作品摘要

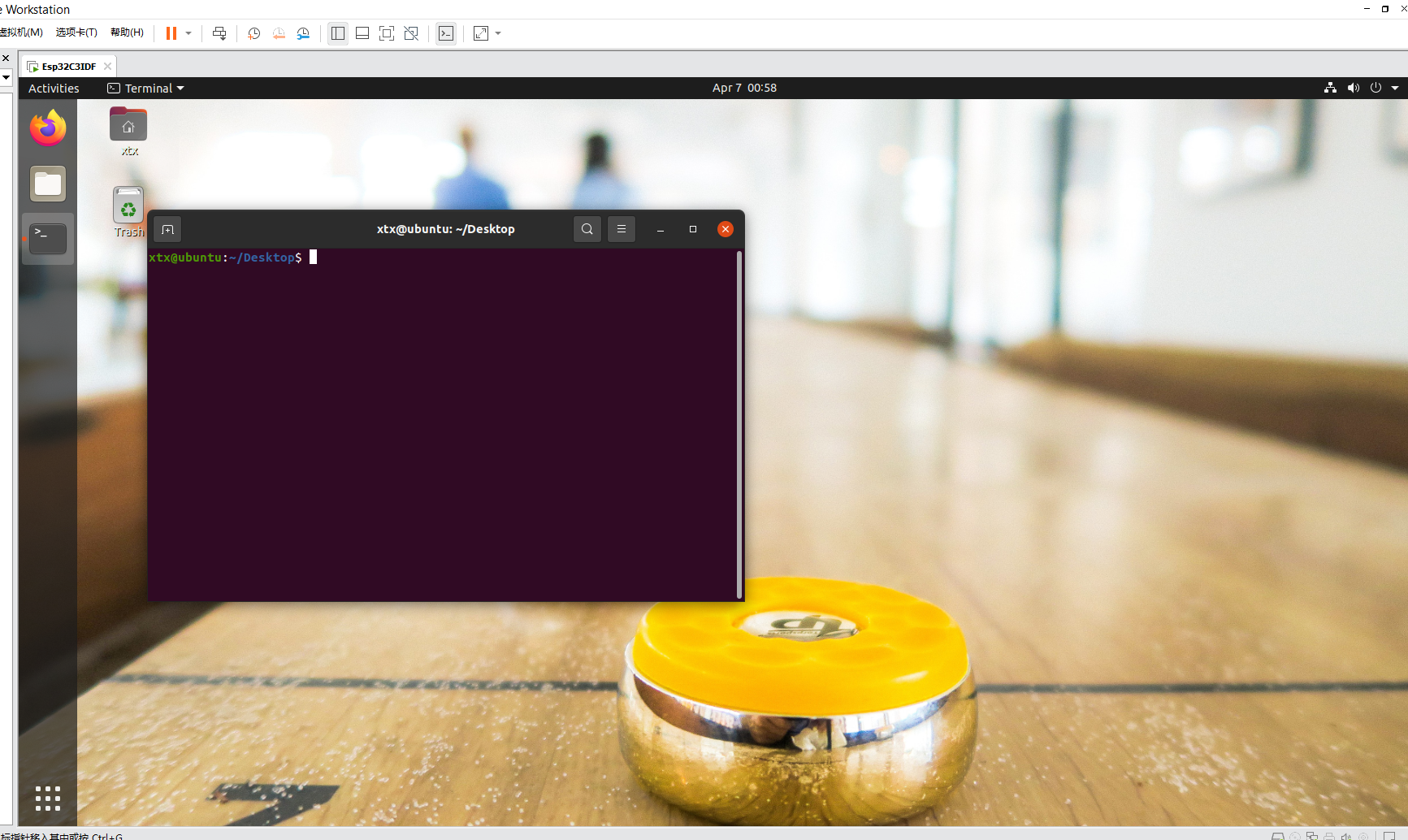
为了深入了解学习计算机组成系统结构，我们小组决定通过ESP32单片机的学习，深入了解CPU的各种外设功能与单片机操作系统RTOS的使用。在初步了解过后，我们小组选择使用TIM和UART外设，前者发送PWM信号驱动RGB呼吸灯，后者接收上位机信号修改灯的颜色。

