

树莓派IO接口简介

2022/3/8

电子系统导论教学团队

实验目的

- 熟悉树莓派的GPIO
- 掌握Python对GPIO的调用方式
- 了解中断的概念和编程

GPIO介绍

■ GPIO – General Purpose I/O

“通用目的输入/输出端口”,是一种灵活的软件控制的端口。通俗地说,就是一些引脚,可以通过它们输出高低电平或者通过它们读入引脚的状态-是高电平或是低电平。在嵌入式系统中,经常需要控制许多结构简单的外部设备或者电路,使用传统的串口或者并口就显得比较复杂,而GPIO解决了这个问题。

■ 基于Python的树莓派GPIO开发方式

本课程的GPIO编程基于Rpi.GPIO库。Rpi.GPIO是一个小型的python库,非常简单好用,但是暂时还没有支持SPI、I2C或者1-wire等总线接口。

如果希望进行I2C的开发,请安装smbus库和I2C tools。

如果希望进行SPI开发,请安装spi-dev库。

如果希望进行串口开发,请安装PySerial库。

具体的I2C,SPI以及串口的开发方式会在后续章节介绍。

树莓派4B GPIO引脚图

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40



树莓派4B

树莓派4B 引脚简介

■ 固定输入输出引脚

- Pin 1,17 (DC 3.3V)
- Pin 2,4 (DC 5V)
- Pin 6,9,14,20,25,30,34,39 (GND)

■ ID EEPROM

- Pin 27,28 (用于和拓展树莓派功能的附加电路板通信)

■ 带有高级功能的GPIO引脚

- Pin 3,5 (I2C)
- Pin 8,10 (UART)
- Pin 19,21,23,24,26 (SPI)

■ 普通GPIO引脚

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

树莓派4B 引脚简介

■ 固定输入输出引脚

- Pin 1,17 (DC 3.3V)
- Pin 2,4 (DC 5V)
- Pin 6,9,14,20,25,30,34,39 (GND)

■ 这些引脚不需要任何的配置和设定，可以直接使用

Pin#	NAME		NAME	Pin#
01	3.3v DC Power	■	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	■	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	■	Ground	06
07	GPIO04 (GPIO_GCLK)	■	(TXD0) GPIO14	08
09	Ground	■	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	■	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	■	Ground	14
15	GPIO22 (GPIO_GEN3)	■	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	■	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	■	Ground	20
21	GPIO09 (SPI_MISO)	■	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	■	(SPI_CE0_N) GPIO08	24
25	Ground	■	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	■	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	■	Ground	30
31	GPIO06	■	GPIO12	32
33	GPIO13	■	Ground	34
35	GPIO19	■	GPIO16	36
37	GPIO26	■	GPIO20	38
39	Ground	■	GPIO21	40

树莓派4B 引脚简介

■ ID EEPROM

- Pin 27,28 (用于和拓展树莓派功能的附加电路板通信)

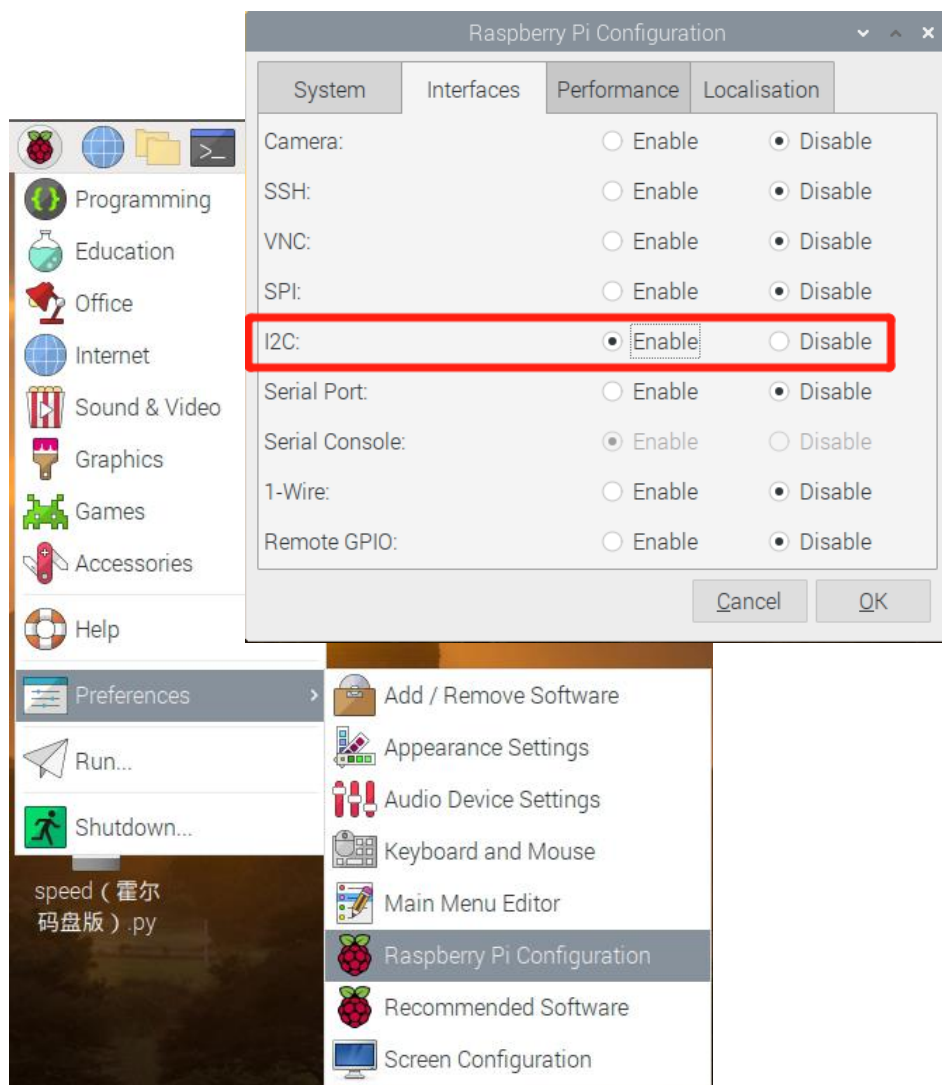
- 如果树莓派现有的IO接口不能满足你的需求，那么你可以利用这两个IO口，在你的树莓派上插上现成的扩展板(Hardware At Top)，或者自行设计扩展板。

