树莓派基础外设 - GPIO

主讲人：曹家英

第一部分：树莓派底层硬件

- 1. 树莓派介绍
- 2. 树莓派环境搭建
- 3. 树莓派与命令行
- 4. 相关知识扩展
- 5. 树莓派基础外设操作
- 6. 树莓派Buildroot SDK环境搭建

课程概览

- 实战3——C语言 LED & Button
- 什么是PWM
- 实战4——Python3 呼吸灯
- 实战5——Python3 舵机控制
- 实战6——wiringPi 呼吸灯

实战3——C语言 LED & Button

实战3——C语言 LED & Button

回顾：

1 初始化：

wiringPiSetup()　　：初始化wiringPi使用wiringPi管脚定义

wiringPiSetupGpio()　　：初始化wiringPi使用BCM管脚定义

wiringPiSetupPhys ()　　：初始化wiringPi使用HW管脚定义（BOARD）

2 核心函数：

void pinMode(int pin, int mode);

配置管脚工作模式。

pin为管角号码，根据初始化的模式选择管角编码。

mode为工作模式：INPUT，OUTPUT，PWM_OUTPUT，GPIO_CLOCK。

需要注意的是仅有管脚1（BCM_GPIO 18）支持PWM_OUTPUT模式，仅有管脚7（BCM_GPIO 4）支持CLOCK输出模式。

void pullUpDnControl(int pin, int pud);

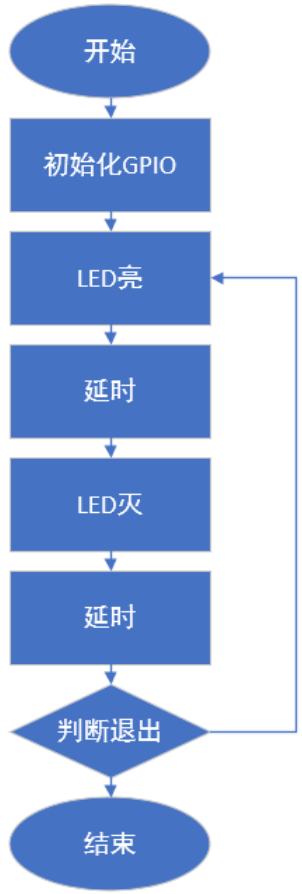
配置管脚的上下拉电阻。

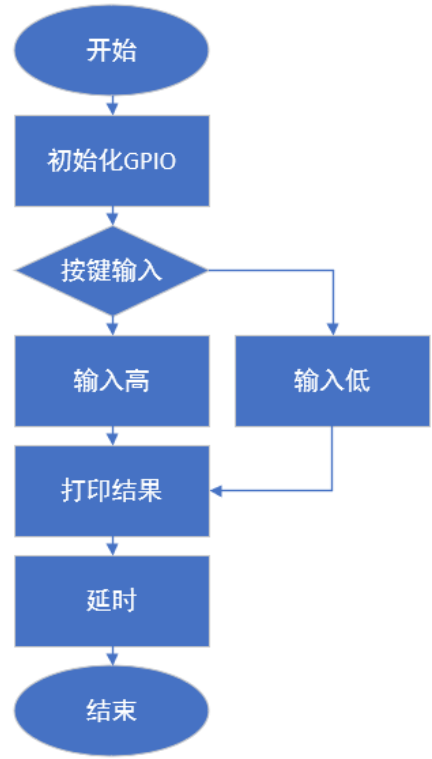
pin同上。

pud为上下拉电阻模式：PUD_OFF，PUD_DOWN，PUD_UP。

内部上拉和下拉电阻有接近50KΩ。

该函数在Sys模式下无作用。如果你需要激活上拉或下拉电阻的话，在启动程序前，可以通过在脚本中调用gpio命令来实现。





实战3——C语言 LED & Button

void digitalWrite(int pin, int value);

