参考网页：

[GPIO & RTC GPIO - ESP32 - — ESP-IDF 编程指南 latest 文档 (espressif.com)](https://docs.espressif.com/projects/esp-idf/zh_CN/latest/esp32/api-reference/peripherals/gpio.html)

[GPIO 上拉下拉解释说明\_嵌入式移动开发-CSDN博客\_gpio的上拉和下拉](https://blog.csdn.net/yinghaijushi/article/details/78991999?spm=1001.2101.3001.6650.8&utm_medium=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~HighlightScore-8.queryctrv2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~BlogCommendFromBaidu~HighlightScore-8.queryctrv2&utm_relevant_index=10)

1. 简介

ESP32 芯片有 40 个物理 GPIO pad。每个 pad 都可用作一个通用 IO，或连接一个内部的外设信号。IO\_MUX、RTC IO\_MUX 和 GPIO 交换矩阵用于将信号从外设传输至 GPIO pad。这些模块共同组成了芯片的 IO 控制。

注意：其中 GPIO 34-­39 仅用作输入管脚，其他的既可以作为输入又可以作为输出管脚。

GPIO6-11通常用于SPI闪存。

注：使用Gpio相关功能时需要添加头文件：

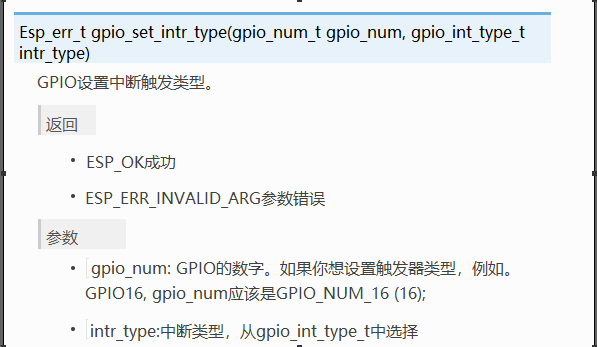
#include "driver/gpio.h"