- 更多资料请参阅

<https://github.com/raspberrypi/hats>

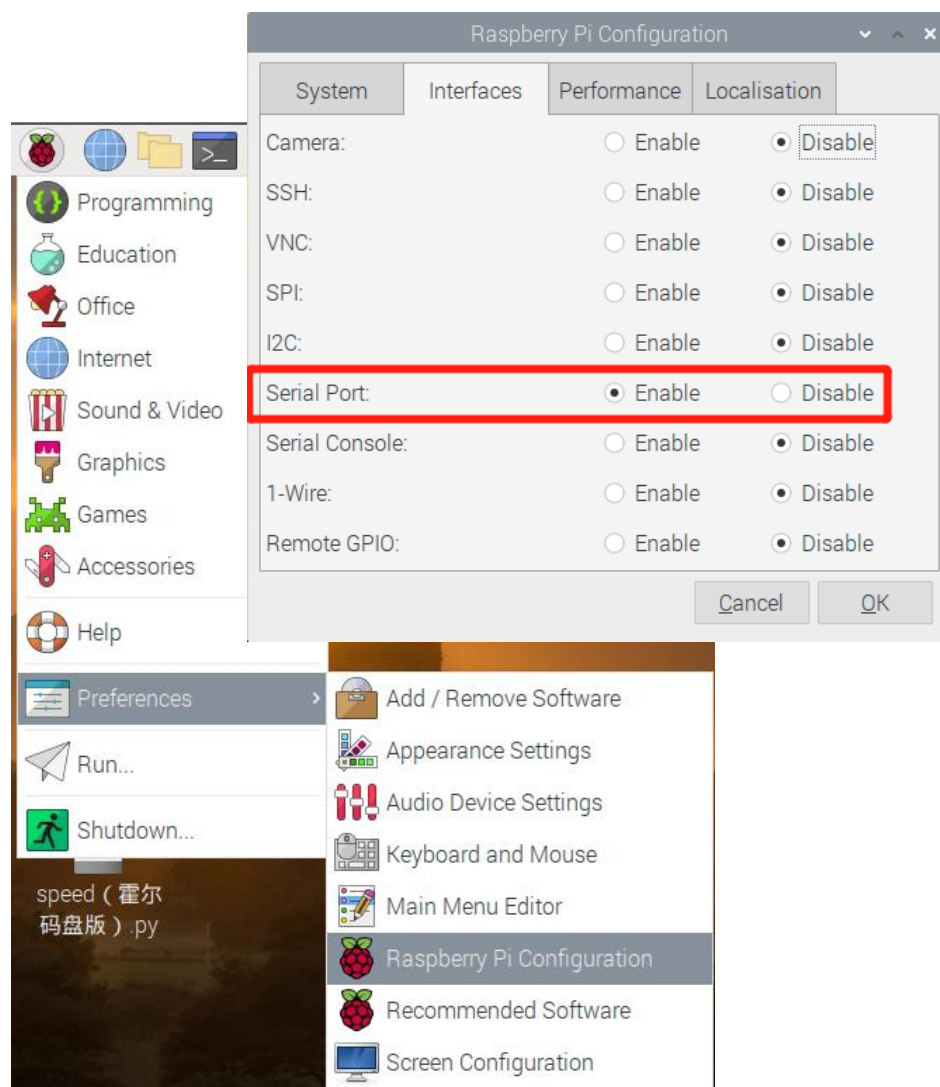
Pin#	NAME		NAME	Pin#
01	3.3v DC Power	■	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	●	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	●	Ground	06
07	GPIO04 (GPIO_GCLK)	●	(TXD0) GPIO14	08
09	Ground	●	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	●	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	●	Ground	14
15	GPIO22 (GPIO_GEN3)	●	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	●	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	●	Ground	20
21	GPIO09 (SPI_MISO)	●	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	●	(SPI_CE0_N) GPIO08	24
25	Ground	●	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	●	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	●	Ground	30
31	GPIO06	●	GPIO12	32
33	GPIO13	●	Ground	34
35	GPIO19	●	GPIO16	36
37	GPIO26	●	GPIO20	38
39	Ground	●	GPIO21	40

I2C功能开启办法（PIN3,5）



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

UART功能开启办法（PIN8,10）



Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

SPI功能开启方法(PIN19,21,23,24,26)

The image shows the Raspberry Pi Configuration tool with the 'Interfaces' tab selected. The 'SPI' option is highlighted with a red box and set to 'Enable'. Below it, the 'I2C' option is also set to 'Disable'. The 'Cancel' and 'OK' buttons are visible at the bottom of the configuration window.

Below the configuration window, a menu is open showing various system settings, including 'Add / Remove Software', 'Appearance Settings', 'Audio Device Settings', 'Keyboard and Mouse', 'Main Menu Editor', 'Raspberry Pi Configuration', 'Recommended Software', and 'Screen Configuration'.