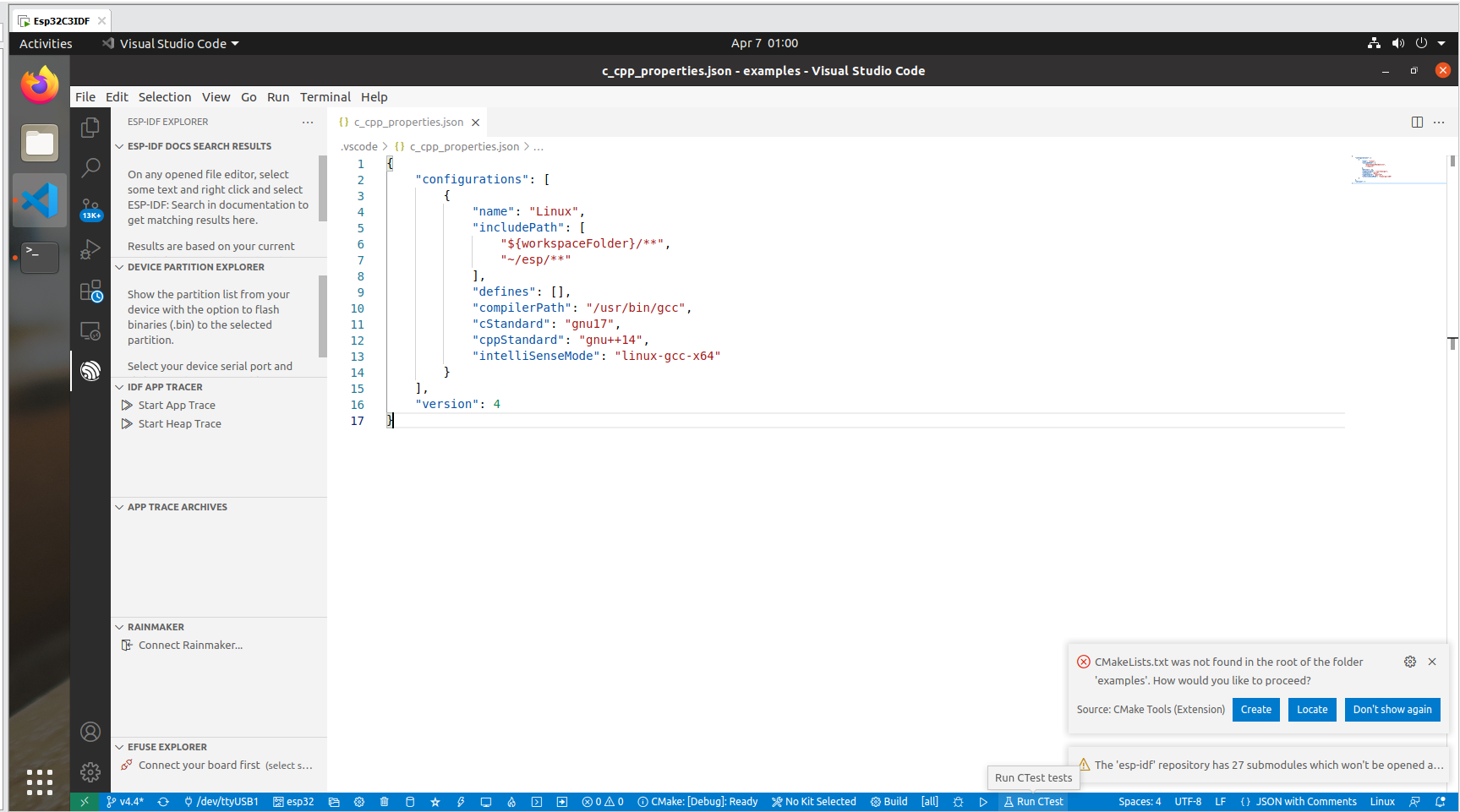
设计步骤

1. 准备步骤

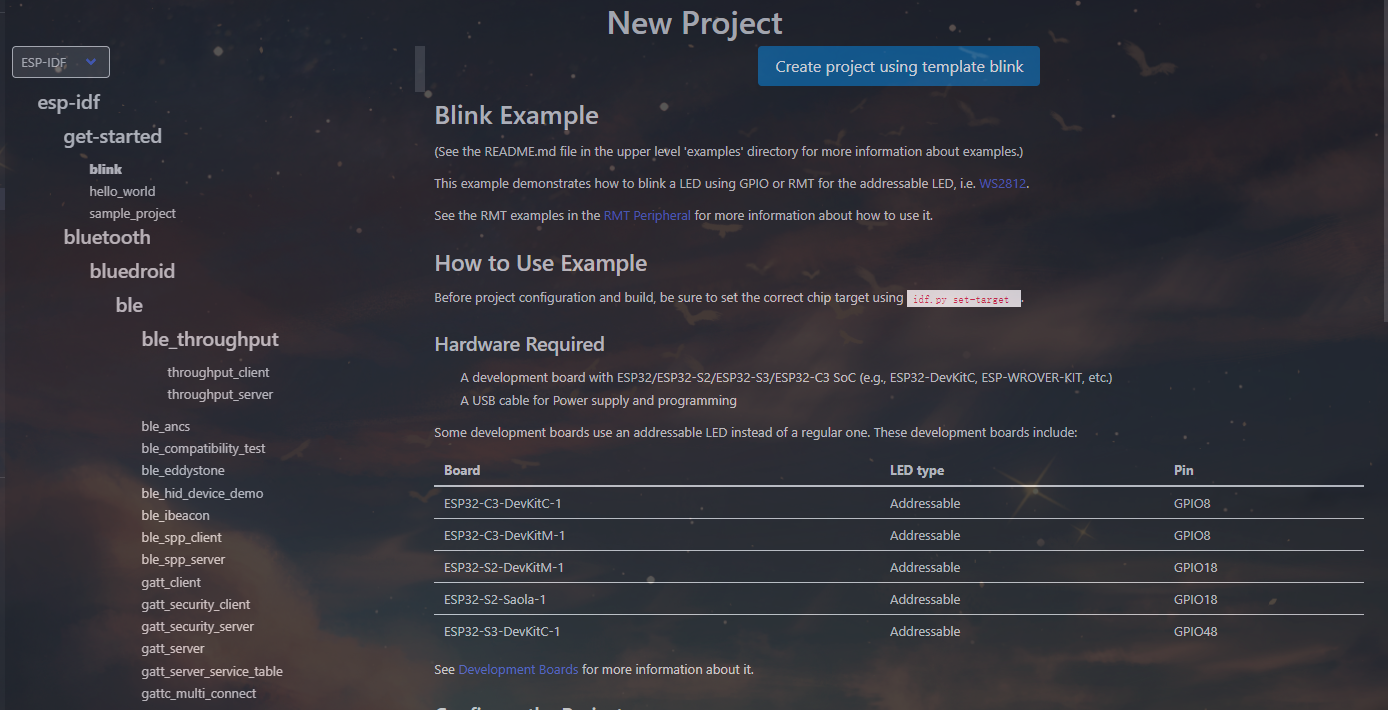
我们小组在淘宝上进行了模组的对比，在对比功能，大小和开发的方便程度后，最终选择了安信可的ESP32-C3-12F。该开发板大小适中，且板载一颗RGB5050灯珠，无需再外接灯珠。板载一块CH340C的串口模组，自动复位烧录，且方便我们初学调试。

