RGB 灯

WS2812 实验(RMT)

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：

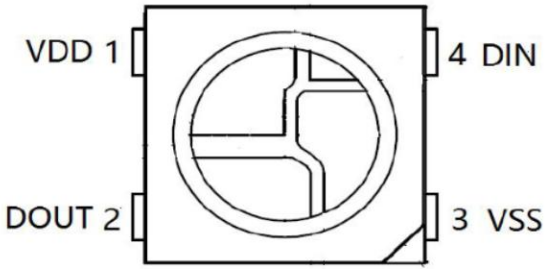
链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

源码下载地址：

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

一、WS2812 介绍

之前的课我们介绍了 LED 的点亮和呼吸灯，但灯的颜色只有一种，一个 IO 口只能操控一个灯，在智能家居开发中，灯可不能这么单调，我们需要一种新的器件，就是本课介绍的 WS2812。WS2812 是集成控制单元以及 RGB 灯珠的器件，集成度高，功能强大，并且可以串接。以下是单个 WS2812 的外形引脚图

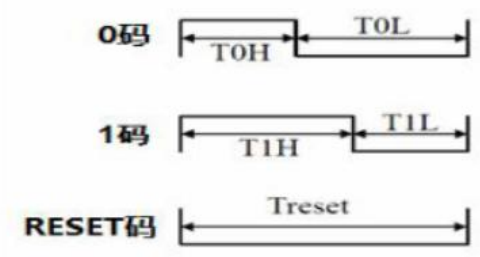


引脚序号	引脚名称	功能
1	VDD	电源 3.5-5.3V，但是实测 3.3V 也能驱动
2	DOUT	数据输出
3	VSS	接地脚
4	DIN	数据输入

WS2812 是单线通信，对于 0 码和 1 码的编码方式如下

T0H	0 码， 高电平时间	220ns~380ns
T1H	1 码， 高电平时间	580ns~1μs
T0L	0 码， 低电平时间	580ns~1μs
T1L	1 码， 低电平时间	220ns~420ns
RES	帧单位，低电平时间	280μs 以上

输入码型：



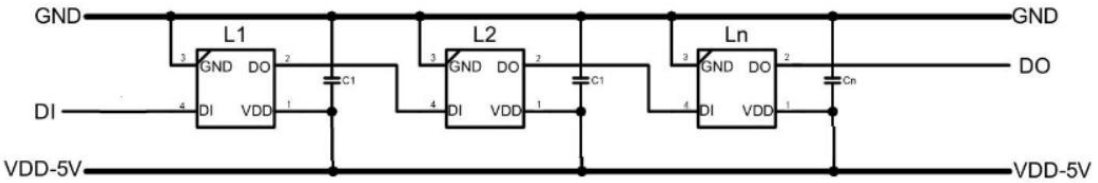
由上可知，0 码和 1 码都是由高低电平组成，只不过高低电平占用时间有区别

而数据帧也很简单，如下

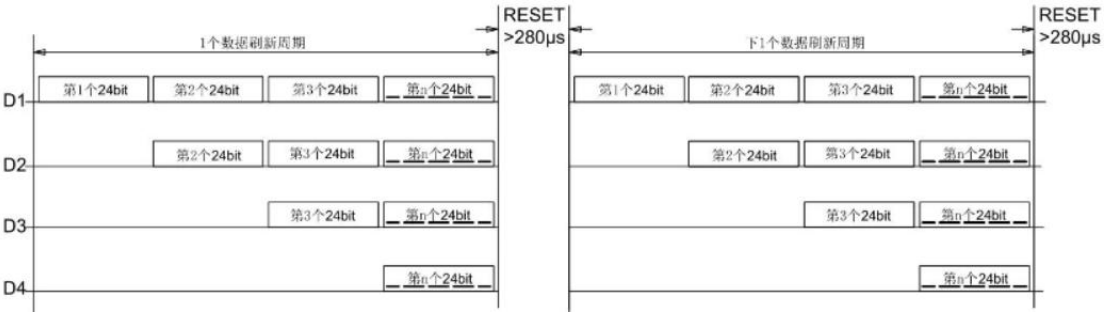
24bit 数据结构

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

对于一个 WS2812 灯珠，颜色描述如下，我们知道所有颜色都是由三原色 RGB 组成的，因此通过对 WS2812 中 RGB 不同的值，就可以显示我们想要的颜色。但一般来说我们电路接的可不只一个 WS2812 灯珠，会有很多个接在一起，也就是前一个 WS2812 的 DOUT 脚接入后一个 WS2812 的 DIN，如下图所示