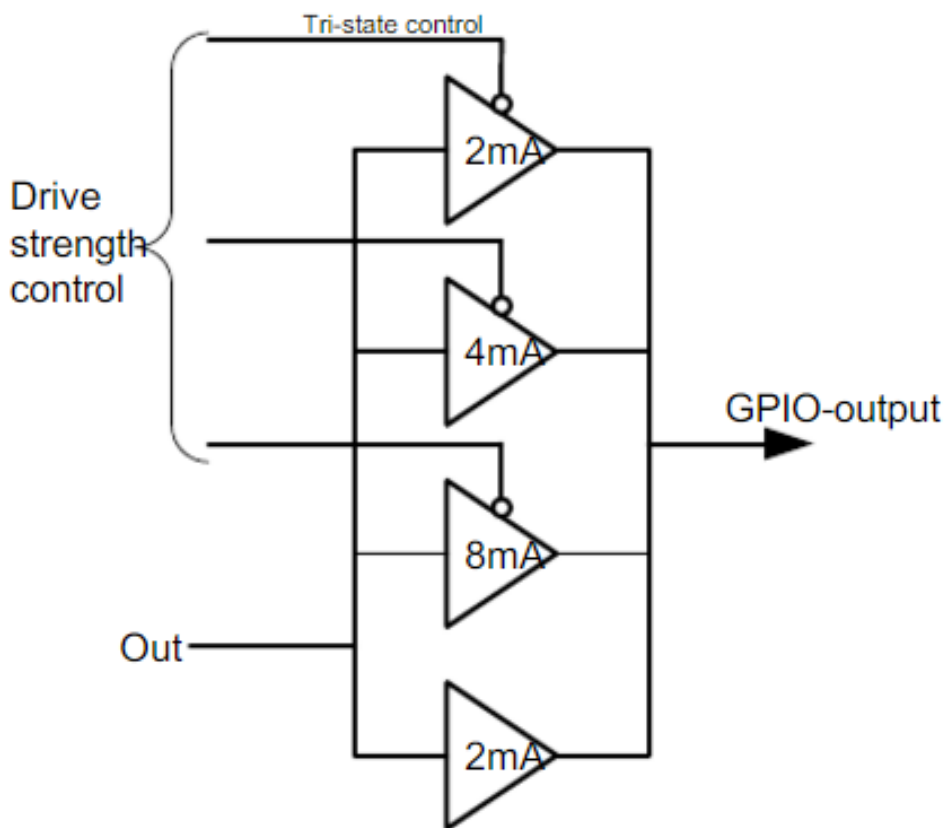
To the right, a table lists the GPIO pins and their functions, with a central diagram of the Raspberry Pi pin header showing the physical connections for pins 19, 21, 23, 24, and 26.

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

树莓派GPIO使用注意事项

- 给树莓派供电的电源不得高过5V (4.75V~5.25V)
- 施加在任何GPIO引脚上的电压不要超过3.3V
- 每个名字为GPIO#的引脚输出的电流不得超过16mA, 总输出电流不得超过51mA
- 3.3V引脚的输出电流应该小于0.05A, 5V引脚的输出电流应该小于0.2A
- 给树莓派通电后, 不要用金属物体接触GPIO接口
- 建议不要使用树莓派直接驱动外部电路, 特别是电机和继电器等含有线圈的模块, 如需直接驱动, 建议加入续流二极管和光耦进行隔离保护