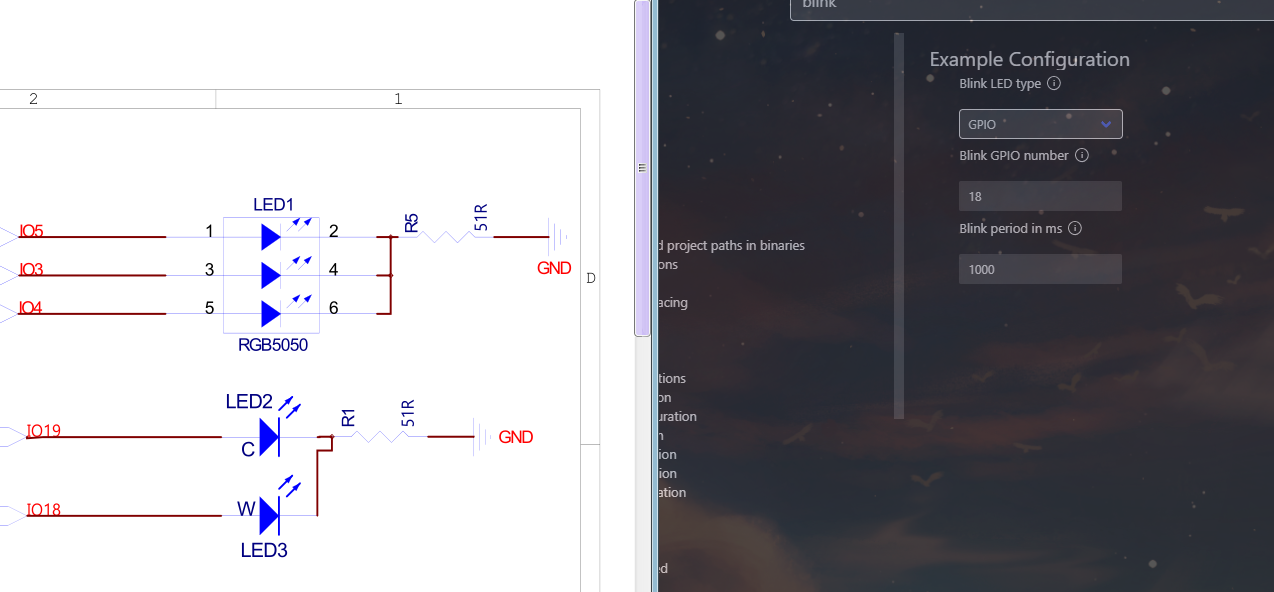
模块到货后，我们首先着手于开发环境的安装工作。开发环境名为ESP-IDF，支持全平台，我们选择了用官方vscode插件并自动安装开发环境，但是在初步安装过程中出现了依赖错误的情况。在查询官方github的issuse后了解到其对环境的纯净性要求较高，因此我们安装了VMware WorkStation Pro 16软件，新建了安装ubuntu20.04的虚拟机来进行下一步的编译和调试工作。我们小组均为第一次安装ubuntu，系统默认为英文版，我们一步步参照网上教程进行了初始化配置和换源并使用命令行安装vscode和ESP-IDF框架的插件。

