引脚 (Trig) 接收到至少 10 微秒的高电平脉冲时，它会发送 8 个周期的 40kHz 超声波脉冲。
3. 接收器监听反射回来的超声波，并将 Echo 引脚置为高电平直到接收到回波为止。此时，Echo 引脚保持高电平的时间长度与超声波往返一次所需的时间成正比。
4. 由于声音传播速度约为 343 米/秒（在 20 摄氏度空气中），因此可以通过测量 Echo 引脚高电平持续时间（秒）\* 34300 / 2
5. 测试距离（单位：厘米）= Echo 引脚高电平持续时间（秒）\* 34300 / 2

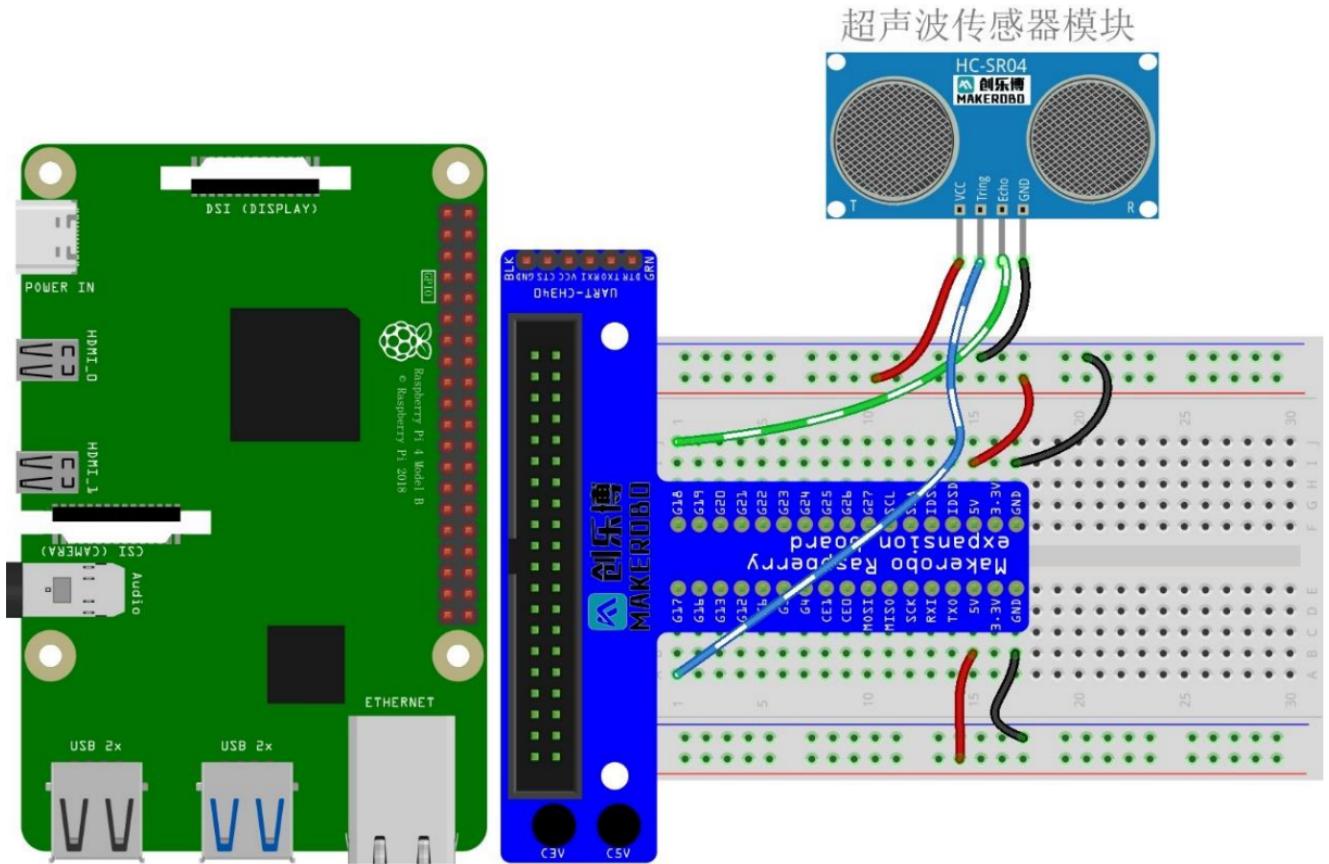
#### 6. 注意事项：

7. 注：Echo 引脚变为高电平时为 5V，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。

### 三、实验步骤

#### 1. 硬件连接：

2. 连接Raspberry Pi、T型转接板和超声波传感器之间的VCC、Trig、Echo和GND引脚。



3. 将超声波传感器的Trig引脚连接到Raspberry Pi的GPIO17（BCM编号），Echo引脚连接到GPIO18（BCM编号），同时确保VCC接到5V电源，GND接地。

4. 编写代码：

5. 导入 RPi.GPIO 库，设置Trig和Echo引脚的BCM编号。

6. 编写函数 `get_distance()`，该函数负责设置Trig引脚输出10微妙的高电平脉冲，然后等待Echo引脚变为高电平，记录开始时间；接着再次等待Echo引脚变为低电平，记录结束时间。最后利用这两个时间点计算出超声波往返一次所花费的时间，并据此换算成实际距离。

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pins for the ultrasonic sensor
5 TRIG = 17 # BCM numbering
6 ECHO = 18 # BCM numbering
7
8 # Setup GPIO mode and pin directions
9 GPIO.setmode(GPIO.BCM)
10 GPIO.setup(TRIG, GPIO.OUT)
11 GPIO.setup(ECHO, GPIO.IN)
12
13 def get_distance():
14     # Ensure TRIG is low initially
15     GPIO.output(TRIG, False)
16     time.sleep(0.2)
17
18     # Send a 10us pulse to TRIG
19     GPIO.output(TRIG, True)
20     time.sleep(0.00001)
21     GPIO.output(TRIG, False)
22
23     # Wait for ECHO to go high
24     while GPIO.input(ECHO) == 0:
25         pulse_start = time.time()
26
27     # Wait for ECHO to go low again
28     while GPIO.input(ECHO) == 1:
29         pulse_end = time.time()
30
31     # Calculate the duration of the pulse
32     pulse_duration = pulse_end - pulse_start
33
34     # Convert pulse duration to distance in centimeters
35     distance = pulse_duration * 17150 # Speed of sound in cm/s divided by 2 (round trip)
36     distance = round(distance, 2)
37
38     return distance
39
40 try:
41     print("Measuring distance...")
42     while True:
43         dist = get_distance()
44         print(f"Distance: {dist} cm")
45         time.sleep(1)
