

## ESP8266 SOC 方案实现远程测控系统

### 实现功能：

- 1.控制一个大功率 LED 灯和一个继电器
- 2.控制 RGB 灯珠，实现颜色调节
- 3.实时获取温湿度（DHT11）
- 4.一个按键短按控制大功率 LED 灯珠亮和灭（同时上报状态），长按实现 SOFTAP 模式；一个按键控制继电器的开和关（同时上报状态），长按实现 AIRLINK 模式。

### 硬件准备：

- 1.ESP12F（32Mbit）一个
- 2.DHT11 一个
- 3.RGB\_LED 一个（由于是 3V 共阴，所以暂时不加限流电阻）
- 4.电容电阻排针若干（参考电路图）
- 5.微动开关 3 个（复位，继电器，LED 灯控制）
- 6.万用板一块
- 7.5V 继电器一个（可以用模块。也可以直接搭建继电器电路）
- 8.3.3V 稳压一个 1117

### GPIO 接线对应

GPIO12-----RGB\_LED 红

GPIO13-----RGB\_LED 绿

GPIO14-----RGB\_LED 蓝

GPIO15-----DHT11 数据

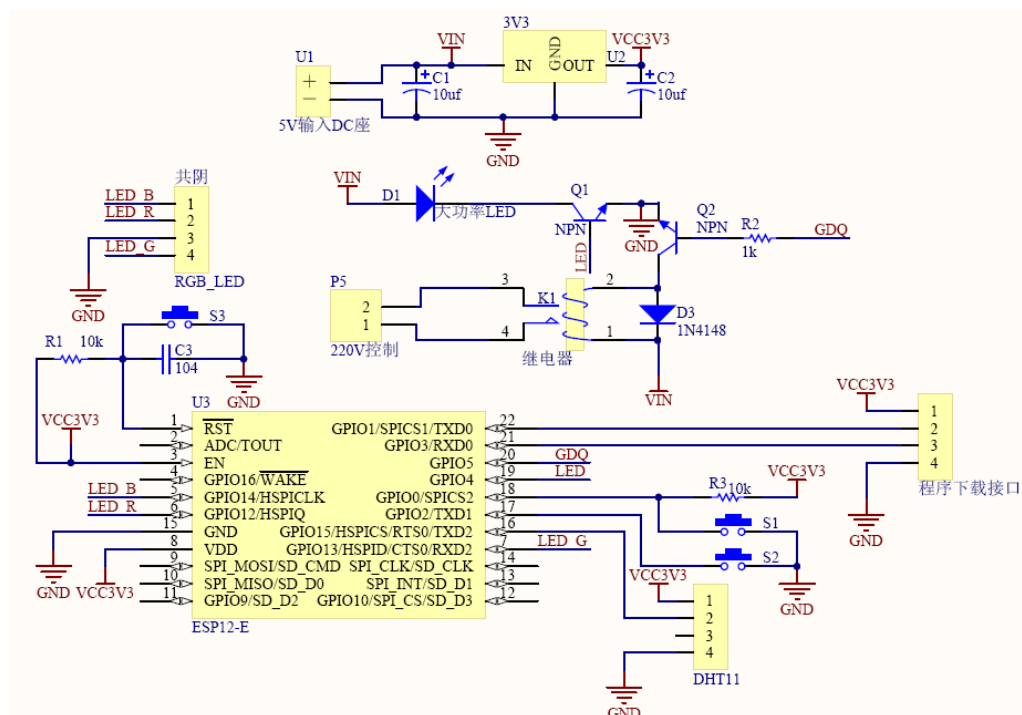
GPIO4-----大功率 LED 控制引脚（驱动 NPN 三极管）

GPIO5-----继电器控制引脚（驱动 NPN 三极管）

GPIO0-----LED 开关控制（SOFTAP）

GPIO2-----继电器开关控制（AIRLINK）

### 设计电路图：

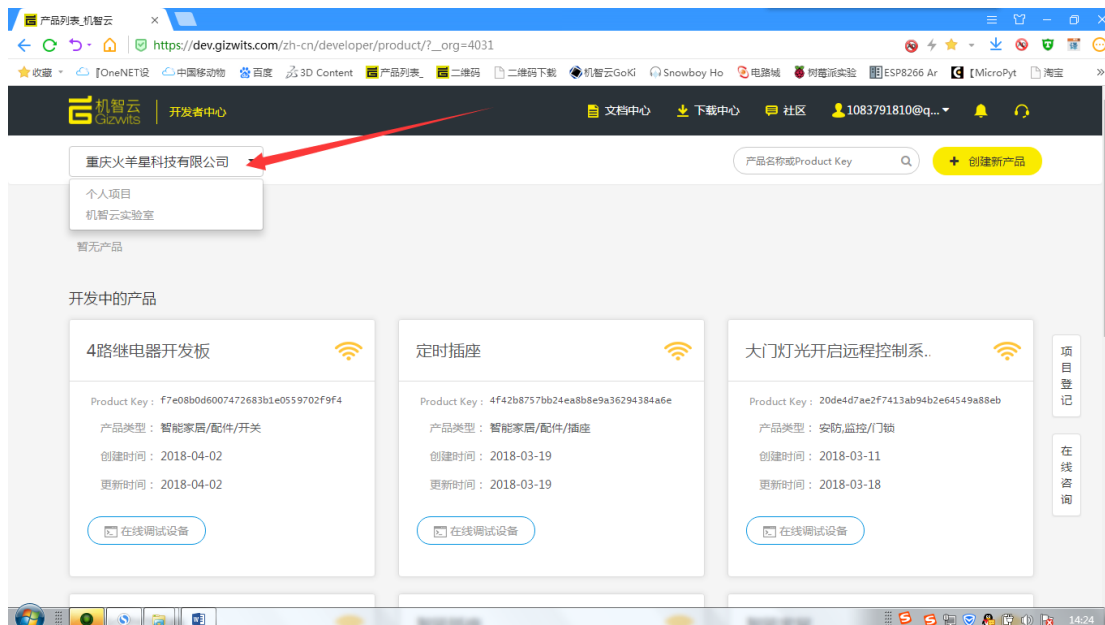


## 制作教程：

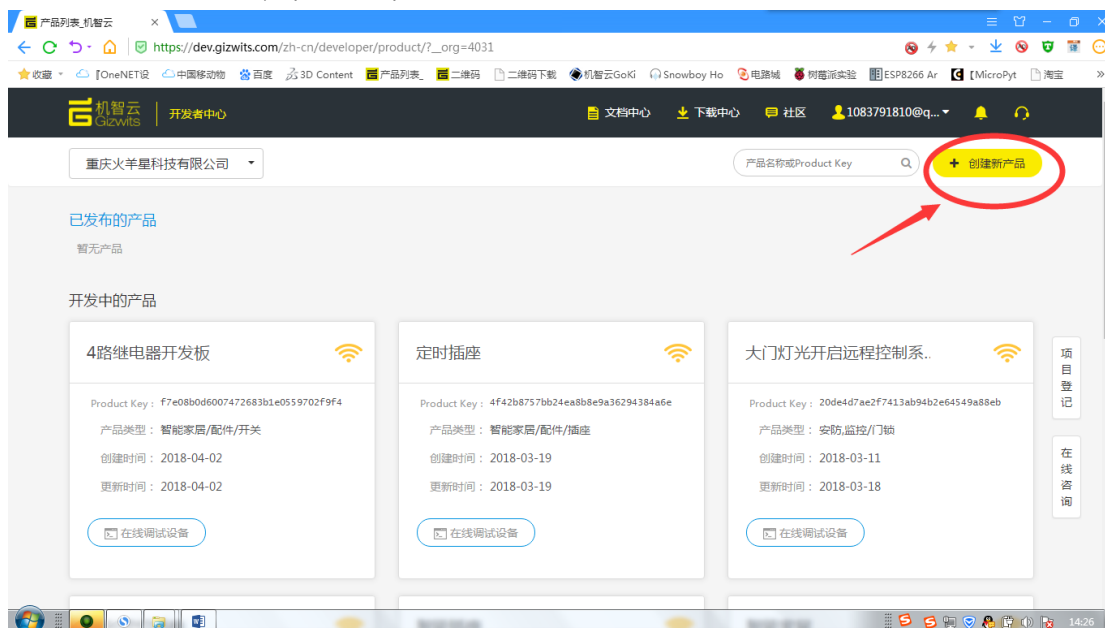
首先进入[开发者中心](https://dev.gizwits.com/zh-cn/developer/)：<https://dev.gizwits.com/zh-cn/developer/>进行账号的注册和登陆。认证之类的自己按照实际情况去认证。此处不在详述，不会可以加 Q 问我：

[1083791810](https://t.me/1083791810)tencent://message/?uin=1083791810&Site=fdfdf&Menu=yes

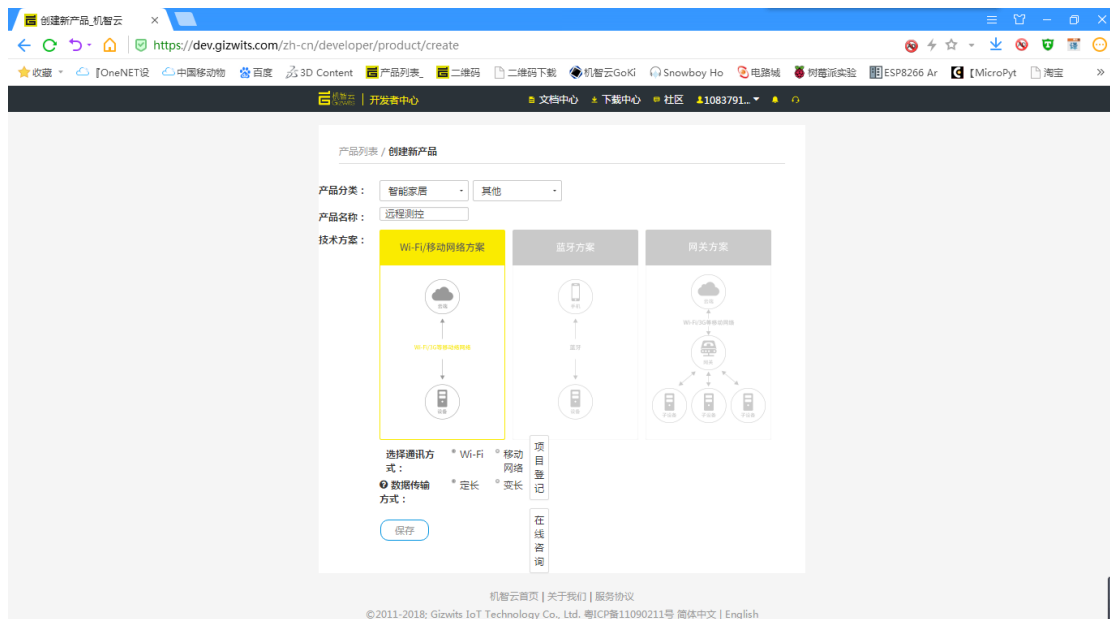
登陆认证过后进入个人项目或者公司项目，都是一样的，看你认证的是个人还是公司。在公司下面创建产品以后就只能在公司目录下浏览产品信息。