树莓派GPIO驱动能力

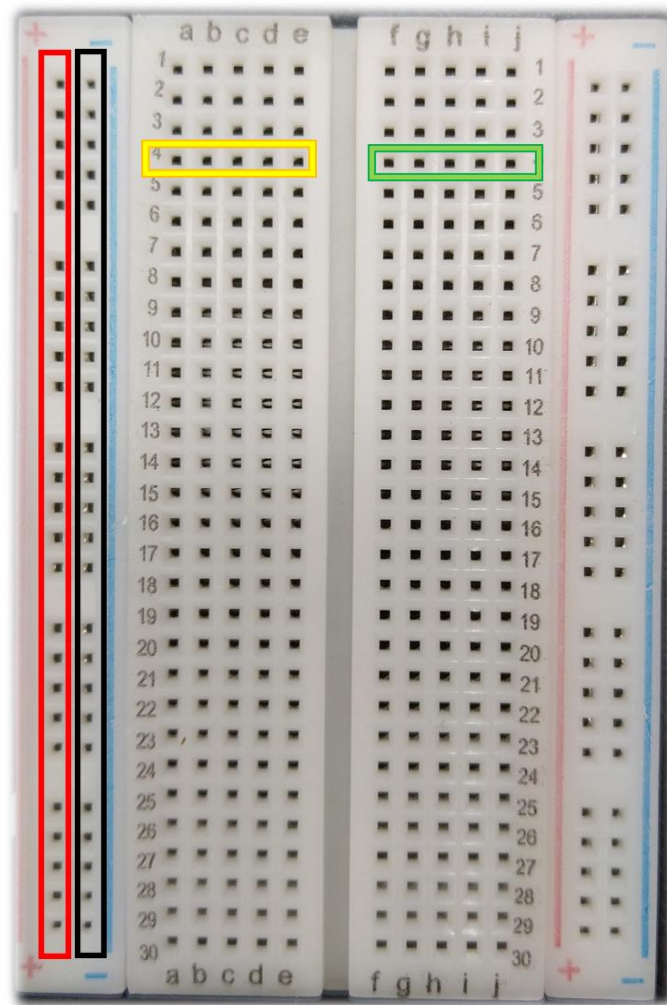


GPIO输出部分内部结构示意图

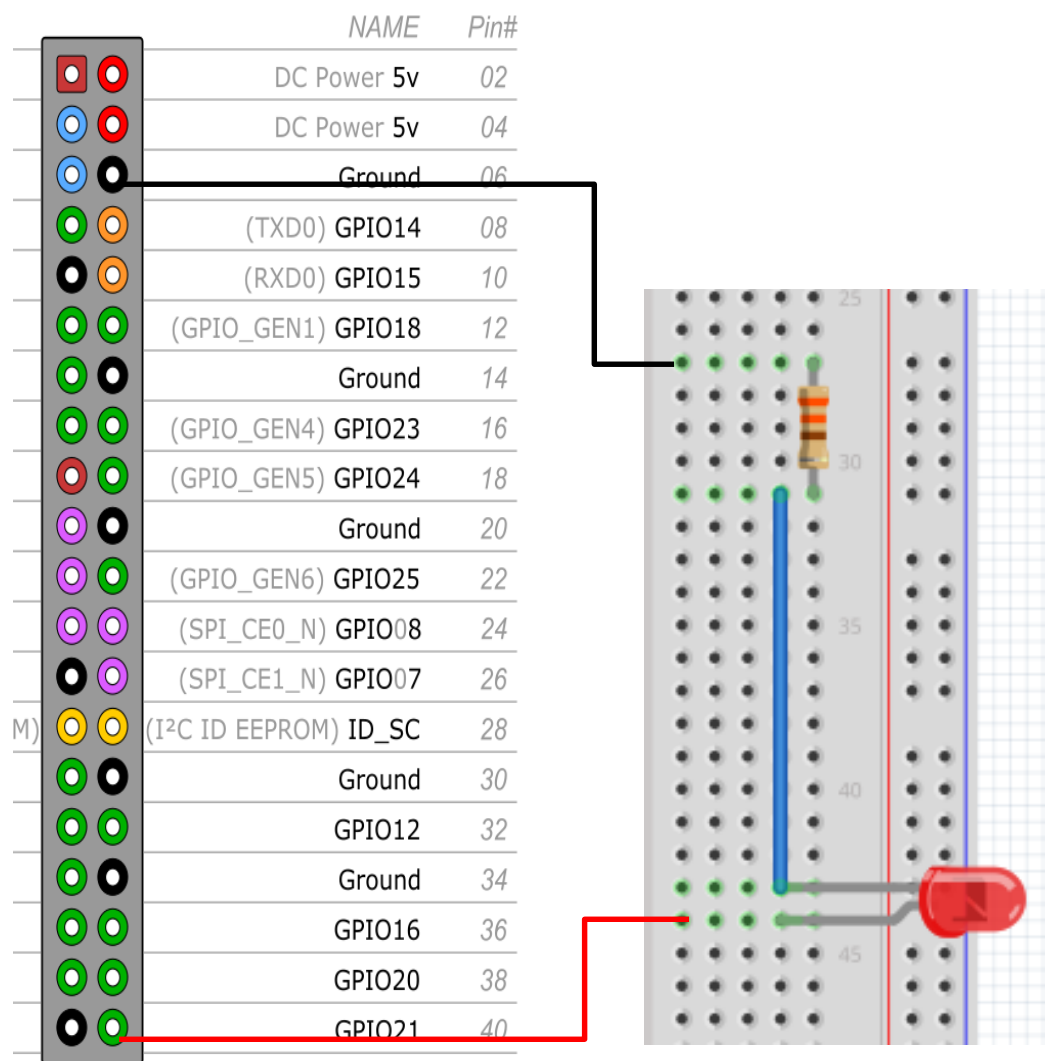
- 树莓派每个GPIO引脚都有一个寄存器可以配置引脚的驱动强度，在保证输出电平质量和保证板子安全的情况下可提供2mA~16mA 的电流，默认值是<3mA。
- 每个GPIO采用CMOS电平
逻辑0：低电平 $\leq 0.3V$
逻辑1：高电平 $\geq 1.8V$

面包板

- 中间每排5个孔相连，通常连接用于器件接口。
- 两侧每竖列相通，通常用于电源和地。



树莓派GPIO驱动LED

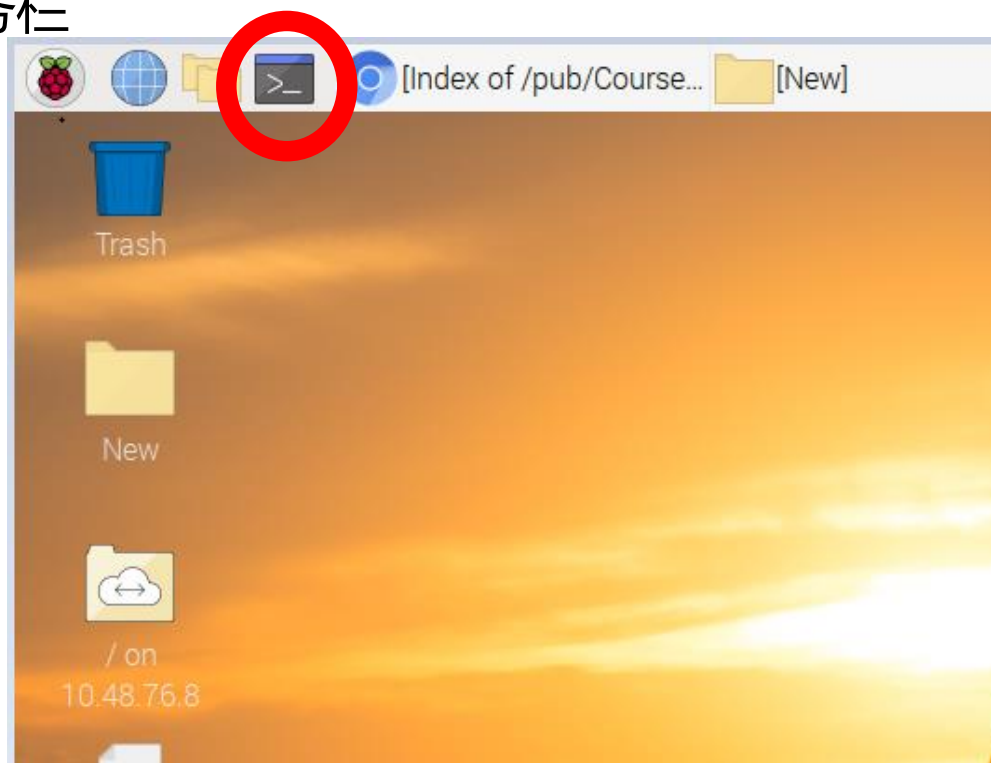


- 在GPIO21和地线之间接入一个LED灯，还有一个用于防止短路的470欧电阻。红色直插LED的导通电压降为2.1V左右。
$$I = (3.3 - 2.2) \text{ V} / 470 \Omega \approx 2 \text{ mA}$$
- 当GPIO21位于高电平时，将有电流通过电路，从而点亮LED灯。

