

微机原理与微系统实验报告

应逸雯 12210159

2024 年 12 月 2 日

目录

1 基本配置	2
2 LED	3
2.1 GPIO 口的使用	3
2.2 代码编辑器	4
2.3 LED 控制	5
3 按键开关	6
3.1 按键开关控制	6
3.2 按键输入控制 LED 输出	7
3.3 倾斜传感器、振动传感器	9
4 模数转换器	10
4.1 PCF8591	10
4.2 I2C 通信	11
4.3 AD 输入传感器数据	12
4.4 DA 输出控制功能	13
4.5 AD-DA 组合功能	13
5 温度传感器	13
6 超声波传感器	13
7 蜂鸣器	13
8 操纵杆	13
9 红外遥控	13

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。

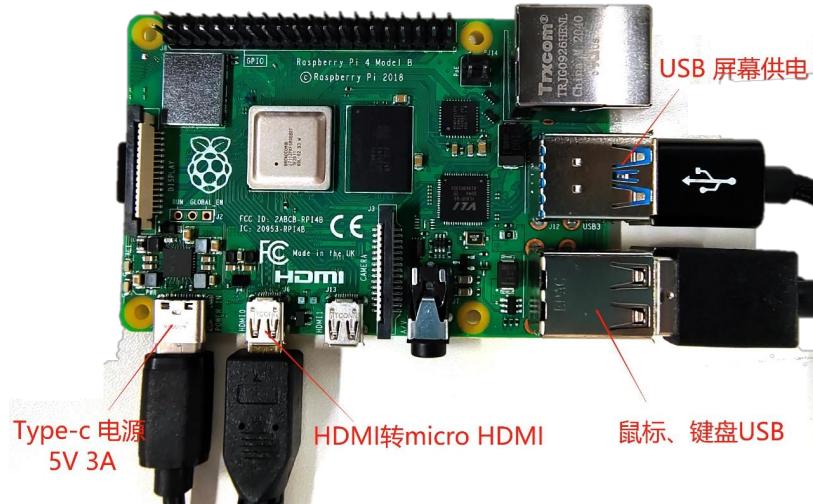


图 1: 树莓派硬件设备

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。

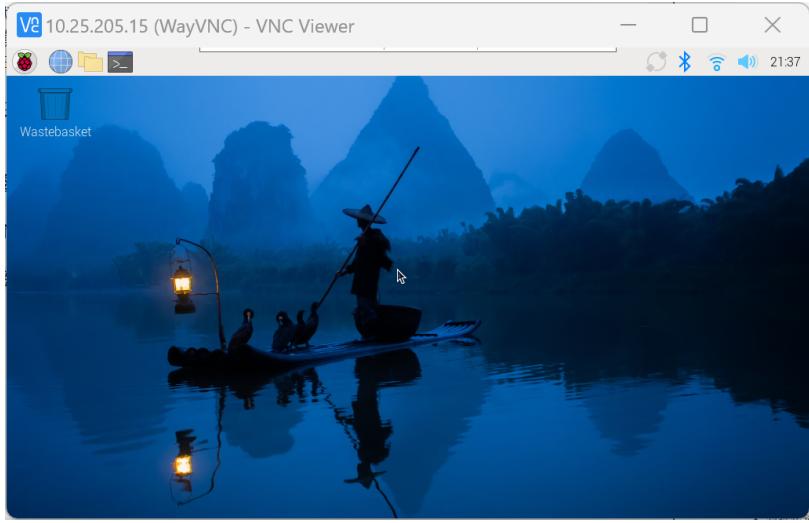


图 2: 树莓派系统界面

