微机原理与微系统实验报告

应逸雯 12210159

2024 年 12 月 5 日

摘要

本报告详细记录了微机原理与微系统实验的过程与结果。通过使用树莓派作为核心控制单元，我们探索了 GPIO 口的基本操作、LED 控制、按键输入、传感器数据采集以及远程控制等多个实验项目。实验涉及了对 PCF8591 模数转换器的应用，实现了温度、超声波传感器的数据读取，以及蜂鸣器和舵机的控制。此外，还包含了 PS2 操纵杆和红外遥控的应用，尝试了树莓派在嵌入式系统和物联网应用中的潜力。通过这些实验，我们验证了各个传感器和外设的功能，并对它们的性能进行了评估，为后续的项目开发和研究提供了宝贵的实践经验。

目录

1 基本配置 **3**

2 LED **4**

2.1 GPIO 口的使用 4

2.2 代码编辑器 5

2.3 LED 控制 6

3 按键开关 **7**

3.1 按键开关控制 7

3.2 按键输入控制 LED 输出 8

3.3 倾斜传感器、振动传感器 10

4 模数转换器 **11**

4.1 PCF8591 11

4.2 I2C 通信 12

4.3 DA 输出传感器数据 13

4.4 AD 输入控制功能 14

4.5 AD-DA 组合功能 15

5 温度传感器 **16**

5.1 传感器原理 16

5.2 读取实时温度 17

6 超声波传感器 **19**
6.1 传感器原理 19
6.2 测量结果 20

7 蜂鸣器 **22**
7.1 蜂鸣器控制 22
7.2 舵机控制 24

8 操纵杆 **24**

9 红外遥控 **26**
9.1 遥控配置 27
9.2 使用遥控进行控制 28

10 中断 **31**

11 结论 **34**

1 基本配置

树莓派（Raspberry Pi）是一款基于 ARM 架构的微型电脑主板，它具备完整的 PC 基本功能，它被设计为低成本、高性能，适合教育和业余爱好者使用。树莓派可以执行各种计算机功能，包括文字处理、游戏、多媒体应用等，并且可以连接到电视和显示器。

树莓派成本低，相对于其他机载 NUC 等主板，更适合进行小项目的开发。尺寸小，易于嵌入到各个设备中。灵活性强，可以运行多种操作系统，如官方的 Raspberry Pi OS，也可以运行其他操作系统如 Ubuntu 等。树莓派也具有通用的输入输出 GPIO 脚、串口、SPI 接口、I2C 接口，易于连接各种传感器和外设。树莓派是一个集成了嵌入式开发板功能和较好计算能力的微型电脑主板。

树莓派 4 使用的是 Broadcom BCM2711 芯片，这是一个四核 Cortex-A72（ARM v8）64 位 SoC，主频可达 1.5GHz。树莓派 4 支持 1GB、2GB、4GB 或 8GB LPDDR4 内存，以及 0GB、8GB、16GB 或 32GB eMMC 闪存。树莓派使用 SD/MicroSD 卡作为存储介质，支持 Micro SD 卡插槽用于加载操作系统和数据存储。树莓派 4 支持 2.4GHz 和 5.0GHz IEEE 802.11b/g/n/ac 无线局域网和蓝牙 5.0 通信。树莓派 4 提供 2 个 micro HDMI 端口（支持 4Kp60）、2-lane MIPI DSI 显示端口、2-lane MIPI CSI 相机端口、4 极立体声音频和复合视频端口。树莓派 4 还提供标准的 40-pin GPIO 口。树莓派 4 支持 H.265（4Kp60 解码）、H.264（1080p60 解码，1080p30 编码）和 OpenGL ES 3.0 图形。树莓派 4 通过 USB-C 连接器或 GPIO 头提供 5V DC 输入，支持通过以太网供电（PoE）。树莓派采用 Raspberry Pi Pico 系列，基于 RP2040 SoC 的微控制器，具有 264KB 的 SRAM 和 2MB 的存储，提供 GPIO 引脚。

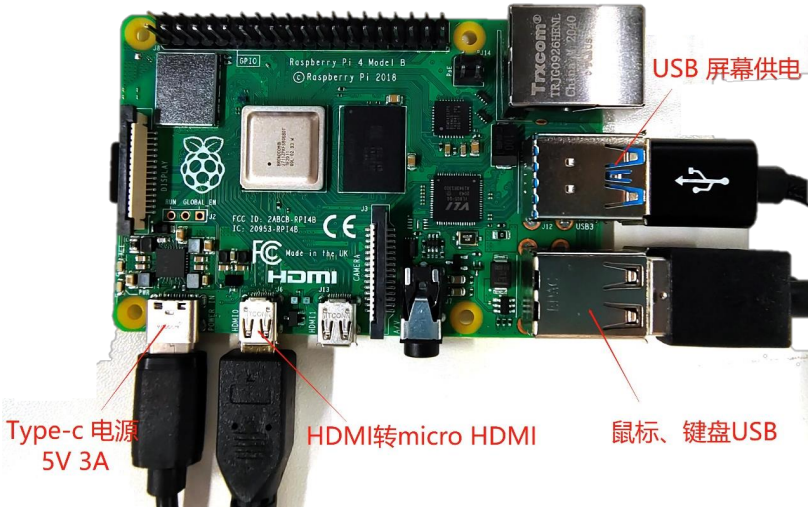


图 1: 树莓派硬件设备

安装 Raspberry Pi 操作系统：需要一台电脑用于下载操作系统镜像和烧录工具，如 Raspberry Pi Imager。将 microSD 卡通过读卡器连接到电脑上，格式化存储卡，而后使用 Raspberry Pi Imager 选择相应的树莓派型号和操作系统镜像进行烧录。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。烧录完成后，将 microSD 卡插入树莓派，连接显示器、键盘、鼠标，并通电启动。

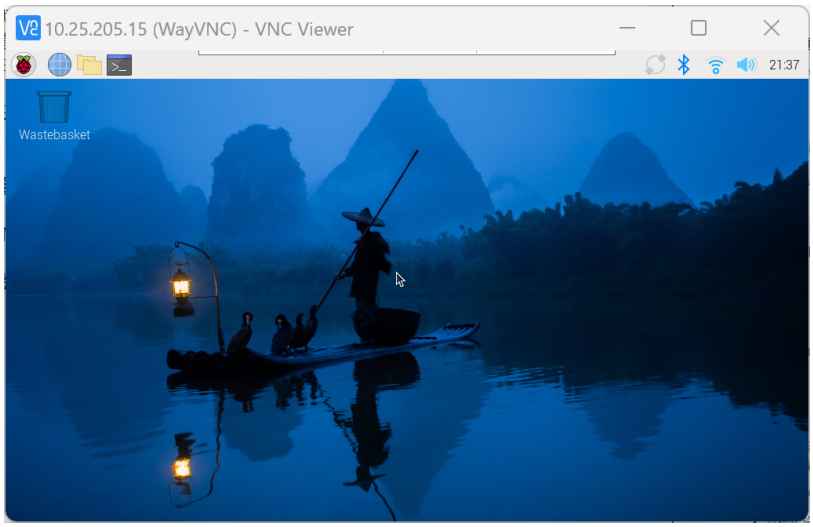


图 2: 树莓派系统界面

SSH（Secure Shell）是一种网络协议，用于在不安全的网络（如互联网）上安全地访问远程计算机。它通过加密技术保护数据传输，确保用户身份验证和数据传输的安全性，防止窃听和中间人攻击。SSH 客户端和服务器之间的通信被加密，使得数据传输即使在公共网络上也是安全的。用户可以通过 SSH 连接到远程服务器，执行命令、管理文件和运行程序，就像直接在服务器的控制台前操作一样。

VNC（Virtual Network Computing）是一种图形桌面共享系统，允许用户远程访问和控制另一台计算机的图形桌面环境。通过 VNC 客户端软件，用户可以查看和交互远程计算机的桌面，执行应用程序、打开文件和使用系统功能，就像坐在该计算机前一样。VNC 通过将远程桌面的屏幕图像编码并传输到客户端，用户的操作也被编码并发送回服务器，从而实现实时的远程控制。这使得 VNC 非常适合需要图形界面操作的远程工作和协作场景。

在树莓派上配置 SSH 和 VNC 服务，首先需要通过图形界面或命令行启用这些服务。可以通过 Raspberry Pi OS 的“Raspberry Pi Configuration”工具，在“Interfacing Options”中找到并启用 SSH 和 VNC 服务。启用后，可以通过 VNC 客户端软件如 RealVNC Viewer，主机与树莓派连接在同一网络下，输入树莓派的 IP 地址和连接密码，实现远程桌面访问。这样，用户就可以在任何地点通过 SSH 和 VNC 远程管理和操作树莓派，无需直接连接显示器和键盘。