树莓派GPIO驱动LED (shell方法)

■ 终端打开方法

- 快捷键 Ctrl+Alt+T
- 桌面任务栏



树莓派GPIO驱动LED (shell方法)

- 初始化GPIO21

```
echo 21 > /sys/class/gpio/export
```

执行后，/sys/class/gpio/下面增加了代表GPIO21的一个目录，目录名就是gpio21。

- 下一步，把GPIO21置于输出状态：

```
echo out > /sys/class/gpio/gpio21/direction
```

- 最后，向GPIO21写入1，从而让PIN处于高电压：

```
echo 1 > /sys/class/gpio/gpio21/value
```

- 使用完毕GPIO21，可以删除该端口：

```
echo 21 > /sys/class/gpio/unexport
```

- 我们用shell命令来控制GPIO21。在Linux中，外部设备经常被表示成文件。向文件写入或读取字符，就相当于向设备输出或者从设备输入。树莓派上的GPIO端口也是如此，其代表文件位于/sys/class/gpio/下。
■ 表示空格

树莓派GPIO驱动LED (Python方法)

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led = 21
GPIO.setup(led, GPIO.OUT)
print("输出高电平")
GPIO.output(led, GPIO.HIGH)
time.sleep(5)
GPIO.output(led, GPIO.LOW)
print("输出低电平")
GPIO.cleanup()
```

- 通常Rpi.GPIO已经包含在树莓派系统中。
关于rpi.gpio的详细介绍, 参看
<https://pypi.python.org/pypi/RPi.GPIO>
或者是爱好者提供的中文手册
https://gitee.com/null_693_8693/RPi.GPIO-use-introduction/attach_files









注意: Rpi.GPIO操作频率最高能达到~400KHz!

树莓派GPIO驱动LED (Python方法)

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led = 21
GPIO.setup(led, GPIO.OUT)
print("输出高电平")
GPIO.output(led, GPIO.HIGH)
time.sleep(5)
GPIO.output(led, GPIO.LOW)
print("输出低电平")
GPIO.cleanup()
```

- 首先调用 *GPIO.setmode* 函数来确定引脚的模式。

在 *RPi.GPIO* 包中定义GPIO针脚的模式：BCM模式和BOARD模式。

11	GPIO17 (GPIO_GEN0)	
13	GPIO27 (GPIO_GEN2)	
15	GPIO22 (GPIO_GEN3)	
17	3.3v DC Power	
19	GPIO10 (SPI_MOSI)	
21	GPIO09 (SPI_MISO)	
23	GPIO11 (SPI_CLK)	
25	Ground	

- 以11号脚为例，当在BOARD模式下时，11号脚即编号11的引脚，也就是GPIO17；当在BCM模式下时，11号脚对应GPIO11，也就是23号引脚。

树莓派GPIO驱动LED (Python方法)

