EE351实验报告

None

Qijun Han

None

Table of contents

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Raspberry Pi OS。

1.1 实验报告列表

作者：韩骐骏

2. 实验报告

2.1 Lab1实验报告：Raspberry Pi系统安装与SSH、VNC服务

2.1.1 一、实验介绍

本实验旨在指导用户如何使用SD卡格式化工具和官方的Raspberry Pi Imager软件进行树莓派操作系统的安装，并学习如何通过SSH和VNC实现对树莓派的远程管理。此过程包括了从准备SD卡到设置初始配置，再到实现安全的远程连接的所有步骤。

2.1.2 二、实验原理

- 系统安装：通过使用SD Formmater工具来格式化SD卡，确保其处于最佳状态以接收新的操作系统镜像；使用Raspberry Pi Imager或 Win32DiskImager等工具将下载的操作系统镜像写入SD卡。
- SSH：Secure Shell是一种网络协议，它允许数据在两个网络实体之间加密传输，可以用于远程登录、执行命令等功能。
- VNC：Virtual Network Computing是一个图形桌面共享系统，它允许一台计算机远程控制另一台计算机。VNC Viewer是客户端软件，它能够连接到启用了VNC服务器的树莓派并显示其桌面环境。

2.1.3 三、实验步骤

- 使用SD Formmater格式化SD卡。
- 使用Raspberry Pi Imager选择并安装所需版本的操作系统。
- 在SD卡中编辑config.txt文件，根据显示器分辨率添加必要的启动参数。
- 硬件接线完成后插入SD卡，开启树莓派电源。
- 设置语言和地区信息，配置新密码（可选）。
- 开启SSH和VNC服务，获取树莓派IP地址。
- 使用Putty、MobaXterm或其他SSH客户端连接至树莓派。
- 使用VNC Viewer连接至树莓派桌面环境，或者通过安装xrdp服务使用Windows远程桌面连接。

2.2 Lab 2

Lab2实验报告：学习知识准备与双色LED实验

一、实验介绍

本实验旨在帮助学生了解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。此外，实验还包括了如何使用Mu和Geany IDE进行Python和C/C++编程的基础教程。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. 在本次实验中，我们主要使用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. **双色LED模块**：
12. 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。确保正确识别所使用的GPIO引脚编号（例如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. 如果遇到问题，检查硬件连接是否正确无误，以及代码逻辑是否有误。必要时可以使用调试工具逐步排查错误。
9. **清理工作**：
10. 实验结束后，记得关闭所有运行中的进程，并断开电源以保护设备安全。

四、PYTHON示例代码

下面提供了一个简单的Python代码示例，用于实现双色LED的红绿交替闪烁：

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the LED (BCM numbering)
RED_PIN = 19  # Red part of the dual-color LED
GREEN_PIN = 20  # Green part of the dual-color LED

# Setup GPIO mode and pin directions
GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_PIN, GPIO.OUT)
GPIO.setup(GREEN_PIN, GPIO.OUT)

try:
    while True:
        # Turn on red LED
        GPIO.output(RED_PIN, GPIO.HIGH)
        GPIO.output(GREEN_PIN, GPIO.LOW)
        print("Red LED is ON")
        time.sleep(1)  # Wait for 1 second

        # Turn on green LED
        GPIO.output(RED_PIN, GPIO.LOW)
        GPIO.output(GREEN_PIN, GPIO.HIGH)
        print("Green LED is ON")
        time.sleep(1)  # Wait for 1 second

except KeyboardInterrupt:
    print("Program stopped by user")

finally:
    # Clean up GPIO settings before exiting
    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。你可以根据需要调整时间间隔或其他参数。

2.3 Lab 3

Lab3实验报告：轻触开关按键实验

一、实验介绍

本实验旨在通过使用轻触开关作为输入设备，学习如何在Raspberry Pi上实现对硬件输入的检测。学生将了解如何配置GPIO引脚以读取开关状态，并根据该状态控制LED灯的颜色变化。此外，还将探讨如何通过编程实现按键功能的扩展，如多模式切换和按键抖动消除。