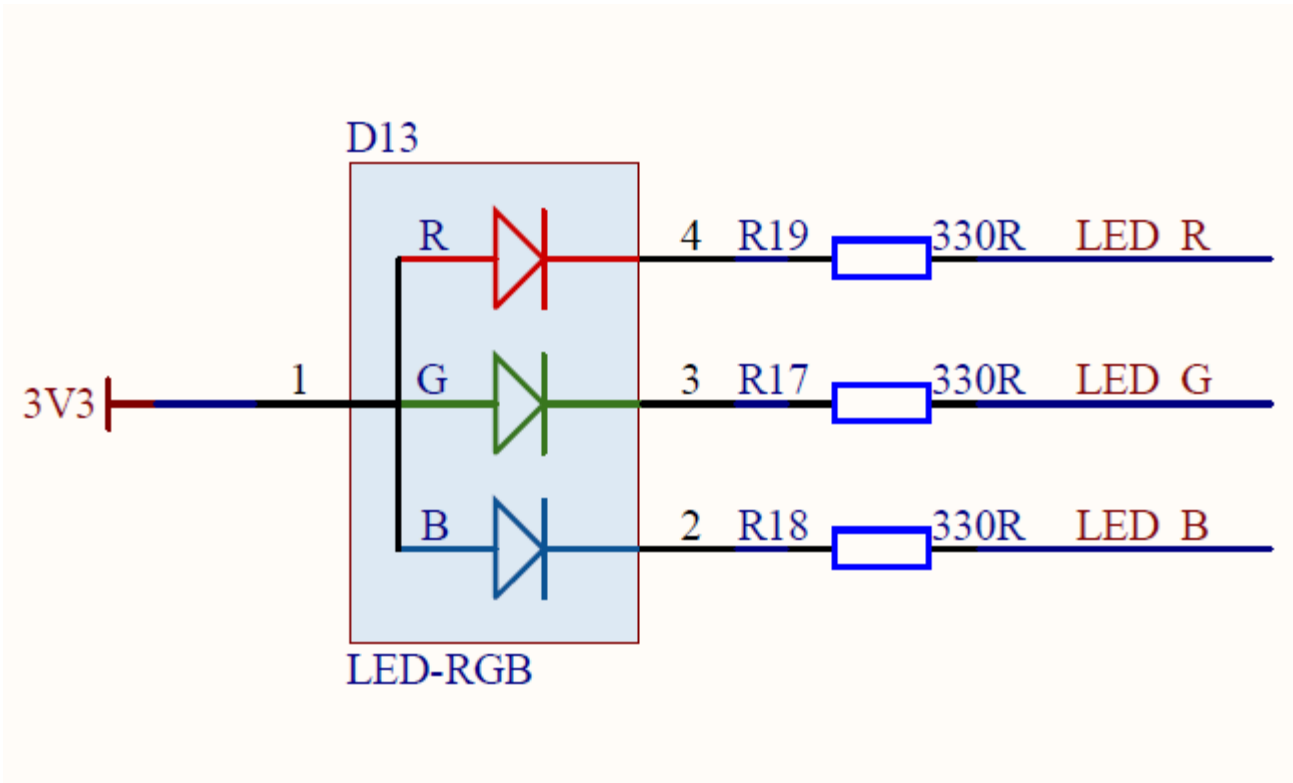
配置管脚的输出值，需要优先将管角设置为OUTPUT。

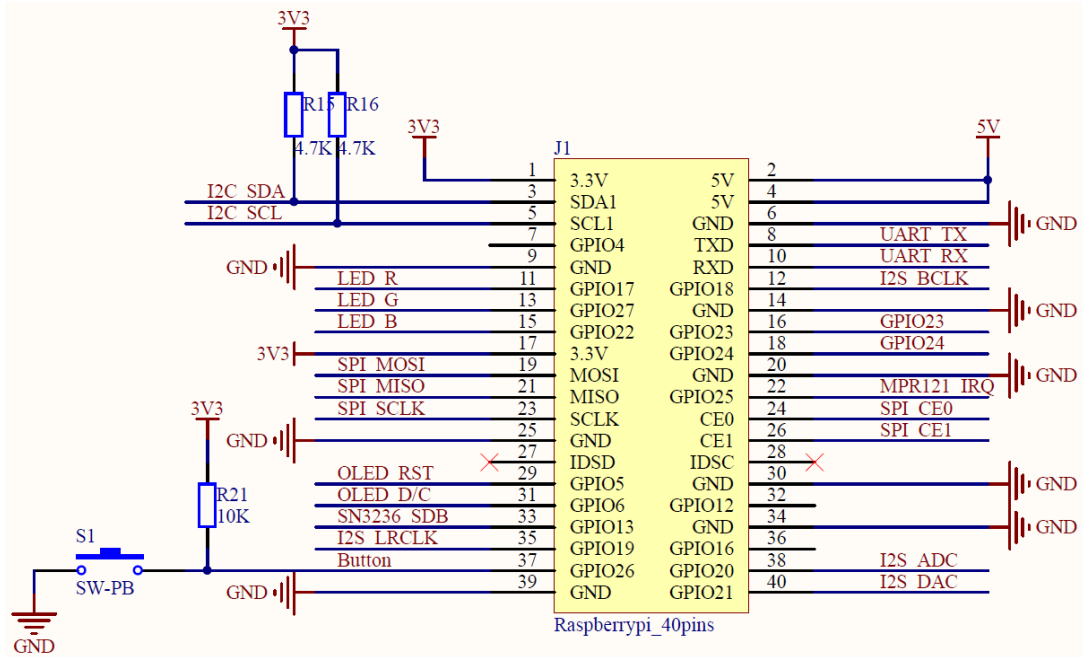
pin同上。

value：可配置HIGH（高）或者LOW（低）

void digitalRead(int pin);

读取管脚上的电平值。





实战3——C语言 LED & Button

LED Test Code

```c
#include <wiringPi.h>
#include <stdio.h>


int main(void) {

    wiringPiSetupGpio() ;    //使用BCM编码GPIO序号
    //wiringPiSetup();       //使用WiringPi编码GPIO序号
    pinMode(17,OUTPUT) ;
    for(;;) {
        digitalWrite(17,HIGH) ;
        delay(500) ;
        digitalWrite(17,LOW) ;
        delay(500) ;
    }
    return 0;
}
```

Button Test Code

```c
#include <wiringPi.h>
#include <stdio.h>

int main(void) {
    int Button_value;
    wiringPiSetupGpio();
    pinMode(26, INPUT);
    pullUpDnControl(26, PUD_UP);
    for (;;) {
        Button_value = digitalRead(26);
        if (Button_value) {
            printf("No trigger!\n");
        }
        else {
            printf("The Button is trigger!\n");
        }
        delay(500);
    }
    return 0;
}
```