1. API参考

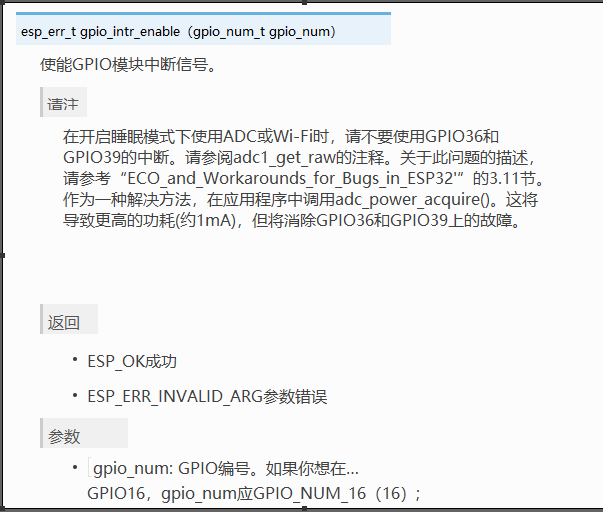




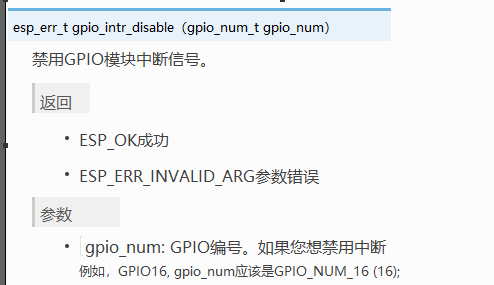




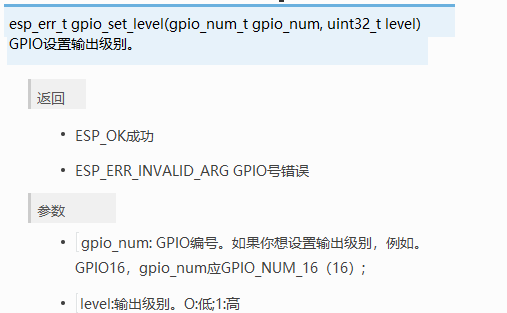




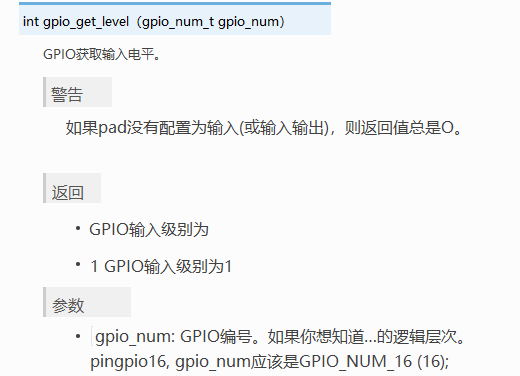




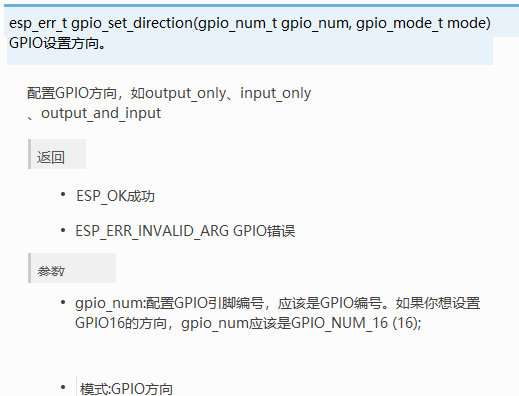




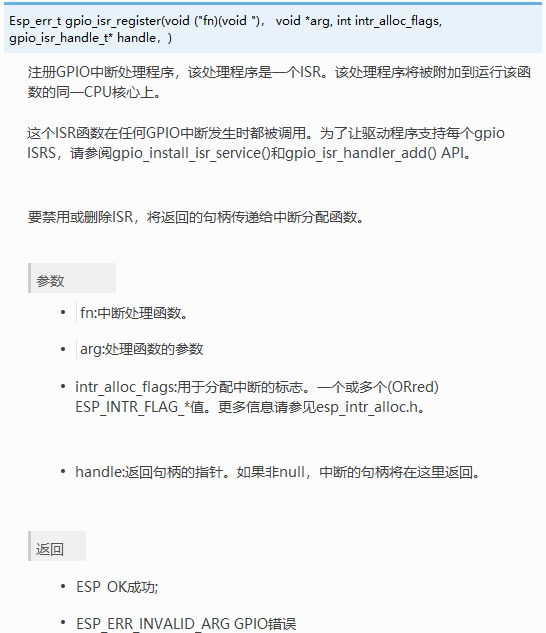




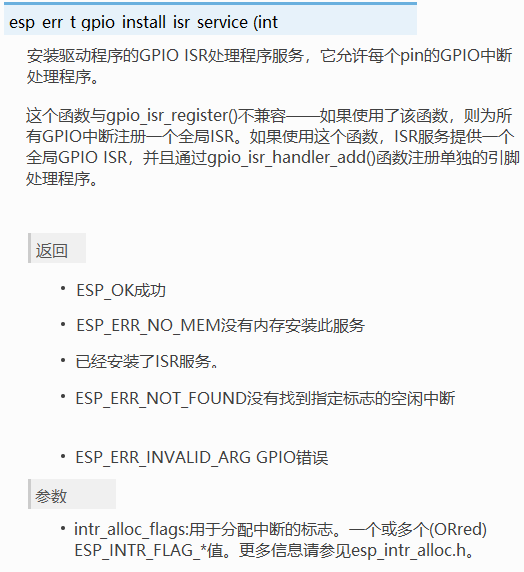




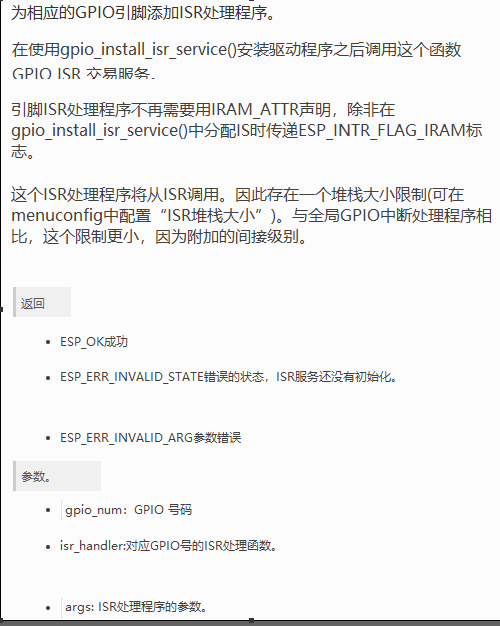




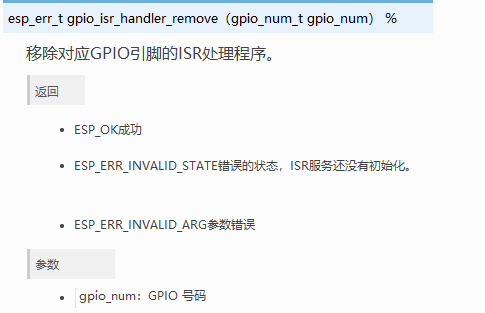












1. 使用Gpio的一般方法

* 选择Gpio口：gpio\_pad\_select\_gpio(GPIO\_NUM\_\*);
* 设置Gpio口的模式：gpio\_set\_direction(GPIO\_NUM\_\*,GPIO\_MODE\_OUTPUT/INPUT);
* 设置Gpio口的电平：gpio\_set\_level(GPIO\_NUM\_\*,HIGH/LOW(1/0));

1. 结构体方法

#define **GPIO\_OUTPUT\_IO\_0** 22 //将22Pin脚命名为GPIO\_OUTPUT\_IO\_0

#define **GPIO\_OUTPUT\_PIN\_SEL** (1ULL<<GPIO\_OUTPUT\_IO\_0) //配置GPIO\_OUT位寄存器

void **gpio\_init**(void)

{

    gpio\_config\_t io\_conf = {};//定义一个gpio\_config\_t类型的结构体io\_config，以下是对其进行参数配置

io\_conf.intr\_type = GPIO\_INTR\_DISABLE;//禁止中断

    io\_conf.mode = GPIO\_MODE\_OUTPUT;//选择为输出模式