2 LED

2.1 GPIO 口的使用

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）。

wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

NXEZ.COM

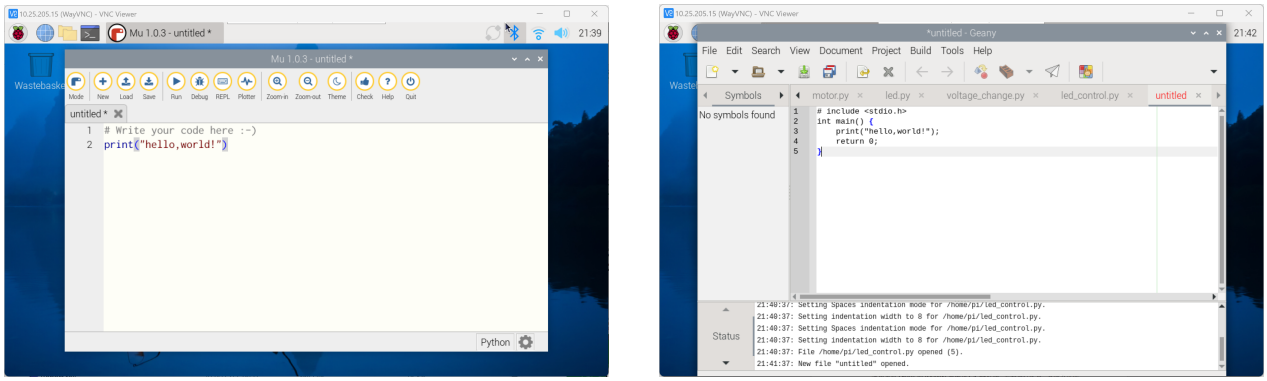
图 3: 树莓派 GPIO 口分布

使用代码控制 GPIO 口，需要配置环境：

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

2.2 代码编辑器

可以在树莓派上使用 mu 编写并运行 python 程序，使用 Geany 编写并运行 C 程序。



(a) MU

(b) Geany

图 4: 编辑器界面

也可以使用超级终端管理器 Mobaxterm 到主机上使用主机编辑器编写程序，并在文字终端中运行程序。

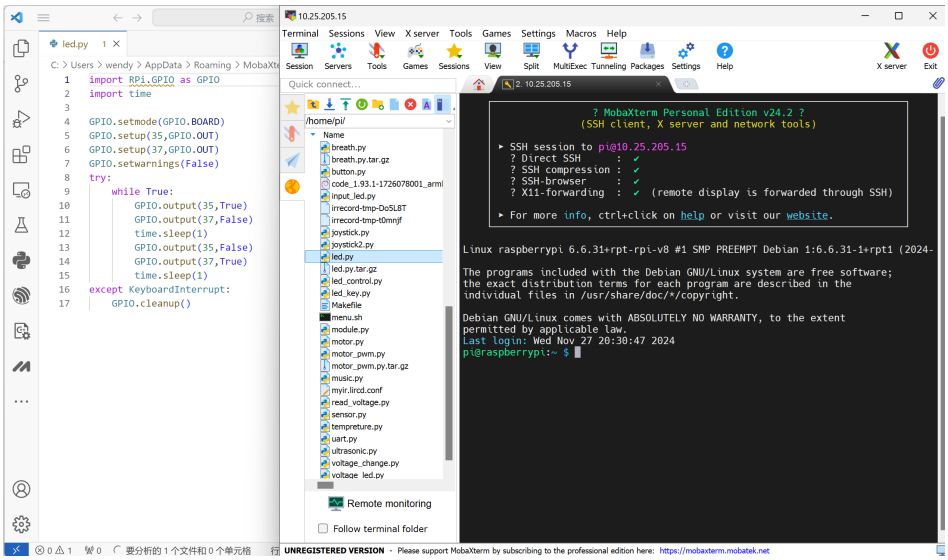


图 5: 主机 VSCode 编辑

2.3 LED 控制

LED 原理图如下：

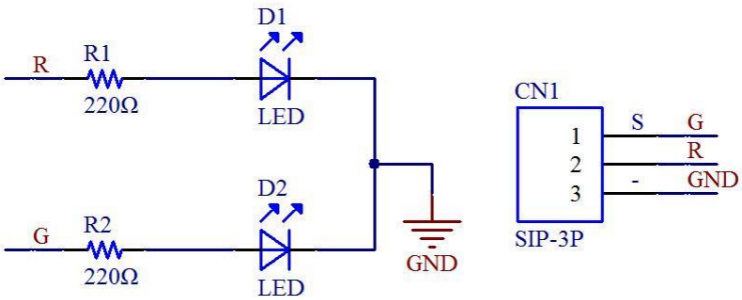


图 6: LED 原理图

在 G 连接的引脚写入 1，亮绿灯；在 R 连接的引脚写入 1，亮红灯；模块与树莓派共地，保持电压稳定。

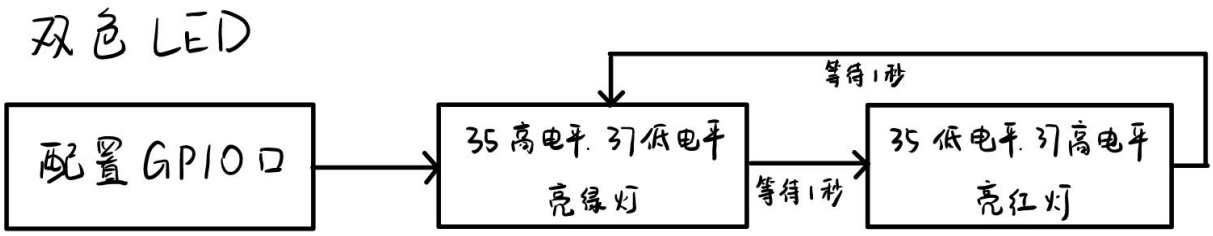
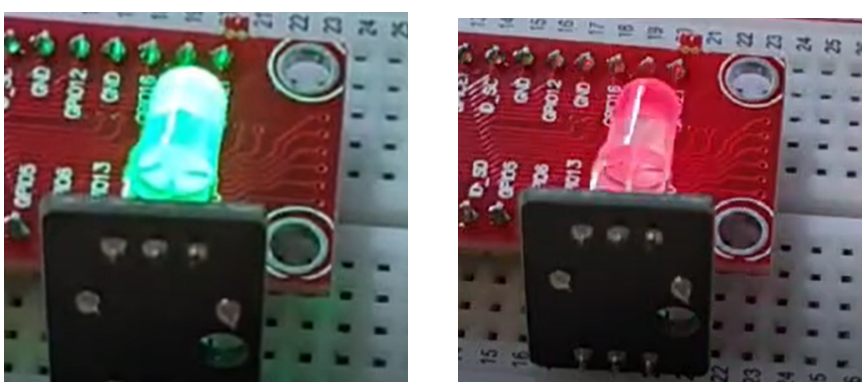


图 7: LED 控制程序流程图

```python
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(35,GPIO.OUT)
GPIO.setup(37,GPIO.OUT)
GPIO.setwarnings(False)

try:
  while True:
    GPIO.output(35,True)
    GPIO.output(37,False)
    time.sleep(1)
    GPIO.output(35,False)
    GPIO.output(37,True)
    time.sleep(1)
except KeyboardInterrupt:
  GPIO.cleanup()
```

运行效果如图：



(a) 亮绿灯　　(b) 亮红灯

图 8: LED 灯自动切换亮灯程序运行效果

3 按键开关

3.1 按键开关控制

按键开关原理图如下：

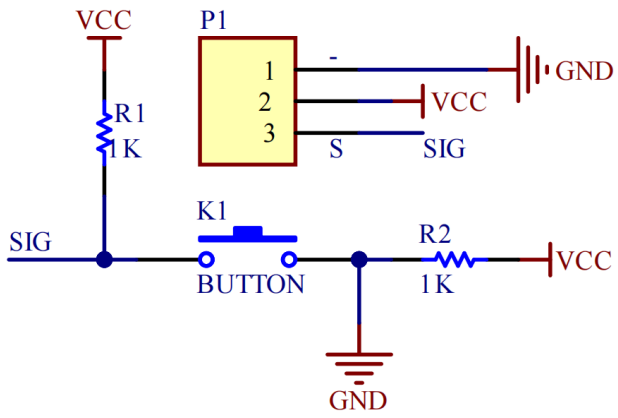


图 9: 按键开关原理图

将 GPIO 口配置成输入模式，可以读取按键输入，增加软件按键消抖，可以得到更适合配置在项目中的按键。

```
def check_input():
  global count
  # 按键检测&消抖
  if GPIO.input(12) == 0:
    time.sleep(0.1)
    if GPIO.input(12) == 0:
      while GPIO.input(12) == 0:
        time.sleep(0.001)
      count += 1
    print(count)
    if count == 5:
      count = 1
```

3.2 按键输入控制 LED 输出

根据按键状态，可以使得输出切换到不同模式，在本次实验中，体现在 LED 的不同闪烁模式中：红灯亮，红灯闪烁，绿灯亮，绿灯闪烁。

```
def react():
  global count
  while True:
    # 状态响应
    if count == 1:
      GPIO.output(35, False)
      GPIO.output(37, True)
    elif count == 2:
      GPIO.output(35, False)
```

```
        GPIO.output(37, True)
        time.sleep(0.5)
        GPIO.output(37, False)
        time.sleep(0.5)
      elif count == 3:
        GPIO.output(37, False)
        GPIO.output(35, True)
      elif count == 4:
        GPIO.output(37, False)
        GPIO.output(35, True)
        time.sleep(0.5)
        GPIO.output(35, False)
        time.sleep(0.5)
```

由于 LED 闪灯程序中存在等待，可能使得系统无法持续接受按键输入响应，故配置多线程。

```
try:
  threading.Thread(target=react).start()
  while True:
    check_input()
except KeyboardInterrupt:
  GPIO.cleanup()
```

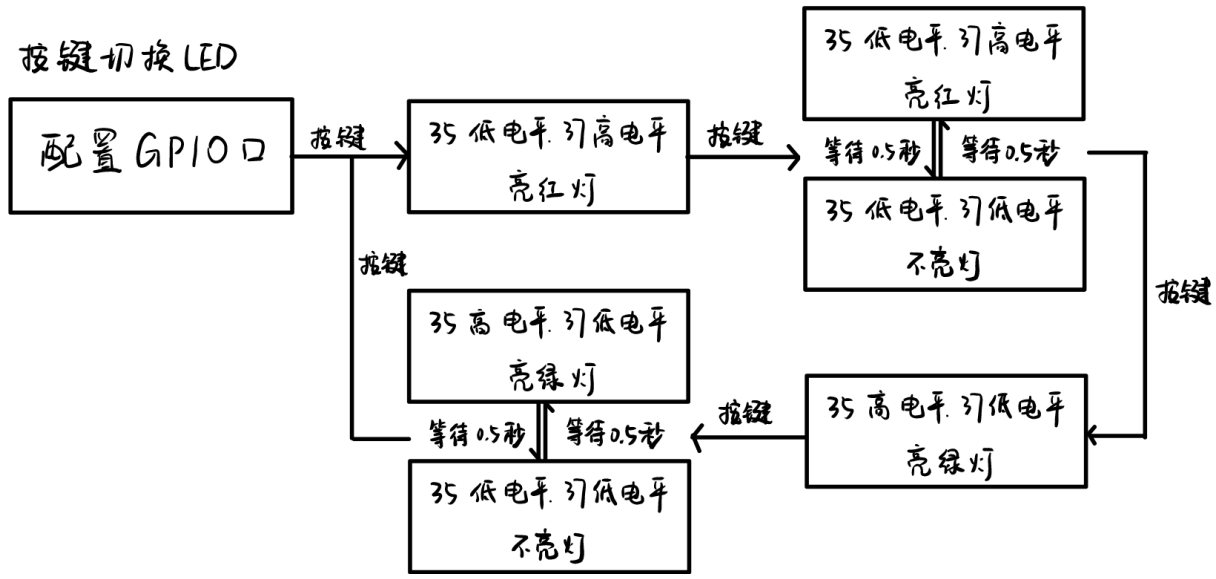


图 10: 按键控制 LED 程序流程图

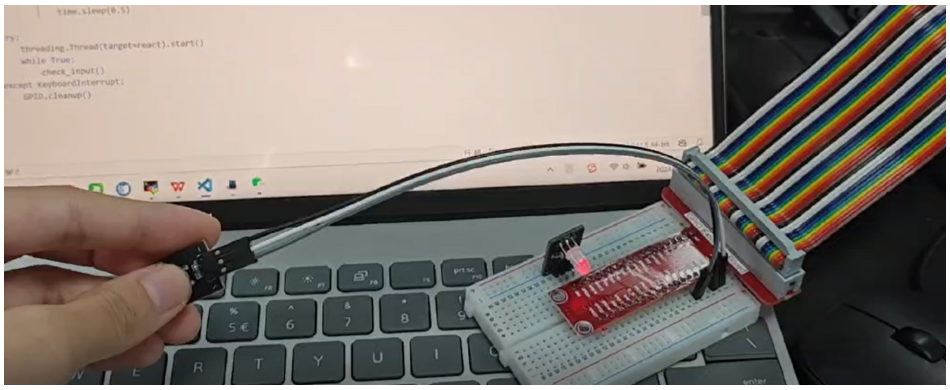


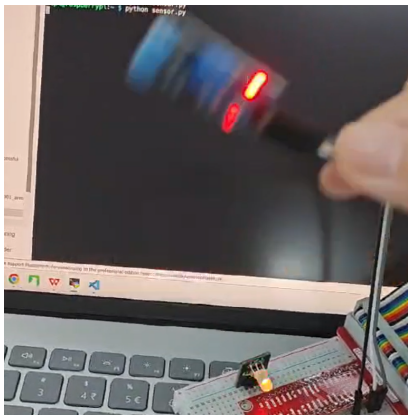
图 11: 按键控制 LED 亮灯模式切换运行效果

3.3 倾斜传感器、振动传感器

与按键开关相同的程序，连接不同的传感器，可以得到不同的工作效果。

```
import RPi.GPIO as GPIO

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(35, GPIO.OUT)
GPIO.setup(37, GPIO.OUT)
GPIO.setup(12,GPIO.IN)

try:
  while True:
    if GPIO.input(12) == 0:
      GPIO.output(35,True)
      GPIO.output(37,False)
    else:
      GPIO.output(35,False)
      GPIO.output(37,True)
except KeyboardInterrupt:
  GPIO.cleanup()
```

