学科实践

实验（二）TrafficLights

姓　　名	邓语苏
学　　号	22920212204066
日　　期	2024 年 3 月 7 日
学　　院	信息学院
课程名称	学科实践（四）

实验（二）TrafficLights

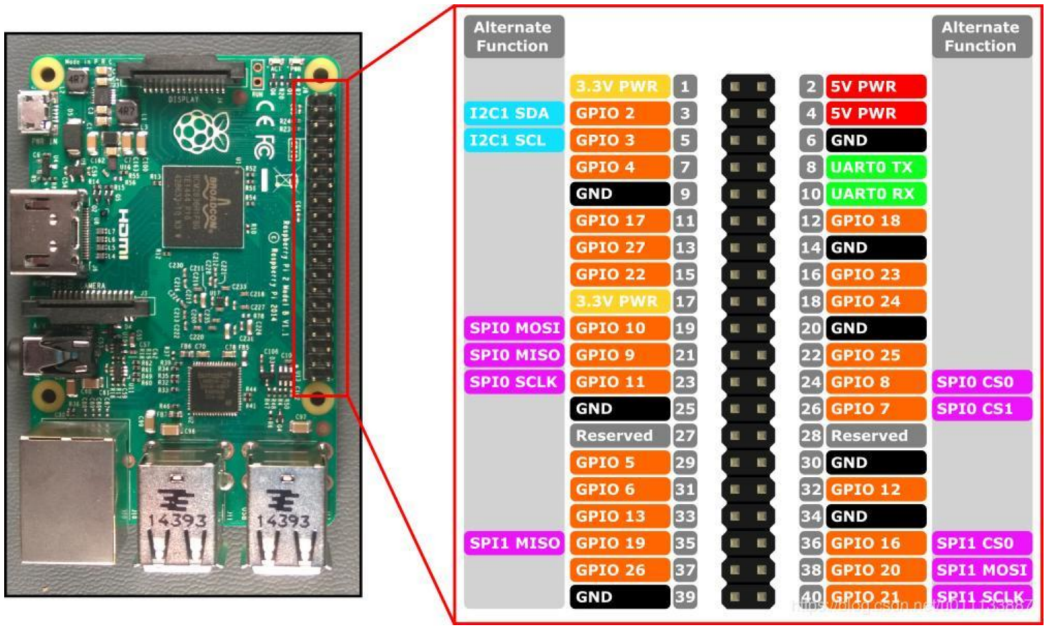
目录

1 Traffic-lights

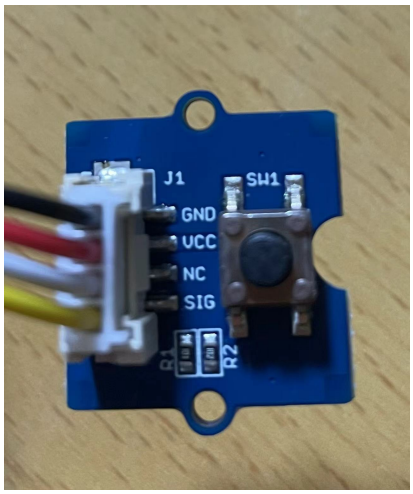
1.1 电路连接

【GPIO 引脚连线图】

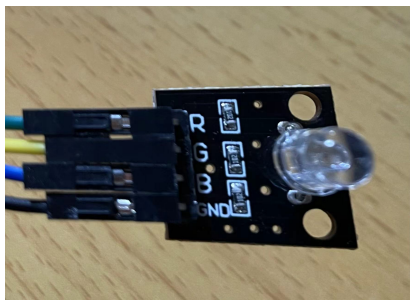
图 1: GPIO 引脚连线图



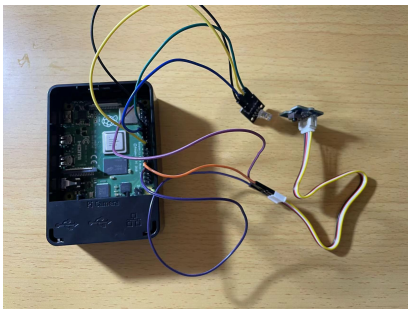
【元件实物图】



按钮



RGB 灯



树莓派连接按钮、RGB 灯

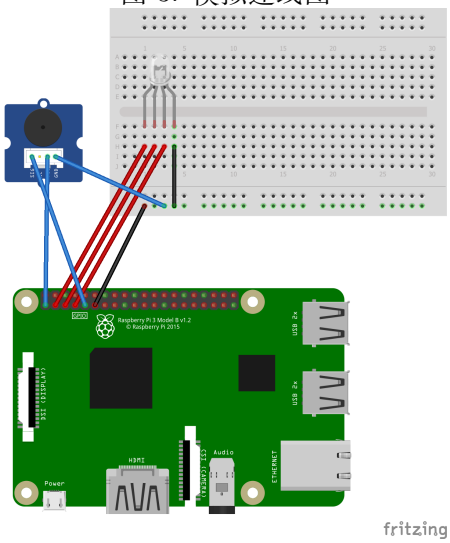
图 2: 元件实物图

- 图 1 为 Button，按钮的状态控制 SIG 引脚的电平值。
- 图 2 为 RGB 灯，有三种颜色：R-red、G-green、B-blue。三种颜色需由不同的 GPIO 引脚控制
- 图 3 为连接好按钮、RGB 灯的树莓派