    io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL;//配置GPIO\_OUT寄存器

    io\_conf.pull\_down\_en = 0;//禁止下拉

    io\_conf.pull\_up\_en = 0;//禁止上拉

**gpio\_config**(&io\_conf);//配置使能

}

//gpio\_set\_level();设置IO口输出电平

**gpio\_set\_level**(GPIO\_OUTPUT\_IO\_0,0);//配置这个GPIO为输出低电平

**gpio\_set\_level**(GPIO\_OUTPUT\_IO\_0,1);//配置这个GPIO为输出高电平

1. Gpio中断例程

* 中断触发方式

// GPIO\_INTR\_DISABLE //禁用GPIO中断

// GPIO\_INTR\_POSEDGE //GPIO中断类型：上升沿

// GPIO\_INTR\_NEGEDGE //下降沿

// GPIO\_INTR\_ANYEDGE //上升沿和下降沿

// GPIO\_INTR\_LOW\_LEVEL //输入低电平触发

// GPIO\_INTR\_HIGH\_LEVEL //输入高电平触发

* 简单使用方法

//选择要使用中断的Gpio口

**gpio\_pad\_select\_gpio**(Gpio\_num\_\*);

//设置Gpio为输入

**gpio\_set\_direction**(GPIO\_NUM\_\*,GPIO\_MODE\_INPUT);

//设置Gpio中断使能

**gpio\_intr\_enable**(GPIO\_NUM\_\*);

//设置Gpio的中断类型，其中GPIO\_INTR\_\*见上：中断触发方式

**gpio\_set\_intr\_type**(GPIO\_NUM\_\*, GPIO\_INTR\_\*);

//设置其上拉使能

**gpio\_pullup\_en**(GPIO\_NUM\_\*);

* 结构体使用方法

#define **GPIO\_INPUT\_IO\_0**     4

#define **GPIO\_INPUT\_PIN\_SEL**  (1ULL<<GPIO\_INPUT\_IO\_0)

void **gpio\_init**(void)