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led = 21
GPIO.setup(led, GPIO.OUT)
print("输出高电平")
GPIO.output(led, GPIO.HIGH)
time.sleep(5)
GPIO.output(led, GPIO.LOW)
print("输出低电平")
GPIO.cleanup()
```

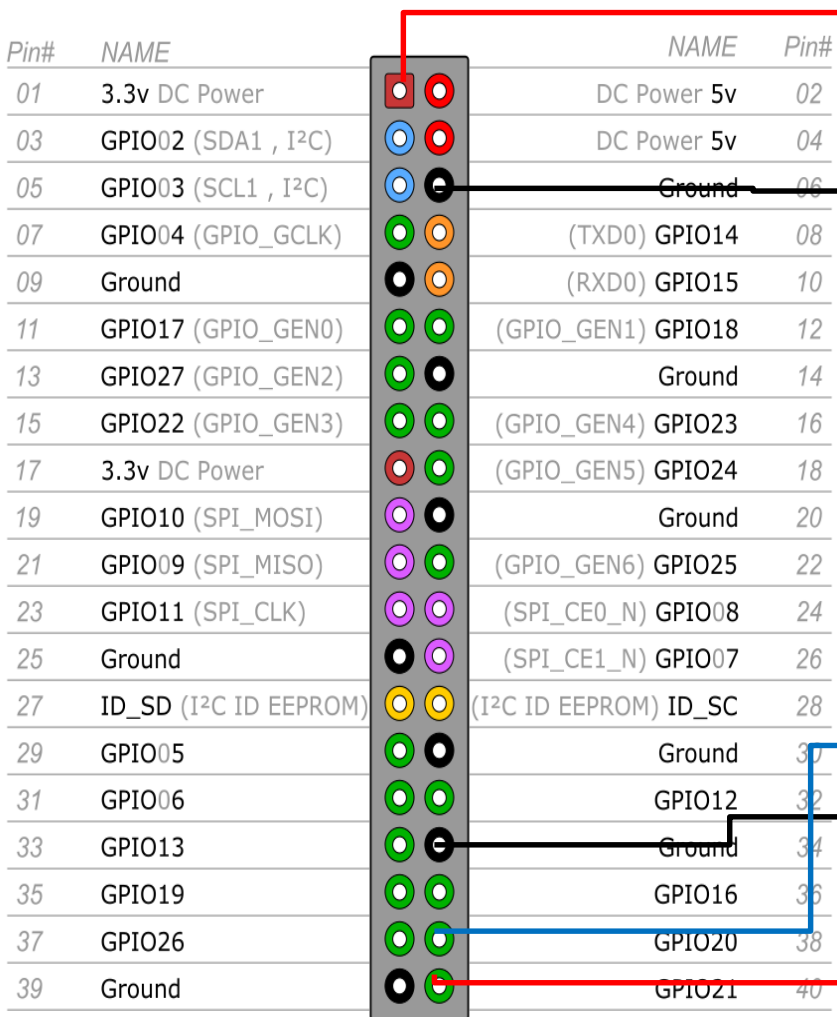
■ 关于 *GPIO.cleanup()* 函数

在任何程序结束后，请养成清理用过的资源的好习惯。

此函数的功能是清除掉之前 *GPIO.setup()* 设置的状态，恢复所有使用过的GPIO状态为输入，避免由于短路意外损坏树莓派。

注意，该操作仅会清理你的代码使用过的GPIO通道。退出程序之前一定要调用，否则下次调用该GPIO的时候会报错。

GPIO按键控制LED (轮询方法)

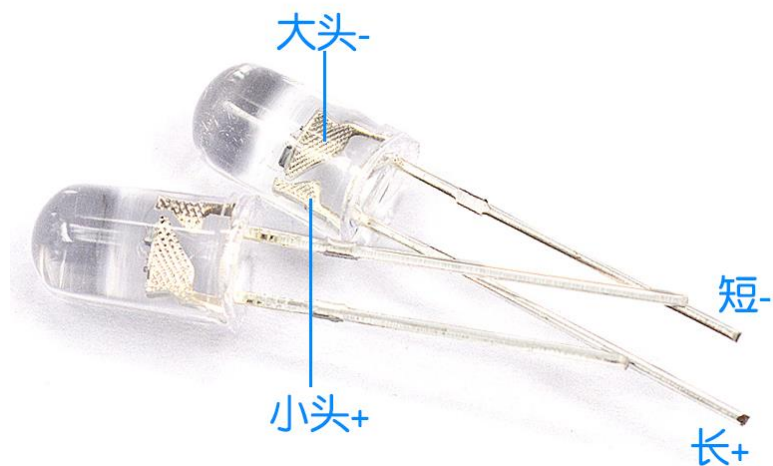


- 在GPIO驱动LED灯的电路基础上，加入一个按键开关，以及一个 $10k\Omega$ 电阻。

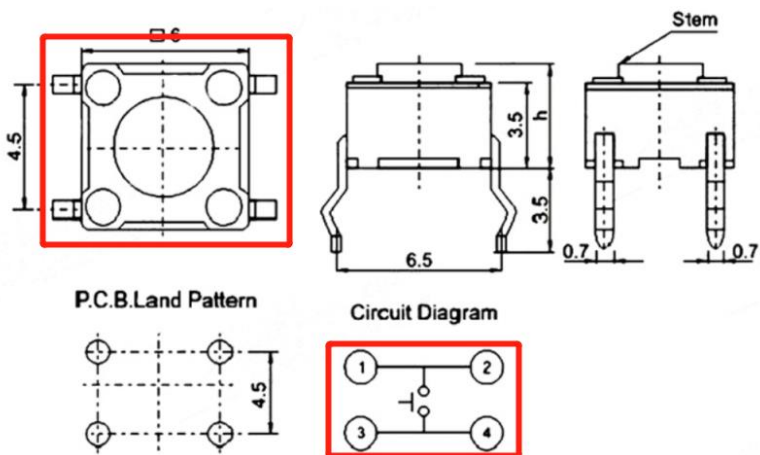
用GPIO20监测按键是否被按下，从而控制LED灯状态。

- 当按键没有按下时，GPIO20上的电平被上拉到3.3V，当按键按下时，GPIO20接地。

LED 和按键开关的管脚图



- 发光二极管LED灯有正负之分
正常一个没被修剪过的LED
长脚为正极，短脚为负极
- 按键开关要注意引脚的方向，
引脚在左右两边时，按键按下
1与3连通，2与4连通。



GPIO按键控制LED (轮询方法)

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led=21
bt=20
GPIO.setup(led,GPIO.OUT)
GPIO.setup(bt,GPIO.IN,pull_up_down=GPIO.PUD_UP)
ledStatus=False
n=1
try:
    while True:
        time.sleep(0.01)
        if(GPIO.input(bt)==GPIO.LOW):
            time.sleep(0.03)
            if(GPIO.input(bt)==GPIO.HIGH):
                print('button pressed',n)
                n=n+1
                ledStatus=not ledStatus
                if ledStatus:
                    GPIO.output(led,GPIO.HIGH)
                else:
                    GPIO.output(led,GPIO.LOW)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

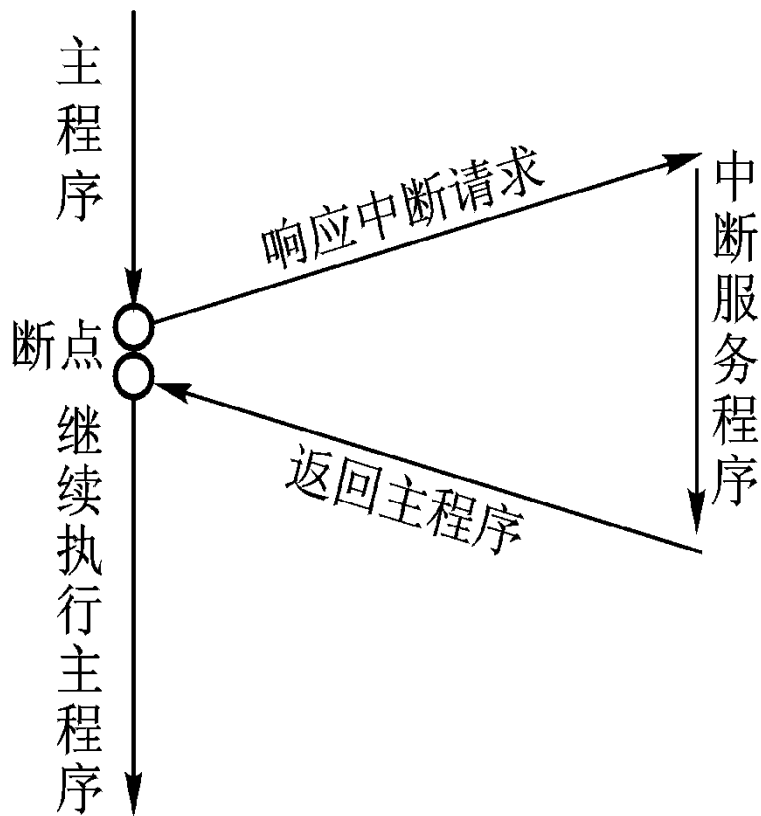
- 如果需要实时监控引脚的状态变化，可以有两种办法。最简单原始的方式是每隔一段时间检查输入的信号值，这种方式被称为轮询。
如果你的程序读取的时机错误，则很可能会丢失输入信号。轮询是在循环中执行的，这种方式比较占用处理器资源。
- 打个简单的比方的话，你一边在宿舍做作业，一边等外卖送到，你每做一会儿作业就拿手机看一次外卖送到了没有，如果送到了，就执行去拿外卖的任务。这种方式会对你做作业的效率造成一定的影响。

GPIO按键控制LED (轮询方法)

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led=21
bt=20
GPIO.setup(led,GPIO.OUT)
GPIO.setup(bt,GPIO.IN,pull_up_down=GPIO.PUD_UP)
ledStatus=False
n=1
try:
    while True:
        time.sleep(0.01)
        if(GPIO.input(bt)==GPIO.LOW):
            time.sleep(0.03)
            if(GPIO.input(bt)==GPIO.HIGH):
                print('button pressed',n)
                n=n+1
                ledStatus=not ledStatus
                if ledStatus:
                    GPIO.output(led,GPIO.HIGH)