(a) 倾斜传感器



(b) 振动传感器

图 12: 多种传感器运行效果

4 模数转换器

4.1 PCF8591

PCF8591 是一款集成了模数转换器（ADC）和数模转换器（DAC）功能的单芯片、单电源、低功耗 8 位 CMOS 数据采集设备。

- 模拟输入与输出：PCF8591 具有四个模拟输入通道（AIN0-AIN3）和一个模拟输出通道（AOUT），支持 8 位的分辨率。
- I2C 总线接口：PCF8591 通过标准的 I2C 总线接口与主控制器或其他设备进行通信，实现对芯片的配置和数据传输。
- 地址引脚：PCF8591 有三个地址引脚 A0、A1 和 A2，用于编程硬件地址，允许在同一 I2C 总线上连接多达八个 PCF8591 设备而不需要额外的硬件。
- 低功耗设计：PCF8591 设计为单电源供电，工作电压范围为 2.5V 至 6.0V，具有低待机电流。
- 模数转换（ADC）：PCF8591 内置的 ADC 可以将模拟信号转换为数字信号，支持 4 个独立的模拟输入通道，实现 8 位的分辨率转换。
- 数模转换（DAC）：PCF8591 还集成了一个 8 位分辨率的 DAC，可以将数字信号转换为相应的模拟输出信号以控制外部设备。
- 电压参考源：PCF8591 芯片内部提供了一个可编程的电压参考源，用于确定模拟输入和输出的参考电压水平。
- 自动递增通道选择：PCF8591 支持自动递增通道选择功能，方便对多个模拟输入通道进行顺序采样。
- 模拟电压范围：PCF8591 的模拟电压范围从 VSS 到 VDD，内置跟踪和保持电路，以及 8 位逐次逼近 A/D 转换功能。

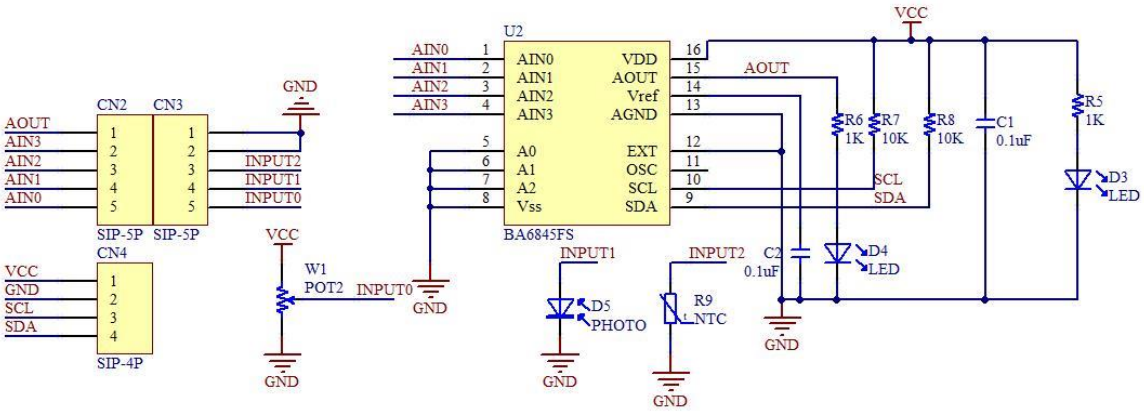


图 13: PCF8591 模数转换器原理图

4.2 I2C 通信

I2C（Inter-Integrated Circuit）通信是一种同步的、多主机、多从机的串行计算机总线，主要用于微控制器和各种外围设备之间的通信。I2C 通信协议由 Philips Semiconductor 开发，广泛应用于嵌入式系统中，因为它只需要两根线（数据线 SDA 和时钟线 SCL），从而减少了所需的引脚数量，简化了硬件设计。

I2C 通信需要两条线连接，数据线（SDA），数据线是双向的，用于传输数据。时钟线（SCL），时钟线是单向的，由主设备控制，用于同步数据传输。I2C 通信的参与者有：主设备（Master），控制 I2C 总线上的通信，生成时钟信号，并开始和结束数据传输。从设备（Slave），响应主设备的通信请求，根据主设备的指令发送或接收数据。

I2C 通信的基本过程：

- 启动条件：主设备通过在 SCL 为高电平时将 SDA 从高电平拉低来生成启动条件。
- 地址加写/读位（Address + R/W bit）：主设备发送从设备地址，地址后跟一个读/写位（R/W bit），指示是读操作（R=1）还是写操作（R=0）。
- 应答位（ACK）：从设备在接收到地址后，如果准备好响应，会发送一个应答位（低电平）。
- 数据传输：根据读写位，主设备和从设备之间开始数据传输。数据传输是 8 位的，每个字节后跟一个应答位。
- 非应答位（NACK）：在数据传输结束时，主设备发送一个非应答位（高电平）。
- 停止条件：主设备通过在 SCL 为高电平时将 SDA 从低电平拉高来生成停止条件。

I2C 有非常多优势，如：多主机能力：I2C 总线允许多个主设备，但一次只能有一个主设备控制总线。多从机能力：多个从设备可以共享同一总线，每个从设备有唯一的地址。速度：I2C 支持不同的速度模式，包括标准

模式（最高 100kbps）、快速模式（最高 400kbps）、快速模式加（最高 1Mbps）和高速模式（最高 3.4Mbps）。总线仲裁：如果两个主设备同时尝试控制总线，I2C 有一套仲裁机制来决定哪个主设备继续控制。

I2C 广泛应用于需要多个外围设备与微控制器通信的场景，如传感器数据读取、EEPROM 数据存储、LCD 显示控制等。

4.3 DA 输出传感器数据

函数 bus.write_byte_data() 是 smbus 库中的一个方法，用于向 I2C 设备的特定寄存器写入一个字节的数据。

```
bus.write_byte_data(i2c_address, command, value)
```

- i2c_address：I2C 设备的地址，这是一个介于 0x03 和 0x77 之间的值（实际可用的地址范围可能受到 I2C 总线和设备的限制）。这个地址用于在 I2C 总线上标识特定的设备。
- command：寄存器的地址或命令码。在 I2C 通信中，某些设备使用寄存器来存储数据，而 command 参数就是用来指定要写入数据的寄存器地址。在某些情况下，command 也可以是一个命令码，用于告诉设备执行特定的操作。
- value：要写入的数据值。这是一个介于 0x00 和 0xFF 之间的字节值（即 0 到 255 的整数），代表了要发送到指定寄存器的具体数据。

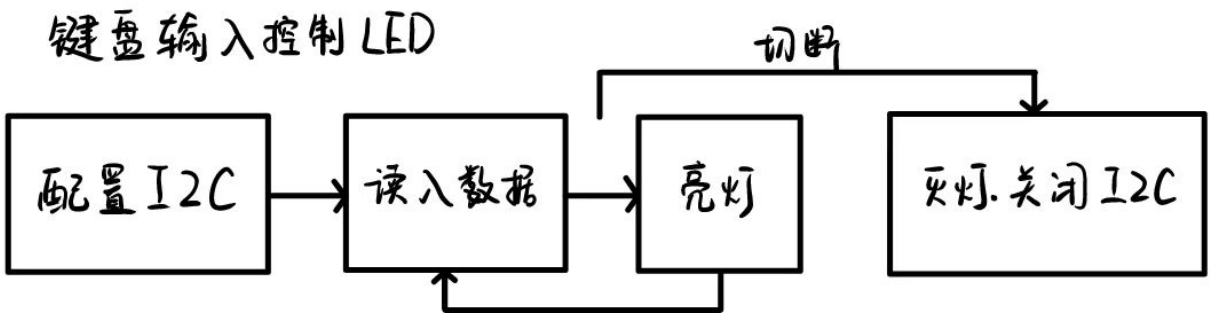


图 14: 键盘控制 LED 程序流程图

```python
import smbus
bus = smbus.SMBus(1)
try:
  while True:
    bright = int(input())
    if 0 < bright < 255:
      bus.write_byte_data(0x48, 0x42, bright)
except KeyboardInterrupt:
  bus.write_byte_data(0x48, 0x42, 0)
  bus.close()
```

通过 input() 函数读入键盘输入，用 bus.write_byte_data 写入数据，数字亮度转换成模拟信号接在小灯上。0x48 表示设备地址，在 PCF8591 中使用该值。0x42 表示寄存器地址，即 AOUT。

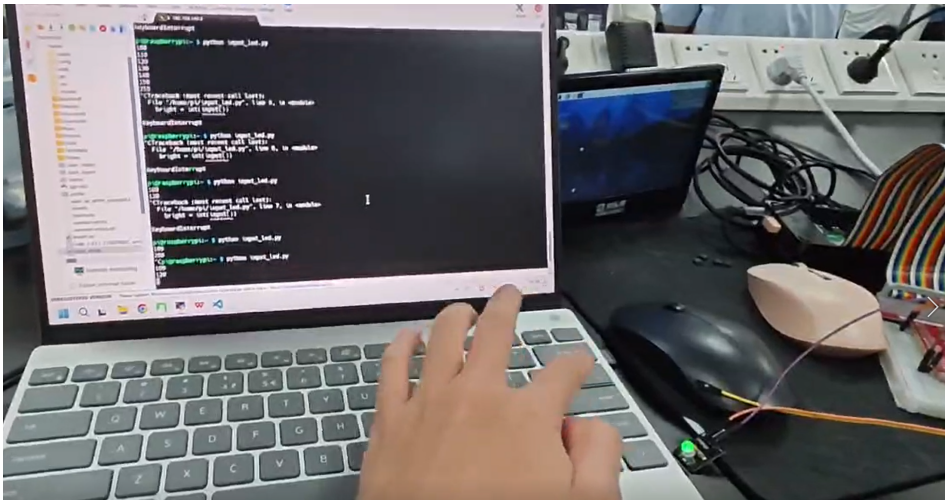


图 15: 键盘输入数字量控制 LED 亮灯运行效果

同样地，我们可以对写入的数字量增加逻辑，使其实现呼吸灯的效果。只需要让它不断增加，减小即可。

```
while True:
  for data in range(0, 255):
    bus.write_byte_data(0x48, 0x42, data)
    time.sleep(0.01)
  for data in range(255, 0, -1):
    bus.write_byte_data(0x48, 0x42, data)
    time.sleep(0.01)
```