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

图 3: 树莓派 GPIO 口分布

使用代码控制 GPIO 口，需要配置环境：

```
1 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。

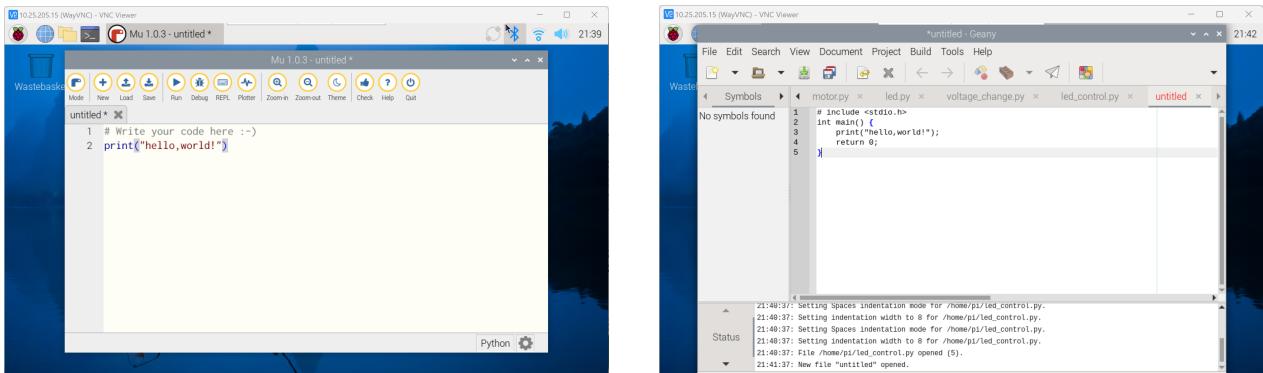


图 4: 编辑器界面

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。

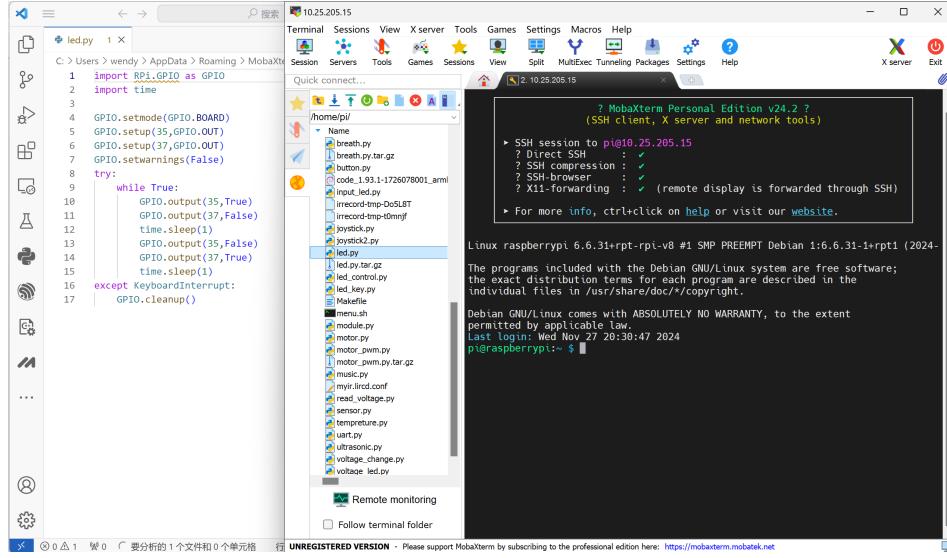


图 5: 主机 VSCode 编辑

2.3 LED 控制

LED 原理图如下：

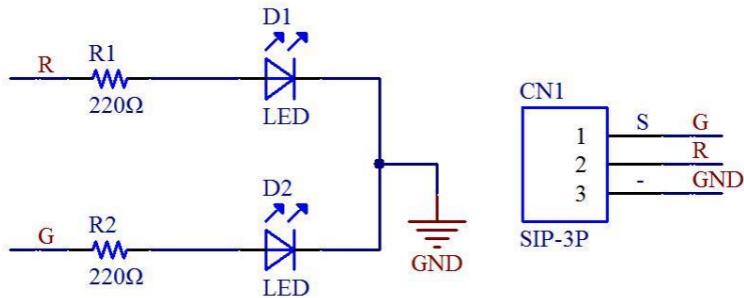


图 6: LED 原理图

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。

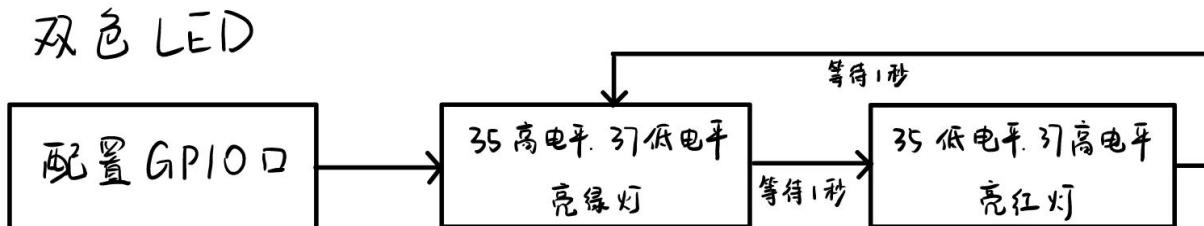


图 7: LED 控制程序流程图