{

gpio\_config\_t io\_conf = {};//定义一个gpio\_config\_t类型的结构体io\_config，以下是对其进行参数配置

io\_conf.intr\_type = GPIO\_INTR\_NEGEDGE;//下降沿触发

    io\_conf.mode = GPIO\_MODE\_INPUT;//选择为输入模式

    io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL;//配置GPIO\_OUT寄存器

    io\_conf.pull\_up\_en = 1;//允许上拉

**gpio\_config**(&io\_conf);//配置使能GPIO\_INPUT\_IO\_0

}

注意：在使用睡眠模式的ADC和wifi时，请不用使用Gpio36和39的中断

* 配置FreeRTOS任务和队列

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#define **ESP\_INTR\_FLAG\_DEFAULT** 0

static xQueueHandle gpio\_evt\_queue = NULL; //定义一个队列返回变量

void IRAM\_ATTR **gpio\_isr\_handler**(void\* arg){

    //把中断消息插入到队列的后面，将gpio的io参数传递到队列中

    uint32\_t gpio\_num = (uint32\_t) arg;

**xQueueSendFromISR**(gpio\_evt\_queue, &gpio\_num, NULL);

注：此函数中不能执行如：printf等耗时操作

}

void **gpio\_task**(void\* arg)

{

**printf**(" \r\n start gpio task ...\r\n  ");

    uint32\_t io\_num;

    for(;;){

        //不断读取gpio队列，读取完后将删除队列

        if(**xQueueReceive**(gpio\_evt\_queue, &io\_num, portMAX\_DELAY)){

**printf**("GPIO[%d] intr, val: %d\n", io\_num, **gpio\_get\_level**(io\_num));

        }

    }

}

void **gpio\_intr\_init**(void)

{

    // 注册中断服务

**gpio\_install\_isr\_service**(ESP\_INTR\_FLAG\_DEFAULT);

    // 设置GPIO的中断回调函数

**gpio\_isr\_handler\_add**(GPIO\_INPUT\_IO\_0, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_0);

    // 创建一个消息队列，从中获取队列句柄

    gpio\_evt\_queue = **xQueueCreate**(10, sizeof(uint32\_t));

    // 创建GPIO检测任务

**xTaskCreate**(gpio\_task         // 任务函数

            , "gpio\_task\_example" // 任务名字

            , 2048                // 任务堆栈大小

            , NULL                // 传递给任务函数的参数

            , 10                  // 任务优先级

            , NULL);              // 任務句柄

}

* 检测Gpio口中断的完整例程：

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

#define **GPIO\_INPUT\_IO\_0**    4

#define **GPIO\_INPUT\_PIN\_SEL**  (1ULL<<GPIO\_INPUT\_IO\_0)  // 配置GPIO\_IN位寄存器

#define **ESP\_INTR\_FLAG\_DEFAULT** 0

static xQueueHandle gpio\_evt\_queue = NULL; //定义一个队列返回变量

/\*//配置中断共四步，本质上是配置四个函数：

1.void IRAM\_ATTR gpio\_isr\_handler(void\* arg)把中断消息插入到队列的后面，将gpio的io参数传递到队列中

2.void gpio\_task(void\* arg)检测中断，并进行相应任务

3.void gpio\_init(void)配置Gpio相关参数

4.void gpio\_intr\_init(void)注册配置中断相关参数

\*/

void IRAM\_ATTR **gpio\_isr\_handler**(void\* arg) {

    //把中断消息插入到队列的后面，将gpio的io参数传递到队列中

    uint32\_t gpio\_num = (uint32\_t) arg;

**xQueueSendFromISR**(gpio\_evt\_queue, &gpio\_num, NULL);

}

void **gpio\_task**(void\* arg) {

**printf**(" \r\n start gpio task ...\r\n  ");

    uint32\_t io\_num;