(a) 暗

(b) 中等亮度

(c) 亮

图 16: 呼吸灯运行效果

4.4 AD 输入控制功能

将输入信号连接在 AIN0 上，寄存器地址写作 0x40，使用 read_byte_data 函数可以读取输入信号。输入的信号读取为数字量，为便于人为核对，可以转换成模拟量。模块接 3.3V 电源，故数字量最大值即 3.3V。

```
data = bus.read_byte_data(0x48, 0x40)
print(f"{data/255*3.3}V")
```

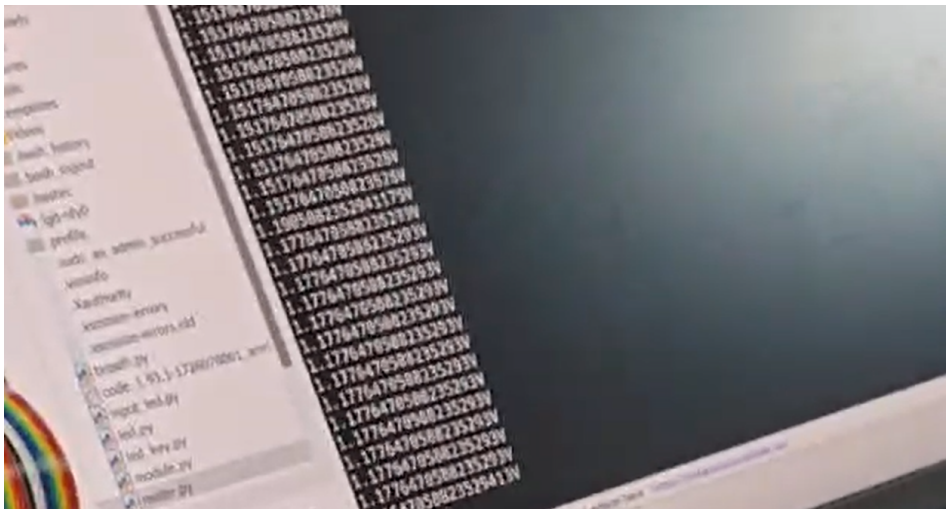


图 17: 读取电压程序运行效果

连续读入一个信号发生器产生的正弦波，可以对于读入的信号作图：

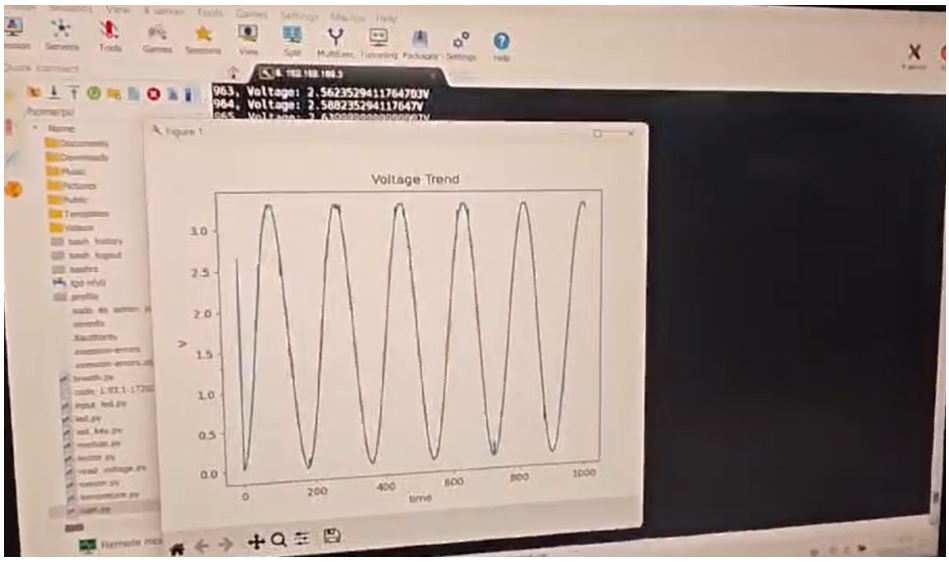


图 18: 读取正弦波电压作图运行效果

4.5 AD-DA 组合功能

综合以上两种功能，可以通过读入的电压信号控制 LED 亮度电压。

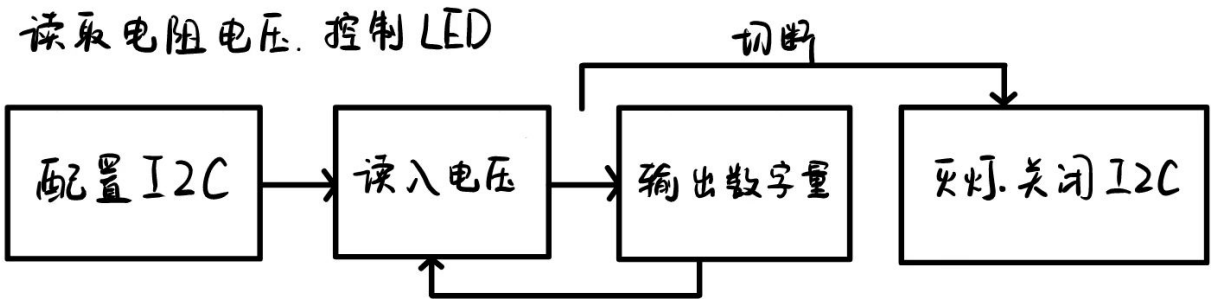


图 19: AD-DA 程序流程图

```
data = bus.read_byte_data(0x48, 0x40)
bus.write_byte_data(0x48, 0x42, data)
```

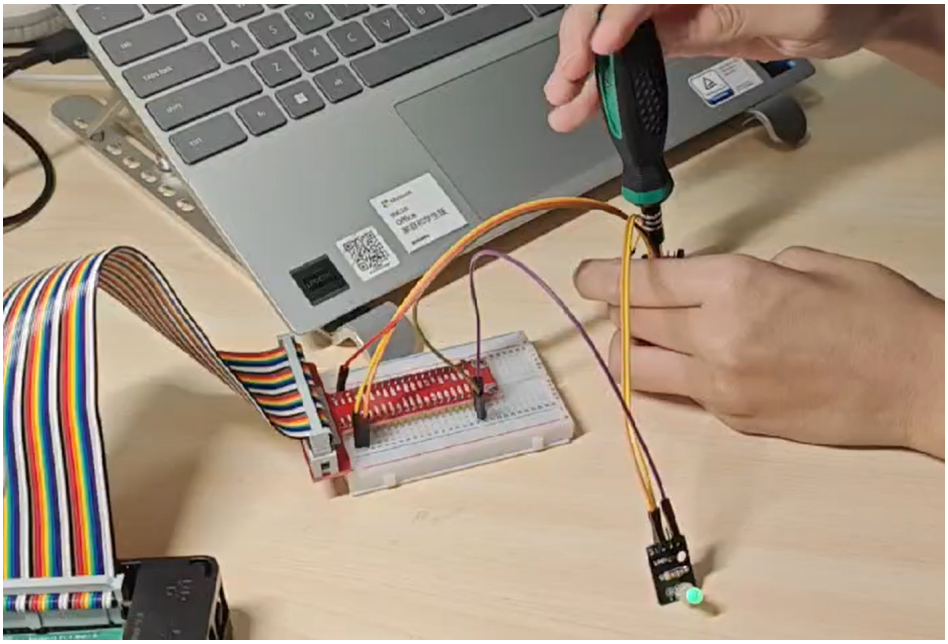


图 20: AD-DA 程序运行效果

5 温度传感器

5.1 传感器原理

温度传感器是一种关键的检测设备，它能够将温度变化转换为电信号，广泛应用于工业自动化、环境监测、家用电器以及医疗设备等领域。它们可以是接触式的，如热电偶和热敏电阻，也可以是非接触式的，如红外传感器。这些传感器能够提供模拟或数字信号输出，以便于监测和控制温度。在选择温度传感器时，需要考虑其测量范围、精度、响应时间以及环境适应性，以确保它们能够在特定应用中提供可靠的性能。此外，为了保持测量的准确性，定期校准也是必要的。

- 热电偶：基于两种不同金属或合金的接触点之间的电压差，这种电压差与接触点的温度有关。
- 热敏电阻（NTC/PTC）：电阻随温度变化而变化，通常用于温度补偿和过热保护。
- 电阻温度检测器（RTD）：如铂电阻温度检测器，其电阻随温度线性变化。
- 红外传感器：可以非接触地测量物体表面的温度。

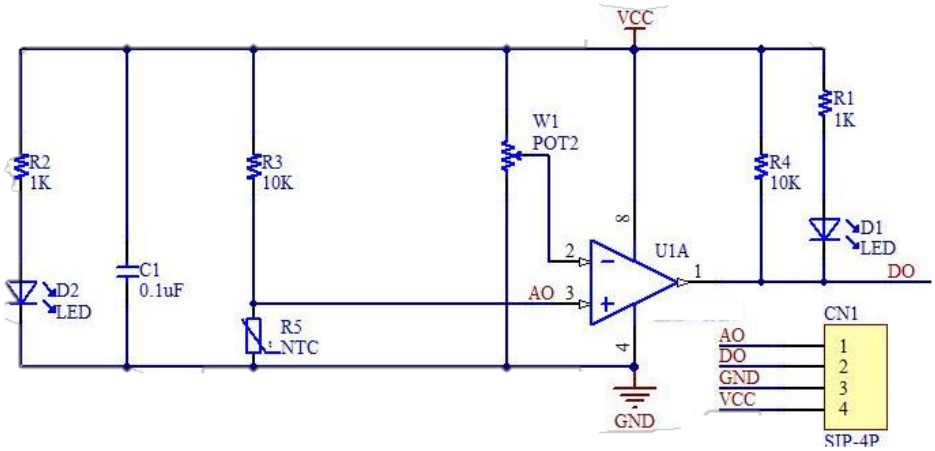


图 21: 温度传感器原理图

5.2 读取实时温度

仍然使用 PCF8591，转换温度传感器读取到的模拟量，以数字量形式输入电脑。根据电路图，可以计算出热敏电阻值。再根据 Steinhart-Hart 方程，可以计算出温度。