46
47 except KeyboardInterrupt:
48     print("Measurement stopped by user")
49
50 finally:
51     GPIO.cleanup() # Clean up GPIO settings before exiting

```

## 2.7 蜂鸣器实验

---

### Lab7实验报告：蜂鸣器实验

#### 一、实验介绍

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。本次实验任务为利用蜂鸣器播放一段音乐（音乐自选），并通过编程控制蜂鸣器发出相应的音符。

#### 二、实验原理

##### 1. 有源蜂鸣器：

2. 内部含有振荡电路，可以将恒定的直流电转化为一定频率的脉冲信号，因此只需给它施加合适的直流电压即可让它发出声音。
3. 在本实验中使用的有源蜂鸣器为低电平触发，即当GPIO引脚设置为低电平时，蜂鸣器会响起；反之则停止发声。

##### 4. 无源蜂鸣器：

5. 没有内置驱动电路，必须由外部提供特定频率的方波信号才能工作。可以通过改变方波的频率来调整发出的声音频率，进而实现不同的音符。
6. PFM (Pulse-Frequency Modulation) 是一种仅使用两个电平（高/低）表示模拟信号的调制方式，在这里用来生成可变频率的脉冲序列以驱动无源蜂鸣器。
7. PWM (Pulse-Width Modulation) 虽然不是本次实验的重点，但作为一种常见的调制技术，它同样适用于控制蜂鸣器或其他设备的输出特性。

#### 8. 编程思路：

9. 对于有源蜂鸣器，只需要简单地配置对应的GPIO引脚状态为高或低就可以控制其开关。
10. 对于无源蜂鸣器，则需要创建一个包含多个音符频率值的列表，并依次遍历这个列表，每次根据当前音符设定适当的PWM频率，使蜂鸣器按照指定旋律发声。

#### 三、实验步骤

##### (1) 有源蜂鸣器

##### 1. 硬件连接：

2. 根据提供的表格，确保正确连接Raspberry Pi、T型转接板和有源蜂鸣器模块之间的I/O、VCC和GND引脚。
3. 注意电源使用3.3V！

##### 4. 编写代码：

5. 使用Python语言编写程序，首先需要安装RPi.GPIO库以控制GPIO引脚。
6. 编写函数 `play_tone()`，该函数负责周期性地切换GPIO引脚的状态，使得蜂鸣器每隔一段时间响一次，模拟出连续的提示音效果。

## 7. 下面是一个简单的代码示例：

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pin for the buzzer (BCM numbering)
5 BUZZER_PIN = 17 # BCM 17, physical pin 11
6
7 # Setup GPIO mode and pin direction
8 GPIO.setmode(GPIO.BCM)
9 GPIO.setup(BUZZER_PIN, GPIO.OUT)
10
11 def play_tone(duration=0.5):
12     """Play a tone using the active buzzer."""
13     try:
14         # Turn on the buzzer (low level trigger)
15         GPIO.output(BUZZER_PIN, GPIO.LOW)
16         time.sleep(duration)
17
18         # Turn off the buzzer
19         GPIO.output(BUZZER_PIN, GPIO.HIGH)
20         time.sleep(0.1) # Short pause between tones
21
22     except KeyboardInterrupt:
23         print("Stopped by user")
24
25 finally:
26     GPIO.cleanup() # Clean up GPIO settings before exiting
27
28 if __name__ == "__main__":
29     print("Playing tone...")
30     while True:
31         play_tone()