什么是PWM

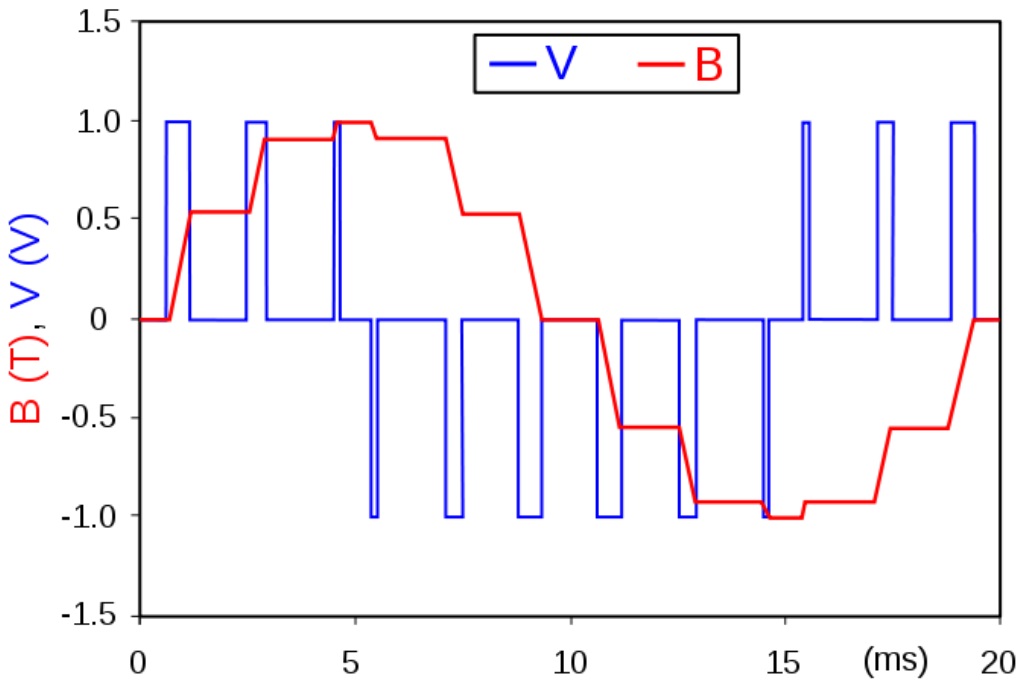
PWM——定义

脉冲宽度调制：Pulse Width Modulation，缩写：PWM，简称脉宽调制，是将模拟信号变换为脉冲的一种技术，一般变换后脉冲的周期固定，但脉冲的占空比会依模拟信号的大小而改变。

在模拟电路中，模拟信号由电压和电流控制，但是非常容易受到干扰，而数字信号相对于模拟信号有更强的抗干扰能力，通过使用高频率调制方波的占空比，从而实现对一个模拟信号的电平进行编码，所以PWM信号经常用来传输模拟量的数值，并在信号传输行业得到大量应用。

PWM的使用场景非常丰富，比如LED调光，电机调速，控制舵机，调制解调。

占空比定义：在一串理想的脉冲序列中（如方波），代表1的正脉冲的持续时间与脉冲总周期的比值。例如，脉冲宽度1μs，信号周期4μs的脉冲序列占空比为0.25。



PWM——原理

原理 [编辑]

脉冲宽度调制使用一个脉冲宽度会被调制的方波，使得波型的平均值会有所变化。如果我们考虑一个周期为 T 的脉冲波 $f(t)$，低值 $y_{\min}$，高值为 $y_{\max}$，跟一个工作循环D(duty cycle)，(参照右图)，此波的平均值为：

$$\bar{y} = \frac{1}{T}\int_0^T f(t)\,dt.$$

当 $f(t)$ 是一个脉冲波，它的值在 $0 < t < D \cdot T$ 是 $y_{\max}$，而在 $D \cdot T < t < T$ 是 $y_{\min}$，上式的描述可以变为：

$$\begin{aligned}\bar{y} &= \frac{1}{T}\left(\int_0^{DT} y_{\max}\,dt + \int_{DT}^{T} y_{\min}\,dt\right)\\ &= \frac{1}{T}\left(D\cdot T\cdot y_{\max} + T\left(1-D\right)y_{\min}\right)\\ &= D\cdot y_{\max} + \left(1-D\right)y_{\min}.\end{aligned}$$

以上表示可以在很多状况下被简化，当 $y_{\min} = 0$ 及 $\bar{y} = D \cdot y_{\max}$。从这是可以看出，波型的平均值非常明显地直接与工作循环之值D有关。

最简单可以产生一个脉冲宽度调制信号的方式是交集性方法(intersective method)，这个方法只需要使用锯齿波或三角波(可以简单地使用震荡器来产生)，以及一个比较器。当参考的信号值(图二的红色波)比锯齿波(图二的蓝色波)来的大，则脉冲调制后的结果会在高状态，反之，则在低状态。

微分调制 [编辑]