二、实验原理

1. **轻触开关模块**：
2. 轻触开关内部有一个按钮，当按下时S引脚输出低电平（0V），松开后恢复高电平（通常为3.3V或5V）。这种特性使得它非常适合用作数字输入信号源。
3. **树莓派GPIO接口**：
4. Raspberry Pi的GPIO引脚可以被设置为输入或输出模式。对于本实验，我们将相关GPIO引脚设置为输入模式来接收来自轻触开关的状态信号。
5. **LED指示**：
6. 使用两种不同颜色的LED来指示按键的不同状态。例如，当按键按下时一个颜色亮起，而按键释放时另一个颜色亮起。
7. **按键抖动处理**：
8. 按键按下瞬间可能会产生电气噪声，导致多次触发。为了确保每次按键操作只触发一次事件，需要在软件层面加入去抖算法，比如设置一个短暂的时间延迟或者利用硬件RC滤波电路。

三、实验步骤

1. **硬件连接**：
2. 根据提供的表格，将轻触开关的SIG(S)引脚连接到Raspberry Pi的GPIO17，VCC连接到5V电源，GND接地。同时，准备两个LED分别用于显示不同的按键状态。
3. 如果是双色LED，则可直接按照建议的方式插入T型板相应的GPIO19, GPIO20, GND三个引脚中；如果是单个LED，则需额外连线至GPIO27和GND。
4. **编写代码**：
5. 编写Python程序，首先导入RPi.GPIO库并初始化GPIO引脚。
6. 设置GPIO17为输入模式，并启用内部上拉电阻以确保未按下时保持高电平。
7. 定义回调函数用于响应按键按下事件，该函数会在检测到电平下降沿时调用。
8. 在主循环中持续监听按键状态的变化，并相应地改变LED的状态。
9. **测试与验证**：
10. 运行编写的Python脚本，尝试按下和释放轻触开关，观察LED是否正确响应。
11. 确认LED能够在按键按下时切换颜色，并且没有因为按键抖动而导致异常闪烁。
12. **功能扩展**：
13. 实现更复杂的逻辑，比如按一下红灯亮起，再按一下红灯闪烁，接着绿灯亮，最后绿灯闪烁，以此类推形成循环。
14. 注意保存最终版本的代码及视频记录，以便提交作业。
15. **清理工作**：

16. 实验结束后，请记得清理所有使用的GPIO引脚，并断开电源以保证安全。

2.4 Lab 4

Lab5实验报告：PCF8591模数转换器实验

一、实验介绍

本实验旨在通过使用PCF8591模数转换器（Analog-to-Digital Converter, ADC），学习如何将模拟信号转换为数字信号，并利用Raspberry Pi对这些数据进行处理。PCF8591是一款具有四个模拟输入通道的8位ADC，支持I2C通信协议，可以轻松地与树莓派相连，用于采集如温度、光强等模拟量的数据。本次实验的任务是通过控制PCF8591来实现LED灯的亮度调节。

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I2C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I2C总线上。
4. **I2C总线通信**：
5. I2C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。
6. **模拟信号采集与处理**：
7. 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。
8. 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. 根据提供的表格，确保正确连接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；如果使用单独的绿色LED，则将其阴极连接到GND，阳极通过限流电阻连接到任意GPIO引脚（如GPIO17）。
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上的模拟值并根据该值调整AOUT输出，进而控制LED亮度。
10. 下面是一个简单的代码示例：

```
import smbus
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit = 0x40  # Command to start conversion on channel 0 (AIN0)

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1
```

```
try:
    while True:
        # Write the control byte to initiate an A/D conversion on channel 0
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
        analog_value = bus.read_byte(address)

        # Print out the raw analog value
        print("Analog Value:", analog_value)

        # Map the analog value to a range suitable for controlling LED brightness
        led_brightness = int((analog_value / 255.0) * 100)

        # Here you would add code to set the LED brightness using PWM or similar method.
        # For demonstration purposes, we'll just print the calculated brightness.
        print("LED Brightness (%):", led_brightness)

        time.sleep(0.1)  # Small delay between readings

except KeyboardInterrupt:
    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亮度是否随电位计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。

2.5 Lab 5

Lab5实验报告：模拟温度传感器实验

一、实验介绍

本实验旨在使用NTC（负温度系数）热敏电阻构建的温度感测模块，通过Raspberry Pi获取当前环境的温度值。该温度传感器能够将温度变化转化为电阻变化，并借助模数转换器PCF8591将这些模拟信号转换为数字信号供Raspberry Pi处理。最终目标是学习如何读取和解析来自温度传感器的数据，以实现对室内/环境温度的有效监测。

二、实验原理

1. **NTC热敏电阻特性**：
2. NTC热敏电阻是一种随温度升高而电阻减小的元件，其阻值与温度之间存在特定关系，这使得它非常适合用来测量温度。
3. 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。
4. **电路设计**：
5. 温度传感器模块由一个NTC热敏电阻和一个固定电阻（例如10kΩ）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。
6. 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**：
8. 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。
9. 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。
10. **Steinhart-Hart方程**：
11. 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10kΩ，B值为3950K。

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**：
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上的模拟值并根据该值计算温度。

10. 下面是一个简单的代码示例：