```

### (2) 无源蜂鸣器

#### 1. 硬件连接：

2. 同样根据提供的表格，确保正确连接Raspberry Pi、T型转接板和无源蜂鸣器模块之间的I/O、VCC和GND引脚。
3. 确保选择支持PWM输出的GPIO引脚（如GPIO18，BCM编号）。
4. 编写代码：
5. 使用Python语言编写程序，首先需要安装RPi.GPIO库以及pigpio库（用于更精确地控制PWM）。
6. 编写函数 `play_music()`，该函数定义了一系列音符及其对应的频率，并通过循环调用这些频率来驱动蜂鸣器发出音乐。

## 7. 下面是一个简单的代码示例：

```

1 import RPi.GPIO as GPIO
2 import pigpio
3 import time
4
5 # Define GPIO pin for the passive buzzer (BCM numbering)
6 BUZZER_PIN = 18 # BCM 18, physical pin 12
7
8 # Initialize pigpio library
9 pi = pigpio.pi()
10
11 # Notes and their frequencies in Hz
12 NOTES = {
13     'C4': 262, 'D4': 294, 'E4': 330, 'F4': 349, 'G4': 392, 'A4': 440, 'B4': 494,
14     'C5': 523, 'D5': 587, 'E5': 659, 'F5': 698, 'G5': 784, 'A5': 880, 'B5': 988,
15 }
16
17 # A simple melody to play
18 MELODY = ['C4', 'D4', 'E4', 'C4', 'E4', 'D4', 'C4']
19
20 # Function to set frequency of the passive buzzer
21 def set_frequency(freq):
22     pi.hardware_PWM(BUZZER_PIN, freq, 500000) # Frequency, Duty cycle (50%)
23
24 def play_music(melody):
25     try:
26         for note in melody:
27             if note in NOTES:
28                 set_frequency(NOTES[note])
29                 time.sleep(0.5) # Duration of each note
30                 set_frequency(0) # Stop sound between notes
31                 time.sleep(0.1) # Short pause between notes
32
33     except KeyboardInterrupt:
34         print("Music stopped by user")
35
36 finally:
37     pi.stop() # Clean up pigpio resources
38     GPIO.cleanup() # Clean up GPIO settings before exiting
39
40 if __name__ == "__main__":
41     print("Playing music...")
42     play_music(MELODY)