以微分调制作为控制脉冲宽度调制的方法，输出信号将会被积分，同时结果也会被拿来与参考信号增减一个偏移量(作为比较的边界)比较。当每一次的积分结果到达边界时，脉冲调制信号便会转变状态如图三。

积分-微分调制 [编辑]

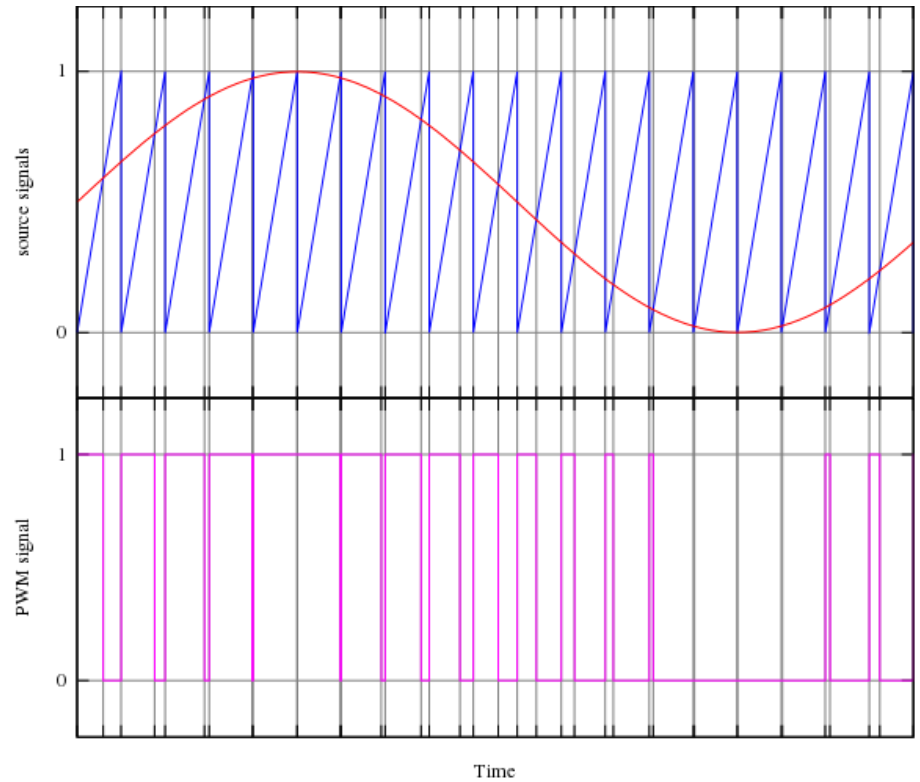
主条目：ΔΣ调制

以积分-微分调制作为控制脉冲宽度调制的方法，参考信号与输出信号会相减得到误差信号。 同时此误差会被积分，若积分超过边界，输出结果便会变换状态(参考图四)。

空间向量调制 [编辑]

主条目：空间向量调制

空间向量调制是一种针对多相位交流信号，控制脉冲宽度调制的算法，先将参考信号正常的采样，接着对于每一次的样本信号，会有一些在参考向量相邻的非零交换向量以及一至多个的零交换向量作为采样之代表，目的是合成出参考向量。



https://zh.wikipedia.org/wiki/%E8%84%88%E8%A1%9D%E5%AF%AC%E5%BA%A6%E8%AA%BF%E8%AE%8A

摩尔吧 MOORE 8

树莓派Python控制PWM

p = GPIO.PWM(channel, frequency)

将GPIO配置为输出PWM的模式

channel：GPIO的编号

frequency：PWM信号频率

p.start(dc)

开始启动PWM信号输出

dc：开始的占空比（0.0 <= dc <= 100.0）

p.ChangeFrequency(freq)