    for(;;){//不断读取gpio队列，读取完后将删除队列

        if(**xQueueReceive**(gpio\_evt\_queue, &io\_num, portMAX\_DELAY)){

**printf**("GPIO[%d] intr, val: %d\n", io\_num, **gpio\_get\_level**(io\_num));

        }}}

void **gpio\_init**(void){

gpio\_config\_t io\_conf = {};//定义一个gpio\_config\_t类型的结构体io\_config，以下是对其进行参数配置

io\_conf.intr\_type = GPIO\_INTR\_NEGEDGE;//下降沿触发

    io\_conf.mode = GPIO\_MODE\_INPUT;//选择为输入模式

    io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL;//配置GPIO\_OUT寄存器

io\_conf.pull\_up\_en = 1;//允许上拉

**gpio\_config**(&io\_conf);//配置使能GPIO\_INPUT\_IO\_0

}

void **gpio\_intr\_init**(void){

    // 注册中断服务

**gpio\_install\_isr\_service**(ESP\_INTR\_FLAG\_DEFAULT);

    // 设置GPIO的中断回调函数

**gpio\_isr\_handler\_add**(GPIO\_INPUT\_IO\_0, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_0);

    // 创建一个消息队列，从中获取队列句柄

    gpio\_evt\_queue = **xQueueCreate**(10, sizeof(uint32\_t));

    // 创建GPIO检测任务

**xTaskCreate**(gpio\_task         // 任务函数

            , "gpio\_task\_example" // 任务名字

            , 2048                // 任务堆栈大小

            , NULL                // 传递给任务函数的参数

            , 10                  // 任务优先级

            , NULL);              // 任務句柄

}

void **app\_main**(void){

**gpio\_init**();

**gpio\_intr\_init**();

    while(1) {

**vTaskDelay**(1000 / portTICK\_PERIOD\_MS);

    }}

1. 官网例程

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"

#include "driver/gpio.h"

/\*\*

 \* Brief:

 \* This test code shows how to configure gpio and how to use gpio interrupt.

 \*

 \* GPIO status:

 \* GPIO18: output

 \* GPIO19: output

 \* GPIO4:  input, pulled up, interrupt from rising edge and falling edge

 \* GPIO5:  input, pulled up, interrupt from rising edge.

 \*

 \* Test:

 \* Connect GPIO18 with GPIO4

 \* Connect GPIO19 with GPIO5

 \* Generate pulses on GPIO18/19, that triggers interrupt on GPIO4/5

 \*

 \*/

#define **GPIO\_OUTPUT\_IO\_0**    18

#define **GPIO\_OUTPUT\_IO\_1**    19

/\*  ULL:unsigned long long  1ULL：表示将 1 的类型声明为unsigned long long

    (1ULL<<GPIO\_OUTPUT\_IO\_0) | (1ULL<<GPIO\_OUTPUT\_IO\_1)  意为：1循环左移GPIO\_OUTPUT\_IO\_0（18）位

    或上 1循环左移GPIO\_OUTPUT\_IO\_1（19）位

    也即：1 000 000 000 000 000 000  | 10 000 000 000 000 000 000 = 11 000 000 000 000 000 000

\*/

#define **GPIO\_OUTPUT\_PIN\_SEL**  ((1ULL<<GPIO\_OUTPUT\_IO\_0) | (1ULL<<GPIO\_OUTPUT\_IO\_1))

#define **GPIO\_INPUT\_IO\_0**     4

#define **GPIO\_INPUT\_IO\_1**     5

#define **GPIO\_INPUT\_PIN\_SEL**  ((1ULL<<GPIO\_INPUT\_IO\_0) | (1ULL<<GPIO\_INPUT\_IO\_1))

#define **ESP\_INTR\_FLAG\_DEFAULT** 0

static xQueueHandle gpio\_evt\_queue = NULL;

static void IRAM\_ATTR **gpio\_isr\_handler**(void\* arg)

{

    uint32\_t gpio\_num = (uint32\_t) arg;

**xQueueSendFromISR**(gpio\_evt\_queue, &gpio\_num, NULL);

}

static void **gpio\_task\_example**(void\* arg)

{

    uint32\_t io\_num;

    for(;;) {

        if(**xQueueReceive**(gpio\_evt\_queue, &io\_num, portMAX\_DELAY)) {

**printf**("GPIO[%d] intr, val: %d\n", io\_num, **gpio\_get\_level**(io\_num));

        }

    }

}

void **app\_main**(void)

{

    //zero-initialize the config structure.

    gpio\_config\_t io\_conf = {};

    //disable interrupt

    io\_conf.intr\_type = GPIO\_INTR\_DISABLE;

    //set as output mode

    io\_conf.mode = GPIO\_MODE\_OUTPUT;