```

### 1. 测试与验证：

2. 分别运行上述编写的Python脚本，观察有源蜂鸣器是否能持续发出提示音，以及无源蜂鸣器是否能够按照预定旋律播放音乐。
3. 如果可能的话，尝试调整代码中的参数（如音符持续时间和间隔），看看是否会对输出效果产生影响。

### 4. 清理工作：

5. 实验结束后，请记得关闭所有运行中的进程，并断开电源以保护设备安全。

## 2.8 PS2操纵杆实验

### Lab8实验报告：PS2操纵杆实验

#### 一、实验介绍

本实验旨在通过使用PS2模拟操纵杆，学习如何在Raspberry Pi上实现对不同LED灯的控制及其亮度变化。PS2操纵杆是一种常见的输入设备，它可以通过两个方向上的电位计来提供X和Y轴的位置信息，并且有一个数字输出用于检测是否按下按钮（Z轴）。本次实验的任务是编写程序读取操纵杆的状态，并根据其位置调整连接到PCF8591模数转换器的LED亮度。

#### 二、实验原理

##### 1. PS2操纵杆工作原理：

2. PS2操纵杆内部有两个垂直安装的电位计，分别对应X轴和Y轴。当用户移动操纵杆时，这两个电位计会产生从0V到5V之间的电压变化，静止状态下通常为2.5V左右。
3. 按下按钮时，SW引脚会输出低电平信号（0V），可用于触发特定事件或功能。

##### 4. 电路设计：

5. 在本实验中，我们将PS2操纵杆的X轴（VRX）和Y轴（VRY）连接到PCF8591的模拟输入端口AIN0和AIN1，而按钮（SW）则可以连接到另一个GPIO引脚或者留空。
6. PCF8591负责将来自操纵杆的模拟电压信号转换为数字值，这些数值可以在Raspberry Pi上进一步处理以确定操纵杆的具体位置。

##### 7. 数据处理与控制逻辑：

8. 通过读取PCF8591提供的数字化后的X轴和Y轴数据，我们可以得知当前操纵杆指向的位置。
9. 根据操纵杆的位置，我们可以改变连接到PCF8591模拟输出端口AOUT的LED亮度。例如，当操纵杆位于中心位置时，LED保持一定亮度；随着操纵杆向任意方向偏移，相应地增加或减少LED的亮度。

#### 三、实验步骤

##### 1. 硬件连接：

2. 根据提供的表格，确保正确连接Raspberry Pi、T型转接板、PCF8591模块以及PS2操纵杆之间的SDA、SCL、VCC、GND、VRX、VRY和SW引脚。
3. 将PS2操纵杆的VRX引脚连接到PCF8591模块的AIN0，VRY引脚连接到AIN1，SW引脚可以根据需要选择性连接到额外的GPIO引脚，VCC引脚接5V电源，GND引脚接地。

##### 4. 配置I2C总线：

5. 点击Raspberry Pi桌面环境中的开始菜单，选择Preferences -> Raspberry Pi Configuration。
6. 进入Interfaces标签页，开启I2C选项，点击OK保存更改并重启系统。

##### 7. 编写代码：

8. 使用Python语言编写程序，首先需要安装smbus库，它可以方便地操作I2C设备。
9. 导入必要的库后，创建一个SMBus实例并与PCF8591建立连接，读取AIN0和AIN1上的模拟值，并根据这些值计算出对应的LED亮度。