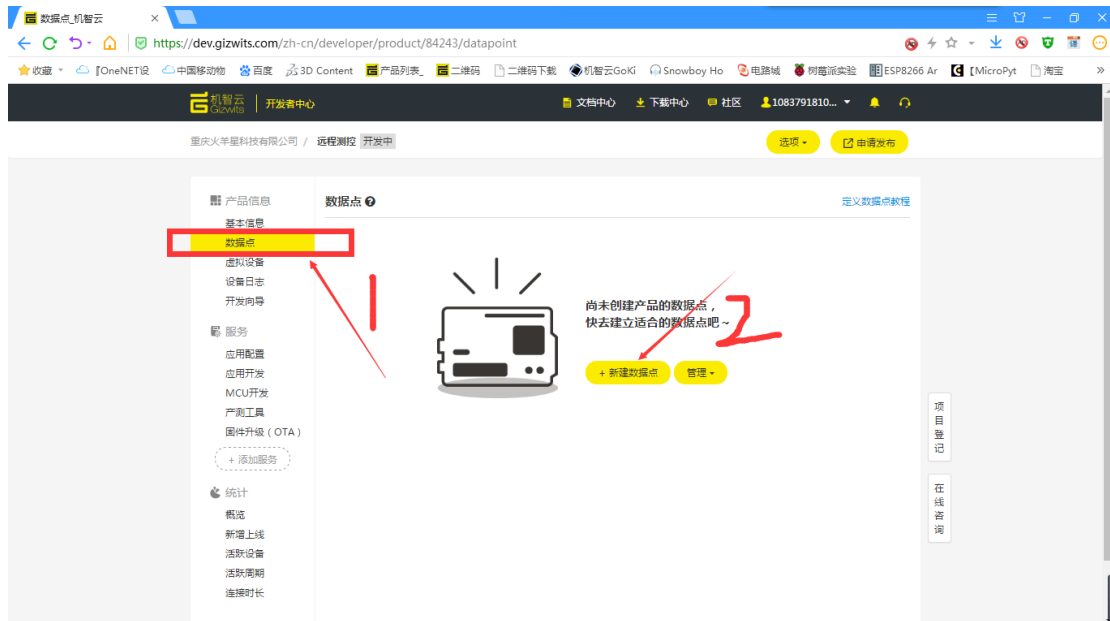
教程从这儿开始，首先点击右上角创建新产品。



按照如图所示设置好产品参数，产品分类随便选相应的就行，乱选不会影响产品；产品名称就是给产品取个名字，可以是中文和英文，此处为了教学，我就写一个远程测控；技术方案选择 wifi/移动网络方案；通讯方式选择 wifi（此处因为是 soc 方案，所以直接用 wifi）；数据传输方式选择定长（定长就是把所有数据点一次全部传输，变长就是只传输变化那部分，个人建议在数据点不多的情况下最好选用定长，在数据点很多的时候才考虑变长），按照如图设置好之后直接点击保存。就完成了产品的创建，会自动跳转到产品页面。



接下来我们需要给产品添加数据点，数据点是网络传输的标识，同一个产品中每个数据点标识名都必须是唯一的，比如有多个开关，每个开关都必须给予唯一的标识名，而且标识名必须是英文，数字，或者 2 个的组合。数据点需要设置三个参数，标识名，读写类型，数据类型，按照下图 1 2 操作可以增加数据点



关于三个参数按照以下规定：(对于不理解数据点的可以加 QQ 问我，先阅读看不懂再问)  
**标识名：**

用于应用层传输，客户端或业务云开发时需要使用。命名规则遵循标准的开发语言变量命名规范，支持英文字母、数字和下划线，以英文字母开头。

**读写类型：**(比如一个继电器的开关量需要在 APP 上进行开关操作，就设置可写，不需在 APP 上操作就只读，设置只读之后 APP 就只能观察状态而不能控制，温度湿度就是只读的，判断温湿度是否不在规定值就是报警，判断 DHT11 是不是正常接线就是故障)

- ① 只读：表示该数据点非控制，数据只支持从设备上报。
- ② 可写：表示该数据点可控制。设备端可上报该数据点数据；云端/客户端可对该数据点

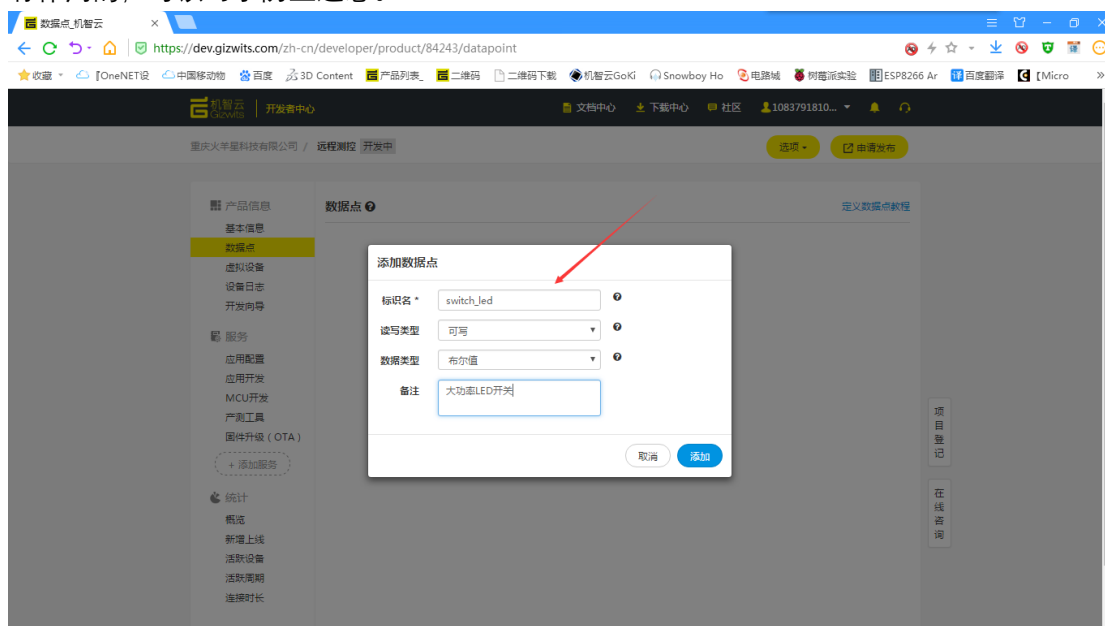
数据做出下发控制。

- ③ 报警：表示该数据点非控制，数据只支持从设备上报，数据类型需为布尔值。
- ④ 故障：表示该数据点非控制，数据只支持从设备上报，数据类型需为布尔值。云端会对设备上报的该数据点做统计，可在“运行状态”查看。

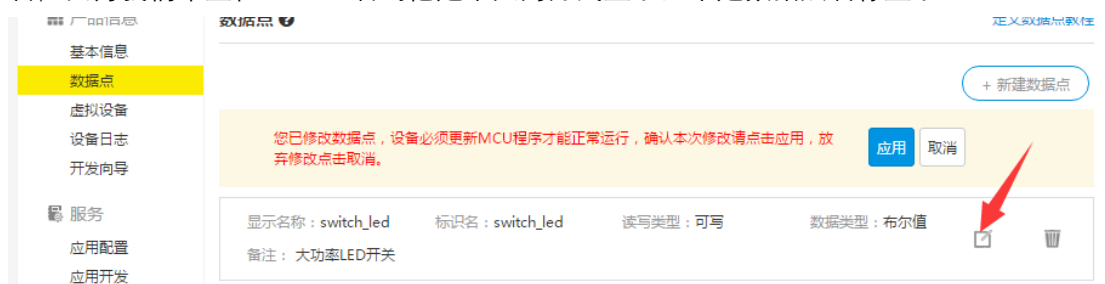
**数据类型：**（比如继电器的开关，就是布尔值，温湿度就是数值）

- ① 布尔值：表示两个状态：0，或 1。如开关状态等，建议使用布尔数据类型。例如打开关闭继电器，打开关闭大功率 LED。
- ② 枚举类型：可定义一个有限的取值集合。当定义的某个功能（元器件）有固定的若干个值。例如设置 RGB 为关闭,自定义组合,红色,绿色,蓝色,黄色,紫色,粉色,白色。
- ③ 数值：填写数值范围，数值可为负数/小数，机智云自动将数值转换为正数。例如温湿度，RGB 色度调节控制等。
- ④ 扩展：填写数据长度，数据内容由用户自定义。对于上述功能点无法满足的复杂功能可采用。机智云不建议使用此类型数据，设备上报该数据点的数据，机智云无法识别。

根据以上内容，我们开始创建数据点，首先建立大功率 LED 的数据点，此处我们就用 switch\_led 表示，由于要在 APP 操作其开关，就选择可写，开关量直接用布尔值。备注没有作用的，可以写了防止遗忘。



设置好参数之后点击确定，然后按照如图箭头所示可以编辑这个数据点，为什么要进行编辑，因为我們希望在 APP 上看到他是中文的方式显示而不是数据点名称显示。



点击编辑之后标识名不用改，改显示名称，数据点创建错了也可以在这儿进行调整修改。此处我修改成电灯。如下图，修改好之后点击确定。

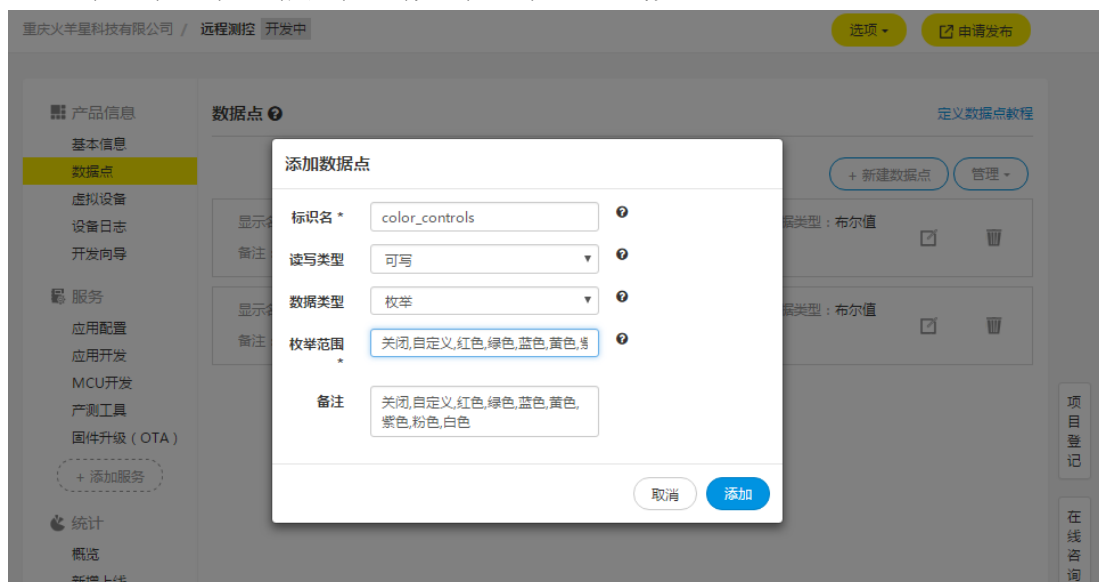
显示名称：电灯	标识名：switch_led	读写类型：可写	数据类型：布尔值		
备注：大功率LED开关					