```
import smbus
import math
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit = 0x40  # Command to start conversion on channel 0 (AIN0)

# Constants for the thermistor calculation
R0 = 10000  # Resistance at 25°C in ohms
B = 3950  # Thermistor constant in Kelvin
T0 = 298.15  # Standard temperature in Kelvin (25°C)
Vcc = 5.0  # Supply voltage in volts

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1

def read_temperature():
    try:
        # Write the control byte to initiate an A/D conversion on channel 0
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
        analog_value = bus.read_byte(address)

        # Calculate the analog voltage
        Vr = (analog_value / 255.0) * Vcc

        # Calculate the resistance of the thermistor
        Rt = R0 * Vr / (Vcc - Vr)

        # Apply the Steinhart-Hart equation to calculate temperature
        temp_kelvin = 1 / (math.log(Rt / R0) / B + 1 / T0)
        temp_celsius = temp_kelvin - 273.15

        return round(temp_celsius, 2)

    except Exception as e:
        print("Error reading temperature:", str(e))
        return None

try:
    while True:
        temperature = read_temperature()
        if temperature is not None:
            print(f"Temperature: {temperature}°C")
        else:
            print("Failed to read temperature.")

        time.sleep(1)  # Small delay between readings

except KeyboardInterrupt:
    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，观察输出结果是否合理，确认温度读数是否稳定且符合实际环境温度。

3. 如果可能的话，尝试改变周围环境温度（比如靠近热源或冷源），检查传感器是否能准确反映温度变化。

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。

2.6 Lab 6

2.6.1 实验介绍

2.6.2 实验原理

2.6.3 实验步骤

Lab6实验报告：超声波传感器测距实验

一、实验介绍

本实验旨在通过使用HC-SR04超声波传感器，学习如何在Raspberry Pi上实现非接触式距离测量。HC-SR04模块能够发射和接收超声波信号，并根据回波时间计算目标物体的距离。此实验不仅有助于理解超声波测距的工作原理，还为开发自动化控制系统提供了实际应用案例。最终目标是编写一段Python代码来读取并显示由超声波传感器测得的距离值。

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引脚高电平持续的时间来确定目标距离。
5. **关键参数说明**：
6. **VCC**：5V电源供电；
7. **Trig**：触发引脚，用于启动超声波发射；
8. **Echo**：回波引脚，表示是否检测到返回的超声波；
9. **GND**：接地。
10. **注意事项**：
11. 因为树莓派GPIO引脚的最大输入电压为3.3V，而Echo引脚输出的是5V逻辑电平，所以在某些情况下建议使用分压电路来保护树莓派。不过，在这个实验中，考虑到Echo引脚高电平时间非常短，可以不使用分压电路。

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. 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`get_distance()`，该函数负责设置Trig引脚输出10微秒的高电平脉冲，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。

7. 下面是一个简单的代码示例：

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the ultrasonic sensor
TRIG = 17   # BCM numbering
ECHO = 18   # BCM numbering

# Setup GPIO mode and pin directions
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def get_distance():
     # Ensure TRIG is low initially
     GPIO.output(TRIG, False)
     time.sleep(0.2)