## 10. 下面是一个简单的代码示例：

```

1 import smbus
2 import time
3
4 # Define the I2C address of the PCF8591 and control bits
5 address = 0x48 # Default address for PCF8591
6 control_bit_x = 0x40 # Command to start conversion on channel 0 (AIN0, X-axis)
7 control_bit_y = 0x41 # Command to start conversion on channel 1 (AIN1, Y-axis)
8
9 # Initialize the SMBus library
10 bus = smbus.SMBus(1) # Use I2C bus 1
11
12 def read_joystick(axis='x'):
13     """Read joystick position from specified axis."""
14     if axis.lower() == 'x':
15         control_bit = control_bit_x
16     elif axis.lower() == 'y':
17         control_bit = control_bit_y
18     else:
19         raise ValueError("Invalid axis. Choose 'x' or 'y'.")
20
21     try:
22         # Write the control byte to initiate an A/D conversion on selected channel
23         bus.write_byte(address, control_bit)
24
25         # Read back the converted value from the PCF8591
26         analog_value = bus.read_byte(address)
27
28     return analog_value
29
30 except Exception as e:
31     print(f"Error reading {axis}-axis:", str(e))
32     return None
33
34 def map_to_brightness(value, in_min=0, in_max=255, out_min=0, out_max=100):
35     """Map joystick value to LED brightness percentage."""
36     return int((value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)
37
38 try:
39     while True:
40         x_value = read_joystick('x')
41         y_value = read_joystick('y')
42
43         if x_value is not None and y_value is not None:
44             print(f"X-axis: {x_value}, Y-axis: {y_value}")
45
46             # Calculate LED brightness based on joystick position
47             led_brightness_x = map_to_brightness(x_value)
48             led_brightness_y = map_to_brightness(y_value)
49
50             # Here you would add code to set the LED brightness using PWM or similar method.
51             # For demonstration purposes, we'll just print the calculated brightness.
52             print(f"LED Brightness X (%): {led_brightness_x}, Y (%): {led_brightness_y}")
53
54         time.sleep(0.1) # Small delay between readings
55
56     except KeyboardInterrupt:
57         pass # Allow the program to exit cleanly with Ctrl+C

```

### 1. 测试与验证：

2. 执行上述编写的Python脚本，观察LED亮度是否随操纵杆位置的变化而相应改变。

3. 检查输出结果是否符合预期，并根据实际情况微调代码逻辑。

### 4. 清理工作：

5. 实验结束后，请记得关闭所有运行中的进程，并断开电源以保护设备安全。

## 2.9 红外遥控实验

### Lab9实验报告：红外遥控实验

#### 一、实验介绍

本实验旨在通过使用红外接收头和LIRC库，在Raspberry Pi上实现对红外遥控器信号的接收与解码。红外通信是一种利用不可见光波段（通常为近红外）进行短距离无线数据传输的技术，广泛应用于电视、空调等家用电器的遥控操作中。本次实验的任务是设置Raspberry Pi以识别来自普通红外遥控器的按键命令，并能够根据接收到的不同脉冲模式执行相应的动作。

#### 二、实验原理

##### 1. 红外通信基础：

2. 红外发射端通过对一个红外LED灯发出经过调制后的载波信号来发送信息；接收端则采用专门设计的红外接收头，它不仅包含光电转换元件（如PIN二极管），还集成了前置放大器和解调电路，可以直接输出已经解调好的数字信号供微处理器进一步处理。

##### 3. 红外接收头工作流程：

4. 当红外接收头捕捉到由遥控器发出的红外信号时，内部的PIN二极管会将光信号转化为电流变化，经过放大和解调后产生代表按键编码的数字脉冲序列。  
5. 每个遥控器按键对应特定的脉冲模式，因此可以通过解析这些脉冲来确定用户按下了哪个键。