$$R_t = 10000 \times \frac{V_r}{5 - V_r} \tag{1}$$

$$R = R_0 exp[B(\frac{1}{T} - \frac{1}{T_0})] \tag{2}$$

其中 R 为测到的电压计算出的阻值，R_0 为传感器型号所设置，B 为 3590，T_0 为-273.15K。

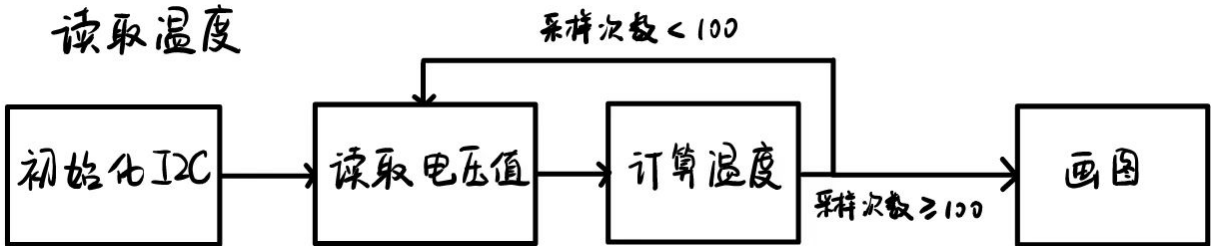
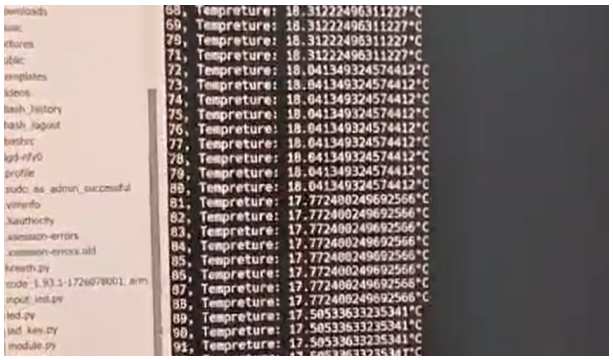


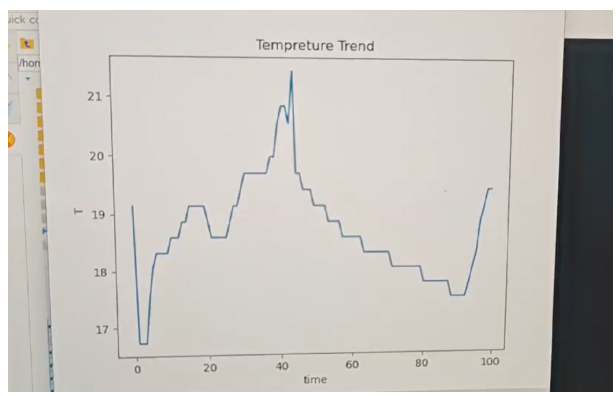
图 22: 温度传感器测量流程图

```python
bus = smbus.SMBus(1)
list = []
try:
  while True:
    data = bus.read_byte_data(0x48, 0x40)
    Vr = 3.3 * float(data) / 255
    Rt = 10000 * Vr / (5 - Vr)
    T = 1 / (1/3950 * math.log(Rt/10000) + 1/273.15) - 273.15
    print(f"{len(list)+1}, Tempreture: {T}°C")
    if len(list) < 100:
      list.append(T)
    if len(list) >= 100:
      break
    time.sleep(0.1)
  plt.plot(list)
  plt.title('Tempreture Trend')
  plt.ylabel('T')
  plt.xlabel('time')
  plt.show()
except KeyboardInterrupt:
  bus.close()
```

实验时，在测量 100 次的时间内，可以通过手触摸传感器增加温度，验证其功能。



(a) 读取单个电压值



(b) 画图效果

图 23: 温度传感器运行效果

6 超声波传感器

6.1 传感器原理

超声波传感器是一种利用超声波信号来检测物体位置、距离或速度的传感设备。超声波传感器的工作原理基于脉冲-回波方法。

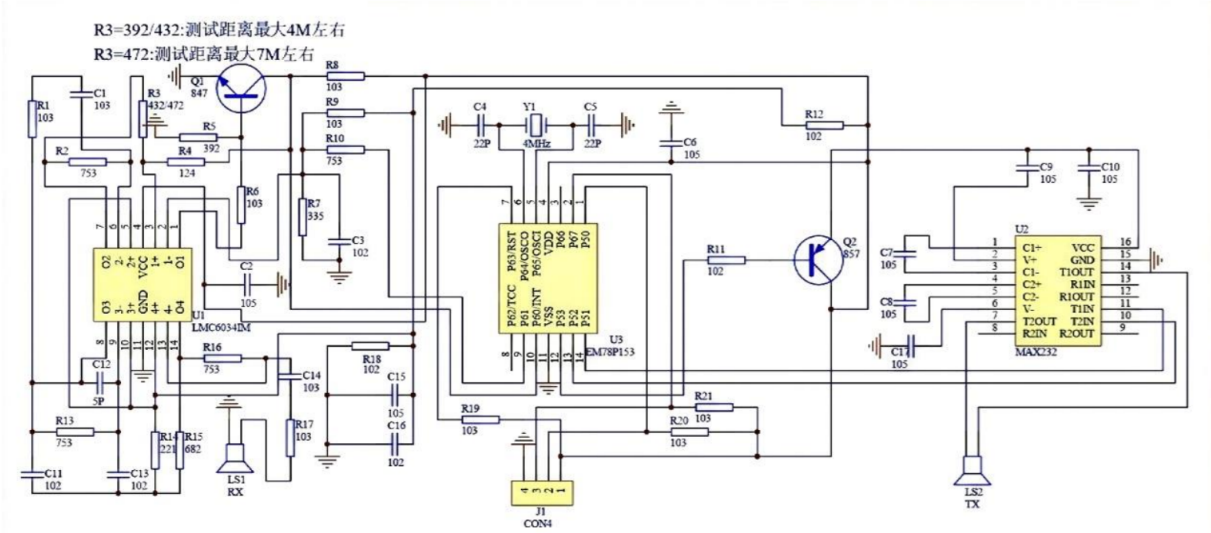


图 24: 超声波传感器原理图

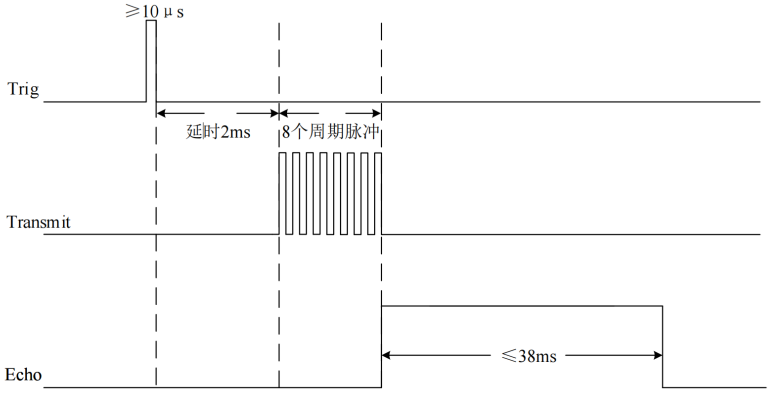


图 25: 超声波传感器工作时序图

- 发射阶段：传感器中的换能器（通常是压电晶体）接收到电信号后开始振动，产生特定频率的超声波脉冲，这些脉冲被定向发送到空气中或其他介质中。
- 传播阶段：发出的超声波以一定速度向前传播，如果前方存在物体或障碍物，超声波就会在这个物体表面发生反射。
- 接收阶段：传感器的换能器同时也作为接收器，等待捕捉从目标反射回来的超声波信号。当反射波返回并

击中换能器时，它再次转换成电信号。

- 数据分析：控制系统会根据两次信号之间的时间差来计算出发射点与障碍物之间的距离。由于已知声波的速度，在空气中的大约为 340 米/秒（随温度变化略有差异），因此可以通过公式 $D = (T * V)/2$ 来得出结果，其中 D 代表距离，T 是从发射到接收到回声所需时间的一半，V 则是声速。

6.2 测量结果

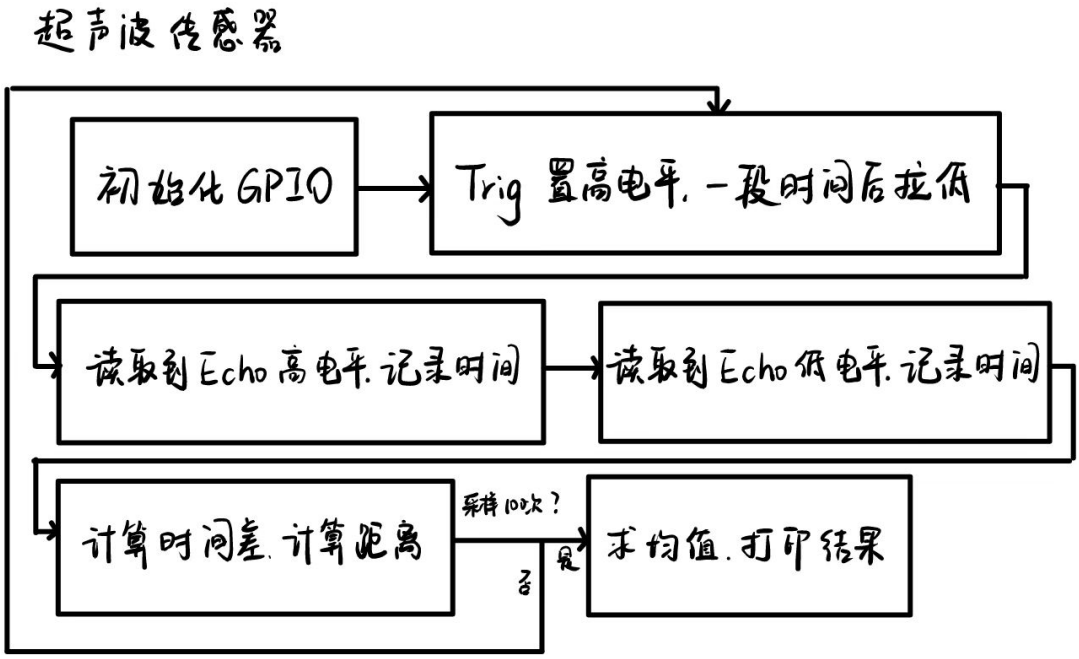
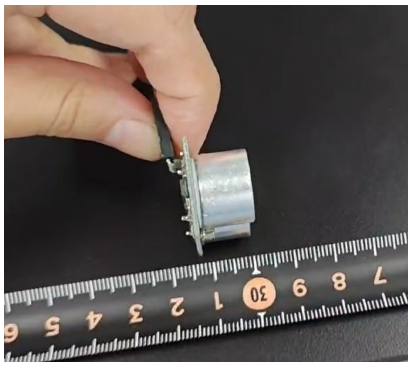