最先一个的 WS2812 的 DIN 接我们 ESP32 的引脚。那既然只有一个 IO，如何控制多个 WS2812？其实这得益于 WS2812 比较特别的控制流程。当我们发送第一个 24 位数据时，第一个 WS2812 会记录并且锁存起来，再发第二个的 24 位的时候，就会把数据直接传输给第二个 WS2812 并记录，再发第三个 24 位数据，第前两个因为已经记录了数据，因此都会将数据流转到下一个，最后也就是第三个会收到数据。如此类推，直到我们发送一个 RESET 信号，本次传输才结束，后续发送数据又从第一个 WS2812 开始记录，具体流程如下图



二、RMT 红外控制

在学习用 ESP32 点亮 WS2812 之前，我们还要了解一下 RMT 红外控制，为什么需要了解这个模块？大家可以看到，驱动 WS2812 要求的时序很高，我们用 IO 口模拟控制比较困难，用 SPI 或者 I2C 时序也对不上，因此红外是最优选择，我们先来了解一下红外的基本原理。

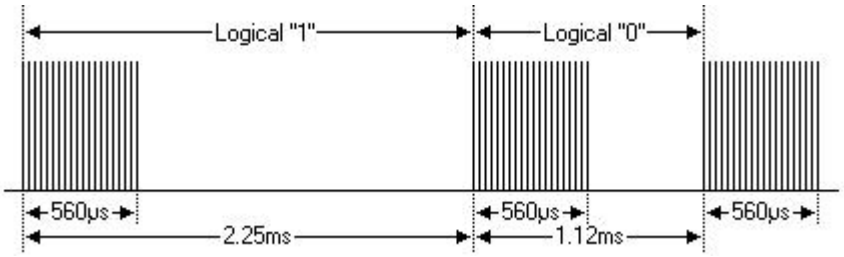
红外辐射，介于可见光和微波之间、波长范围为 0.76-1000 微米的红外波段的电磁波。它是频率比红光低的不可见光，因此我们肉眼是无法看见红外的。