##### 6. LIRC库的作用：

7. LIRC (Linux Infrared Remote Control) 是一个开源项目，提供了多种接口用于管理和配置红外遥控设备。在本实验中，我们将使用LIRC库读取并解释从红外接收头获取的数据流，从而实现对各种遥控指令的支持。

#### 三、实验步骤

##### 1. 安装LIRC及相关配置：

2. 使用以下命令安装LIRC软件包及其依赖项：

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

3. 修改 /boot/config.txt 文件中的红外模块部分，确保启用了红外接收功能，并指定了正确的GPIO引脚编号（例如接收引脚为22，发射引脚为23）。添加或修改如下行：

```
1 dtoverlay= gpio-ir,gpio_pin=22
2 dtoverlay= gpio-ir-tx,gpio_pin=23
```

##### 4. 调整驱动设置：

5. 编辑位于 /etc/lirc/lirc\_options.conf 的LIRC配置文件，更改默认驱动程序和设备路径：

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

将内容更改为：

```
1 driver = default
2 device = /dev/lirc0
```

##### 6. 重启系统：

7. 执行完上述配置更改后，请重启Raspberry Pi以使新的设置生效：

```
1 sudo reboot
```

##### 8. 测试IR接收器：

9. 重启完成后，可以使用 `irw` 命令查看当前接收到的红外信号。打开终端窗口并输入：

```
1  irw
```

10. 此时按下遥控器上的任意按键，你应该能在屏幕上看到对应的十六进制代码输出。

11. 编写控制逻辑：

12. 根据实际需求开发Python或其他语言的应用程序，监听来自LIRC的服务端口，解析收到的红外命令，并据此触发预设的操作（比如播放音乐、切换频道等）。

13. 下面是一个简单的Python示例，展示了如何读取并打印出所有接收到的红外事件：

```
1  import subprocess
2
3  def listen_to_remote():
4      try:
5          process = subprocess.Popen(['irw'], stdout=subprocess.PIPE)
6
7      while True:
8          line = process.stdout.readline().decode('utf-8').strip()
9          if not line:
10              break
11
12      print("Received IR command:", line)
13
14  except KeyboardInterrupt:
15      print("\nListening stopped.")
16
17  if __name__ == "__main__":
18      print("Listening for IR commands...")
19      listen_to_remote()
```

1. 验证结果：

2. 运行编写的Python脚本，尝试用遥控器发送不同的按键信号，观察是否能正确接收到相应的编码。

3. 如果一切正常，接下来就可以根据具体应用场景扩展程序的功能了，比如关联某些按键到特定任务上。

4. 清理工作：

5. 实验结束后，请记得关闭所有运行中的进程，并断开电源以保护设备安全。

## 2.10 中断实验

---

### Lab10实验报告：中断实验

#### 一、实验介绍

本实验旨在通过使用外部中断机制，学习如何在Raspberry Pi上实现对不同外接设备（如按键开关）的及时响应。外部中断允许系统暂停当前任务以优先处理紧急事件，例如硬件设备输入或定时器到期等。本次实验的任务是设置树莓派能够监听特定GPIO引脚上的状态变化，并在检测到有效边沿时触发预定义的动作，如点亮LED灯。

#### 二、实验原理

##### 1. 树莓派中断函数：

2. 使用 `GPIO.add_event_detect()` 方法来监控指定GPIO引脚的状态改变。此方法接受四个参数：

- `channel`：需要监测的GPIO引脚编号。
- `edge`：指定要监测的边沿类型，可以是上升沿（`GPIO.RISING`）、下降沿（`GPIO.FALLING`）或者两者皆可（`GPIO.BOTH`）。
- `callback`：当检测到状态变化时调用的回调函数（可选）。
- `bouncetime`：用于消除机械按键抖动的时间间隔（毫秒单位），即两次有效状态变化之间所需的最长时间差（可选）。

##### 3. 阻塞式等待边缘触发：

4. 另一种方式是使用 `GPIO.wait_for_edge()` 函数，在检测到指定边沿之前阻止程序继续执行。这种方法占用较少CPU资源，但会使主程序处于等待状态直到条件满足。