                else:
                    GPIO.output(led,GPIO.LOW)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

- try/except 语句用来检测try 语句块中的错误，让except语句捕获异常信息并处理。使用except KeyboardInterrupt ,使我们能够用Ctrl + C的方法中止While循环，执行cleanup清理GPIO.setup()的配置后退出。
- *GPIO.setup(bt,GPIO.IN,...)* 是对输入引脚的配置。如果输入引脚处于悬空状态，引脚的值将是漂动的，其读取到的值是未知的，由于干扰的影响，输入的值可能会反复的变化，所以根据电路，这里预先配置为PUD_UP

GPIO按键控制LED (中断方法)

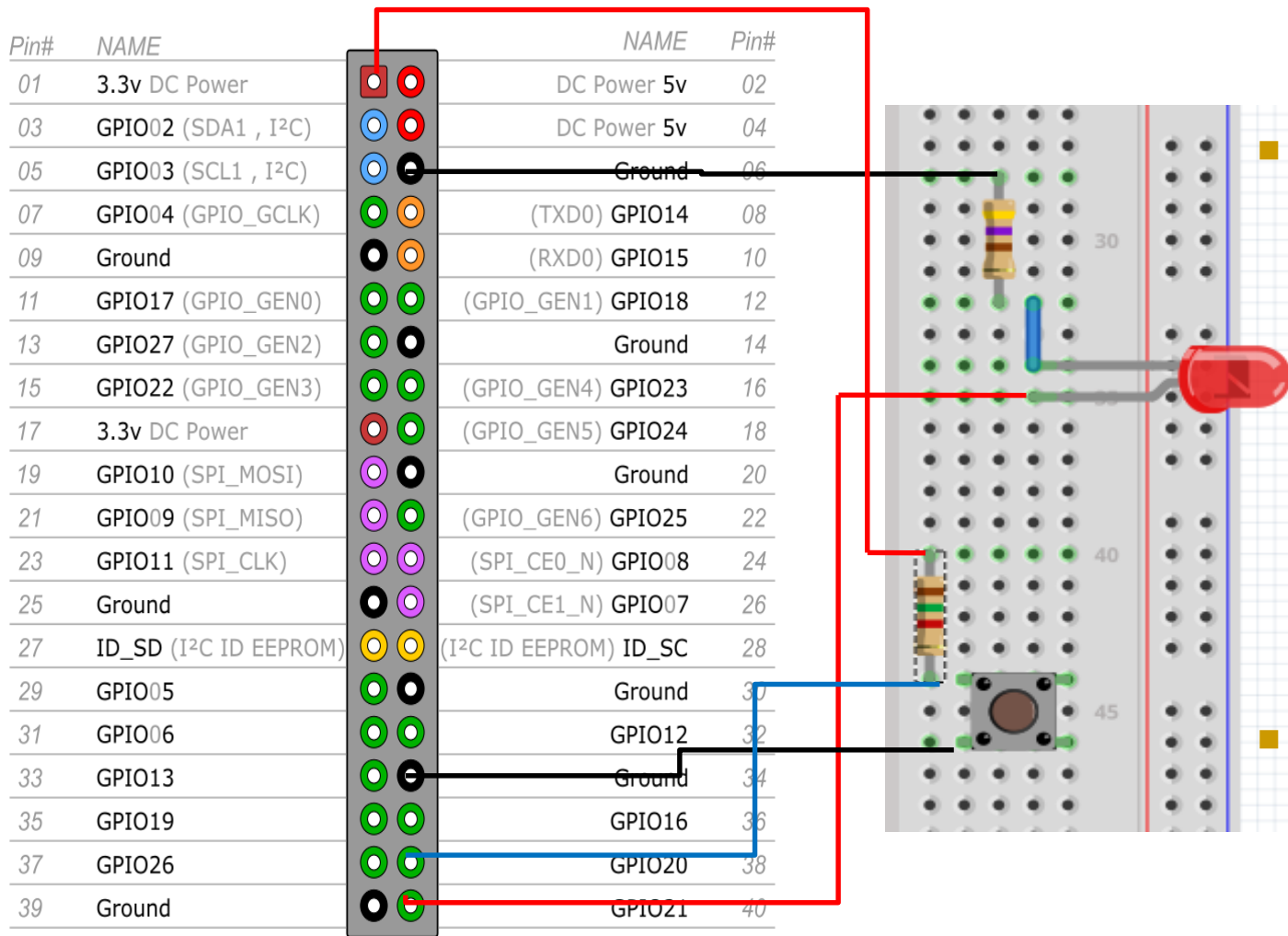


■ 中断的概念

CPU在运行目前任务时，当有特别事件发生时，CPU应当暂停正在执行的程序，转向执行处理该事件的子程序；事件处理完毕后，恢复原来的状态，再继续执行原来的程序。这种对这些事件的处理模式，称为程序中断。

举个生活化的例子，现在你在宿舍做作业，突然外卖送到楼下了（中断源），于是你下楼取外卖（中断服务程序），然后再上楼做作业（继续执行主程序）。

GPIO按键控制LED (中断方法)



■ 另一种响应GPIO输入的方式是使用中断（边缘检测），这里的边缘是指信号从高到低的变换（下降沿）或从低到高的变换（上升沿）。。

■ 依旧使用之前的硬件电路

GPIO按键控制LED (中断方法)

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led = 21
bt = 20
GPIO.setup(led, GPIO.OUT)
GPIO.setup(bt,GPIO.IN,pull_up_down =
GPIO.PUD_UP)
ledStatus = True
def my_callback(channel):
    print ("button pressed")
    global ledStatus
    ledStatus = not ledStatus
    if ledStatus:
        GPIO.output(led, GPIO.HIGH)
    else:
        GPIO.output(led,GPIO.LOW)

GPIO.add_event_detect(bt,GPIO.FALLING,call
back = my_callback,bouncetime=200)
try:
    while True:
        print("I LOVE Raspberry Pi")
        time.sleep(2)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

- *GPIO.add_event_detect(bt, GPIO.FALLING, callback=my_callback)* 给bt引脚添加一个事件函数，触发条件是：捕获到上升沿（GPIO.RISING）。这个参数还可以是：GPIO.FALLING（下降沿）、GPIO.BOTH（两者都有）。
- 对于中断模式的检测按键，想给这种程序添加去抖程序的话，只要在 *GPIO.add_event_detect()* 中加入 *bouncetime* 参数，*bouncetime* 的单位是ms

GPIO按键控制LED (中断方法)

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
led = 21
bt = 20
GPIO.setup(led, GPIO.OUT)
GPIO.setup(bt,GPIO.IN,pull_up_down =
GPIO.PUD_UP)
ledStatus = True
def my_callback(channel):
    print ("button pressed")
    global ledStatus
    ledStatus = not ledStatus
    if ledStatus:
        GPIO.output(led, GPIO.HIGH)
    else:
        GPIO.output(led,GPIO.LOW)

GPIO.add_event_detect(bt,GPIO.FALLING,call
back = my_callback,bouncetime=200)
try:
    while True:
        print("I LOVE Raspberry Pi")
        time.sleep(2)
except KeyboardInterrupt:
    pass
GPIO.cleanup()
```