修改当前对象p的PWM信号频率

freq：要修改的频率

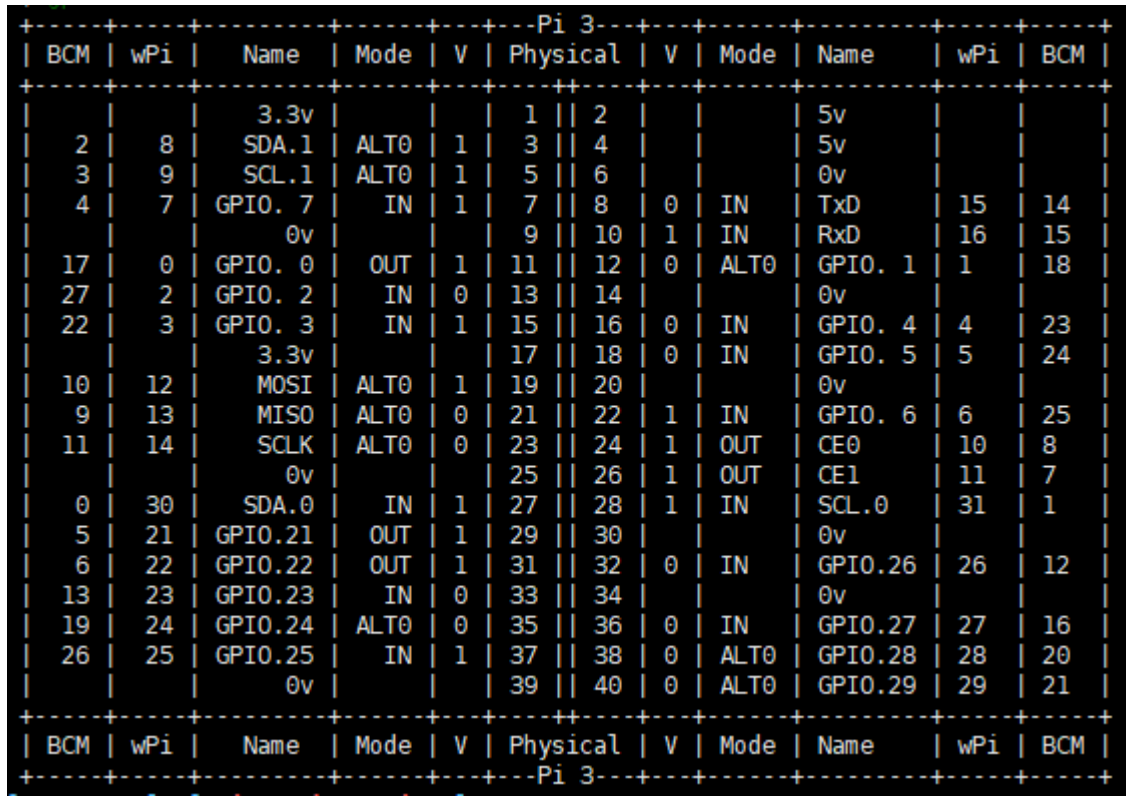
p.ChangeDutyCycle(dc)

修改当前对象p的占空比

dc：要修改的占空比数值

p.stop()

停止PWM



BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
					Pi 3					
		3.3v			1 || 2			5v		
2	8	SDA.1	ALT0	1	3 || 4			5v		
3	9	SCL.1	ALT0	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	0	IN	TxD	15	14
		0v			9 || 10	1	IN	RxD	16	15
17	0	GPIO. 0	OUT	1	11 || 12	0	ALT0	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	1	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	0	IN	GPIO. 5	5	24
10	12	MOSI	ALT0	1	19 || 20			0v		
9	13	MISO	ALT0	0	21 || 22	1	IN	GPIO. 6	6	25
11	14	SCLK	ALT0	0	23 || 24	1	OUT	CE0	10	8
		0v			25 || 26	1	OUT	CE1	11	7
0	30	SDA.0	IN	1	27 || 28	1	IN	SCL.0	31	1
5	21	GPIO.21	OUT	1	29 || 30			0v		
6	22	GPIO.22	OUT	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	ALT0	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	1	37 || 38	0	ALT0	GPIO.28	28	20
		0v			39 || 40	0	ALT0	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
					Pi 3					

树莓派wiringPi控制PWM

void pwmSetMode(int mode);

配置PWM输出模式，

mode：模式选择，PWM_MODE_BAL，WM_MODE_MS

void pwmSetRange(unsigned int range);

该函数用来设置PWM发生器的范围寄存器。

默认值1024

void pwmSetClock(int divisor);

该函数用来设置PWM时钟的分频值。

void pwmWrite(int pin, int value);