     # Send a 10us pulse to TRIG
     GPIO.output(TRIG, True)
     time.sleep(0.00001)
     GPIO.output(TRIG, False)

     # Wait for ECHO to go high
     while GPIO.input(ECHO) == 0:
          pulse_start = time.time()

     # Wait for ECHO to go low again
     while GPIO.input(ECHO) == 1:
          pulse_end = time.time()

     # Calculate the duration of the pulse
     pulse_duration = pulse_end - pulse_start

     # Convert pulse duration to distance in centimeters
     distance = pulse_duration * 17150  # Speed of sound in cm/s divided by 2 (round trip)
     distance = round(distance, 2)

     return distance

try:
     print("Measuring distance...")
     while True:
          dist = get_distance()
          print(f"Distance: {dist} cm")
          time.sleep(1)

except KeyboardInterrupt:
     print("Measurement stopped by user")

finally:
     GPIO.cleanup()  # Clean up GPIO settings before exiting
```

1. **测试与验证**：

2. 运行上述编写的Python脚本，观察输出结果是否合理，确认测量的距离值是否稳定且符合实际情况。

3. 可以尝试改变超声波传感器前方障碍物的位置，检查传感器是否能准确反映距离变化。

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。

2.7 Lab 7

Lab7实验报告：蜂鸣器实验

一、实验介绍

本实验旨在通过使用蜂鸣器，学习如何在Raspberry Pi上产生声音信号。蜂鸣器分为有源和无源两种类型，前者内置振荡电路可以直接发出固定频率的声音，而后者则需要外部提供调制后的脉冲信号才能发声。本次实验的任务是利用这两种类型的蜂鸣器分别播放提示音和一段音乐（例如简单的旋律），从而掌握不同类型蜂鸣器的工作原理及其编程控制方法。

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：

```
import RPi.GPIO as GPIO
import time

# Define GPIO pin for the buzzer (BCM numbering)
BUZZER_PIN = 17  # BCM 17, physical pin 11

# Setup GPIO mode and pin direction
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER_PIN, GPIO.OUT)
```

```
def play_tone(duration=0.5):
    """Play a tone using the active buzzer."""
    try:
        # Turn on the buzzer (low level trigger)
        GPIO.output(BUZZER_PIN, GPIO.LOW)
        time.sleep(duration)

        # Turn off the buzzer
        GPIO.output(BUZZER_PIN, GPIO.HIGH)
        time.sleep(0.1)  # Short pause between tones

    except KeyboardInterrupt:
        print("Stopped by user")

finally:
    GPIO.cleanup()  # Clean up GPIO settings before exiting

if __name__ == "__main__":
    print("Playing tone...")
    while True:
        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：

```
import RPi.GPIO as GPIO
import pigpio
import time

# Define GPIO pin for the passive buzzer (BCM numbering)
BUZZER_PIN = 18  # BCM 18, physical pin 12

# Initialize pigpio library
pi = pigpio.pi()

# Notes and their frequencies in Hz
NOTES = {
    'C4': 262, 'D4': 294, 'E4': 330, 'F4': 349, 'G4': 392, 'A4': 440, 'B4': 494,
    'C5': 523, 'D5': 587, 'E5': 659, 'F5': 698, 'G5': 784, 'A5': 880, 'B5': 988,
}

# A simple melody to play
MELODY = ['C4', 'D4', 'E4', 'C4', 'E4', 'D4', 'C4']

# Function to set frequency of the passive buzzer
def set_frequency(freq):
    pi.hardware_PWM(BUZZER_PIN, freq, 500000)  # Frequency, Duty cycle (50%)

def play_music(melody):
    try:
        for note in melody:
            if note in NOTES:
                set_frequency(NOTES[note])
                time.sleep(0.5)  # Duration of each note
                set_frequency(0)  # Stop sound between notes
                time.sleep(0.1)  # Short pause between notes

    except KeyboardInterrupt:
        print("Music stopped by user")

finally:
    pi.stop()  # Clean up pigpio resources
    GPIO.cleanup()  # Clean up GPIO settings before exiting

if __name__ == "__main__":
```

```
    print("Playing music...")
    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 Lab 8

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：

```
import smbus
import time

# Define the I2C address of the PCF8591 and control bits
address = 0x48  # Default address for PCF8591
control_bit_x = 0x40  # Command to start conversion on channel 0 (AIN0, X-axis)
```