同样的方法创建继电器的数据点，标识名 switch\_relay，显示为插座，可写，创建好之后如下图所示

显示名称：电灯	标识名：switch_led	读写类型：可写	数据类型：布尔值		
备注：大功率LED开关					

显示名称：插座	标识名：switch_relay	读写类型：可写	数据类型：布尔值		
备注：无					

接着我们创建一个颜色控制的枚举可控数据点，用来快速控制 RGB 灯的颜色。枚举范围一定要是英文输入下的逗号。数据点标识名为 color\_controls，可写，枚举，枚举范围：关闭,自定义,红色,绿色,蓝色,黄色,紫色,粉色,白色，显示名称：RGB 控制



显示名称：电灯	标识名：switch_led	读写类型：可写	数据类型：布尔值		
备注：大功率LED开关					

显示名称：插座	标识名：switch_relay	读写类型：可写	数据类型：布尔值		
备注：无					

显示名称：RGB控制	标识名：color_controls	读写类型：可写	数据类型：枚举		
枚举范围：0.关闭 1.自定义 2.红色 3.绿色 4.蓝色 5.黄色 6.紫色 7.粉色 8.白色					
备注：关闭,自定义,红色,绿色,蓝色,黄色,紫色,粉色,白色					

接下来创建 3 个 RGB 自由调节的可写数值数据点：标识名分别为：LED\_R，LED\_G，LED\_B，可写，数据类型：数值，数据范围 0-254，分辨率 1。显示名称红色(R)| 绿色(G) 蓝色(B)，创建好之后如图所示。

显示名称：红色(R)	标识名：LED_R	读写类型：可写	数据类型：数值		
数据范围：0 - 254	分辨率：1	增量：0			
备注：RGB灯红色调节					

显示名称：绿色(G)	标识名：LED_G	读写类型：可写	数据类型：数值		
数据范围：0 - 254	分辨率：1	增量：0			
备注：RGB灯绿色调节					

显示名称：蓝色(B)	标识名：LED_B	读写类型：可写	数据类型：数值		
数据范围：0 - 254	分辨率：1	增量：0			
备注：RGB灯蓝色调节					

接下来温湿度显示，范围就取正常的 DHT11 可控范围。温度范围 0-50.湿度 0-100，设置好后如下图所示，创建完之后点击最上面的应用

显示名称：温度(°C) 标识名：temperature 读写类型：只读 数据类型：数值 数据范围：0 - 50 分辨率：1 增量：0 备注：DHT11 测量温度

显示名称：湿度(%) 标识名：humidity 读写类型：只读 数据类型：数值 数据范围：0 - 100 分辨率：1 增量：0 备注：DHT11 测量湿度

显示名称：温度(°C)	标识名：temperature	读写类型：只读	数据类型：数值		
数据范围：0 - 50	分辨率：1	增量：0			
备注：DHT11测量温度					

显示名称：湿度(%)	标识名：humidity	读写类型：只读	数据类型：数值		
数据范围：0 - 100	分辨率：1	增量：0			
备注：DHT11测量湿度					

关于对分辨率和精度的理解可以参考这段说明，在定义数值型数据点的时候，取值范围可以使用包括小数、负数等非 uint 类型数值，熟悉嵌入式开发的开发者会知道，这些数值在设备端都是不被支持的。

机智云为了让设备功能定义更加简单直接、所见即所得，研究出来一套算法，用于将用户输入的数值转换成设备能够识别的 uint 类型，这套算法的核心公式是： $y=kx+m$ 。

y 表示“显示值”，就是用户可见的最终数值，也是数据点定义时输入的值。包括 Ymin(最小值) 和 Ymax(最大值)。

x 表示“传输值”，就是实际指令间传输使用的数值，云端/客户端接收到的值。一定是 uint 格式。也包括 Xmin 和 Xmax。

k 表示“分辨率”，就是用户输入的分辨率一值，确定了每个取值的步进。

m 表示“取值偏移量”或“增量”，算法通过 m 值将 y 值偏移以满足 x 值 uint 格式的要求，m 值默认等于 Ymin，确保 Xmin = 0。

以下用一个电子温度计举例说明换算过程 数据点内容：取值范围：-30 (Ymin) ~ 50 (Ymax)，分辨率：0.1

根据公式： $y=kx+m$ ，m 默认等于 Ymin -30

$Xmin = (-30+30) / 0.1 = 0$

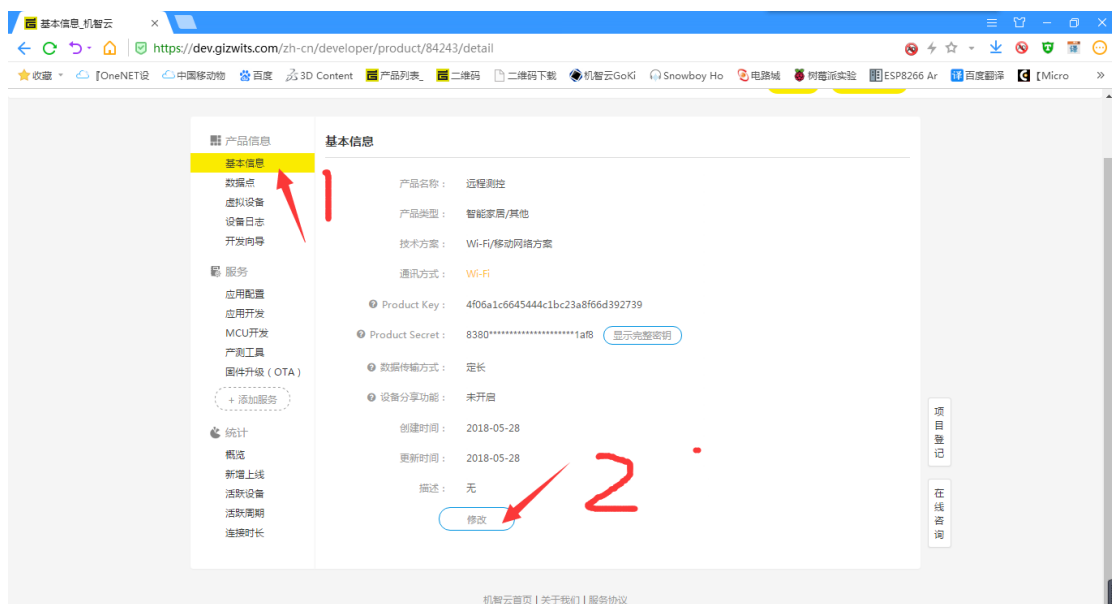
$Xmax = (50+30) / 0.1 = 800$

所有数据点如下表格所示，到这儿所有的数据点就创建完毕，可以从以下表中进行

复制粘贴。

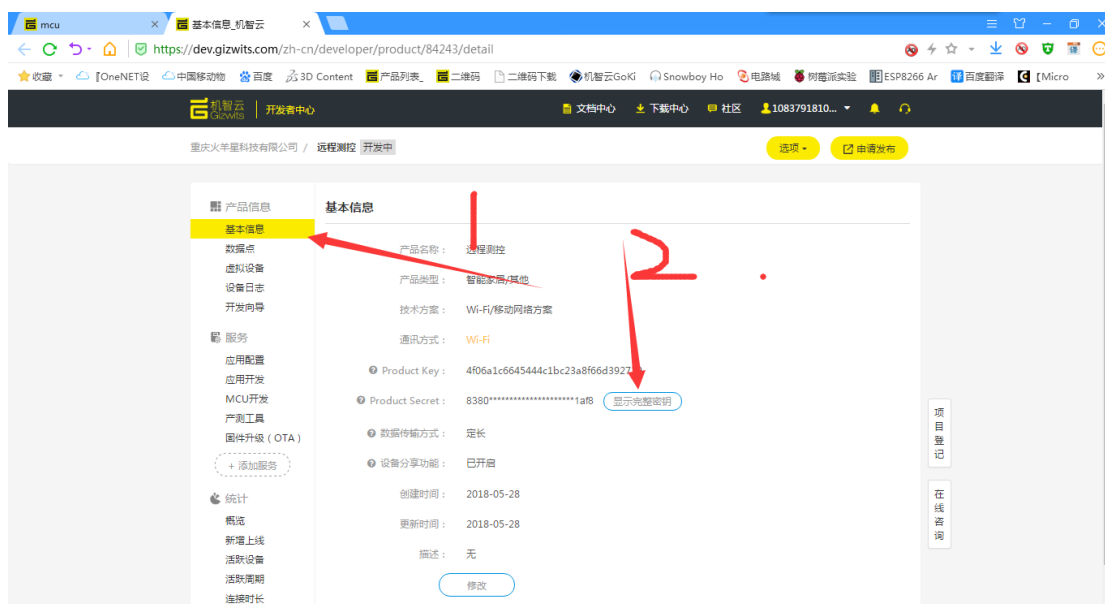
标识名	备注	读写类型	数据类型	数值				枚举选项	扩展长度
				分辨率	增量	数据范围最小值	数据范围最大值		
switch_led	大功率 LED 开关	可写	布尔值						
switch_relay		可写	布尔值						
color_controls	关闭,自定义,红色,绿色,蓝色,黄色,紫色,粉色,白色	可写	枚举					关闭,自定义,红色,绿色,蓝色,黄色,紫色,粉色,白色	
LED_R	RGB灯红色调节	可写	数值	1	0	0	254		
LED_G	RGB灯绿色调节	可写	数值	1	0	0	254		
LED_B	RGB灯蓝色调节	可写	数值	1	0	0	254		
temperature	DHT11测量温度	只读	数值	1	0	0	50		
humidity	DHT11测量湿度	只读	数值	1	0	0	100		

设备共享的开启：设备共享开启之后，可以通过 APP 进行分享，具体开启方法是电机基本设置，修改。设备分享功能：后面点一下开启，弹出框选择开启，就完成设备分享功能。



接下来我们需要生成 SOC 的代码。生成之前我们先去基本信息里面复制 Product Secret，点击显示完整密钥，输入登录密码，查看就能复制了，复制之后保存一会儿会用到。