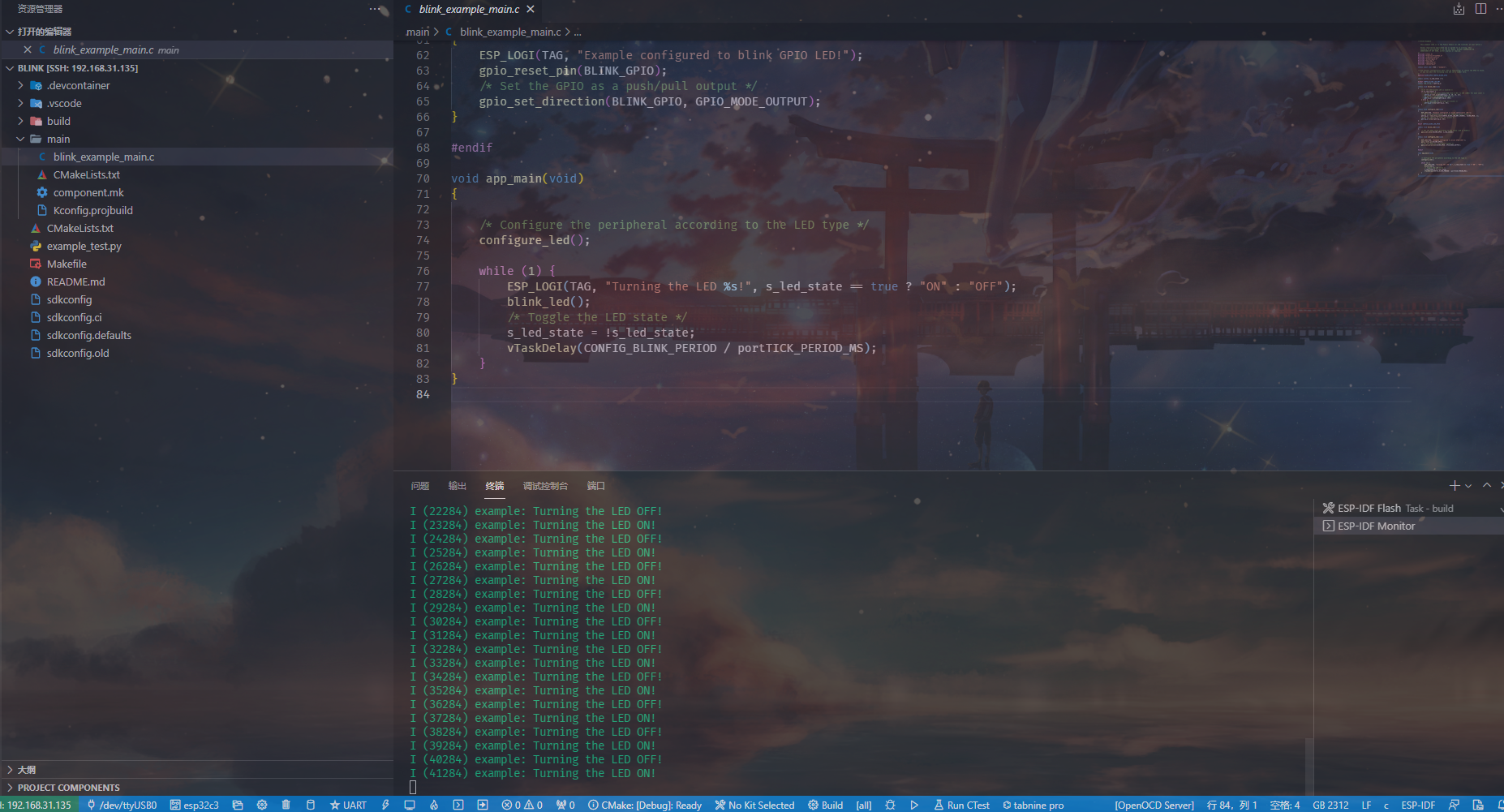




在ubuntu配置完成后，我们小组考虑到主要使用的系统是windows，所以配置了ssh密钥和vscode远程连接，成功在windows端修改代码。跟随官方wiki的指示，我们使用模板程序创建了一个点灯Blink程序。使用SDK Configuration editor面板，对照开发板的原理图修改GPIO，成功编译烧录了第一个程序









1. 分析学习RTOS和GPIO的使用

我们小组决定从BLink例程的代码入手分析学习RTOS和GPIO的使用。

首先程序入口是函数app\_main，其具体代码如下：

1. void app\_main(void)
2. {
3. */\* Configure the peripheral according to the LED type \*/*
4. configure\_led();
5. while (1) {
6. ESP\_LOGI(TAG, "Turning the LED %s!", s\_led\_state == true ? "ON" : "OFF");
7. blink\_led();
8. */\* Toggle the LED state \*/*
9. s\_led\_state = !s\_led\_state;
10. vTaskDelay(CONFIG\_BLINK\_PERIOD / portTICK\_PERIOD\_MS);
11. }
12. }

首先导入到了configure\_led函数中，这个函数主要负责配置GPIO口

GPIO=General Purpose Input Output，意思是通用输入输出。我们在查找其作用时了解到，其既可以作为一个电源输出3.3v或可以作为一个地输出0v，还可以用于输出甚至复用为其他的功能。

1. static void configure\_led(void)
2. {
3. ESP\_LOGI(TAG, "Example configured to blink GPIO LED!");
4. gpio\_reset\_pin(BLINK\_GPIO);
5. */\* Set the GPIO as a push/pull output \*/*
6. gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT);
7. }

