LED 操作

和呼吸灯实现

淘宝店铺：

PC 端：

http://n-xytrt8gqu585po94mwj5atokcyd4.taobao.com/index.htm

手机端：

https://shop.m.taobao.com/shop/shop_index.htm?sellerId=755668508&shopId=104493595&inShopPageId=423890608&pathInfo=shop/index2

资料下载地址：

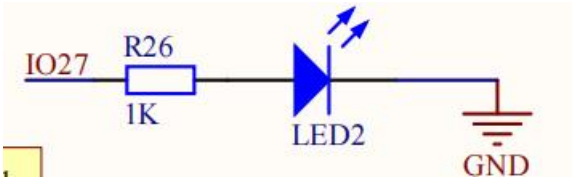
链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

源码下载地址：

https://gitee.com/vi-iot/esp32-board.git

一、LED 操作介绍

学嵌入式程序，第一步是 Helloworld，第二步就是点亮 LED 了，对于有开发经验的同学来说，点亮 LED 不是什么问题，本质上就是 GPIO 的高低电平操作，大家看下图是一个简单的 LED 电路，



GPIO27 管脚通过一个电阻串接一个 LED 到地，当我们的 IO27 管脚输出高电平 3.3V 时，电路中就会产生电流流过 LED，LED 就会点亮，当我们的 IO27 输出低电平 0V 时，就没有电流，LED 就会熄灭，非常简单，在所有的嵌入式开发中，LED 硬件都是类似的，通过程序我们可以玩出更多花样，我们先来看看 esp-idf 框架中关于 GPIO 的程序是怎样写的。

在之前章节中我们介绍了基于 esp-idf 框架的工程目录，现在我们直接在 main.c 修改代码，新增一个 LED 初始化函数（源码位于 esp32-board/ledc）

```
//定义 LED 的 GPIO 口
#define LED_GPIO  GPIO_NUM_27

//LED 闪烁初始化
void led_flash_init(void)
{
    gpio_config_t led_gpio_cfg = {
        .pin_bit_mask = (1<<LED_GPIO),          //指定 GPIO
        .mode = GPIO_MODE_OUTPUT,               //设置为输出模式
        .pull_up_en = GPIO_PULLUP_DISABLE,      //禁止上拉
        .pull_down_en = GPIO_PULLDOWN_DISABLE,  //禁止下拉
        .intr_type = GPIO_INTR_DISABLE,         //禁止中断
    };
    gpio_config(&led_gpio_cfg);
    xTaskCreatePinnedToCore(led_run_task,"led",2048,NULL,3,NULL,1);
}
```

这个函数中，会定义一个关于 gpio 的配置结构体，然后通过 gpio_config 函数将配置设置到底层，通过这步，我们就完成了 gpio 的初始化，这里我们将 gpio 设置成输出，这里注意，一般来说，GPIO 有四种较为常见的工作模式：

输出：可以设置 GPIO 的高低电平

输入：可以获取外部输入的高低电平信息，一般要设置加上拉电阻或下拉电阻

浮空输出：可以设置 GPIO 的高低电平，但要在电路外部中增加上拉电阻

开漏输入：可以获取外部输入的高低电平信息，但要在电路外部中增加上拉电阻

回到程序，启用一个任务，循环点亮 LED，任务内容如下：

```
//LED 闪烁运行任务
void led_run_task(void* param)
{
    int gpio_level = 0;
```

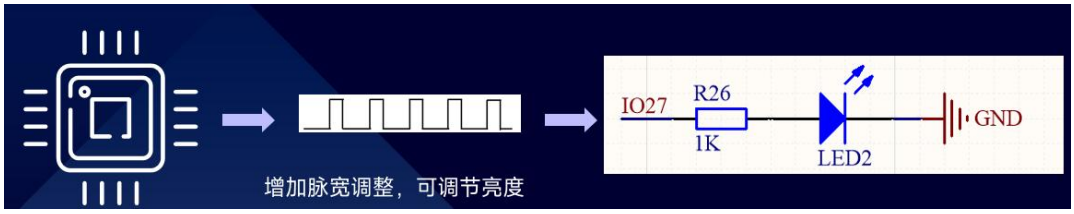
```
    while(1)
    {
        vTaskDelay(pdMS_TO_TICKS(500));
        gpio_set_level(LED_GPIO,gpio_level);
        gpio_level = gpio_level?0:1;
    }
}
```