使用该函数可以将值写入指定管脚的PWM寄存器中。

树莓派板上仅有一个PWM管脚，即wiringPi管脚1（BCM_GPIO 18，物理管脚号为12）。可以配置的值为0~1024。

当在Sys模式时，该函数不可以控制树莓派的板上PWM。

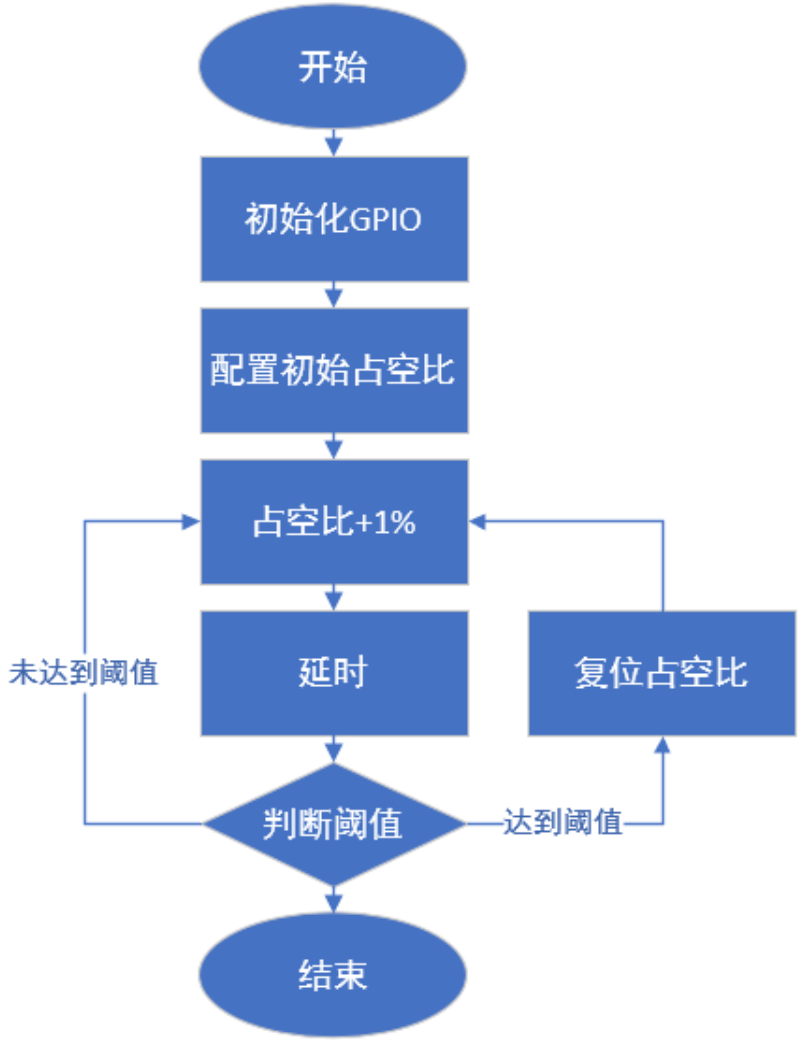
实战4——Python3 呼吸灯

实战4——Python3 呼吸灯

PWM信号的占空比，表现在LED灯的现象就是LED灯的亮度变化，通过不断的调整PWM占空比，实现LED有灭倒亮的变化。

使用到的方法：

```
p = GPIO.PWM(channel, frequency)
p.start(dc)
p.ChangeFrequency(freq)
p.ChangeDutyCycle(dc)
p.stop()
```



实战4——Python3 呼吸灯

PWM控制LED实现呼吸灯效果:

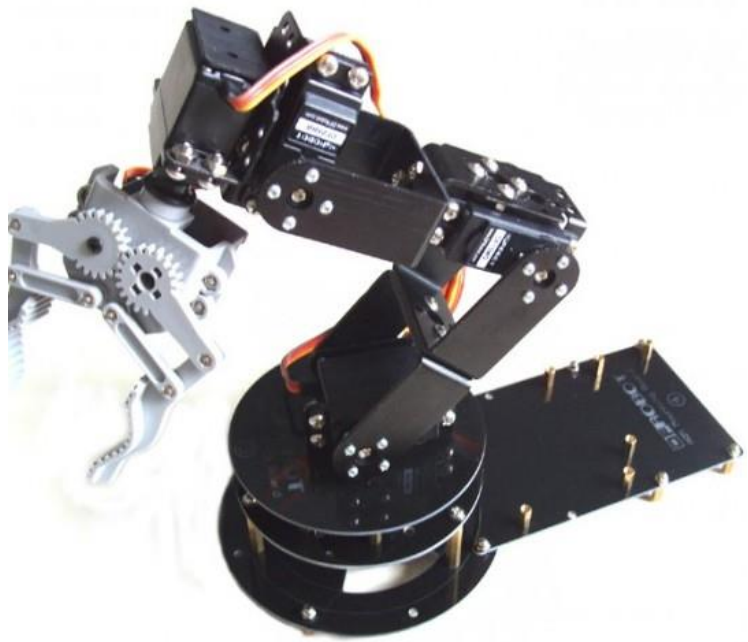
```python
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)
p = GPIO.PWM(17,1000)
p.start(0)

i = 0
while(1) :
    i = i + 1
    time.sleep(0.015)
    if(i == 100) :
        i = 0
    p.ChangeDutyCycle(i)

```

实战5——Python3 舵机控制

舵机介绍

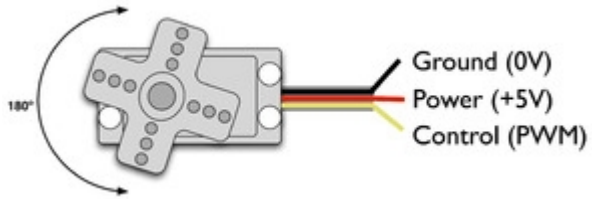
舵机：是一种位置（角度）伺服的驱动器，适用于那些需要角度不断变化并可以保持的控制系统。目前，在高档遥控玩具，如飞机、潜艇模型，遥控机器人中已经得到了普遍应用。

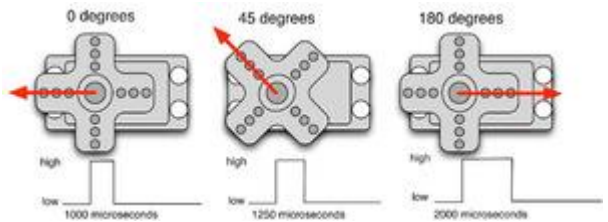
在需要操作机械结构的系统或项目中舵机的操作必不可免，掌握舵机操作原理，操作方法，就可设计属于自己的舵机设备，加入机械机构及算法实现，距离机器人也就指日可待了。