    //bit mask of the pins that you want to set,e.g.GPIO18/19

    io\_conf.pin\_bit\_mask = GPIO\_OUTPUT\_PIN\_SEL;

    //disable pull-down mode

    io\_conf.pull\_down\_en = 0;

    //disable pull-up mode

    io\_conf.pull\_up\_en = 0;

    //configure GPIO with the given settings

**gpio\_config**(&io\_conf);

    //interrupt of rising edge

    io\_conf.intr\_type = GPIO\_INTR\_POSEDGE;

    //bit mask of the pins, use GPIO4/5 here

    io\_conf.pin\_bit\_mask = GPIO\_INPUT\_PIN\_SEL;

    //set as input mode

    io\_conf.mode = GPIO\_MODE\_INPUT;

    //enable pull-up mode

    io\_conf.pull\_up\_en = 1;

**gpio\_config**(&io\_conf);

    //change gpio intrrupt type for one pin

**gpio\_set\_intr\_type**(GPIO\_INPUT\_IO\_0, GPIO\_INTR\_ANYEDGE);

    //create a queue to handle gpio event from isr

    gpio\_evt\_queue = **xQueueCreate**(10, sizeof(uint32\_t));

    //start gpio task

**xTaskCreate**(gpio\_task\_example, "gpio\_task\_example", 2048, NULL, 10, NULL);

    //install gpio isr service

**gpio\_install\_isr\_service**(ESP\_INTR\_FLAG\_DEFAULT);

    //hook isr handler for specific gpio pin

**gpio\_isr\_handler\_add**(GPIO\_INPUT\_IO\_0, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_0);

    //hook isr handler for specific gpio pin

**gpio\_isr\_handler\_add**(GPIO\_INPUT\_IO\_1, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_1);

    //remove isr handler for gpio number.

**gpio\_isr\_handler\_remove**(GPIO\_INPUT\_IO\_0);

    //hook isr handler for specific gpio pin again

**gpio\_isr\_handler\_add**(GPIO\_INPUT\_IO\_0, gpio\_isr\_handler, (void\*) GPIO\_INPUT\_IO\_0);

**printf**("Minimum free heap size: %d bytes\n", **esp\_get\_minimum\_free\_heap\_size**());

    int cnt = 0;

    while(1) {

**printf**("cnt: %d\n", cnt++);

**vTaskDelay**(1000 / portTICK\_RATE\_MS);//延时1s

        //注：若想设置Gpio口的电平，需先设置为输出模式

        //注：若想读取Gpio口的电平，需先设置为输入模式

**gpio\_set\_direction**(GPIO\_OUTPUT\_IO\_0,GPIO\_MODE\_OUTPUT);

**gpio\_set\_level**(GPIO\_OUTPUT\_IO\_0, cnt % 2);//设置GPIO\_OUTPUT\_IO\_0为 高/低电平

**gpio\_set\_direction**(GPIO\_OUTPUT\_IO\_0,GPIO\_MODE\_INPUT);

**printf**("Gpio\_18 leve：%d\n", **gpio\_get\_level**(GPIO\_OUTPUT\_IO\_0));

**gpio\_set\_direction**(GPIO\_OUTPUT\_IO\_1,GPIO\_MODE\_OUTPUT);

**gpio\_set\_level**(GPIO\_OUTPUT\_IO\_1, cnt % 2);//设置GPIO\_OUTPUT\_IO\_1为 高/低电平

**gpio\_set\_direction**(GPIO\_OUTPUT\_IO\_1,GPIO\_MODE\_INPUT);

**printf**("Gpio\_19 leve：%d\n", **gpio\_get\_level**(GPIO\_OUTPUT\_IO\_1));

/\*

        测试蜂鸣器

        if(cnt < 100){

            gpio\_set\_level(GPIO\_OUTPUT\_IO\_0,0);

            vTaskDelay(1 / portTICK\_RATE\_MS);

            gpio\_set\_level(GPIO\_OUTPUT\_IO\_0,1);

            vTaskDelay(1 / portTICK\_RATE\_MS);

        }

        else{

            vTaskDelay(2 / portTICK\_RATE\_MS);

        }

        cnt++;

        cnt %= 200;

\*/

    }

}