每隔 500ms 切换一次电平输出。

app_main()，函数直接调用 led_flash_init()初始化，编译 idf.py build 和 idf.py flash 烧录到板子上后，我们就可以看到 led 在闪烁了。

二、LEDC 模块介绍

上一章我们简单学习了一下，GPIO 的操作，并点亮了一个 LED，现在我们想要做得更好，我们希望 LED 可以改变亮度，但从原理图上来看，似乎从硬件的角度无法实现这个功能，GPIO 也无法输出其他值，那要怎么做呢？这时就需要用到了 PWM 脉宽调制，如下图所示，



当我们在 GPIO 引脚上增加一段高低电平的脉冲时，我们会看到灯一闪一闪，高低电平脉冲切换速度达到一定程度时（大约是 25Hz），我们人眼是看不出来一闪一闪的效果，只会看到 LED 较暗，那到底暗多少，这就需要 PWM 脉宽调制的占空比来决定，所谓占空比，简单来说就是高电平时间占 PWM 周期的百分比时间，比如占空比是 50%，我们看到 LED 就是半亮，占空比 100%就是全亮，和直接输出高电平没区别，如果我们动态的改变占空比，那么就可以看到 LED 从暗到亮，从亮到暗的变化，这就是呼吸灯的效果

好了理论知识讲解完，接下来在 esp-idf 中是如何操作的？在说明前，我们程序功能计划如下，LED 从亮缓慢熄灭，再从熄灭状态逐渐变亮，也就是演示呼吸灯效果。新增一个函数 led_breath_init

```
//LED 呼吸灯初始化
void led_breath_init(void)
{
    //初始化一个定时器
    ledc_timer_config_t ledc_timer = {
        .speed_mode       = LEDC_MODE,      //低速模式
        .timer_num        = LEDC_TIMER,     //定时器 ID
        .duty_resolution  = LEDC_DUTY_RES,  //占空比分辨率，这里是 13 位，
2^13-1
        .freq_hz          = LEDC_FREQUENCY,  // PWM 频率,这里是 5KHZ
        .clk_cfg          = LEDC_AUTO_CLK    // 时钟
    };
    ESP_ERROR_CHECK(ledc_timer_config(&ledc_timer));

    //ledc 通道初始化
```

```
    ledc_channel_config_t ledc_channel = {
        .speed_mode     = LEDC_MODE,         //低速模式
        .channel        = LEDC_CHANNEL,      //PWM 通道 0-7
        .timer_sel      = LEDC_TIMER,        //关联定时器，也就是上面初始化好
的那个定时器
        .intr_type      = LEDC_INTR_DISABLE,//不使能中断
        .gpio_num       = LEDC_OUTPUT_IO,    //设置输出 PWM 方波的 GPIO 管脚
        .duty           = 0, // 设置默认占空比为 0
        .hpoint         = 0
    };
    ESP_ERROR_CHECK(ledc_channel_config(&ledc_channel));
    //开启硬件 PWM
    ledc_fade_func_install(0);
    //创建一个事件组，用于通知任务渐变完成
    s_ledc_ev = xEventGroupCreate();
    //配置 LEDC 渐变
    ledc_set_fade_with_time(LEDC_MODE,LEDC_CHANNEL,LEDC_DUTY,2000);
    //启动渐变
    ledc_fade_start(LEDC_MODE,LEDC_CHANNEL,LEDC_FADE_NO_WAIT);
    //设置渐变完成回调函数
    ledc_cbs_t cbs = {.fade_cb=ledc_finish_cb,};
    ledc_cb_register(LEDC_MODE,LEDC_CHANNEL,&cbs,NULL);
    xTaskCreatePinnedToCore(ledc_breath_task,"ledc",2048,NULL,3,NULL,1);
}
```