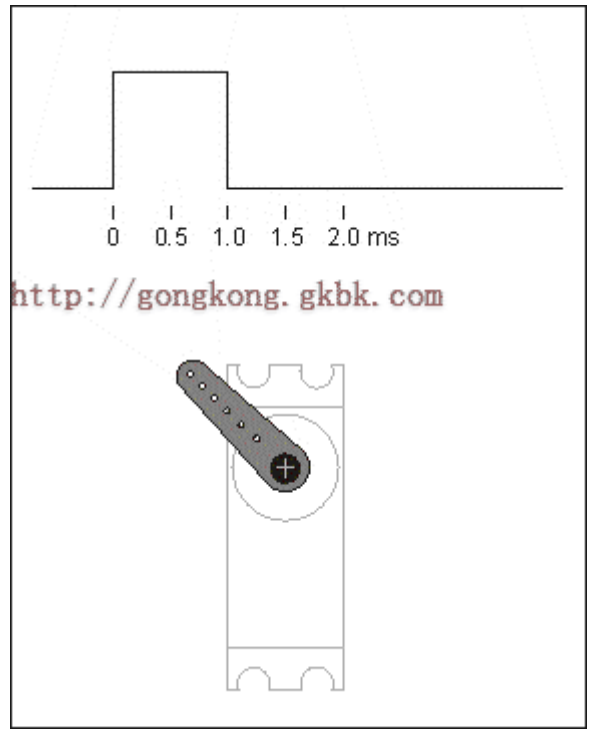


舵机操作方法

控制：控制信号由接收机的通道进入信号调制芯片，获得直流偏置电压。它内部有一个基准电路，产生周期为20ms，宽度为1.5ms的基准信号，将获得的直流偏置电压与电位器的电压比较，获得电压差输出。最后，电压差的正负输出到电机驱动芯片决定电机的正反转。当电机转速一定时，通过级联减速齿轮带动电位器旋转，使得电压差为0，电机停止转动。

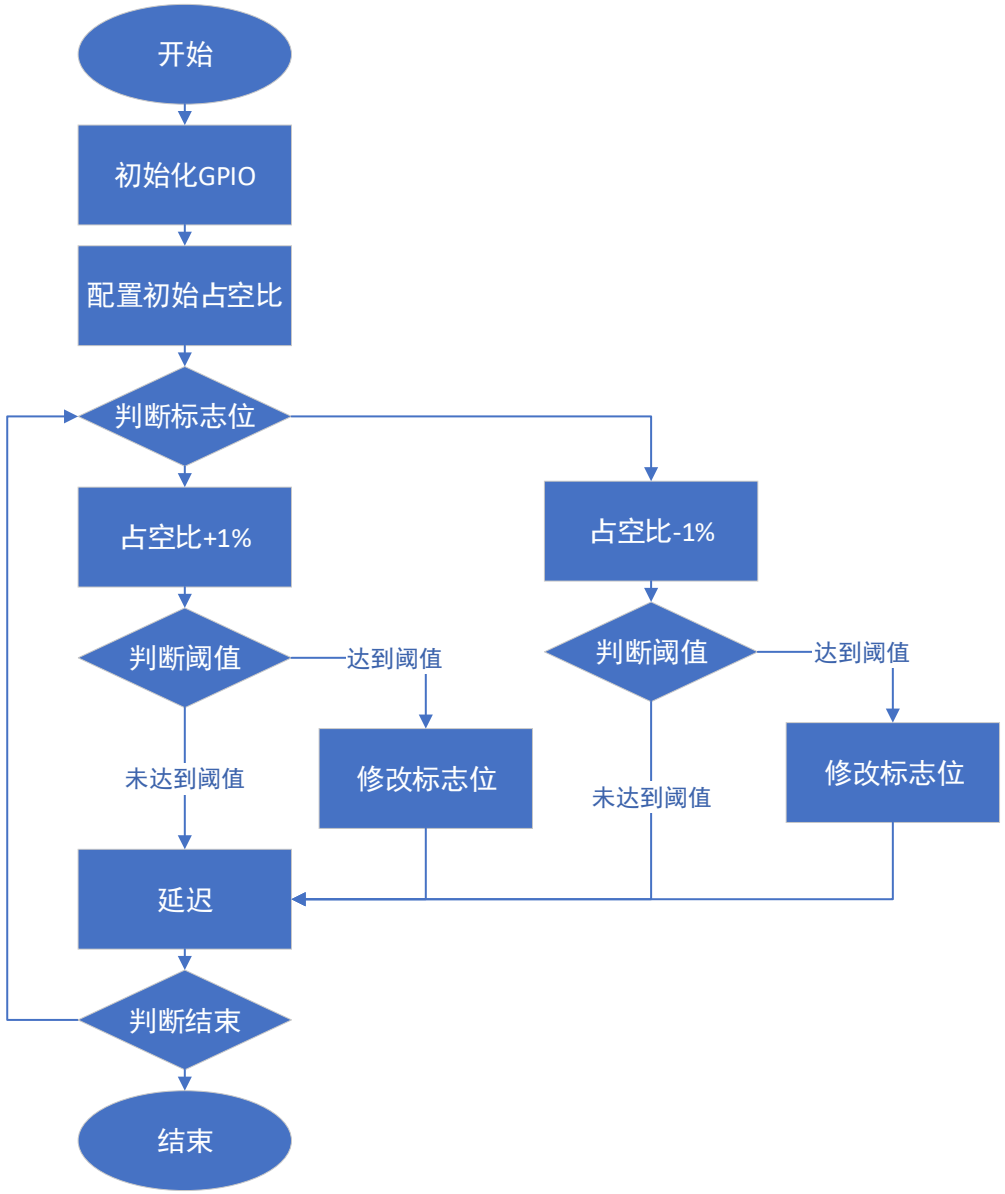






选型：需求不用，选择的舵机也是不同的，一般会根据扭矩，转速这两个重要参数来选择。当然在机械结构上也会有不同，比如大小，重量等等参数。

使用Code




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

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(18,GPIO.OUT)
p = GPIO.PWM(18,50)
p.start(3)