【模拟连线图】

为了方便连接，RGB LED 灯的三个引脚接 GPIO 2、3、4，GND 接 GND。按钮的 SIG 引脚接 GPIO 17，GND 接 GND。

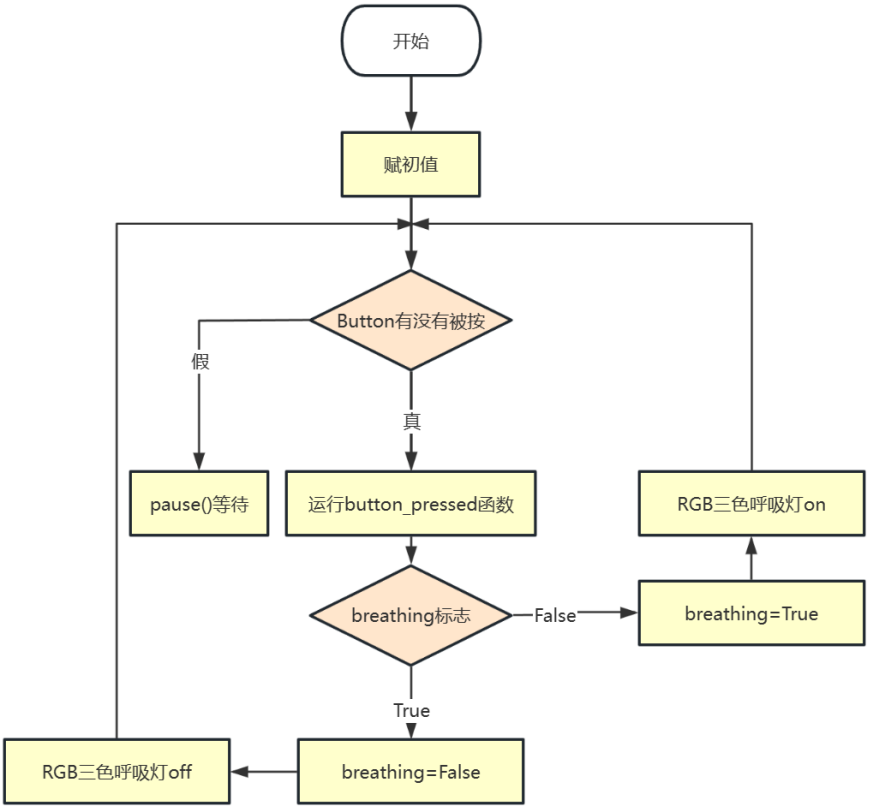
图 3: 模拟连线图



1.2 代码实现

1.2.1 流程图

图 4: 流程图



1.2.2 树莓派 python 环境配置

1. 在 Vscode 中使用 SSH 连接树莓派
2. 按 Ctrl+Shift+~ 调用 bash
3. 在命令行中输入 python -m venv myenv 创建虚拟环境，其中 myenv 是自定义的虚拟环境名称
4. 在命令行中输入 source myenv/bin/activate 激活虚拟环境
5. 在本次实验中需要安装以下包：

代码 1 安装包的指令

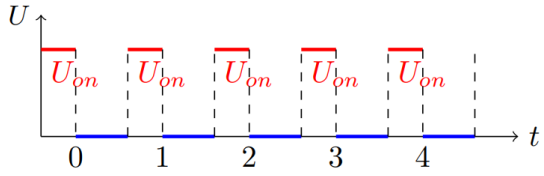
```
pip install gpiozero
pip install rpi-gpio
pip install lgpio
```

6. 在命令行中输入 deactivate 退出虚拟环境

1.2.3 三色呼吸灯实现

为了实现呼吸灯，首先需要控制灯的亮度实现渐变。TrafficLights 类提供参数 pwm，使用 PWM 调光实现亮度的渐变。PWM 调光的原理如下图所示。通过占空比来改变电子元件的功率。

图 5: PWM 原理图



因此可以写出代码如下，完成的功能是：在命令行输入 python breathe.py 后，RGB 灯以红、绿、蓝的颜色顺序循环呼吸，每个颜色呼吸 3 秒

代码 2 PWM 控制

```
# 呼吸灯效果函数
def breathe():
    while breathing:
        red_led.on()
        red_led.pulse()
        sleep(3)
        red_led.off()

        green_led.on()
        green_led.pulse()
        sleep(3)
        green_led.off()

        blue_led.on()
        blue_led.pulse()
        sleep(3)
        blue_led.off()
breathe()
```

1.2.4 三色呼吸灯 + 按钮控制

加入按钮后。设置了一个布尔类型的 breathing 作为标志。当每次按下 button 的时候，调用 `button_pressed` 函数，当此时 breathing 为 False，也就是当时没有在呼吸时，将 breathing 设为 True，并调用 breathe 函数开始呼吸；当此时 breathing 为 True，也就是当时在呼吸时，将 breathing 设为 False，此时 breathe 函数内部每 0.1 秒检查一次 breathing 的值，此时检查为 False 则停止呼吸。循环进行，直到下一次按按钮。