这个函数需要初始化两个内容，分别是 ledc_timer_config 和 ledc_channel_config，ledc_timer_config 用于初始化用到的定时器，ledc_channel_config 用于初始化 ledc 输出通道以及将 timer 关联起来，而初始内容大家看上述代码的注释，很清晰，这里就不多说了。初始化完成后我们就可以使用 ledc 的一些 API 了。

这里有几个函数比较关键

ledc_set_fade_with_time 设置一个 PWM 占空比目标值和渐变周期，这里代码示例是，需要在 2000ms，将目前的占空比渐变至 LEDC_DUTY（满占空比）

ledc_fade_start 函数启动渐变，通过 LEDC_FADE_NO_WAIT 参数设置为立刻返回，那我们怎么知道渐变完成呢？可以通过 ledc_cb_register 函数，注册一个回调函数，当渐变完成的时候会调用我们的回调函数

然后我们再新建一个任务，处理渐变完成的事项。我们来看下回调函数和任务内容

```
//用于通知渐变完成
static EventGroupHandle_t   s_ledc_ev = NULL;
//关灯完成事件标志
#define LEDC_OFF_EV  (1<<0)
//开灯完成事件标志
#define LEDC_ON_EV   (1<<1)
//渐变完成回调函数
```

```
bool IRAM_ATTR ledc_finish_cb(const ledc_cb_param_t *param, void *user_arg)
{
    BaseType_t xHigherPriorityTaskWoken;
    if(param->duty)
    {
        xEventGroupSetBitsFromISR(s_ledc_ev,LEDC_ON_EV,&xHigherPriorityT
askWoken);
    }
    else
    {
        xEventGroupSetBitsFromISR(s_ledc_ev,LEDC_OFF_EV,&xHigherPriority
TaskWoken);
    }
    return xHigherPriorityTaskWoken;
}
```

由上可知，回调函数是在中断中执行的，我们不能做过多的操作，当渐变完成后，我们将事件标记一下就可退出

接下来看下任务函数

```
//ledc 渐变任务
void ledc_breath_task(void* param)
{
    EventBits_t ev;
    while(1)
    {
        ev =
xEventGroupWaitBits(s_ledc_ev,LEDC_ON_EV|LEDC_OFF_EV,pdTRUE,pdFALSE,pdM
S_TO_TICKS(5000));
        if(ev)
        {
            //设置 LEDC 开灯渐变
            if(ev & LEDC_OFF_EV)
            {
                ledc_set_fade_with_time(LEDC_MODE,LEDC_CHANNEL,LEDC_DUTY,
2000);
                ledc_fade_start(LEDC_MODE,LEDC_CHANNEL,LEDC_FADE_NO_WAIT)
;
            }
            else if(ev & LEDC_ON_EV)    //设置 LEDC 关灯渐变
            {
                ledc_set_fade_with_time(LEDC_MODE,LEDC_CHANNEL,0,2000);
                ledc_fade_start(LEDC_MODE,LEDC_CHANNEL,LEDC_FADE_NO_WAIT)
;
```

```
            }
            //再次设置回调函数
            ledc_cbs_t cbs = {.fade_cb=ledc_finish_cb,};
            ledc_cb_register(LEDC_MODE,LEDC_CHANNEL,&cbs,NULL);
        }
    }
}
```

上述任务函数等待渐变完成事件，如果渐变结束后占空比是 0，则重新启动渐变至满占空比的操作；如果渐变结束后占空比是满占空比，则重新启动渐变至 0 占空比的操作。

完成的例程在 esp32-board/ledc 中，大家下载下来后，idf.py build 和 idf.py flash 烧录至开发板后，可以看到呼吸灯的效果，从亮缓慢熄灭，再缓慢点亮，如此循环