ESP\_LOGI作为官方函数库的方法，它可以打印在默认的串口里，在我们的开发板上，它默认打印到我们接入的串口中，TAG代表该消息是由什么事件打印。

在这里，它的定义是

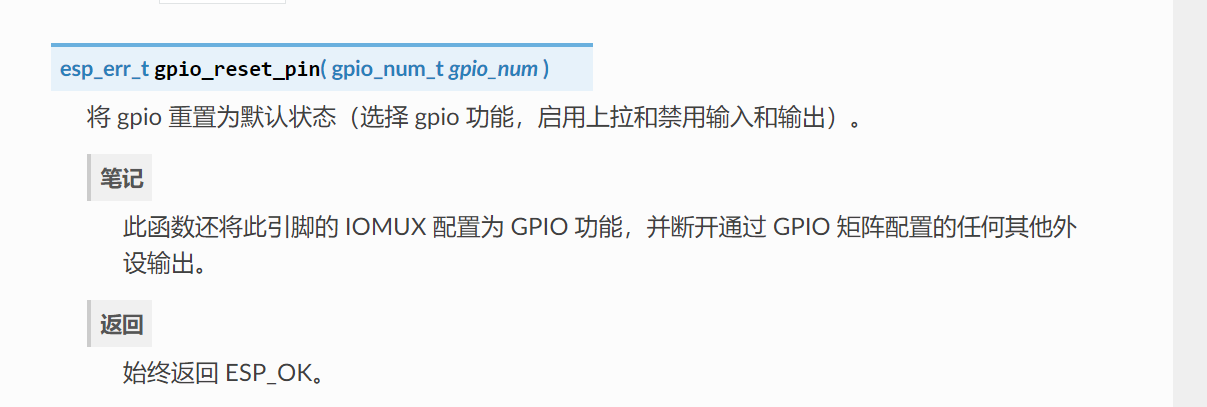


其输出效果是如下

所有官方函数方法都有在线文档，如gpio\_reset\_pin方法

是将某个GPIO重置为默认状态，防止该IO重复配置

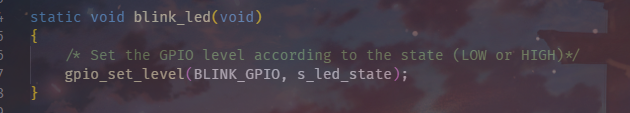
该方法的参数是io口的宏定义



gpio\_set\_direction方法，设定GPIO的输入还是输出方向，在这里是需要点灯，因此要改为输出模式。

完成IO口配置后进入while循坏，使用ESP\_LOGI输出当前灯的状态

然后调用blink\_led函数执行gpio\_set\_level方法。



该方法的第一个参数是io口的宏定义，第二个就是设置的高低电平。

当s\_led\_state为1时，其就代表把io口拉高。

当s\_led\_state为0是，其就代表把io口拉低。

根据我们对io口的学习，拉高是把电平设置为3.3v，拉低即设置为0v

然后是把s\_led\_state变量反转，并执行vTaskDelay(CONFIG\_BLINK\_PERIOD / portTICK\_PERIOD\_MS)方法

该方法是FreeRTOS的延时函数，用于在FreeRTOS系统内进行延时，括号内的参数据我们查资料是一种固定写法，其包含宏定义

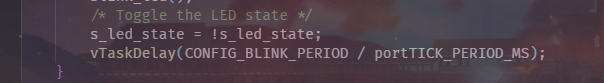
#define CONFIG\_BLINK\_PERIOD 1000

#define portTICK\_PERIOD\_MS ((TickType\_t)(1000 / configTICK\_RATE\_HZ))

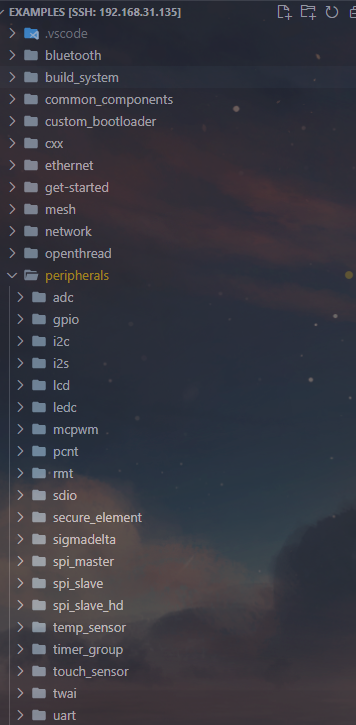
#define configTICK\_RATE\_HZ ( 1000 )

其含义为

心跳频率configTICK\_RATE\_H是1000，那么时间片是1ms。那么，vTaskDelay(CONFIG\_BLINK\_PERIOD / portTICK\_PERIOD\_MS)就是延迟1000个时间片，也就是延迟1s。这就是其一秒钟闪灯的由来。



1. 使用例程分析pwm波的使用

IDF SDK提供了非常丰富的函数例程和在线文档

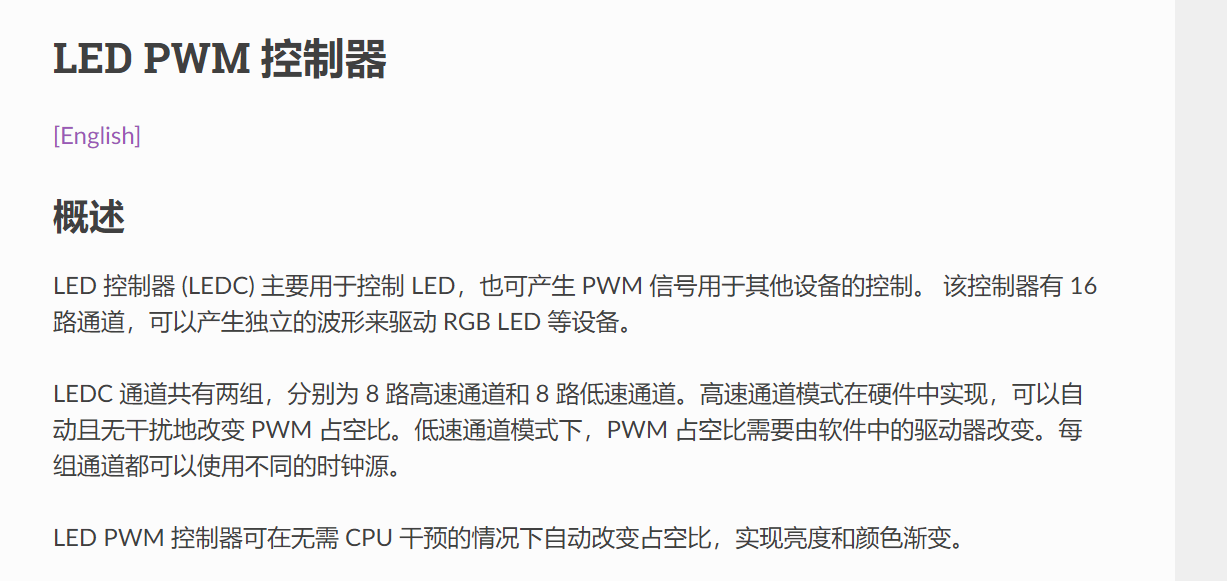
我们小组发现其中包含一个外设叫LED PWM 控制器，PWM全程Pulse-width modulation简称脉宽调制，是将模拟信号变换为脉冲的一种技术。在模拟电路中，其将输出一系列由高低电平组成的，频率固定的方波。占空比是高电平占整个周期的时间。随着占空比的调整，该模拟信号的电压有效值将被改变。