```

1 import RPi.GPIO as GPIO
2 import time
3
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(35,GPIO.OUT)
6 GPIO.setup(37,GPIO.OUT)
7 GPIO setwarnings(False)
8
9 try:
10     while True:
11         GPIO.output(35,True)
12         GPIO.output(37,False)
13         time.sleep(1)
14         GPIO.output(35,False)
15         GPIO.output(37,True)
16         time.sleep(1)
17 except KeyboardInterrupt:
18     GPIO.cleanup()

```

运行效果如图：

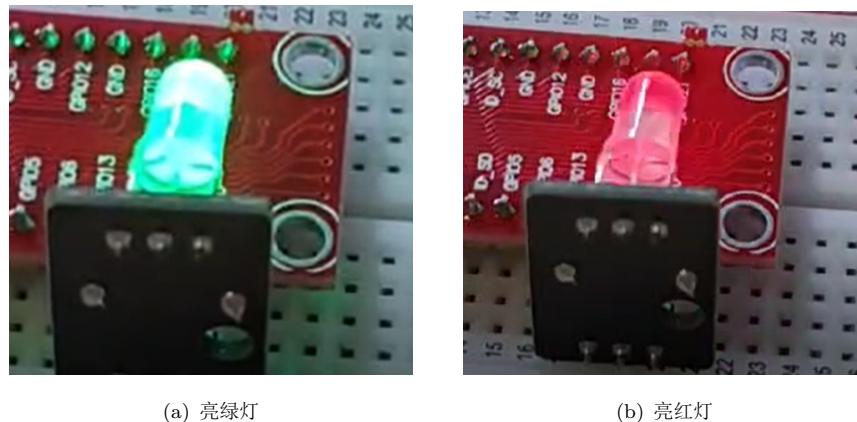


图 8: LED 灯自动切换亮灯程序运行效果

3 按键开关

3.1 按键开关控制

按键开关原理图如下：

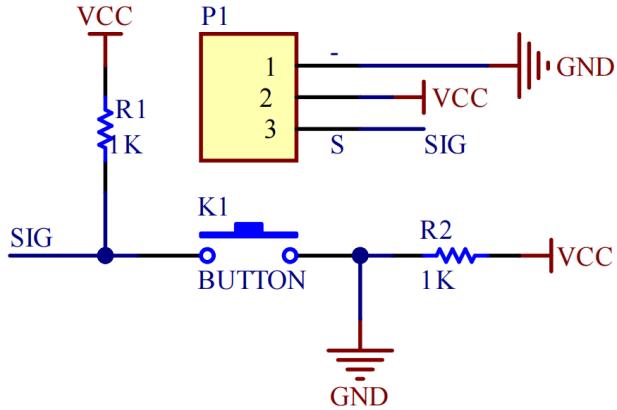


图 9: 按键开关原理图

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。

```

1 def check_input():
2     global count
3     # 按键检测&消抖
4     if GPIO.input(12) == 0:
5         time.sleep(0.1)
6         if GPIO.input(12) == 0:
7             while GPIO.input(12) == 0:
8                 time.sleep(0.001)
9             count += 1
10    print(count)
11    if count == 5:
12        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。

```

1 def react():
2     global count
3     while True:
4         # 状态响应
5         if count == 1:
6             GPIO.output(35, False)
7             GPIO.output(37, True)
8         elif count == 2:
9             GPIO.output(35, False)

```

```

10    GPIO.output(37, True)
11    time.sleep(0.5)
12    GPIO.output(37, False)
13    time.sleep(0.5)
14    elif count == 3:
15        GPIO.output(37, False)
16        GPIO.output(35, True)
17    elif count == 4:
18        GPIO.output(37, False)
19        GPIO.output(35, True)
20        time.sleep(0.5)
21        GPIO.output(35, False)
22        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。