我们来看下常用的 NEC 红外编码数据格式

经典 NEC 红外编码使用 38kHz 载波频率；

命令帧由 起始位 + 地址码 + 地址码反码 + 命令码 + 命令码反码 组成。

起始码：9ms 高电平（38KHZ 脉冲）+4.5ms 低电平



逻辑"0"：562.5μs 的高电平 + 562.5μs 的低电平，总时长为 1.125ms。

逻辑"1"：562.5μs 的高电平 + 1.6875ms 的低电平，总时长为 2.25ms

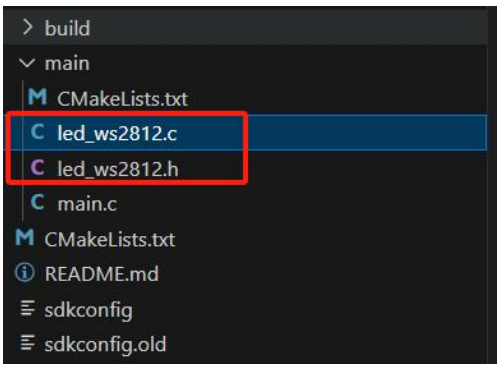
结束码：562.5μs 的有效脉冲

注意这里的高电平是指 38kHz，占空比大概是 1/3 的载波，低电平为 0，然而在使用 RMT 驱动 WS2812 控制中，我们不需要载波。

三、RMT 程序控制 WS2812

控制 WS2812 程序稍微有点复杂，但都是一劳永逸，我们写好了，日后拿过来就可以直接用。

打开 esp32-board/ws2812 工程，目录如下



最核心的代码在 led_ws2812.c 中，我们先来看看初始化函数代码 ws2812_init

```
/** 初始化 WS2812 外设
 * @param gpio 控制 WS2812 的管脚
 * @param maxled 控制 WS2812 的个数
 * @param led_handle 返回的控制句柄
 * @return ESP_OK or ESP_FAIL
*/
esp_err_t ws2812_init(gpio_num_t gpio,int maxled,ws2812_strip_handle_t*
handle)
{
    struct ws2812_strip_t* led_handle = NULL;
    //新增一个 WS2812 驱动描述
    led_handle = calloc(1, sizeof(struct ws2812_strip_t));
    assert(led_handle);
    //按照 led 个数来分配 RGB 缓存数据
    led_handle->led_buffer = calloc(1,maxled*3);
    assert(led_handle->led_buffer);
    //设置 LED 个数
    led_handle->led_num = maxled;
    //定义一个 RMT 发送通道配置
    rmt_tx_channel_config_t tx_chan_config = {
        .clk_src = RMT_CLK_SRC_DEFAULT,         //默认时钟源
        .gpio_num = gpio,                       //GPIO 管脚
        .mem_block_symbols = 64,                //内存块大小，即 64 * 4 = 256
字节
        .resolution_hz = LED_STRIP_RESOLUTION_HZ,    //RMT 通道的分辨率
10000000hz=0.1us，也就是可以控制的最小时间单元
        .trans_queue_depth = 4,                 //底层后台发送的队列深度
    };
    //创建一个 RMT 发送通道
```

```
    ESP_ERROR_CHECK(rmt_new_tx_channel(&tx_chan_config,
&led_handle->led_chan));
    //创建自定义编码器（重点函数），所谓编码，就是发射红外时加入我们的时序控制
    ESP_ERROR_CHECK(rmt_new_led_strip_encoder(&led_handle->led_encoder))
;
    //使能 RMT 通道
    ESP_ERROR_CHECK(rmt_enable(led_handle->led_chan));
    //返回 WS2812 操作句柄
    *handle = led_handle;
    return ESP_OK;
}
```

源码大致解读如下

ws2812_strip_t 结构体为 WS2812 驱动描述，保存了 RMT 通道、RGB buffer 数据、led 个数信息。

rmt_tx_channel_config_t 结构体用于定义 RMT 发送通道配置，包含时钟源、GPIO 管脚、分辨率、发送缓存等。最后调用 rmt_new_tx_channel 函数设置。

rmt_new_led_strip_encoder 函数是本篇的重点，里面设定了编码器，对发送的数据格式进行配置

现在看下这个函数的定义

```
** 创建一个基于 WS2812 时序的编码器
 * @param ret_encoder 返回的编码器，这个编码器在使用 rmt_transmit 函数传输时
会用到
 * @return ESP_OK or ESP_FAIL
*/
esp_err_t rmt_new_led_strip_encoder(rmt_encoder_handle_t *ret_encoder)
{
    esp_err_t ret = ESP_OK;
    //创建一个自定义的编码器结构体，用于控制发送编码的流程
    rmt_led_strip_encoder_t *led_encoder = NULL;
    led_encoder = calloc(1, sizeof(rmt_led_strip_encoder_t));
    ESP_GOTO_ON_FALSE(led_encoder, ESP_ERR_NO_MEM, err, TAG, "no mem for
led strip encoder");
    led_encoder->base.encode = rmt_encode_led_strip;    //这个函数会在 rmt
发送数据的时候被调用，我们可以在这个函数增加额外代码进行控制
    led_encoder->base.del = rmt_del_led_strip_encoder;  //这个函数在卸载
rmt 时被调用
    led_encoder->base.reset = rmt_led_strip_encoder_reset;  //这个函数在复
位 rmt 时被调用
    //新建一个编码器配置(0,1 位持续时间，参考芯片手册)
    rmt_bytes_encoder_config_t bytes_encoder_config = {
        .bit0 = {
            .level0 = 1,
```

```
            .duration0 = 0.3 * LED_STRIP_RESOLUTION_HZ / 1000000, //
T0H=0.3us
            .level1 = 0,
            .duration1 = 0.9 * LED_STRIP_RESOLUTION_HZ / 1000000, //
T0L=0.9us
        },
        .bit1 = {
            .level0 = 1,
            .duration0 = 0.9 * LED_STRIP_RESOLUTION_HZ / 1000000, //
T1H=0.9us
            .level1 = 0,
            .duration1 = 0.3 * LED_STRIP_RESOLUTION_HZ / 1000000, //
T1L=0.3us
        },
        .flags.msb_first = 1 //高位先传输
    };
    //传入编码器配置，获得数据编码器操作句柄
    rmt_new_bytes_encoder(&bytes_encoder_config,
&led_encoder->bytes_encoder);
    //新建一个拷贝编码器配置，拷贝编码器一般用于传输恒定的字符数据，比如说结束
位
    rmt_copy_encoder_config_t copy_encoder_config = {};
    rmt_new_copy_encoder(&copy_encoder_config,
&led_encoder->copy_encoder);
    //设定结束位时序
    uint32_t reset_ticks = LED_STRIP_RESOLUTION_HZ / 1000000 * 50 / 2; //
分辨率/1M=每个 ticks 所需的 us，然后乘以 50 就得出 50us 所需的 ticks
    led_encoder->reset_code = (rmt_symbol_word_t) {
        .level0 = 0,
        .duration0 = reset_ticks,
        .level1 = 0,
        .duration1 = reset_ticks,
    };
    //返回编码器
    *ret_encoder = &led_encoder->base;
    return ESP_OK;
err:
    if (led_encoder) {
        if (led_encoder->bytes_encoder) {
            rmt_del_encoder(led_encoder->bytes_encoder);
        }
        if (led_encoder->copy_encoder) {
            rmt_del_encoder(led_encoder->copy_encoder);
        }
```

```
        free(led_encoder);
    }
    return ret;
}
```