图 26: 超声波传感器测量程序流程图

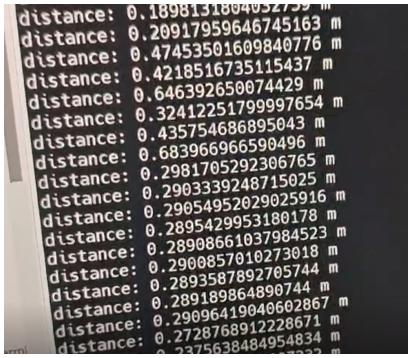
```
import RPi.GPIO as GPIO
import time
import numpy as np

GPIO.setmode(GPIO.BOARD)
GPIO.setup(38,GPIO.OUT)
GPIO.setup(40,GPIO.IN)
GPIO.setwarnings(False)

try:
  while True:
    d_list = []
    for _ in range(1,100):
      GPIO.output(38,True)
```

```
        time.sleep(1e-5)
        GPIO.output(38,False)
        while True:
          if GPIO.input(40) == 1:
            break
        t1 = time.time()
        while True:
          if GPIO.input(40) == 0:
            break
        t2 = time.time()
        d = (t2-t1) * 343 /2
      d_list.append(d)
      d_list = np.array(d_list)
      print(f"distance: {d_list.mean()} m")
except KeyboardInterrupt:
  GPIO.cleanup()
```

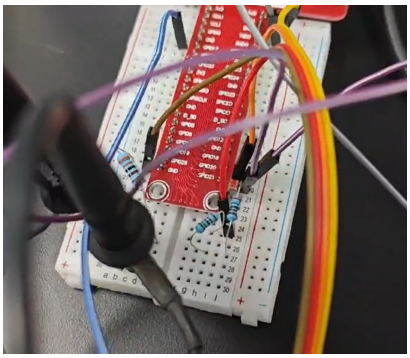
把 Trig 置高电平一段时间后，传感器发送超声波，Echo 接收到的高电平时间即可计算出 ToF 距离。由于单次测量可能不准，而测量速度非常快，可以连续采样取均值作为测量结果。



(a) 测量位置



(b) 测量结果



(c) 电路分压接线

图 27: 超声波传感器运行效果

测量序号	1	2	3	4	5	6	7	8	9	10
实测距离	19.52cm	20.75cm	22.48cm	15.19cm	9.68cm	6.83cm	24.80cm	29.97cm	35.88cm	32.32cm
超声测距	20cm	21cm	23cm	15cm	10cm	7cm	25cm	30cm	36cm	33cm
误差	-0.48cm	-0.25cm	-0.52cm	0.19cm	-0.32cm	-0.17cm	-0.20cm	-0.03cm	-0.12cm	-0.68cm

表 1: 超声波传感器实测结果

7 蜂鸣器

7.1 蜂鸣器控制

蜂鸣器是一种电子元件，当通电时会发出声音。蜂鸣器由振荡器、电磁线圈、磁铁、振动膜片及外壳等组成。接通电源后，振荡器产生的音频信号电流通过电磁线圈，使电磁线圈产生磁场。振动膜片在电磁线圈和磁铁的相互作用下，周期性地振动发声。PWM（Pulse Width Modulation，脉冲宽度调制）是一种利用微处理器的数字输出来控制模拟电路的技术。可以通过改变 PWM 信号的占空比和频率来控制蜂鸣器的音量和音调。

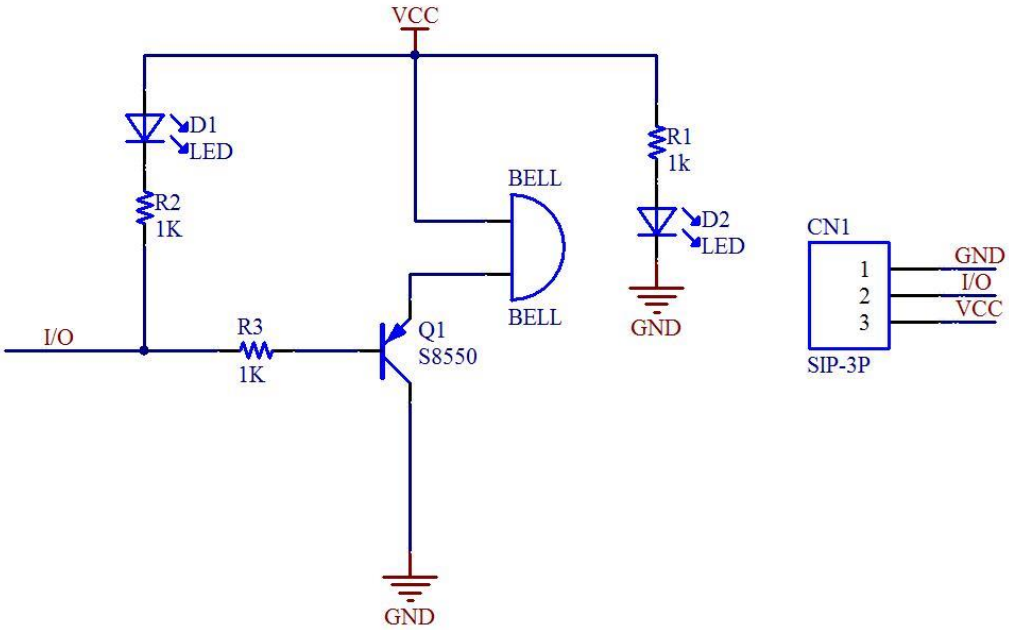


图 28: 蜂鸣器原理图

调整不同的频率，可以控制音调。因此，这样可以使用蜂鸣器演奏简单的音乐。

图 29: 蜂鸣器演奏流程图

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
```

```
GPIO.setup(11,GPIO.OUT)
GPIO.setwarnings(False)

def pwm(freq, dur):
  pi_pwm.start(50)
  pi_pwm.ChangeFrequency(freq)
  time.sleep(0.3*dur)
  pi_pwm.stop()
  time.sleep(0.1*dur)

def music(tune):
  for i in range(0, len(tune)):
    pwm(pitch_to_freq[tune[i][0]], tune[i][1])

pitch_to_freq = {
  'C': 261, # Do
  'D': 293, # Re
  'E': 329, # Mi
  'F': 349, # Fa
  'G': 392, # So
  'A': 440, # La
  'B': 493  # Si
}

try:
  pi_pwm = GPIO.PWM(11, 440)

  tune = [
  ('C',1),('C',1),('G',1),('G',1),('A',1),('A',1),('G',2),
  ('F',1),('F',1),('E',1),('E',1),('D',1),('D',1),('C',2),
  ('G',1),('G',1),('F',1),('F',1),('E',1),('E',1),('D',2),
  ('G',1),('G',1),('F',1),('F',1),('E',1),('E',1),('D',2),
  ('C',1),('C',1),('G',1),('G',1),('A',1),('A',1),('G',2),
  ('F',1),('F',1),('E',1),('E',1),('D',1),('D',1),('C',2)
  ]
  music(tune)

  GPIO.cleanup()

```

```
except KeyboardInterrupt:
  pi_pwm.stop()
  GPIO.cleanup()
```

7.2 舵机控制

同样地，可以用 PWM 输出控制舵机。需要注意，使舵机逐渐转到目标位置，防止瞬时电流过大烧坏电路。

```
def set_angle(angle):
  duty = angle / 9 + 2.5
  pwm.ChangeDutyCycle(duty)
  print(duty)

try:
  pwm = GPIO.PWM(11,50)
  pwm.start(2.5)
  last_angle = 0
  while True:
    angle = int(input())
    if angle < last_angle:
      for i in range(last_angle, angle, -1):
        set_angle(i)
        time.sleep(0.01)
    elif angle > last_angle:
      for i in range(last_angle, angle):
        set_angle(i)
        time.sleep(0.01)
    last_angle = angle
except KeyboardInterrupt:
  pwm.stop()
  GPIO.cleanup()
```

8 操纵杆

PS2 操纵杆是一种常用于视频游戏和机器人控制的输入设备，它允许用户通过操纵杆的移动来控制设备的方向或角度。

PS2 操纵杆有两个模拟输出，分别对应 X 轴和 Y 轴的偏移量。这两个轴反映了摇杆在水平和垂直方向上的位置变化。PS2 操纵杆还有一个数字输出，表示按键（Z 轴）是否被按下。当用户按下摇杆时，电路接通，数字

信号输出状态发生改变。PS2 摇杆本质上是一个双向十字的电位器。在初始状态下，X、Y 轴的输出电压约为 2.5V。当摇杆往某个方向推动时，电位器的抽头阻值会随之变化，导致输出的相应轴电压值增加或减小。电压变化范围为 0V 到 5V，通过检测电压的变化，连接的设备可以确定摇杆的位置。

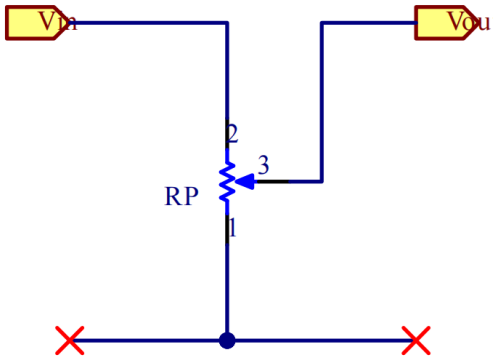


图 30: PS2 操纵杆原理图

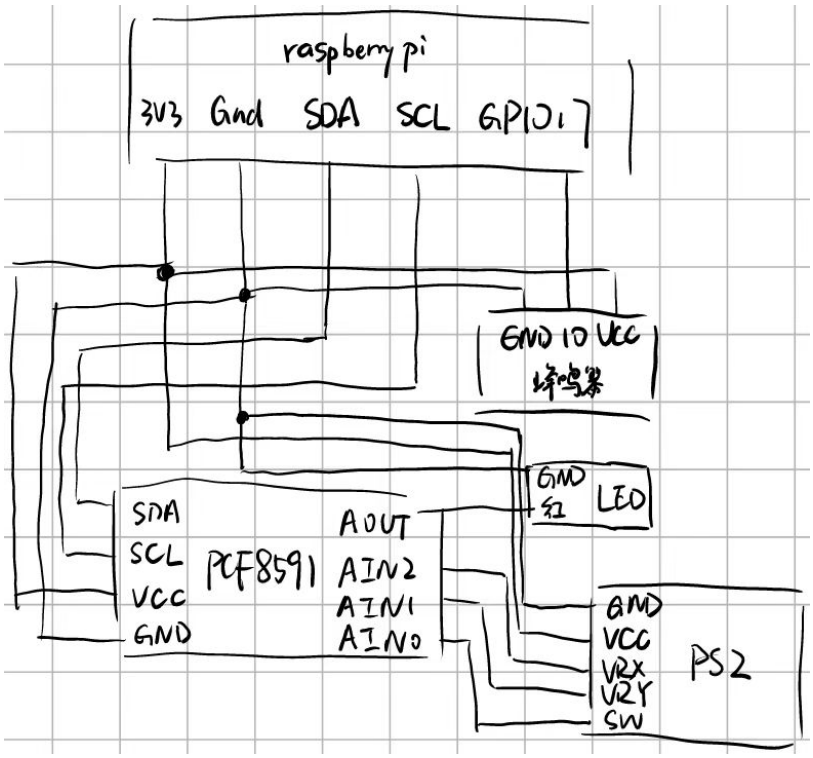


图 31: PS2 操纵杆工作接线图

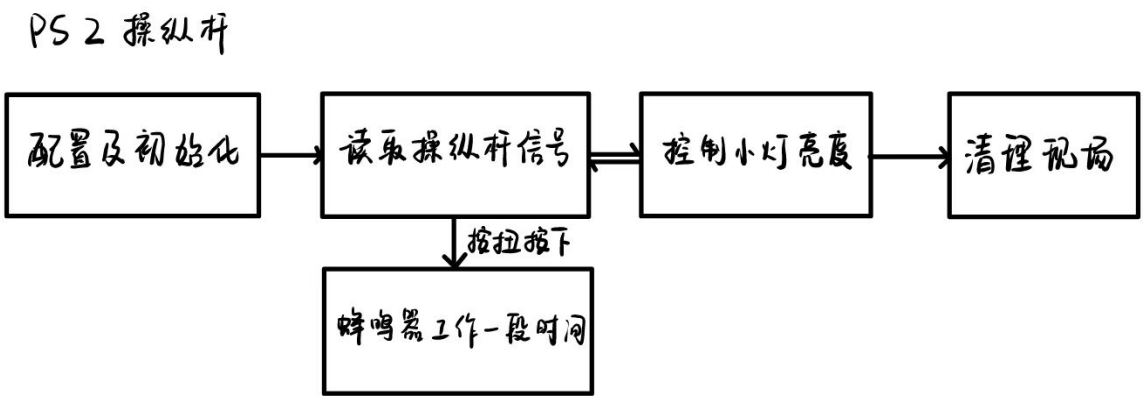


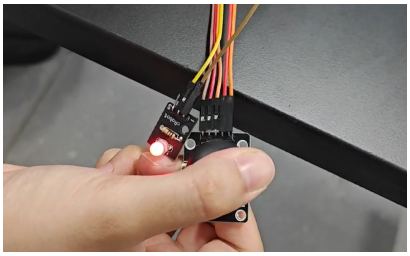
图 32: PS2 操纵杆工作流程图

将其与模数转换器相连，易于读取数据。

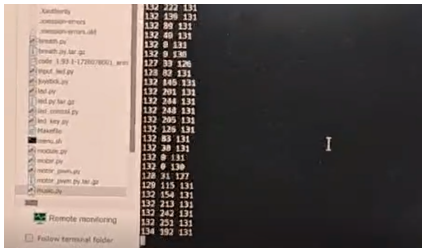
```
data0 = bus.read_byte_data(0x48, 0x40)
data1 = bus.read_byte_data(0x48, 0x41)
data2 = bus.read_byte_data(0x48, 0x42)
```

也可以通过其结果再输出控制其他设备，如蜂鸣器，小灯。

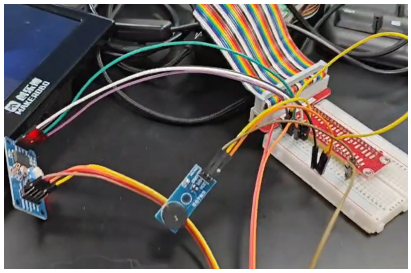
```
bus.write_byte_data(0x48, 0x42, int(255-data2))
if data1 == 0:
  pi_pwm.start(50)
else:
  pi_pwm.stop()
```



(a) 小灯随操纵杆灯亮



(b) 测量结果



(c) 实物电路接线

图 33: PS2 操纵杆运行效果

9 红外遥控

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，接收头输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。

9.1 遥控配置

使用红外遥控，首先需要配置 lirc 库：

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

然后在/boot/firmware/config.txt 中配置引脚信息。