如当占空比为100%，即一直为高电平。对于ESP32-C3芯片，高电平为3.3v，所以该输出为3.3v。

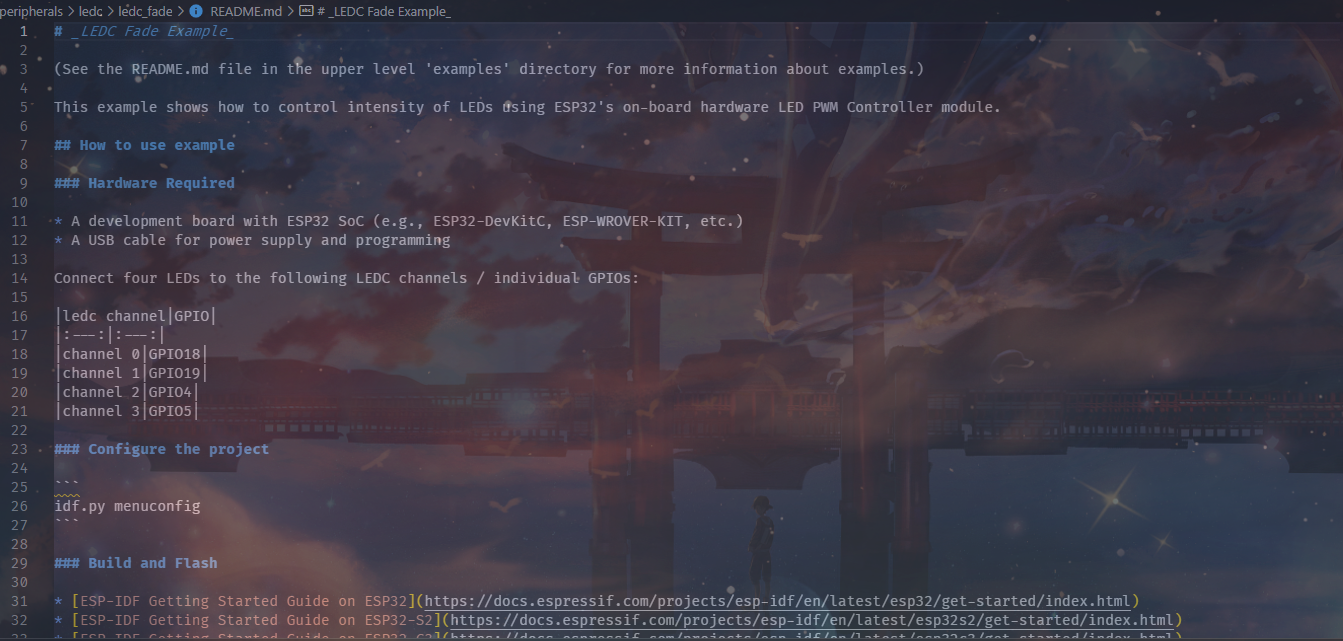
如当占空比为50%，即有50%时间为高电平，根据有效值计算可得，有效值为1.65v。

同时我们注意到对于有效值的计算的前提是，该PWM频率必须大于约10Hz。

明白了这些基础知识后，我们开始阅读LED PWM 控制器的文档



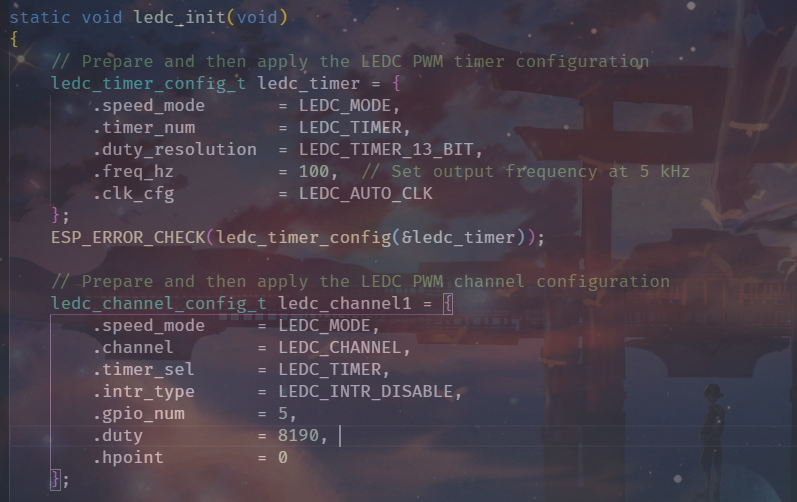
根据在线文档的指示，我们找到了例程文件夹下peripherals/ledc/ledc\_fade的例程项目