```
control_bit_y = 0x41  # Command to start conversion on channel 1 (AIN1, Y-axis)

# Initialize the SMBus library
bus = smbus.SMBus(1)  # Use I2C bus 1

def read_joystick(axis='x'):
    """Read joystick position from specified axis."""
    if axis.lower() == 'x':
        control_bit = control_bit_x
    elif axis.lower() == 'y':
        control_bit = control_bit_y
    else:
        raise ValueError("Invalid axis. Choose 'x' or 'y'.")

    try:
        # Write the control byte to initiate an A/D conversion on selected channel
        bus.write_byte(address, control_bit)

        # Read back the converted value from the PCF8591
        analog_value = bus.read_byte(address)

        return analog_value

    except Exception as e:
        print(f"Error reading {axis}-axis:", str(e))
        return None

def map_to_brightness(value, in_min=0, in_max=255, out_min=0, out_max=100):
    """Map joystick value to LED brightness percentage."""
    return int((value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)

try:
    while True:
        x_value = read_joystick('x')
        y_value = read_joystick('y')

        if x_value is not None and y_value is not None:
            print(f"X-axis: {x_value}, Y-axis: {y_value}")

            # Calculate LED brightness based on joystick position
            led_brightness_x = map_to_brightness(x_value)
            led_brightness_y = map_to_brightness(y_value)

            # Here you would add code to set the LED brightness using PWM or similar method.
            # For demonstration purposes, we'll just print the calculated brightness.
            print(f"LED Brightness X (%): {led_brightness_x}, Y (%): {led_brightness_y}")

        time.sleep(0.1)  # Small delay between readings

except KeyboardInterrupt:
    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 Lab 9

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： `bash`
   ```
   sudo apt-get update
   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： `text`
   ```
   dtoverlay=gpio-ir,gpio_pin=22
   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： `bash`
   ```
   sudo nano /etc/lirc/lirc_options.conf
   ```
   将内容更改为： `text`
   ```
   driver = default
   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： `bash`
   ```
   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： `bash`
   ```
   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：

```
import subprocess

def listen_to_remote():
    try:
        process = subprocess.Popen(['irw'], stdout=subprocess.PIPE)

        while True:
            line = process.stdout.readline().decode('utf-8').strip()
            if not line:
                break

            print("Received IR command:", line)

    except KeyboardInterrupt:
        print("\nListening stopped.")

if __name__ == "__main__":
    print("Listening for IR commands...")
    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 Lab 10

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小时间差（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：

```
import RPi.GPIO as GPIO
import time

# Define GPIO pins for the LED and button (BCM numbering)
RED_LED_PIN = 17   # BCM 17, physical pin 11
GREEN_LED_PIN = 27   # BCM 27, physical pin 13
BUTTON_PIN = 23   # BCM 23, physical pin 16

def setup_gpio():
    """Setup GPIO mode and pin directions."""
    GPIO.setmode(GPIO.BCM)
```

```
    # Setup LEDs as output
    GPIO.setup(RED_LED_PIN, GPIO.OUT)
    GPIO.setup(GREEN_LED_PIN, GPIO.OUT)

    # Setup button as input with pull-up resistor
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def button_pressed_callback(channel):
    """Callback function called when the button is pressed."""
    if channel == BUTTON_PIN:
        print("Button pressed!")

        # Toggle between red and green LED
        if GPIO.input(RED_LED_PIN):
            GPIO.output(RED_LED_PIN, GPIO.LOW)
            GPIO.output(GREEN_LED_PIN, GPIO.HIGH)
        else:
            GPIO.output(RED_LED_PIN, GPIO.HIGH)
            GPIO.output(GREEN_LED_PIN, GPIO.LOW)

try:
    setup_gpio()

    # Add event detection on the button pin with debouncing
    GPIO.add_event_detect(BUTTON_PIN, GPIO.FALLING, callback=button_pressed_callback,
bouncetime=200)

    print("Waiting for button press...")
    while True:
        time.sleep(1)  # Keep script running to allow callbacks to work

except KeyboardInterrupt:
    print("\nProgram stopped by user")

finally:
    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。