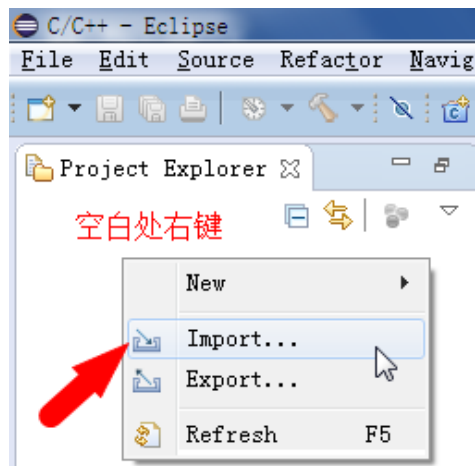
复制好之后点击左侧的 MCU 开发，然后选择 SOC 方案，32M，粘贴刚才复制的 Product Secret，然后点击下面的生成代码包，等到进度条完成之后，代码就生成了，下载代码所生成，到这儿，所有的云端工作就已经完成了。接下来就是代码的修改。



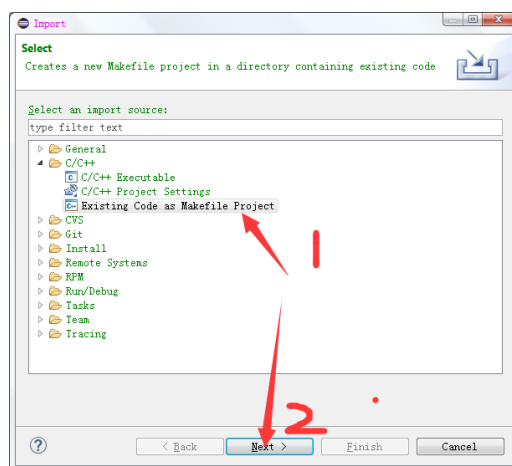
接下来是代码的修改，修改代码和编译代码用的是乐鑫的专用 IDE，没有开发环境或者需要这个开发环境的可以去群 [438373554](https://jq.qq.com/?_wv=1027&k=5nxGDB1) [https://jq.qq.com/?\\_wv=1027&k=5nxGDB1](https://jq.qq.com/?_wv=1027&k=5nxGDB1) 的群文件进行下载，安装遇到问题可以 QQ 联系我，

安装软件之后打开软件，在软件的左侧导航右键，导入项目（Import），

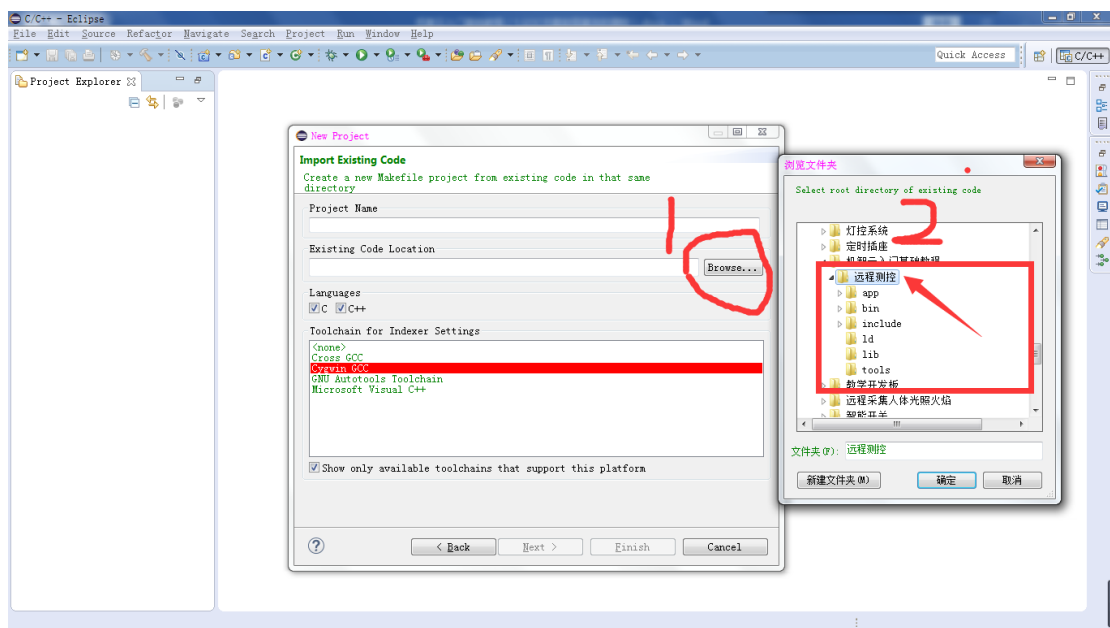




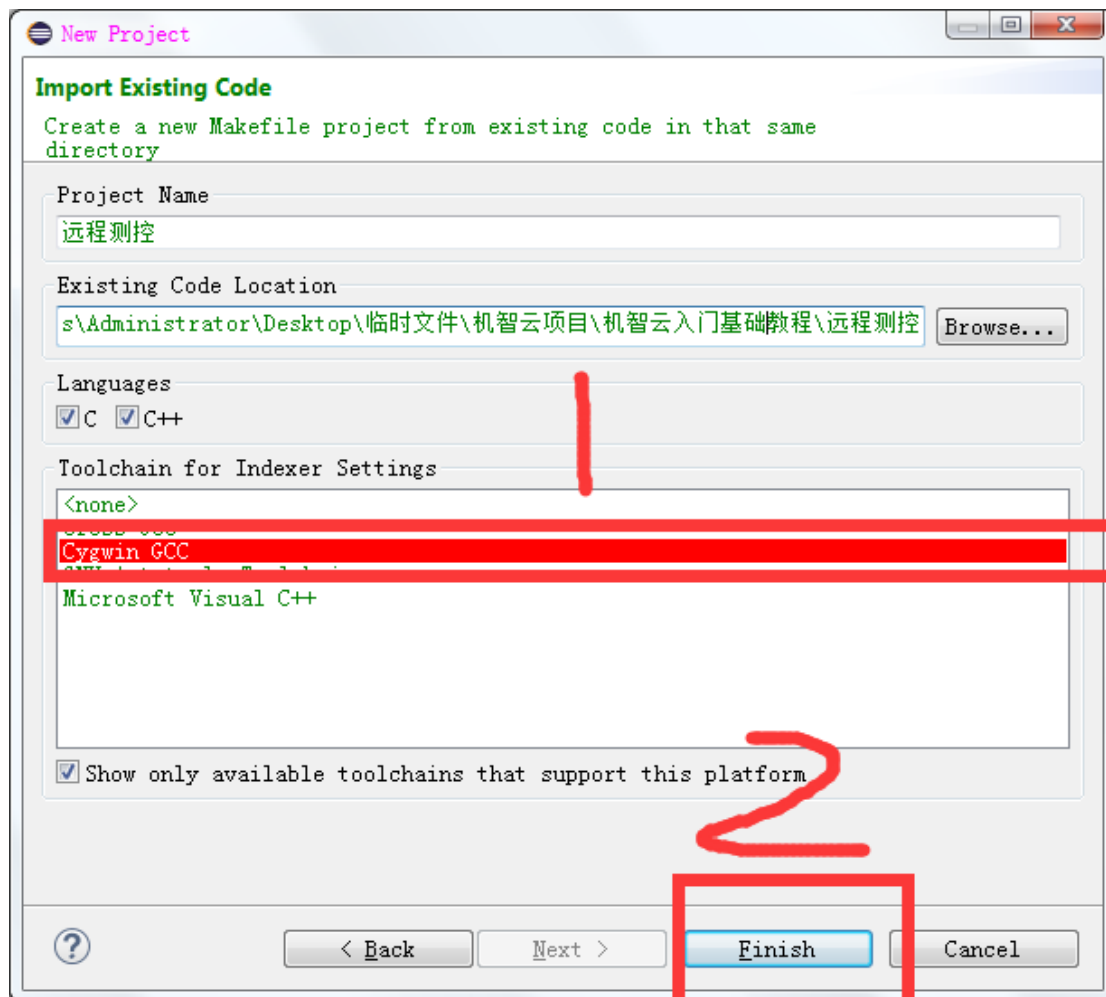
选择 Makefile 项目，然后点击下一步，



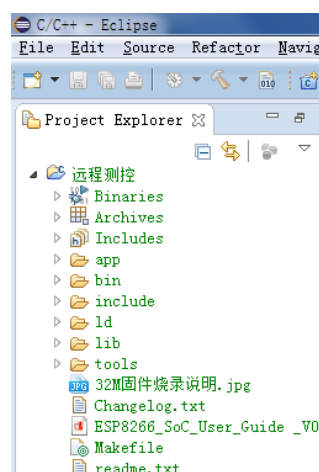
选择 BROWSE 按钮，载入 Makefile 所在的根目录（这个目录很重要，一定要是解压过后的 Makefile 所在的根目录，目录下面包含了 APP,BIN ID 等多个文件夹，如果选错，将不能正确导入项目。）



选择好目录之后，需要选择编译器这儿我们用 Cygwin GCC 编译器，然后点击 Finish 就能成功导入项目。



成功导入的项目如图所示，到这儿我们的项目已经成功导入到了软件之中，我们接下来要进行代码修改。



首先我们先要修改根目录下面的 Makefile 文件，双击打开文件，找到文件的 22 行，我们需要修改其内容，其目的是对应我们单片机的 flash 的通讯方式。将