这个函数重点在于 rmt_bytes_encoder_config_t 结构体配置，这个结构体里面定义了 RMT 控制 0 码和 1 码的高低电平时间，这时候就可以把 WS2812 的 0 码、1 码的高低电平时间填入。其中 LED_STRIP_RESOLUTION_HZ 宏定义的是时钟分辨率，例程里是 10MHz，也就是低于 0.1us 的脉冲无法发出，LED_STRIP_RESOLUTION_HZ/10000000 表示的是 1us 对应时钟节拍 tick 数。完成配置后，通过 rmt_new_bytes_encoder 函数设置并返回一个编码器，这个编码器用于将我们要发送的字节数据转为 RMT 格式的数据，在 esp-idf 中 RMT 格式的数据称为 RMT 符号，对应的是 rmt_symbol_word_t 结构体。

另外一个较重点的是 rmt_encode_led_strip 函数，这个函数会在执行红外发送的时候调用，其目的就是将数据编码成 RMT 格式，编码 RESET 符号，控制结束流程。

具体和其余的代码，大家通过看代码中的注释即可知道什么意思。

现在我们看一下怎么应用这个代码，我们看下 led_ws2812.h 头文件的接口

```
typedef struct ws2812_strip_t *ws2812_strip_handle_t;

/** 初始化 WS2812 外设
 * @param gpio 控制 WS2812 的管脚
 * @param maxled 控制 WS2812 的个数
 * @param led_handle 返回的控制句柄
 * @return ESP_OK or ESP_FAIL
*/
esp_err_t ws2812_init(gpio_num_t gpio,int maxled,ws2812_strip_handle_t*
led_handle);
/** 反初始化 WS2812 外设
 * @param handle 初始化的句柄
 * @return ESP_OK or ESP_FAIL
*/
esp_err_t ws2812_deinit(ws2812_strip_handle_t handle);
/** 向某个 WS2812 写入 RGB 数据
 * @param handle 句柄
 * @param index 第几个 WS2812（0 开始）
 * @param r,g,b RGB 数据
 * @return ESP_OK or ESP_FAIL
*/
esp_err_t ws2812_write(ws2812_strip_handle_t handle,uint32_t
index,uint32_t r,uint32_t g,uint32_t b);
```

只有三个接口，初始化、反初始化、设置 RGB 值。

现在我们主要看下 ws2812_write 函数如何使用

ws2812_strip_handle_t 句柄是 ws2812_init 函数返回的句柄，我们对 WS2812 操作都是

基于这个句柄进行。

index 要操作的灯珠下标，0 开始。比如说开发板配套的 WS2812 器件上有 12 个灯珠，我们要操作第 5 个灯珠，index=4。

rgb rgb 数据每个值最大 255,3 个值就是 24 位

在 app_main()中，调用如下

```
void app_main(void)
{
    ws2812_strip_handle_t ws2812_handle = NULL;
    int index = 0;
    ws2812_init(WS2812_GPIO_NUM,WS2812_LED_NUM,&ws2812_handle);
    while(1)
    {
        //红色跑马灯
        for(index = 0;index < WS2812_LED_NUM;index++)
        {
            uint32_t r = 230,g = 20,b = 20;
            ws2812_write(ws2812_handle,index,r,g,b);
            vTaskDelay(pdMS_TO_TICKS(80));
        }
        //绿色跑马灯
        for(index = 0;index < WS2812_LED_NUM;index++)
        {
            uint32_t r = 20,g = 230,b = 20;
            ws2812_write(ws2812_handle,index,r,g,b);
            vTaskDelay(pdMS_TO_TICKS(80));
        }
        //蓝色跑马灯
        for(index = 0;index < WS2812_LED_NUM;index++)
        {
            uint32_t r = 20,g = 20,b = 230;
            ws2812_write(ws2812_handle,index,r,g,b);
            vTaskDelay(pdMS_TO_TICKS(80));
        }
    }
}
```

编译并烧录进我们的开发板后，就会看到红绿蓝三种颜色的跑马灯在循环闪动。

大家可以直接将这两个 led_ws2812.h、led_ws2812.c 应用在自己工程中，减少后面重复的编码工作