根据README的描述，该例程可以做到对LED的PWM波的后台平滑调整

再根据在线文档的函数描述，我们找到了几个核心函数，并复制到新的空白项目中

我们新建了函数ledc\_init

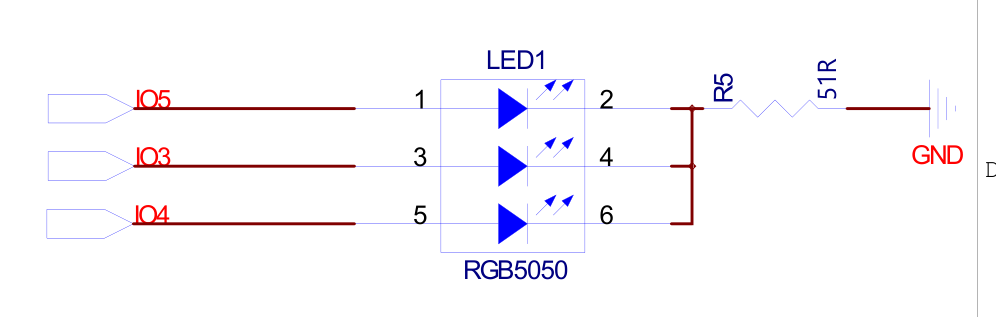


ledc\_timer\_config\_t结构体和ledc\_timer\_config方法用于配置定时器时钟。

定时器的配置决定了freq\_hz频率，clk\_cfg时钟源以及占空比分辨率duty\_resolution。还有就是选择需要配置哪个定时器timer\_num。

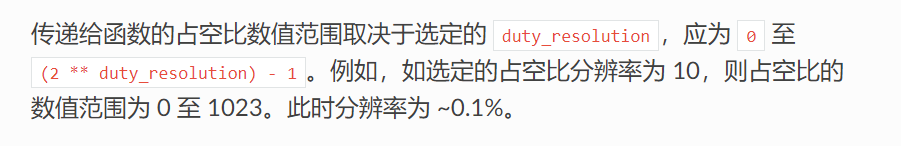
ledc\_channel\_config\_t结构体和ledc\_channel\_config用于配置定时器通道

其需要配置通道号channel，定时器号timer\_sel，映射到的物理IO口gpio\_num还有占空比duty。对于我们的ESP32-C3-12F开发板，根据原理图可查得其板载的RGB灯是使用IO3,4,5连接，并且为共地RGB灯。



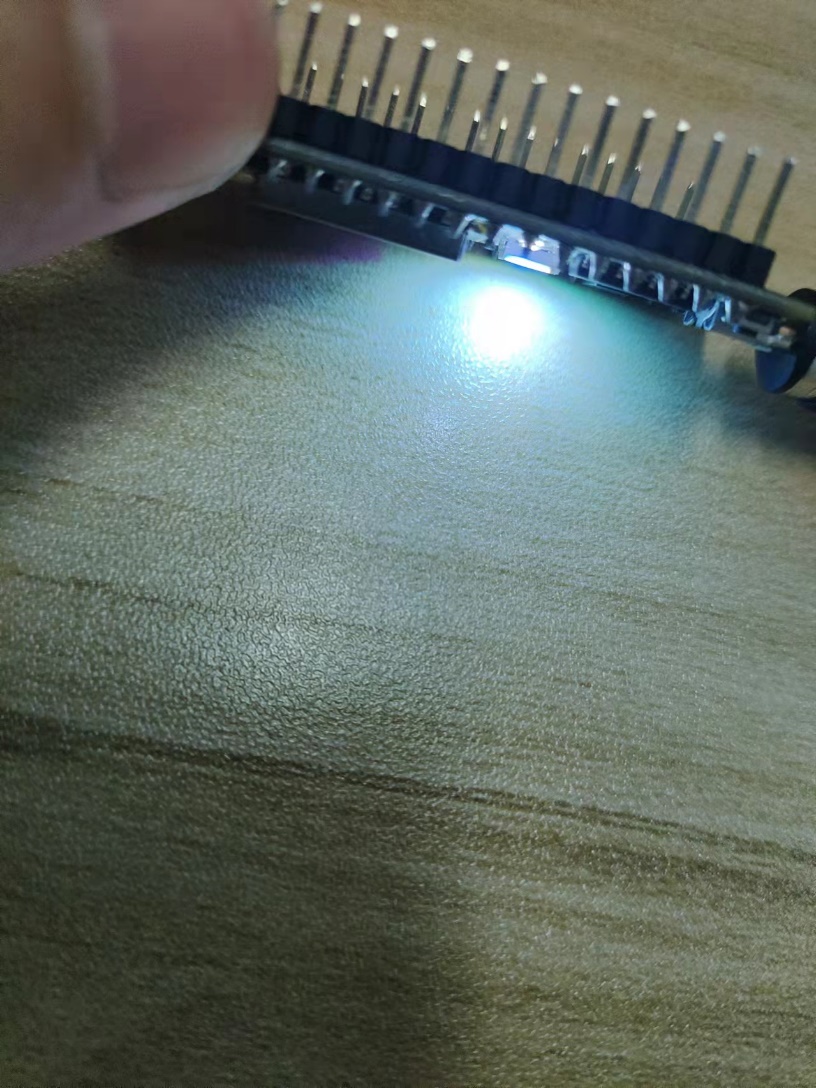
在这段代码里我们配置了其物理IO口映射到GPIO5，下面还有两段相似的代码用于配置IO4和IO3。