BOOT?=none

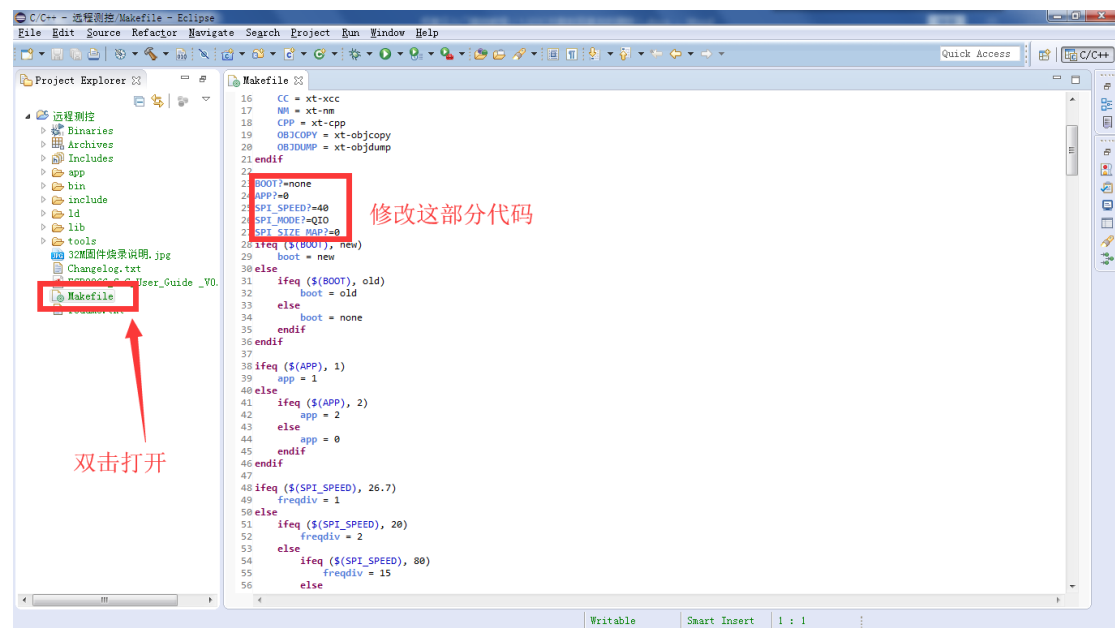
APP?=0

SPI\_SPEED?=40  
SPI\_MODE?=QIO  
SPI\_SIZE\_MAP?=0

内容修改为：

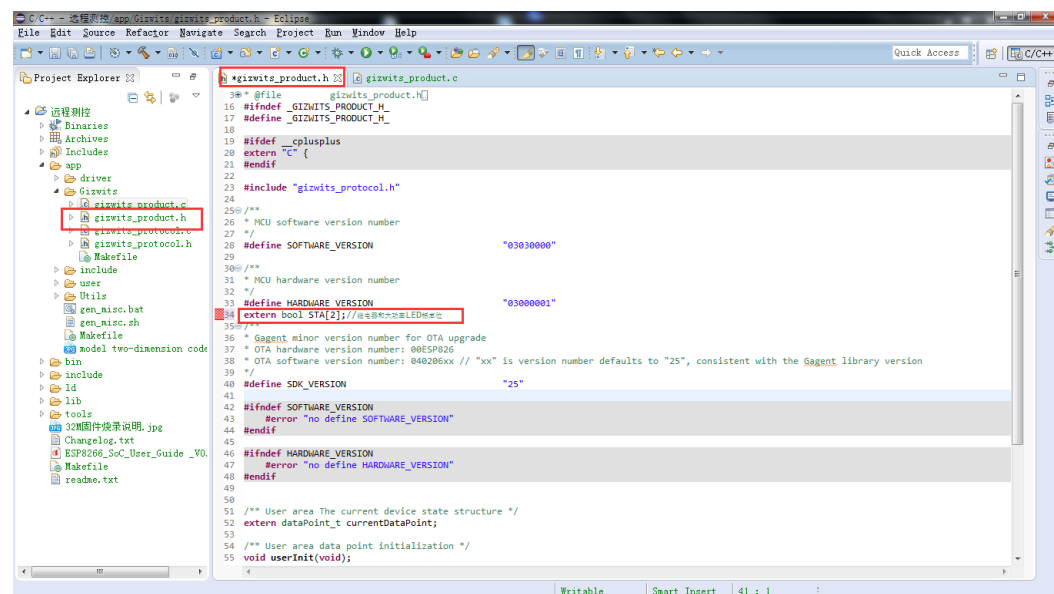
BOOT?=new  
APP?=1  
SPI\_SPEED?=40  
SPI\_MODE?=QIO  
SPI\_SIZE\_MAP?=6

修改完成之后保存关闭。

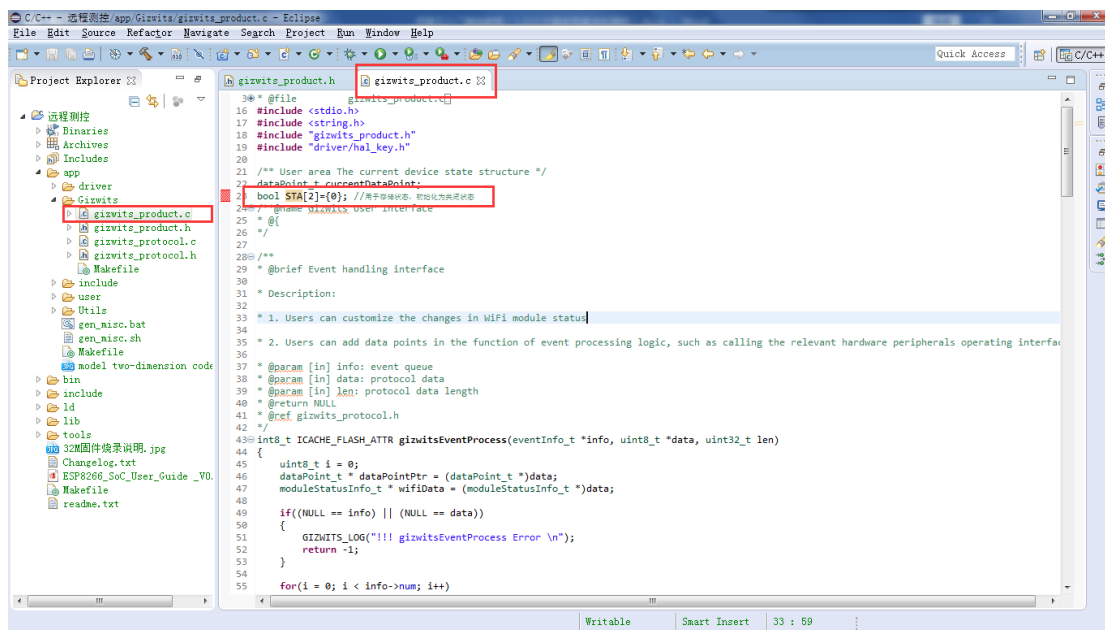


接下来我们创建一个布尔数组来保存继电器和大功率 LED 的开关状态，我们需要在 gizwits\_product.h 文件里面定义，然后再 gizwits\_product.c 里面进行全局定义，方便其他文件进行调用。

在 gizwits\_product.h 文件里面添加代码 `extern bool STA[2];` //继电器和大功率 LED 标志位



在 gizwits\_product.c 里面添加代码 bool STA[2]={0}; //用于存储状态，初始化为关闭状态



接下来我们需要修改按键，实现 2 个按键对应的功能，打开 user\_main.c 文件。需要对长短按键进行修改，首先我们原理图用的是 GPIO0 和 GPIO2 引脚，GPIO0 对应的是 LED 开关控制和 SOFTAP（长按），GPIO2 对应的是继电器开关控制和 AIRLINK（长按）。

找到 user\_main.c 里面的按键重定义，在第 45 行附近：

```
#define KEY_0_IO_MUX      PERIPHS_IO_MUX_GPIO0_U
#define KEY_0_IO_NUM      0
#define KEY_0_IO_FUNC     FUNC_GPIO0
#define KEY_1_IO_MUX      PERIPHS_IO_MUX_MTMS_U
#define KEY_1_IO_NUM      14
#define KEY_1_IO_FUNC     FUNC_GPIO14
```

将其内容修改为：

```
#define KEY_0_IO_MUX      PERIPHS_IO_MUX_GPIO0_U
#define KEY_0_IO_NUM      0
#define KEY_0_IO_FUNC     FUNC_GPIO0
#define KEY_1_IO_MUX      PERIPHS_IO_MUX_GPIO2_U
#define KEY_1_IO_NUM      2                                     ///  
#define KEY_1_IO_FUNC     FUNC_GPIO2
```

PERIPHS\_IO\_MUX\_GPIO2\_U 和 FUNC\_GPIO2 是在头文件 eagle\_soc.h 当中 240 行左右。

修改完定义之后我们需要修改长短按键的内容，将功能修改成我们之前设定好的。

GPIO0-----短按 L E D 开关控制，长按 SOFTAP 模式

GPIO2-----短按继电器开关控制，长按 AIRLINK 模式

将下面的内容：

```
LOCAL void ICACHE_FLASH_ATTR key1ShortPress(void)
{
    GIZWITS_LOG("#### KEY1 short press ,Production Mode\n");
    gizwitsSetMode(WIFI_PRODUCTION_TEST);
}
```

```
LOCAL void ICACHE_FLASH_ATTR key1LongPress(void)
{
    GIZWITS_LOG("#### key2 short press, soft ap mode \n");
    gizwitsSetMode(WIFI_SOFTAP_MODE);
}
LOCAL void ICACHE_FLASH_ATTR key2ShortPress(void)
{
    GIZWITS_LOG("#### key2 short press, soft ap mode \n");
    gizwitsSetMode(WIFI_SOFTAP_MODE);
}
LOCAL void ICACHE_FLASH_ATTR key2LongPress(void)
{
    GIZWITS_LOG("#### key2 long press, airlink mode\n");
    gizwitsSetMode(WIFI_AIRLINK_MODE);
}
```

**修改为：**

```
LOCAL void ICACHE_FLASH_ATTR key1ShortPress(void)
{
    STA[0]=!STA[0];//说明一下，此处的取反就是短按一次开关状态改变一次
    if(STA[0]) GIZWITS_LOG("#### GPIO0-ShortPress,High-power-LED ON \n");
    else GIZWITS_LOG("#### GPIO0-ShortPress,High-power-LED OFF \n");
}
LOCAL void ICACHE_FLASH_ATTR key1LongPress(void)
{
    GIZWITS_LOG("#### GPIO0-LongPress,soft ap mode \n");
    gizwitsSetMode(WIFI_SOFTAP_MODE);
}
LOCAL void ICACHE_FLASH_ATTR key2ShortPress(void)
{
    STA[1]=!STA[1]; //说明一下，此处的取反就是短按一次开关状态改变一次
    if(STA[1]) GIZWITS_LOG("#### GPIO2-ShortPress,Relay ON \n");
    else GIZWITS_LOG("#### GPIO2-ShortPress,Relay OFF \n");
}
LOCAL void ICACHE_FLASH_ATTR key2LongPress(void)
{
    GIZWITS_LOG("#### GPIO2-LongPress,airlink mode\n");
    gizwitsSetMode(WIFI_AIRLINK_MODE);
}
```

到此按键部分就修改完成了，接下来我们修改继电器和大功率的控制部分，要作为输出 IO，首先就要对其进行初始化。在这儿可以新建一个初始化函数或者直接在初始化按键的函数里面进行初始化。

找到按键初始化函数 LOCAL void ICACHE\_FLASH\_ATTR keyInit(void)

**在函数末尾增加以下内容：**

```
PIN_FUNC_SELECT(PERIPHS_IO_MUX_GPIO4_U, FUNC_GPIO4);//配置大功率 LED 管脚
```

为输出

```
GPIO_DIS_OUTPUT(GPIO_ID_PIN(4));
PIN_FUNC_SELECT(PERIPHS_IO_MUX_GPIO5_U, FUNC_GPIO5); //配置继电器管脚为输出
GPIO_DIS_OUTPUT(GPIO_ID_PIN(5));
GPIO_OUTPUT_SET(GPIO_ID_PIN(4), 1); //输出高电平
GPIO_OUTPUT_SET(GPIO_ID_PIN(5), 1); //输出高电平
```

增加初始化之后需要处理云端的开关指令，找到 gizwits\_product.c 的 gizwitsEventProcess 函数，需要对 valueswitch\_led 和 valueswitch\_relay 进行修改：

首先修改 valueswitch\_led，找到 gizwitsEventProcess 函数里面的 valueswitch\_led 的响应程序：if(0x01 == currentDataPoint.valueswitch\_led)

```
{
//user handle
}
else
{
//user handle
}
```