```
dtoverlay=gpio-ir,gpio_pin=23
dtoverlay=gpio-ir-tx,gpio_pin=24
```

在/etc/lirc/lirc_options.conf 中修改驱动文件信息。

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

然后就可以测试遥控是否正常工作，当按下按钮时，应当接收到几个 pulse 和 timeout 信息。

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

接下来就可以开始录入遥控信息。由于红外可识别距离较远，录入时需避免过多同类型红外信号的干扰。

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

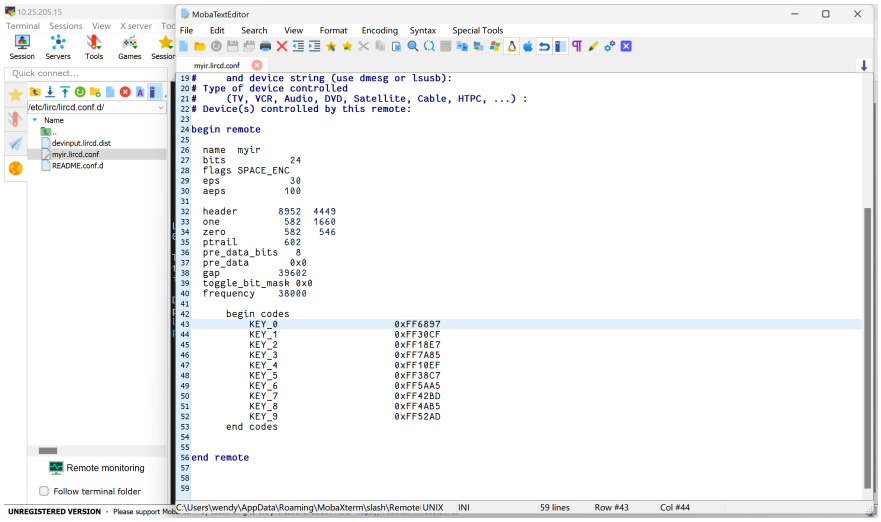


图 34: 遥控器录入结果

录入完成后，可以测试是否录入成功：

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

而后在/etc/lirc/lirc 文件中可以配置按钮与别名，需要使用的文件名相关联。

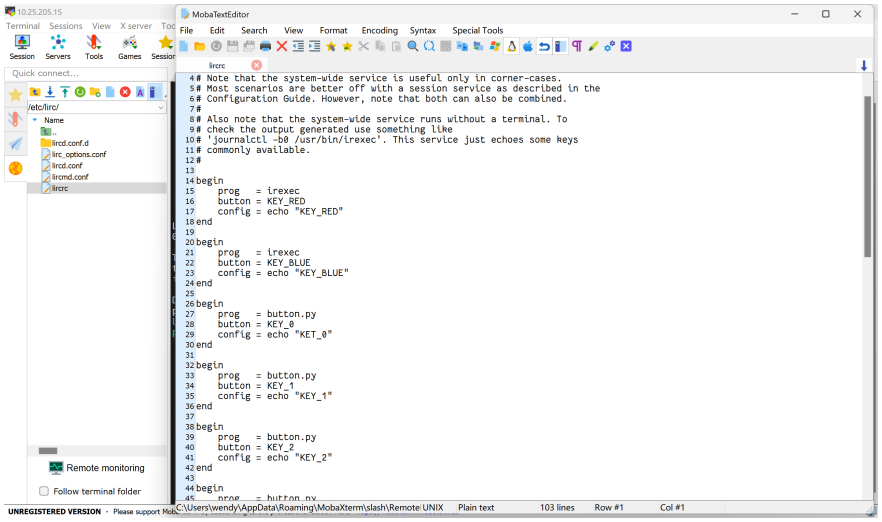


图 35: 遥控器关联结果

9.2 使用遥控进行控制

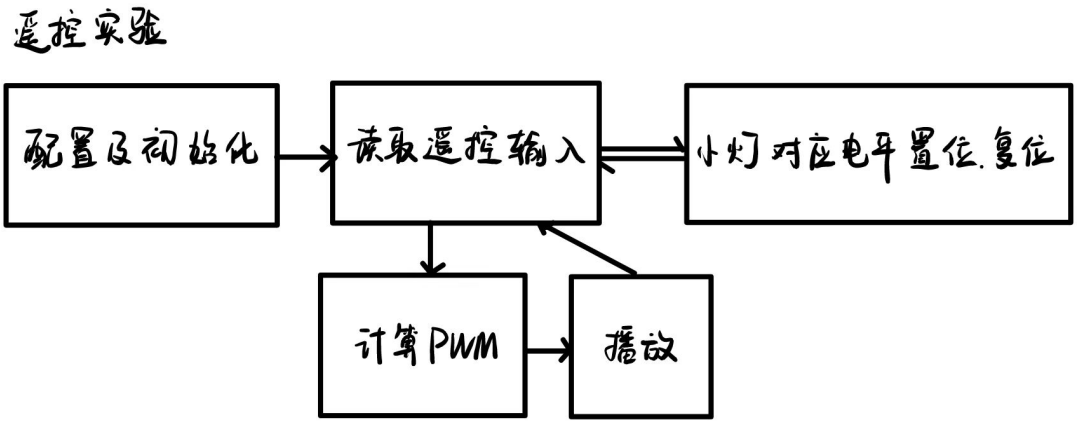


图 36: 遥控器工作流程图

可以沿用与之前实验中类似的，使用蜂鸣器播放音乐的函数：

```
def pwm(freq, dur):
  pi_pwm.start(50)
  pi_pwm.ChangeFrequency(freq)
```

```
  time.sleep(0.3*dur)
  pi_pwm.stop()
  time.sleep(0.1*dur)

def music(tune):
  pwm(pitch_to_freq[tune[0]], tune[1])

pitch_to_freq = {
  'C': 261, # Do
  'D': 293, # Re
  'E': 329, # Mi
  'F': 349, # Fa
  'G': 392, # So
  'A': 440, # La
  'B': 493 # Si
}
```

对于遥控器的输入，在关联了该脚本后，能够读取到遥控信息。

```
def pasreset(data):
  if data == 'echo "KEY_1"':
    print("1 Pressed")
    return 1
  elif data == 'echo "KEY_2"':
    print("2 Pressed")
    return 2
  elif data == 'echo "KEY_3"':
    print("3 Pressed")
    return 3
  elif data == 'echo "KEY_4"':
    print("4 Pressed")
    return 4
  elif data == 'echo "KEY_5"':
    print("5 Pressed")
    return 5
  elif data == 'echo "KEY_6"':
    print("6 Pressed")
    return 6
  elif data == 'echo "KEY_7"':
    print("7 Pressed")
    return 7
```

```
  elif data == 'echo "KEY_8"':
    print("8 Pressed")
    return 8
  elif data == 'echo "KEY_9"':
    print("9 Pressed")
    return 9
```

配置完函数后，可以编写主逻辑，使系统能够根据遥控的 1-6 演奏音乐，7-9 控制小灯。