i = 5
k = 0
for j in range(0,1000,1) :
    if(k == 0) :
        i = i + 1
        time.sleep(0.5)
    else :
        i = i - 1
        time.sleep(0.5)
    if(i == 12) :
        k = 1
    elif(i == 5) :
        k = 0
    p.ChangeDutyCycle(i)
```

实战6——wiringPi 呼吸灯

wiringPi 呼吸灯

使用到的函数：

void pwmWrite(int pin, int value);

使用该函数可以将值写入指定管脚的PWM寄存器中，控制占空比。

pin：树莓派板上仅有一个PWM管脚，即wiringPi管脚1（BCM_GPIO 18，物理管脚号为12）。

value：可以配置的值为0~1024。

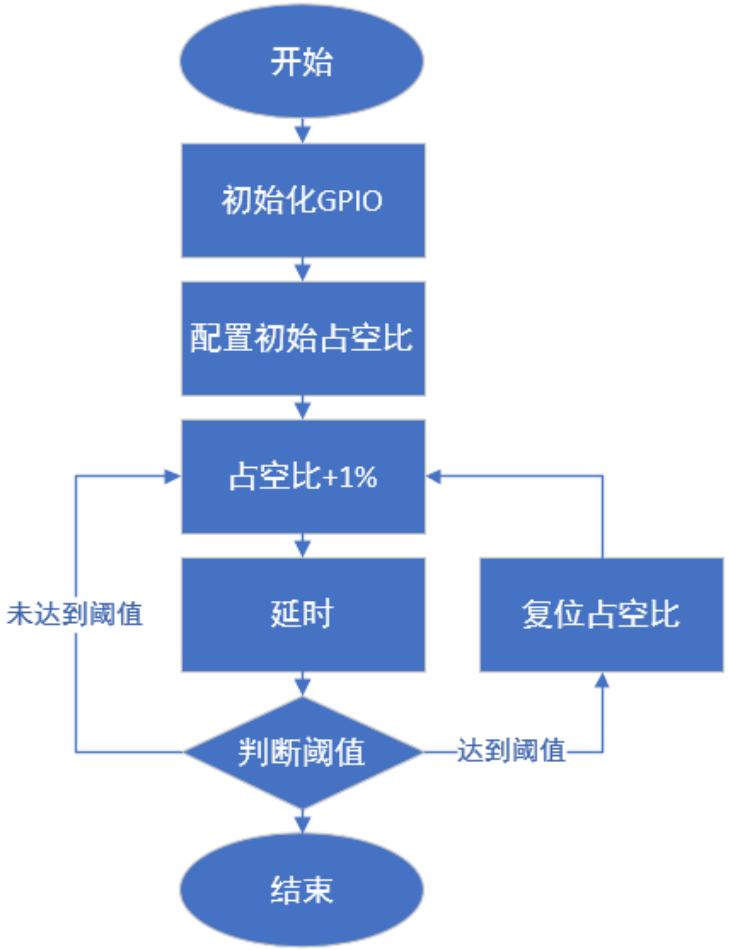
当在Sys模式时，该函数不可以控制树莓派的板上PWM。

所以只能操作GPIO18。

wiringPiSetupGpio()

pinMode(pin,mode)

```
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |  OUT | 1 | 11 || 12 | 0 | ALT0 | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |   IN | 1 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI | ALT0 | 1 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 1 | IN   | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
|   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
|   5 |  21 | GPIO.21 |  OUT | 1 | 29 || 30 |   |      | 0v      |     |     |
|   6 |  22 | GPIO.22 |  OUT | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
|  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
|  19 |  24 | GPIO.24 | ALT0 | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
|  26 |  25 | GPIO.25 |   IN | 1 | 37 || 38 | 0 | ALT0 | GPIO.28 | 28  | 20  |
|     |     |      0v |      |   | 39 || 40 | 0 | ALT0 | GPIO.29 | 29  | 21  |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+---Pi 3---+---+------+---------+-----+-----+
```



wiringPi 呼吸灯

```c
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int bright ;
    printf ("wiringPi PWM test program\n") ;
    if (wiringPiSetupGpio()== -1) {
        exit(1) ;
    }

    pinMode(18, PWM_OUTPUT) ;
    for(;;) {
        for(bright = 0; bright < 1024; ++bright) {
            pwmWrite(18, bright) ;
            delay(1) ;
        }
        for(bright = 1023; bright >= 0; --bright) {
            pwmWrite(18, bright) ;
            delay(1) ;
        }
    }
    return 0 ;
}

```

Python PWM实现与 wiringPi PWM的实现区别:

1. Python由软件实现PWM。
   wiringPi由硬件实现。

2. Python消耗CPU资源多。
   wiringPi基本不消耗CPU资源。

3. Python实现受硬件影响小。
   wiringPi严重受硬件影响。

谢谢您的观看

www.moore8.com

微信公众平台：
moore_8

QQ群： 327350729