修改其内容为：

```
if(0x01 == currentDataPoint.valueswitch_led)
{
    STA[0]=1; //打开大功率 LED
}
else
{
    STA[0]=0; //关闭大功率 LED
}
```

同样也修改下面的 valueswitch\_relay，修改为

```
if(0x01 == currentDataPoint.valueswitch_relay)
{
    STA[1]=1; //打开继电器
}
else
{
    STA[1]=0; //关闭继电器
}
```

修改完成之后我们找到定时上报函数：**userHandle(void)函数**

GPIO\_OUTPUT\_SET(GPIO\_ID\_PIN(4), !STA[0]); //大功率 LED 开关，NPN 型三极管，低电平有效，取反操作

GPIO\_OUTPUT\_SET(GPIO\_ID\_PIN(5), !STA[1]); //继电器开关，NPN 型三极管，低电平有效，取反操作

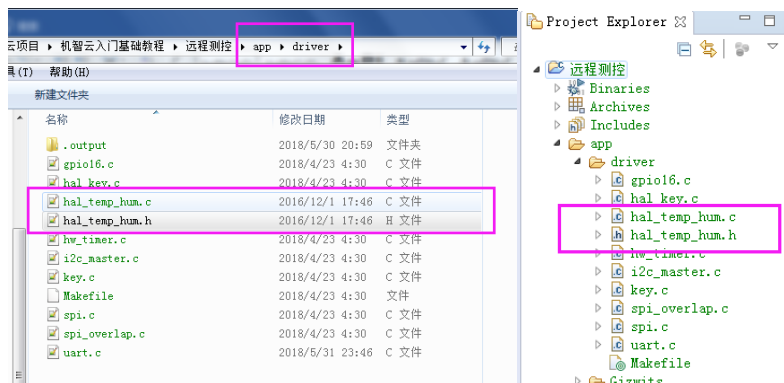
if(time\_updata==20) //定时 50ms\*20，修改定时器之后不改变上报频率

```
{
    time_updata=0;
    currentDataPoint.valueswitch_led = STA[0]; //大功率 LED 开关状态上报
```

```
currentDataPoint.valueswitch_relay = STA[1]; //继电器开关状态上报
}
else time_updata++;
```

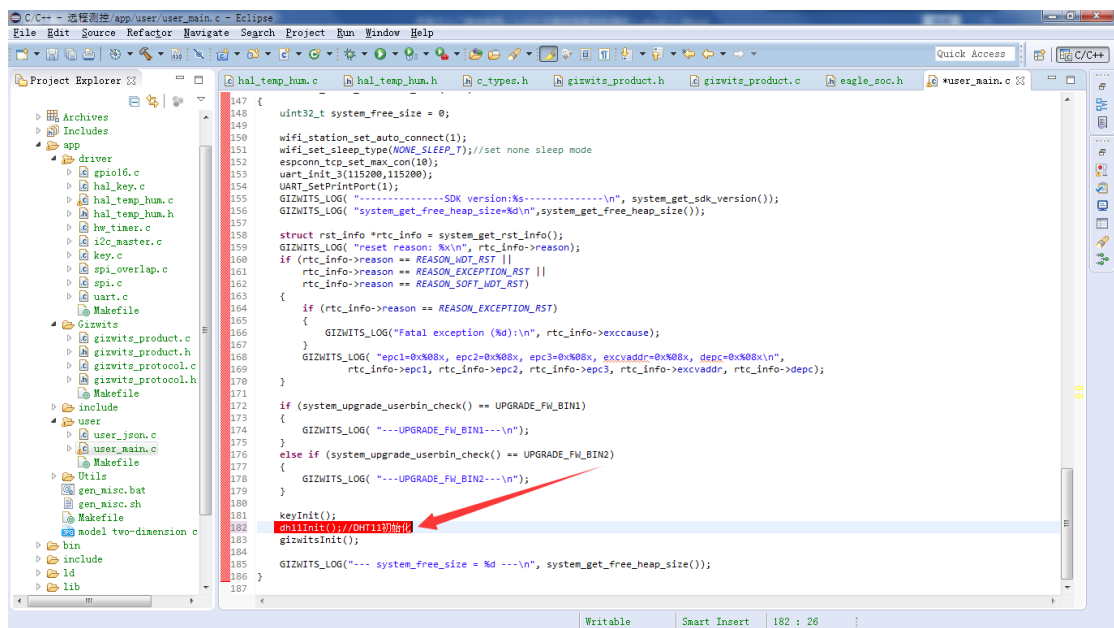
在函数里面增加上面内容，userHandle(void)函数是定时上报的函数，定时时间我们修改一下，修改成 50ms，默认 1 秒，因为我们需要把控制加在里面，50ms 可以减小控制延迟，修改方法是打开 user\_main.c 文件，找到 gizwits\_protocol.c 文件里面的#define USER\_TIME\_MS 1000 修改成#define USER\_TIME\_MS 50 然后需要定义一下 time\_updata，在 gizwits\_product.c 开头定义就行了。到此为止，继电器和 LED 已经修改完成。

接下来我们增加 DHT11 温湿度的函数，使其实现温湿度的采集，将驱动文件 hal\_temp\_hum.h 和 hal\_temp\_hum.c 文件复制到项目的 driver 文件夹下面。如下图所示。



增加进去之后驱动库我到时候会在文件里面直接打包上传，或者直接跟我索要也行。需要注意的是这儿使用的是 GPIO15，如果是其他 GPIO，请修改管脚配置相关内容。

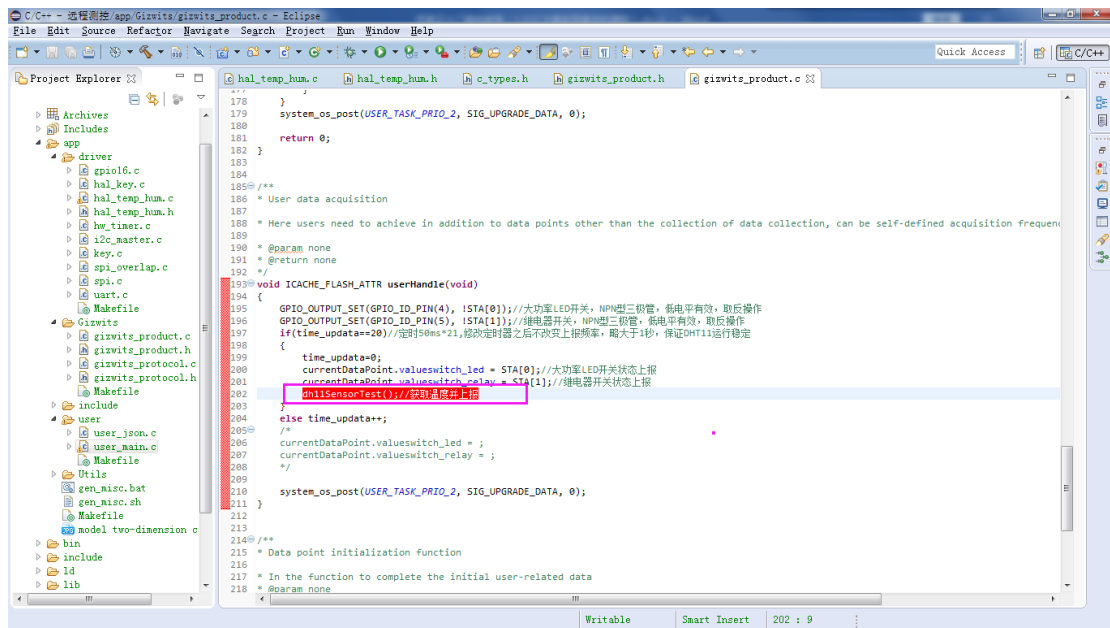
找到 gizwits\_product.c 文件，在文件靠头增加#include "driver/hal\_temp\_hum.h"头文件，增加头文件之后我们需要对 DHT11 进行初始化，找到 user\_main.c 文件，在最末尾的增加初始化 dh11Init(); //DHT11 初始化。如图



初始化完成之后我们需要对采集的数据进行上报，上报在 dh11SensorTest();函数里面的，所以我们在定时上报函数里面需要增加 dh11SensorTest(); //获取温度并上报，在增加后如



图所示。



接下来我们修改开机数据点上报的默认值。找到 gizwits\_product.c 文件最后的那个函数 void ICACHE\_FLASH\_ATTR userInit(void); 将其内容修改为：

```
void ICACHE_FLASH_ATTR userInit(void)
{
    gizMemset((uint8_t *)&currentDataPoint, 0, sizeof(dataPoint_t));
    /** Warning !!! DataPoint Variables Init , Must Within The Data Range **/
    currentDataPoint.valueswitch_led = 0;
    currentDataPoint.valueswitch_relay = 0;
    currentDataPoint.valuecolor_controls = 0;
    currentDataPoint.valueLED_R = 0;
    currentDataPoint.valueLED_G = 0;
    currentDataPoint.valueLED_B = 0;
    currentDataPoint.valuetemperature = 0;
    currentDataPoint.valuehumidity = 0;
}
```

修改完初始化上报之后我们需要修改全彩灯控制程序，这里我们直接利用#include "pwm.h"库，这儿我把驱动写成了 2 个独立的库文件 RGB\_light.h 和 RGB\_light.c，所以直接把我的库复制到驱动文件夹 driver 里面。然后再 user\_main.c 和 gizwits\_product.c 里面包含头文件。添加好之后在刚在初始化 DHT11 的位置在初始化一个 RGB，添加下面代码

```
RGB_light_init();//初始化
RGB_light_set_period(500);//设置周期
RGB_light_set_color(0,0,0);//关闭 rgb
```