```

1 try:
2     threading.Thread(target=react).start()
3     while True:
4         check_input()
5 except KeyboardInterrupt:
6     GPIO.cleanup()

```

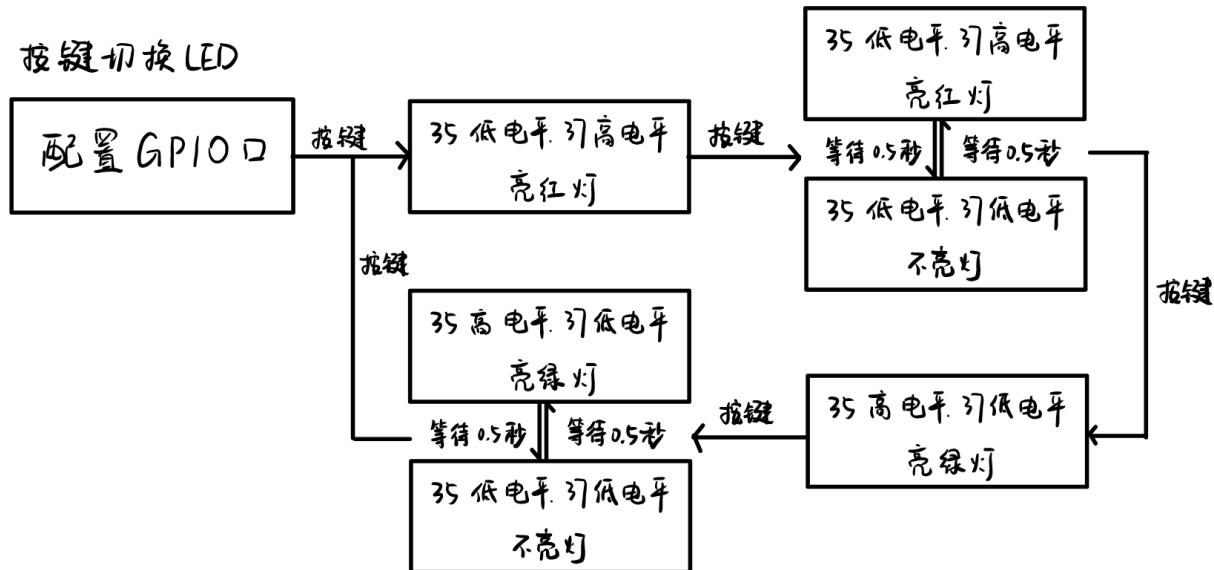


图 10: 按键控制 LED 程序流程图

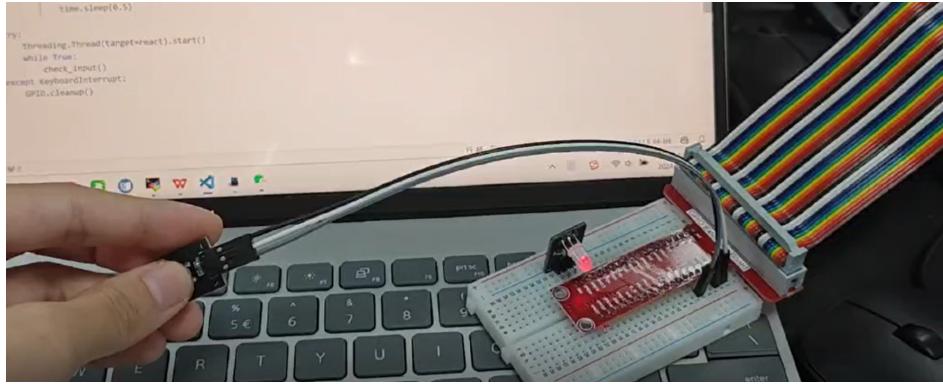


图 11: 按键控制 LED 亮灯模式切换运行效果

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。