- my_callback是我们自己定义的一个回调函数。RPi.GPIO库会为回调函数另外开启一个线程。这意味着这回调函数可以和你的主程序同时运行，及时的对边沿做出响应。
- 回调函数和普通库函数有何区别？打个简单的比方，某旅馆提供叫醒服务，但要求旅客自己决定叫醒的方法。可以是派服务员去敲门，也可以要求往自己头上浇盆水。这里“叫醒”这个行为是旅馆提供的，但是叫醒的方式是由旅客定义并告诉旅馆的，这就是回调函数的特点。

树莓派GPIO中断优先级

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup([24,26], GPIO.IN, GPIO.PUD_UP)
def my_callback_one(channel):
    print('--my_callback_one start--')
    for i in range(0, 100000):
        pass
    print('--my_callback_one end--')
def my_callback_two(channel):
    print('--my_callback_two start--')
    for i in range(0, 100000):
        pass
    print('--my_callback_two end--')
GPIO.add_event_detect(24, GPIO.FALLING,
my_callback_one, bouncetime=200)
GPIO.add_event_detect(26, GPIO.FALLING,
my_callback_two, bouncetime=200)
while True:
    time.sleep(1)
GPIO.cleanup()
```

- 经过对树莓派多个GPIO的测试，示例代码为对PIN24和PIN26的测试，把24和26引脚连在一起，同时给下降沿，发现它们虽然都被触发了，但还是会先执行一个，执行完后再执行下一个，而不会出现嵌套现象。
- 再尝试先给24引脚一个下降沿，当24引脚的中断被触发，开始执行回调函数时但还没有退出回调的时候，马上给26引脚一个下降沿。发现依旧要等24脚的回调函数执行完后才会执行26脚的。
- 根据BCM2711手册，树莓派不存在默认的优先中断，但可通过FIQ寄存器配置，详见
https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf

参考资料

1. 树莓派4B型原理图、官方文档, <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>
2. GPIO Pads Control2, <https://zh.scribd.com/doc/101830961/GPIO-Pads-Control2>
3. 树莓派GPIO输出电流限制, <http://spicliff.blogspot.jp/2016/08/impedance-cpu17gpio-gpio-impedance.html>
4. rpi.gpio官方文档, <https://pypi.python.org/pypi/RPi.GPIO>
5. rpi.gpio爱好者中文手册, https://gitee.com/null_693_8693/RPi.GPIO-use-introduction/attach_files
6. bcm2711datasheet, https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf

实验内容

1. 搭建硬件，用Shell命令通过GPIO控制LED亮灭。
 2. 用Python编程通过GPIO控制LED亮灭。
 3. 轮询和中断方式实现按键控制LED亮灭。
- 完成每个实验后需要由教师或者助教验收。

实验报告中需要回答的问题

1. 轮询式和中断式IO的主要区别是什么？
2. 如果实验中发现按键无法控制LED，请论述如何分步排查故障。

致谢

- 本课件由以下同学协助编写
 - 高晓航(13307130023)