添加完之后在 gizwits\_product.c 定义一个数组存放云端数据，用来缓存我们需要的数据。  
uint32\_t mode\_Cloud\_data[4]={0}; //云端数据缓存 0 模式 1R 2G 3B  
然后找到 gizwitsEventProcess 函数，处理云端数据，把数据缓存到我们创建的数组里面。  
主要修改下面几个数据点的缓存，

currentDataPoint.valuecolor\_controls)

currentDataPoint.valueLED\_R

currentDataPoint.valueLED\_G

currentDataPoint.valueLED\_B

**修改完成之后代码为：**

```
case EVENT_color_controls:
    currentDataPoint.valuecolor_controls = dataPointPtr->valuecolor_controls;
    GIZWITS_LOG("Evt: EVENT_color_controls %d\n",
currentDataPoint.valuecolor_controls);
    switch(currentDataPoint.valuecolor_controls)
    {
    case color_controls_VALUE0:
        mode_Cloud_data[0]=0;//关闭
        break;
    case color_controls_VALUE1:
        mode_Cloud_data[0]=1;//自定义
        break;
    case color_controls_VALUE2:
        mode_Cloud_data[0]=2;//设置红色
        break;
    case color_controls_VALUE3:
        mode_Cloud_data[0]=3;//设置绿色
        break;
    case color_controls_VALUE4:
        mode_Cloud_data[0]=4;//设置蓝色
        break;
    case color_controls_VALUE5:
        mode_Cloud_data[0]=5;//设置黄色
        break;
    case color_controls_VALUE6:
        mode_Cloud_data[0]=6;//设置紫色
        break;
    case color_controls_VALUE7:
        mode_Cloud_data[0]=7;//设置粉色
        break;
    case color_controls_VALUE8:
        mode_Cloud_data[0]=8;//设置白色
        break;
    default:
        break;
    }
    break;

case EVENT_LED_R:
```

```
currentDataPoint.valueLED_R= dataPointPtr->valueLED_R;
GIZWITS_LOG("Evt:EVENT_LED_R %d\n",currentDataPoint.valueLED_R);
mode_Cloud_data[1]=currentDataPoint.valueLED_R;//红色值
if(mode_Cloud_data[1]!=1)
{
    mode_Cloud_data[0]=1;//自定义
}
break;
case EVENT_LED_G:
currentDataPoint.valueLED_G= dataPointPtr->valueLED_G;
GIZWITS_LOG("Evt:EVENT_LED_G %d\n",currentDataPoint.valueLED_G);
mode_Cloud_data[2]=currentDataPoint.valueLED_G;//绿色值
if(mode_Cloud_data[2]!=1)
{
    mode_Cloud_data[0]=1;//自定义
}
break;
case EVENT_LED_B:
currentDataPoint.valueLED_B= dataPointPtr->valueLED_B;
GIZWITS_LOG("Evt:EVENT_LED_B %d\n",currentDataPoint.valueLED_B);
mode_Cloud_data[3]=currentDataPoint.valueLED_B;//蓝色值
if(mode_Cloud_data[0]!=1)
{
    mode_Cloud_data[0]=1;//自定义
}
break;
```

到这儿我们就成功缓存了所有的数据，接下来就到定时上报函数里面处理数据。在 else 语句的后面增加以下内容（所有代码会打包上传，不清楚哪儿加看代码）

```
switch(mode_Cloud_data[0])
{
    case 0 :
        RGB_light_set_color(0,0,0);//关闭灯
        currentDataPoint.valuecolor_controls = 0;
        currentDataPoint.valueLED_R = 0;
        currentDataPoint.valueLED_G = 0;
        currentDataPoint.valueLED_B = 0;
        break;

    case 1 :

        RGB_light_set_color(mode_Cloud_data[1],mode_Cloud_data[2],mode_Cloud_data[3]);//
        自定义

        currentDataPoint.valuecolor_controls = 1;
        currentDataPoint.valueLED_R = mode_Cloud_data[1];
```

```
currentDataPoint.valueLED_G = mode_Cloud_data[2];  
currentDataPoint.valueLED_B = mode_Cloud_data[3];  
break;
```

case 2 :

```
RGB_light_set_color(255,0,0);//红色  
currentDataPoint.valuecolor_controls = 2;  
currentDataPoint.valueLED_R = 255;  
currentDataPoint.valueLED_G = 0;  
currentDataPoint.valueLED_B = 0;  
break;
```

case 3 :

```
RGB_light_set_color(0,255,0);//绿色  
currentDataPoint.valuecolor_controls = 3;  
currentDataPoint.valueLED_R = 0;  
currentDataPoint.valueLED_G = 255;  
currentDataPoint.valueLED_B = 0;  
break;
```

case 4 :

```
RGB_light_set_color(0,0,255);//蓝色  
currentDataPoint.valuecolor_controls = 4;  
currentDataPoint.valueLED_R = 0;  
currentDataPoint.valueLED_G = 0;  
currentDataPoint.valueLED_B = 255;  
break;
```

case 5 :

```
RGB_light_set_color(255,255,0);//黄色  
currentDataPoint.valuecolor_controls = 5;  
currentDataPoint.valueLED_R = 255;  
currentDataPoint.valueLED_G = 255;  
currentDataPoint.valueLED_B = 0;  
break;
```

case 6 :

```
RGB_light_set_color(255,0,255);//紫色  
currentDataPoint.valuecolor_controls = 6;  
currentDataPoint.valueLED_R = 255;  
currentDataPoint.valueLED_G = 0;  
currentDataPoint.valueLED_B = 255;  
break;
```

case 7 :

```
RGB_light_set_color(255,52,179);//粉色（估计不是粉色）
currentDataPoint.valuecolor_controls = 7;
currentDataPoint.valueLED_R = 255;
currentDataPoint.valueLED_G = 52;
currentDataPoint.valueLED_B = 179;
break;
```

case 8 :

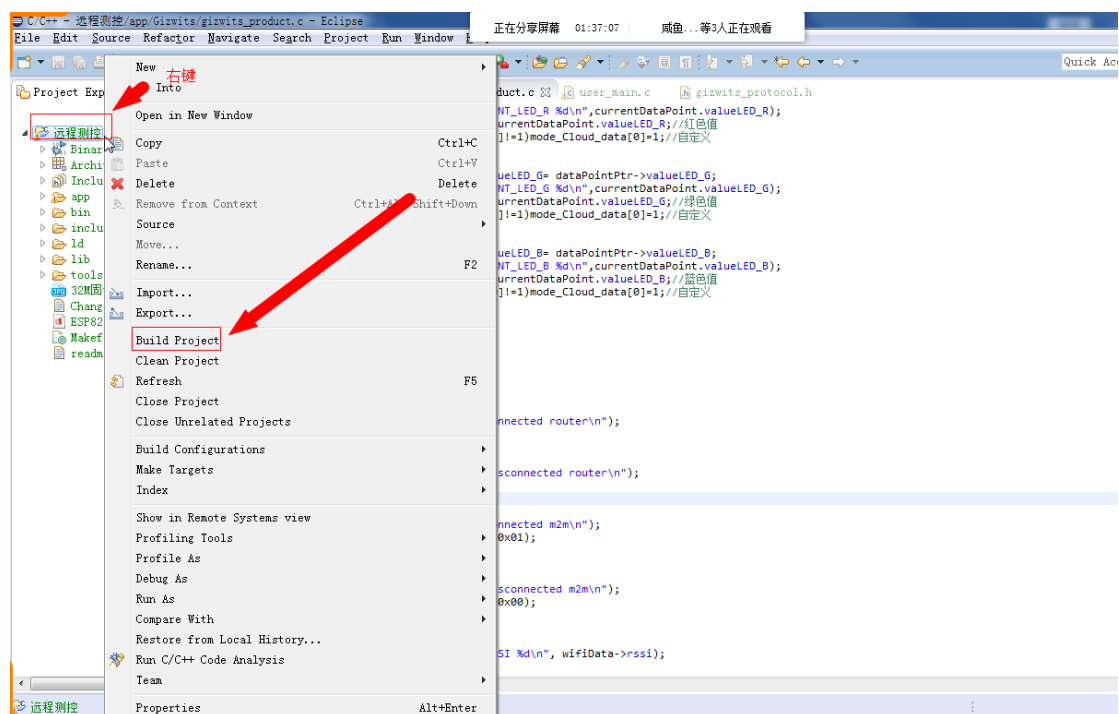
```
RGB_light_set_color(255,255,255);//白色
currentDataPoint.valuecolor_controls = 8;
currentDataPoint.valueLED_R = 255;
currentDataPoint.valueLED_G = 255;
currentDataPoint.valueLED_B = 255;
break;
```

default:

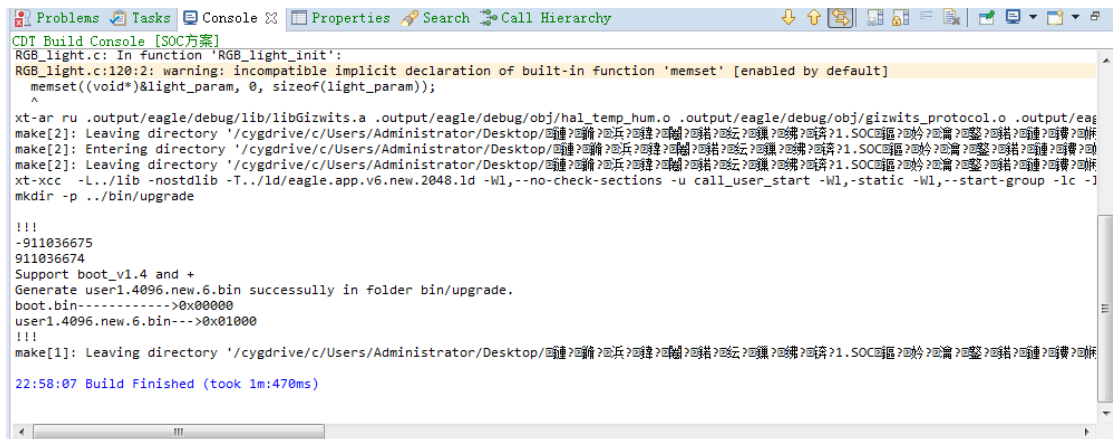
```
break;
```

```
}
```

修改完所有代码之后，编译代码。在项目地方右键，build 项目，生成固件。



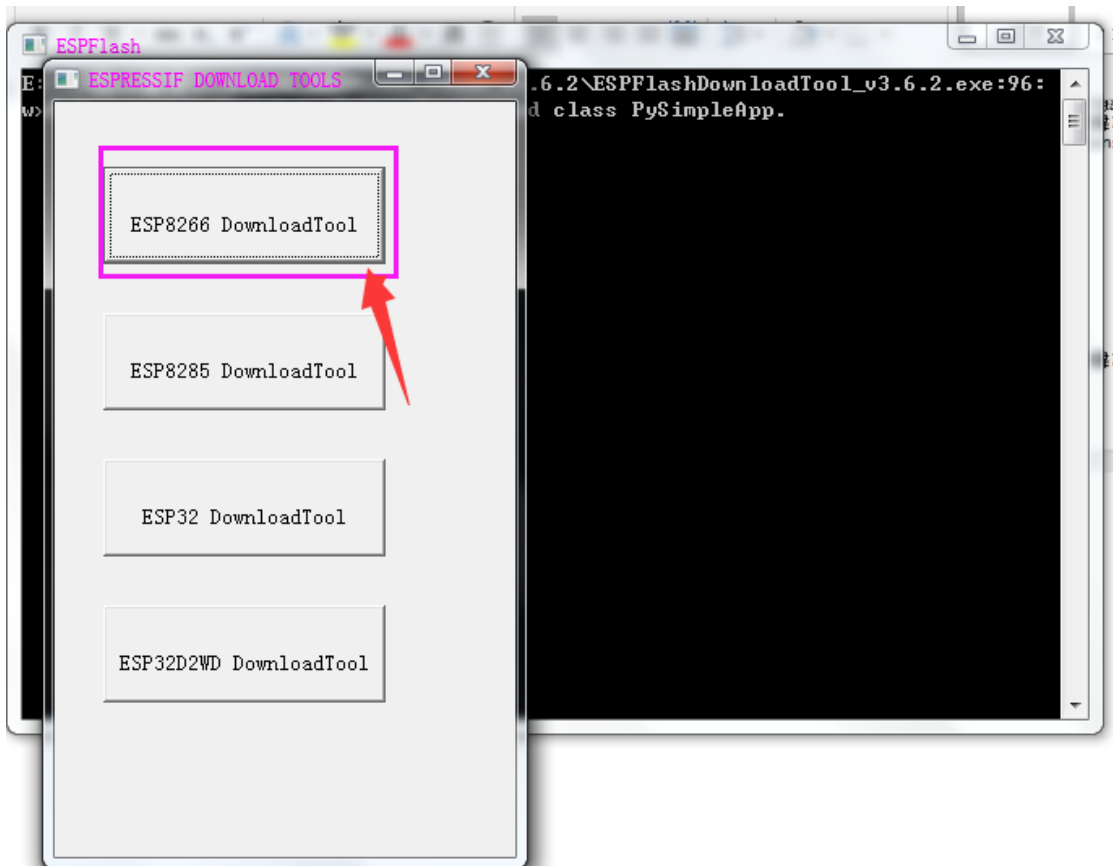
编译过后成功过后如下图所示，所有代码以项目文件为准。



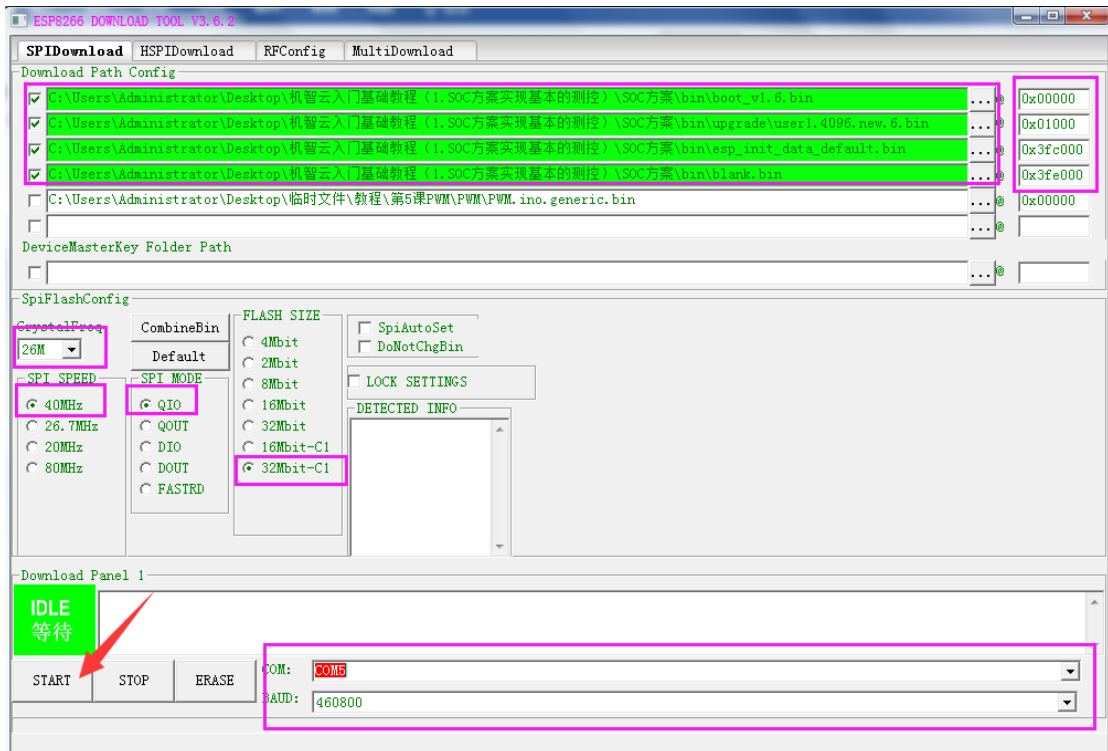
```
CDT Build Console [SOC方案]
RGB_light.c: In function 'RGB_light_init':
RGB_light.c:120:2: warning: incompatible implicit declaration of built-in function 'memset' [enabled by default]
    memset((void*)&light_param, 0, sizeof(light_param));
    ^
xt-ar ru .output/eagle/debug/lib/libGizwits.a .output/eagle/debug/obj/hal_temp_hum.o .output/eagle/debug/obj/gizwits_protocol.o .output/eagle/debug/obj/gizwits_main.o
make[2]: Leaving directory '/cygdrive/c/Users/Administrator/Desktop/ESP8266 DownloadTool'
make[2]: Entering directory '/cygdrive/c/Users/Administrator/Desktop/ESP8266 DownloadTool'
make[2]: Leaving directory '/cygdrive/c/Users/Administrator/Desktop/ESP8266 DownloadTool'
xt-cc -L../lib -nostdlib -T../ld/eagle.app.v6.new.2048.ld -Wl,--no-check-sections -u call_user_start -Wl,--static -Wl,--start-group -lc -ld
mkdir -p ../bin/upgrade

!!!
-911036675
911036674
Support boot_v1.4 and +
Generate user1.4096.new.6.bin successfully in folder bin/upgrade.
boot.bin----->0x000000
user1.4096.new.6.bin--->0x01000
!!!
make[1]: Leaving directory '/cygdrive/c/Users/Administrator/Desktop/ESP8266 DownloadTool'
22:58:07 Build Finished (took 1m:470ms)
```

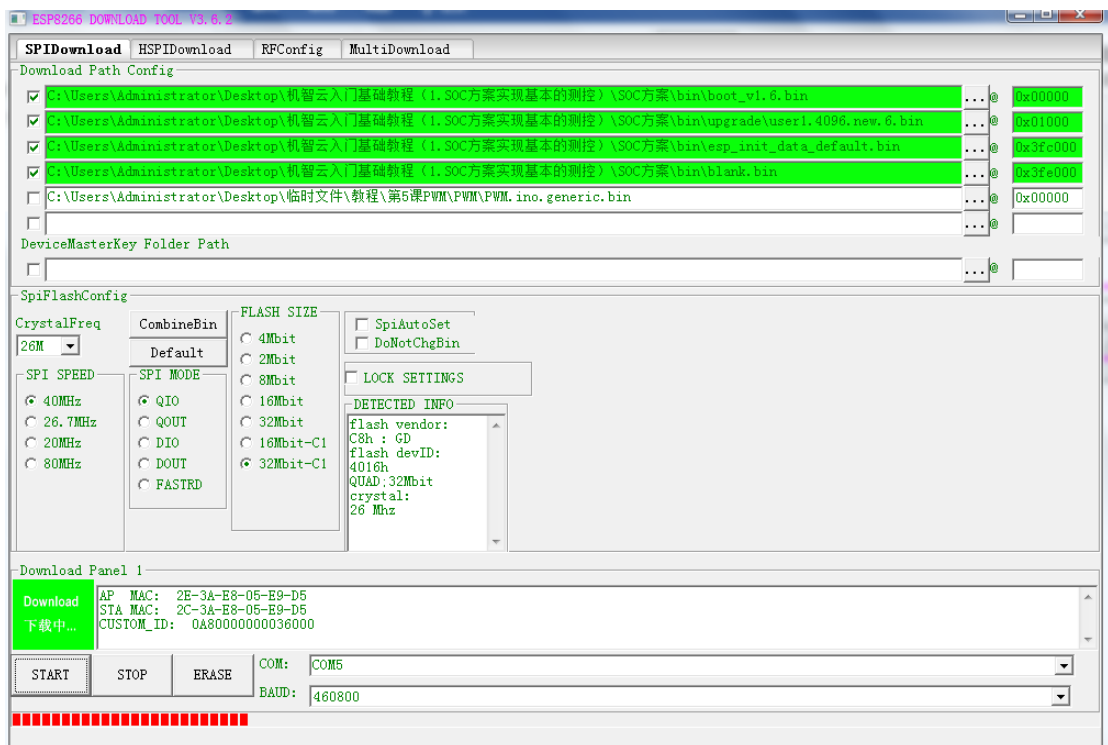
打开 ESP8266 下载软件



选择固件，对应地址，和复选框，波特率串口之类的，然后点击 start 下载程序

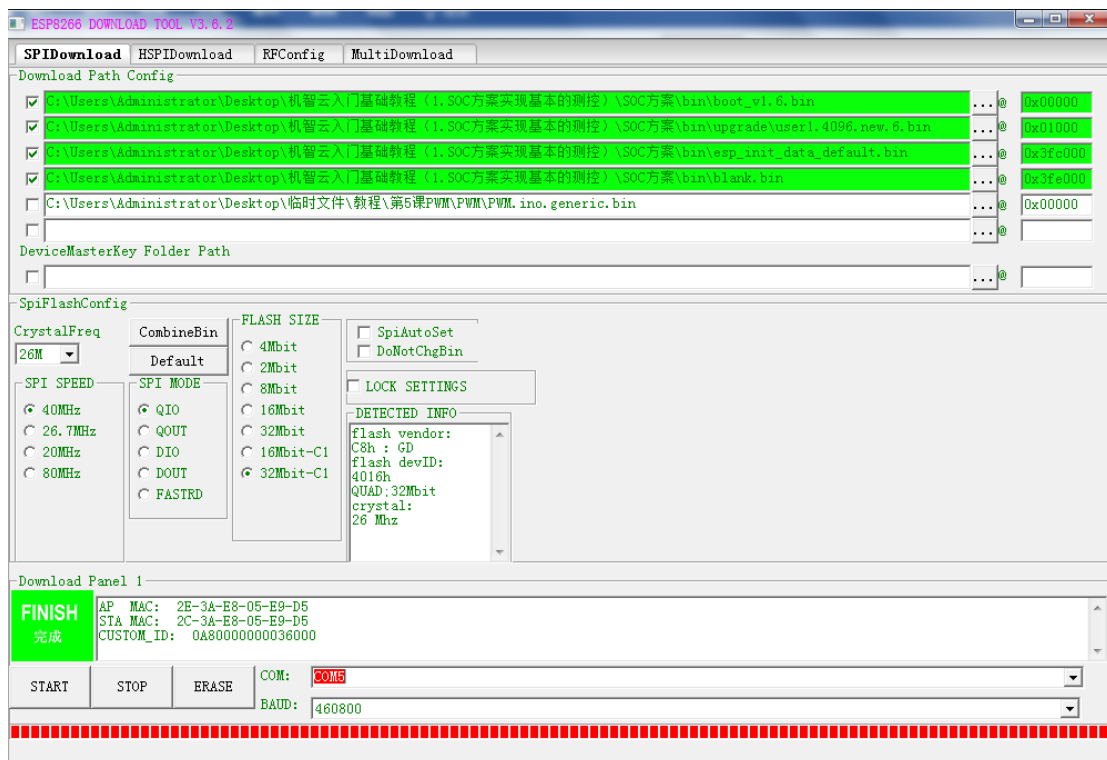


下载中



下载完成





教程到此结束