查阅资料时我们发现duty用于占空比调节，但是其范围不是我们想象的0%到100%的调节，而是0~(2 ^ duty\_resolution) – 1，duty\_resolution在这里使用的是13bit的宏定义，即其Duty调整范围为0~8191。



颜色上，我们希望灯的颜色可以做到我们想要的颜色，我们想到可以让其模拟RGB颜色中的任意颜色，当其为白色时，则RGB三个通道的占空比全部设置为1即可，此时降低任意颜色的占空比即可混合出不同的颜色。

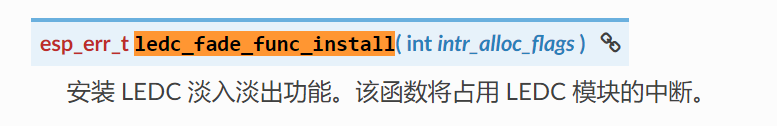
如颜色代码#66ccff，RGB通道颜色分别为102，204，255。对应占空比0.4,0.8,1.0。对应duty约为3276,6552,8190。其中8190如上图所配置。

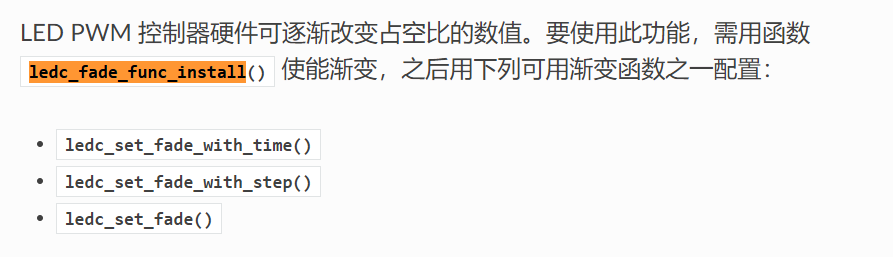


如下图所示，现在它正在显示#66ccff色（由于相机与亮度原因，其可能存在色偏）

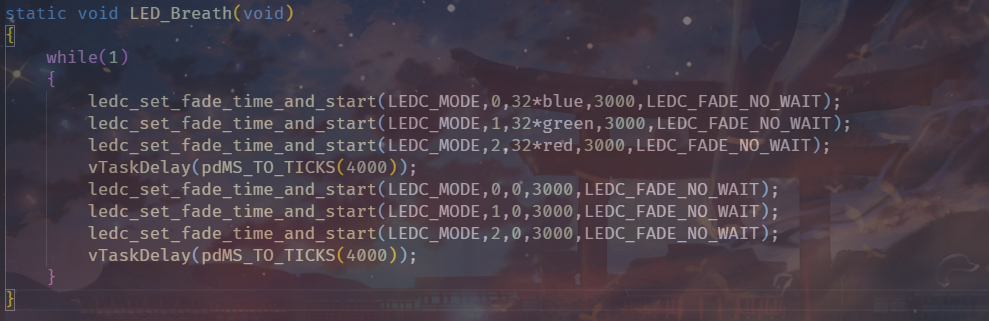
对于呼吸，我们发现LED PWM 控制器有一个函数可以直接后台控制修改占空比以做到呼吸的功能

首先调用ledc\_fade\_func\_install方法安装该功能后即可使用其方法





使用ledc\_set\_fade\_time\_and\_start方法启动duty的调控



其中，blue，green和red变量分别存放0~255的RGB颜色值

我们把该功能封装为LED\_Breath函数，并启动该任务

对于xTaskCreate，其作用为创建动态任务，是FreeRTOS的基础任务创建函数。

被创建的任务所调用的函数必须放在while里，除非调用任务删除释放函数

xTaskCreate的常用调用方法与参数为

1. xReturn = xTaskCreate((TaskFunction\_t )AppTaskCreate,*/\* 任务入口函数 \*/*
2. (const char\* )"AppTaskCreate",*/\* 任务名字 \*/*
3. (uint16\_t )512, */\* 任务栈大小 \*/*
4. (void\* )NULL,*/\* 任务入口函数参数 \*/*
5. (UBaseType\_t )1, */\* 任务的优先级 \*/*
6. (TaskHandle\_t\*)&AppTaskCreate\_Handle));*/\*任务控制块指针\*/*

使用

xTaskCreate(&LED\_Breath, "LED\_Breath", 2048, NULL, 12, NULL);

然后即可实现呼吸的功能。

1. 使用Uart串口实现调色功能

前文所述，我们实现了显示#66ccff色的呼吸灯，但是我们更进一步想进行灯的变色

我们想到了使用Uart串口