```
with lirc.LircdConnection("button.py",) as conn:
  while True:
    string = conn.readline()
    key = pasreset(string)
    if key == 1:
      music(['C',1])
    elif key == 2:
      music(['D',1])
    elif key == 3:
      music(['E',1])
    elif key == 4:
      music(['F',1])
    elif key == 5:
      music(['G',1])
    elif key == 6:
      music(['A',1])
    elif key == 7:
      GPIO.output(35, True)
      GPIO.output(37, False)
    elif key == 8:
      GPIO.output(35, False)
      GPIO.output(37, True)
    elif key == 9:
      GPIO.output(35, False)
      GPIO.output(37, False)
```

图 37: 遥控器运行效果

10 中断

中断是计算机系统中一个关键的机制，它允许硬件或软件在关键时刻暂停当前任务，以处理紧急事件。在计算机硬件层面，中断通常由外部设备（如键盘、鼠标、定时器等）或系统内部条件（如定时器溢出）触发，这些事件需要操作系统立即注意并作出响应。当一个中断发生时，CPU 会暂停当前正在执行的程序，保存当前任务的状态（如程序计数器和寄存器），然后跳转到中断服务例程（Interrupt Service Routine, ISR）。ISR 是一段专门用来处理特定中断的代码，它会执行必要的操作来响应中断源的请求。处理完毕后，中断服务例程会恢复 CPU 的状态，并返回到被中断的程序继续执行。

中断机制使得操作系统能够同时处理多个任务，提高了系统的响应能力和效率。中断的类型包括硬件中断和软件中断。硬件中断通常由物理设备触发，而软件中断则由程序请求操作系统服务时产生。中断处理是操作系统和硬件交互的重要部分，它涉及到中断的识别、处理和优先级管理。

在树莓派这类基于 Linux 的单板计算机中，软件中断通常与系统调用相关联。系统调用是用户空间程序请求操作系统内核服务的一种方式，它们通过软件中断来实现。当用户程序需要执行超出其权限范围的操作，如访问硬件资源或管理进程时，它会触发一个软件中断。Linux 内核通过定义一组中断向量来响应这些系统调用，执行相应的内核代码来完成请求。这种机制确保了系统的安全性和稳定性，同时也允许用户程序高效地利用操作系统提供的功能。

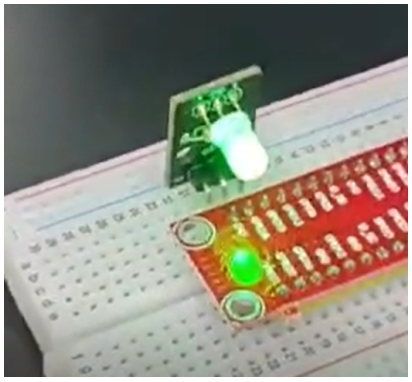
GPIO.wait_for_edge 是 RPi.GPIO 库中的一个函数，它用于等待一个 GPIO 引脚的状态发生变化，即等待一个边缘触发事件。这个函数是阻塞式的，意味着它会暂停程序的执行，直到检测到指定的边缘变化。

```
GPIO.wait_for_edge(channel, event, timeout=None, bouncetime=None)
```

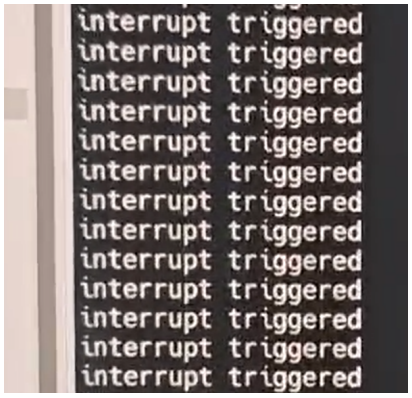
- channel：要监视的 GPIO 引脚编号。
- event：要等待的边缘类型，可以是 GPIO.RISING（上升沿）或 GPIO.FALLING（下降沿）。
- timeout：可选参数，指定等待边缘的最大时间（以秒为单位）。如果设置为 None，则无限期等待。
- bouncetime：可选参数，用于设置去抖动时间（以毫秒为单位）。如果设置为 None，则使用 RPi.GPIO 的默认去抖动时间。

使用该函数需要配置库：

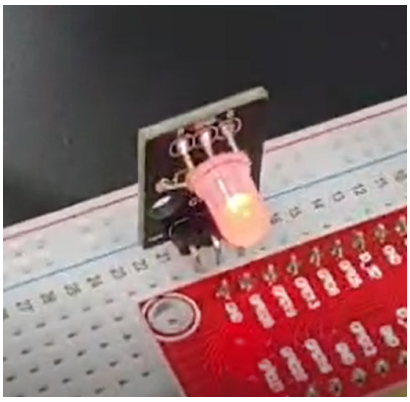
```
sudo apt remove python3-rpi.gpio
sudo apt update
sudo apt install python3-rpi-lgpio
```



(a) 初始状态——红绿交替闪烁



(b) 触发中断



(c) 中断状态——红灯常亮

图 38: PS2 操纵杆运行效果

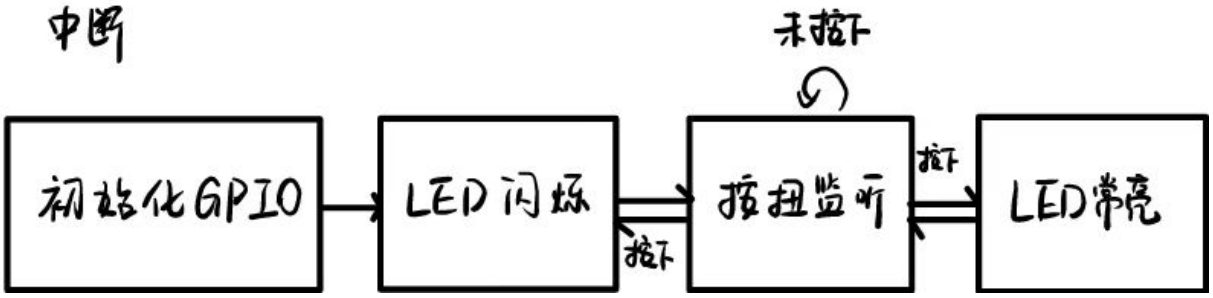


图 39: 中断实验流程图

代码初始化了两个 LED 灯和一个按钮的 GPIO 引脚，并创建了两个线程：一个用于交替闪烁 LED 灯，另一个用于监听按钮按下事件。当按钮被按下时，程序会停止 LED 灯的闪烁，并更新状态，以便在下一次按钮按下时改变 LED 灯的闪烁模式。程序持续运行，直到被外部中断（如键盘中断），此时会清理 GPIO 资源并退出。

```
def control_leds():
  while True:
  if state == 0:
    stop_blinking.clear()
    while not stop_blinking.is_set():
      GPIO.output(LED_RED, GPIO.LOW)
      GPIO.output(LED_GREEN, GPIO.HIGH)
      time.sleep(0.5)
      GPIO.output(LED_RED, GPIO.HIGH)
      GPIO.output(LED_GREEN, GPIO.LOW)
      time.sleep(0.5)
    time.sleep(0.01)

def button_interrupt():
  global state
  while True:
    GPIO.wait_for_edge(BUTTON, GPIO.FALLING)
    time.sleep(0.01)
    if GPIO.input(BUTTON) == 0:
      stop_blinking.set()
      state = (state + 1) % 2

def interrupt_callback():
  GPIO.output(LED_RED, GPIO.HIGH)
  GPIO.output(LED_GREEN, GPIO.LOW)
  time.sleep(0.1)
  print("interrupt triggered")

def main():
  setup_gpio()
  led_thread = threading.Thread(target=control_leds)
  button_thread = threading.Thread(target=button_interrupt)
  led_thread.start()
  button_thread.start()
  try:
    while True:
```

```
        if state == 1:
            stop_blinking.set()
            interrupt_callback()
    except KeyboardInterrupt:
        pass
    finally:
        cleanup_gpio()
```

11 结论

在本课程中，我们探索了树莓派在微机原理与微系统实验中的应用，也使用树莓派实践了微机原理与微系统的理论知识。

- 基础配置与 GPIO 口操作：我们成功地展示了如何通过树莓派的 GPIO 口控制 LED 灯和读取按键输入，验证了 GPIO 口的基本功能和编程接口的可靠性。
- 传感器数据采集：通过 PCF8591 模数转换器，我们实现了对温度和超声波传感器数据的采集，展示了树莓派在数据采集和处理方面的能力。
- PWM 输出控制：实验中，我们通过 PWM 信号控制蜂鸣器和舵机，证明了树莓派在控制简单机械和音频输出方面的有效性。
- PS2 操纵杆与红外遥控：我们探索了 PS2 操纵杆和红外遥控在树莓派上的应用，展示了树莓派在接收和处理外部控制信号方面的能力。

树莓派具有小巧、高性能、丰富的外设的特点，广泛被使用与电子类项目中。

- **工业自动化与监控系统**

  树莓派在工业自动化与监控系统中的应用日益广泛，其灵活性和低成本特性使其成为理想的数据采集与处理单元。例如，通过与温湿度传感器等外接传感器结合，树莓派能够实时监控生产环境的关键参数，并将数据上传至监控中心。这种集成化的解决方案不仅提高了生产效率，还增强了对生产过程的控制和优化能力，确保了工业环境的安全与稳定。此外，树莓派还可以作为软 PLC 使用，运行 Linux 操作系统，几乎可以在其上运行任何东西，包括高级语言编程和自动化解决方案。

- **环境检测**

  在环境检测领域，树莓派结合空气质量传感器和气象传感器，提供了一种低成本、高效率的监测工具。这些传感器能够收集关于空气质量、温度、湿度等关键环境指标的数据，帮助研究人员更好地理解环境变化，为环境保护政策的制定提供科学依据。树莓派环境监测系统可以实时监测环境数据，并通过网络将数据共享给用户，用户可以通过浏览器或移动设备访问树莓派的 Web 服务器，实时查看环境数据的变化。

- **娱乐**

  树莓派在娱乐领域的应用广泛。例如，BlueCherry Apps 是一个基于树莓派的操作系统，用于创建功能强大的多媒体系统。这款开源软件允许用户轻松地安装、配置和管理各种应用程序，以满足个性化需求。通

过 KODI 应用程序，用户可以在大屏幕上欣赏高清电影和电视节目，而 Plex 媒体服务器可以帮助管理和流式传输个人媒体文件。此外，树莓派还可以作为游戏机模拟器，让用户回味经典游戏，或作为家庭安全监控系统，实时查看、录制和回放视频片段。

- **机器人操作平台**

  在机器人技术领域，树莓派的计算能力使其成为机器人控制单元的理想选择。结合视觉传感器和距离传感器，树莓派能够使机器人实现自主导航和避障，执行复杂的任务。树莓派不仅支持多种编程语言，还对 Python 支持得非常好，使得使用 Python 语言可以快速地在树莓派上开发软件去控制机器人的传感器。此外，树莓派还能够运行人工智能相关的算法，比如 SVM，能简单地对数据进行分类，这为机器人的智能化提供了强大的支持。树莓派还允许配置 ubuntu 系统，可以便捷地使用机器人操作系统（ROS）平台进行开发，轻松调用封装库实现机器人的控制，搭建机器人操作平台，便于在此基础上展开各类研究。