代码 3 加上按钮控制的三色呼吸灯 (关键代码)

```
# 是否处于呼吸状态的标志
breathing = False
# 按钮按下时的处理函数
def button_pressed():
    global breathing, breathe_thread
    print(breathing)
    # 创建并启动一个新线程来监视 breathing 变量的状态
    if breathing==False:
        # 如果不处于呼吸状态，则开始呼吸灯效果
        breathing = True
        # 创建并启动一个新线程来执行呼吸灯效果
        breathe_thread = Thread(target=breathe)
        breathe_thread.start()

    else:
        # 否则停止呼吸灯效果
        breathing = False
        red_led.off()
        green_led.off()
        blue_led.off()
# 呼吸灯效果函数
def breathe():
    while breathing:

        red_led.on()
        red_led.pulse()
        for i in range(30):
            sleep(0.1)
            if not breathing:
                return
        red_led.off()
        ...
# 创建按钮对象，代表控制按钮
button = Button(12,bounce_time=0.2)
while(True):
    # 当按键被按下时调用 button_pressed 函数
    button.when_pressed= button_pressed
    # 进入暂停模式，等待中断信号
    pause()
```

2 dweet_led.py

首先，代码 4中的 resolve_thing_name 函数负责为设备生成一个唯一的名字，如果以前没有生成过，就生成一个 UUID 写入相应的配置文件，否则就从文件中进行读取。、

代码 4 resolve_thing_name 函数

```
def resolve_thing_name(thing_file):
    """Get existing, or create a new thing name"""
    if os.path.exists(thing_file):                                    # (3)
        with open(thing_file, 'r') as file_handle:
            name = file_handle.read()
            logger.info('Thing name ' + name + ' loaded from ' + thing_file)
            return name.strip()
    else:
        name = str(uuid1())[:8]  # UUID object to string.             # (4)
        logger.info('Created new thing name ' + name)

        with open(thing_file, 'w') as f:                              # (5)
            f.write(name)

    return name
```

然后，代码 5 6中的 poll_dweets_forever 函数负责从 dweet.io 中每隔一段时间获取一次最新的请求，如果有请求，则调用 process_dweet 函数进行处理。然而这种方式并不是最高效的，因为轮询的方式会浪费大量的时间在等待上。因此，我们可以使用 dweet.io 提供的流式接口，即函数 stream_dweets_forever，它会一直等待新的请求，当有请求时，会立刻返回。利用流式的操作，既减少了请求次数，又使得代码更加简洁清晰。 最后，在程序的入口点，我们首先从 dweet.io 中获取最后的请求，然后根据其处理 LED

代码 5 获取请求 1

```
def poll_dweets_forever(delay_secs=2):
    """Poll dweet.io for dweets about our thing."""
    while True:
        dweet = get_latest_dweet()
        if dweet is not None:
            process_dweet(dweet)
            sleep(delay_secs)
```

灯的状态，然后调用 stream_dweets_forever 函数或者 poll_dweets_forever 函数，进入请求处理模式。

该程序实现的是根据 dweet.io 的请求来控制 LED 灯的状态，状态有三种：on、off 和 blink，分别对应灯亮、灯灭和灯闪烁。代码 7中的 process_dweet 函数负责根据请求的内容。

代码 6 获取请求 2

```
def stream_dweets_forever():
    resource = URL + '/listen/for/dweets/from/' + thing_name
    logger.info('Streaming dweets from url %s', resource)

    session = requests.Session()
    request = requests.Request("GET", resource).prepare()

    while True:  # while True to reconnect on any disconnections.
        try:
            response = session.send(request, stream=True, timeout=1000)

            for line in response.iter_content(chunk_size=None):
                if line:
                    try:
                        json_str = line.splitlines()[1]
                        json_str = json_str.decode('utf-8')
                        dweet = json.loads(eval(json_str))  # json_str is a string in a string.
                        logger.debug('Received a streamed dweet %s', dweet)

                        dweet_content = dweet['content']
                        process_dweet(dweet_content)
                    except Exception as e:
                        logger.error(e, exc_info=True)
                        logger.error('Failed to process and parse dweet json string %s', json_str)

        except requests.exceptions.RequestException as e:
            pass

        except Exception as e:
            logger.error(e, exc_info=True)
```

代码 7 处理请求

```
def process_dweet(dweet):
    """Inspect the dweet and set LED state accordingly"""
    global last_led_state

    if not 'state' in dweet:
        return
    led_state = dweet['state']
    if led_state == last_led_state:
        return  # LED is already in requested state.

    if led_state == 'on':
        led.on()
    elif led_state == 'blink':
        led.blink()
    else:  # Off, including any unhandled state.
        led_state = 'off'
        led.off()

    if led_state != last_led_state:
        last_led_state = led_state
        logger.info('LED ' + led_state)
```