```
1 import RPi.GPIO as GPIO
2
3 GPIO.setwarnings(False)
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(35, GPIO.OUT)
6 GPIO.setup(37, GPIO.OUT)
7 GPIO.setup(12,GPIO.IN)
8
9 try:
10     while True:
11         if GPIO.input(12) == 0:
12             GPIO.output(35,True)
13             GPIO.output(37,False)
14         else:
15             GPIO.output(35,False)
16             GPIO.output(37,True)
17     except KeyboardInterrupt:
18         GPIO.cleanup()
```

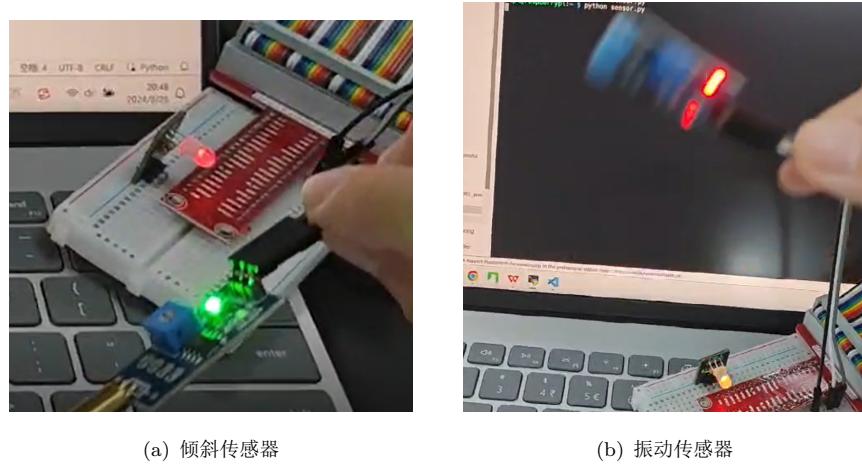


图 12: 多种传感器运行效果

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。

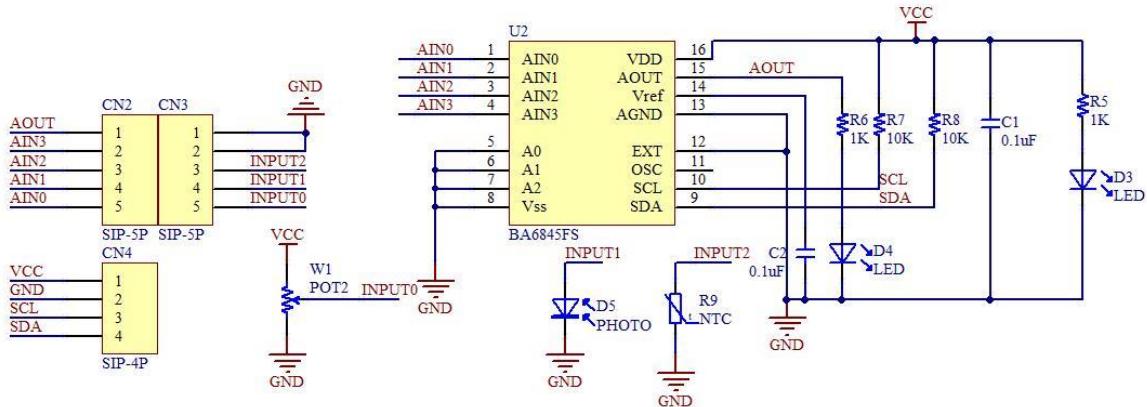


图 13: PCF8591 模数转换器原理图

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 AD 输入传感器数据

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。

```
1 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。

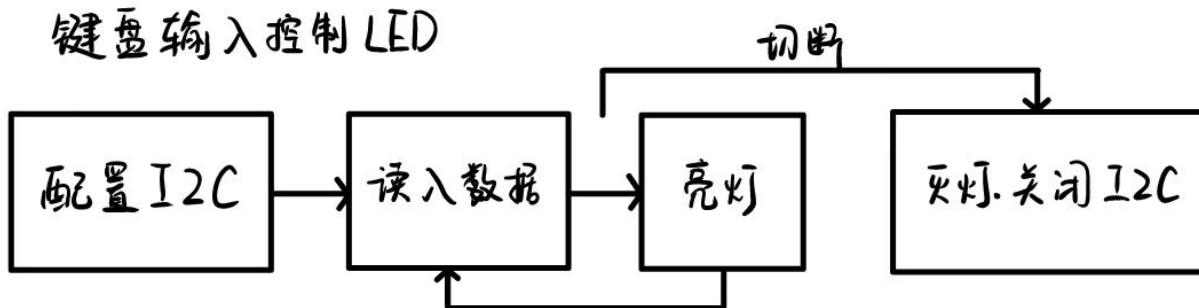


图 14: 键盘控制 LED 程序流程图

```
1 import smbus  
2  
3 bus = smbus.SMBus(1)  
4  
5 try:  
6     while True:  
7         bright = int(input())  
8         if 0 < bright < 255:  
9             bus.write_byte_data(0x48, 0x42, bright)  
10    except KeyboardInterrupt:
```

```
11 bus.write_byte_data(0x48, 0x42, 0)
12 bus.close()
```

4.4 DA 输出控制功能

4.5 AD-DA 组合功能

5 温度传感器

6 超声波传感器

7 蜂鸣器

8 操纵杆

9 红外遥控