##### 5. 按键去抖动：

6. 由于物理按键按下时可能会产生短暂的电压波动（即“抖动”），因此在实际应用中通常会加入软件延时或者硬件滤波来确保每个按键动作只被记录一次。

#### 三、实验步骤

##### 1. 建立电路：

2. 根据提供的表格，确保正确连接Raspberry Pi、T型转接板和轻触按键模块之间的SIG(S)、VCC和GND引脚。

3. 将轻触按键模块的SIG(S)引脚连接到Raspberry Pi的GPIO23（BCM编号），VCC引脚接5V电源，GND引脚接地。

4. 同样地，准备一个或多个LED用于指示按键状态的变化。例如，红色LED的阳极通过限流电阻连接到GPIO17，阴极接地；绿色LED则连接到GPIO27。

##### 5. 编写代码：

6. 使用Python语言编写程序，首先需要安装RPi.GPIO库以控制GPIO引脚。

7. 编写函数 `setup_gpio()` 初始化GPIO模式和方向，以及配置按键引脚为输入并启用内部上拉电阻。

8. 定义回调函数 `button_pressed_callback()`，该函数将在每次按键按下时被调用，并负责切换LED的颜色。

## 9. 下面是一个简单的代码示例：

```

1 import RPi.GPIO as GPIO
2 import time
3
4 # Define GPIO pins for the LED and button (BCM numbering)
5 RED_LED_PIN = 17 # BCM 17, physical pin 11
6 GREEN_LED_PIN = 27 # BCM 27, physical pin 13
7 BUTTON_PIN = 23 # BCM 23, physical pin 16
8
9 def setup_gpio():
10     """Setup GPIO mode and pin directions."""
11     GPIO.setmode(GPIO.BCM)
12
13     # Setup LEDs as output
14     GPIO.setup(RED_LED_PIN, GPIO.OUT)
15     GPIO.setup(GREEN_LED_PIN, GPIO.OUT)
16
17     # Setup button as input with pull-up resistor
18     GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
19
20 def button_pressed_callback(channel):
21     """Callback function called when the button is pressed."""
22     if channel == BUTTON_PIN:
23         print("Button pressed!")
24
25     # Toggle between red and green LED
26     if GPIO.input(RED_LED_PIN):
27         GPIO.output(RED_LED_PIN, GPIO.LOW)
28         GPIO.output(GREEN_LED_PIN, GPIO.HIGH)
29     else:
30         GPIO.output(RED_LED_PIN, GPIO.HIGH)
31         GPIO.output(GREEN_LED_PIN, GPIO.LOW)
32
33 try:
34     setup_gpio()
35
36     # Add event detection on the button pin with debouncing
37     GPIO.add_event_detect(BUTTON_PIN, GPIO.FALLING, callback=button_pressed_callback, bouncetime=200)
38
39     print("Waiting for button press...")
40     while True:
41         time.sleep(1) # Keep script running to allow callbacks to work
42
43 except KeyboardInterrupt:
44     print("\nProgram stopped by user")
45
46 finally:
47     GPIO.cleanup() # Clean up GPIO settings before exiting

```

### 1. 测试与验证：

2. 执行上述编写的Python脚本，尝试按下轻触按键，观察LED是否能够在红绿之间交替亮起。

3. 检查输出结果是否符合预期，并根据实际情况微调代码逻辑，比如调整按键去抖时间（`bouncetime`参数）。

### 4. 功能扩展：

5. 在基础版本的基础上，可以进一步开发更复杂的交互逻辑，例如实现多模式切换（按一下红灯亮，再按一下红灯闪烁，接着绿灯亮，再次按一下绿灯闪烁...如此循环）。

6. 注意保存最终版本的代码及视频记录，以便提交作业。

### 7. 清理工作：

8. 实验结束后，请记得关闭所有运行中的进程，并断开电源以保护设备安全。