

**Doctors of Intelligence &Technology(DOIT)**

博流BL602&BL604开发教程

(V1.0)

2020.12.11

目录

[一、开发板介绍 3](#_Toc58596349)

[1.1.BL602/BL604芯片介绍： 3](#_Toc58596350)

[1.2.芯片主要特点 3](#_Toc58596351)

[1.3.开发板介绍 5](#_Toc58596352)

[二. 开发环境搭建 5](#_Toc58596353)

[2.1. 串口驱动的安装 5](#_Toc58596354)

[2.2.编译环境搭建 10](#_Toc58596355)

[3.3.测试编译环境 11](#_Toc58596356)

[3.4.程序烧录 11](#_Toc58596357)

[三.外设基础实验 14](#_Toc58596358)

[3.1. Hello world 14](#_Toc58596359)

[3.2.GPIO控制LED灯 14](#_Toc58596360)

[3.3.PWM呼吸灯 16](#_Toc58596361)

[3.4. ADC采集电压值实验 17](#_Toc58596362)

[3.5.FLASH读写操作 18](#_Toc58596363)

[3.6.定时器控制LED闪烁 19](#_Toc58596364)

[3.7.串口收发 20](#_Toc58596365)

[四.蓝牙通讯 23](#_Toc58596366)

[4.1.蓝牙通讯 23](#_Toc58596367)

[4.2.蓝牙配网 26](#_Toc58596368)

[五．WIFI通讯 29](#_Toc58596369)

[5.1.WIFI连接 29](#_Toc58596370)

[5.2.TCP CLIENT 32](#_Toc58596371)

[5.3.TCP SERVER 39](#_Toc58596372)

[六．综合项目 44](#_Toc58596373)

[6.1.Dohome智能全彩灯具 44](#_Toc58596374)

[6.2. dolphin蓝牙跳蛋 53](#_Toc58596375)

[6.3.AT指令 60](#_Toc58596376)

[免责申明和版权公告 66](#_Toc58596377)

# 一、开发板介绍

## 1.1.BL602/BL604芯片介绍：

BL602 / BL604是用于超低功耗应用的Wi-Fi + BLE组合芯片组。无线子系统包括2.4G无线电、Wi-Fi 802.11b/g/n和蓝牙LE5.0基带/MAC设计。微控制器子系统包含低功耗32位RISC-V CPU、高速缓存和存储器.电源管理单元控制低功耗模式.此外，还支持各种安全特性。

外围接口包括SDIO、SPI、UART、I2C、IR Remote、PWM、ADC、DAC、ACOMP、PIR等。

支持灵活的GPIO配置。BL602共有16个GPO，BL 604共有23个GPO。



## 1.2.芯片主要特点

**无线** （第1层RF性能）

* Wi-Fi 802.11 b / g / n
* 低功耗蓝牙®5.0
* 具有BLE协助的Wi-Fi快速连接
* Wi-Fi和BLE共存
* Wi-Fi安全WPS / WEP / WPA / WPA2 / WPA3
* STA，SoftAP和嗅探器模式
* 多云连接
* 2.4 GHz射频收发器
* 集成射频巴伦，PA / LNA

**周边设备**

* SDIO 2.0从站（AP主机）
* SPI主/从
* 两个UART
* I2C主/从
* 五个PWM通道
* 10位通用DAC
* 12位通用ADC
* 两个通用模拟比较器
* PIR（被动红外）检测
* 红外遥控硬件加速器
* 灵活的16个GPIO（BL602）/ 23个GPIO（BL604）

**微控制器子系统**

* 带FPU的32位RISC CPU
* L1快取
* RTC计时器长达一年
* 两个32b通用定时器
* 四个DMA通道
* 1MHz至192MHz的动态频率
* JTAG开发支持
* XIP QSPI闪存支持

**记忆**

* 276KB SRAM
* 128KB ROM
* 1Kb电子保险丝
* 嵌入式Flash（可选）

**安全性 （完整的安全性功能）**

* 安全启动
* 安全调试
* XIP QSPI即时AES解密（OTFAD）
* AES 128/192/256
* SHA-1 / 224/256
* TRNG（真随机数生成器）
* PKA（公钥加速器）

**时钟**

* 支持XTAL 24/26/32 / 38.4 / 40MHz
* 支持XTAL 32 / 32.768KHz
* 内部RC 32KHz和32MHz振荡器
* 内部系统PLL

**功耗模式 （超低功耗模式）**

* Off
* Hibernate
* Power Down Sleep (flexible)
* Active

## 1.3.开发板介绍

供电

开发板可以通过 USB 接口。板上集成有 AMS1117-3.3V 稳压芯片，把输入的 5V 电压稳压到 3.3V， 用于给板上的设备和模块供电。

USB 接口

USB 接口有三种功能：

 供电：只要插上 USB 线到 USB 接口即可。

 下载：USB 下载会有单独的一节说明。

 串口调试：串口调试在实验中有说明。

注：当使用 USB 接口进行调试或者程序下载的时候，需先按住D8键后按EN键进入下载模式。

按键

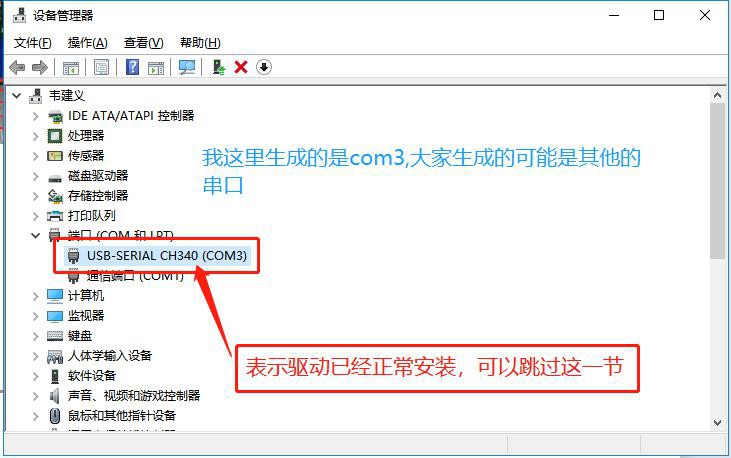
开发板共有两个按钮，分别为D8和EN，D8为进入下载模式所需按键，同时可作为普通按键连接至GPIO8使用，EN为复位按键，当需要进入下载模式时需先按住D8然后按一下EN。

# 二. 开发环境搭建

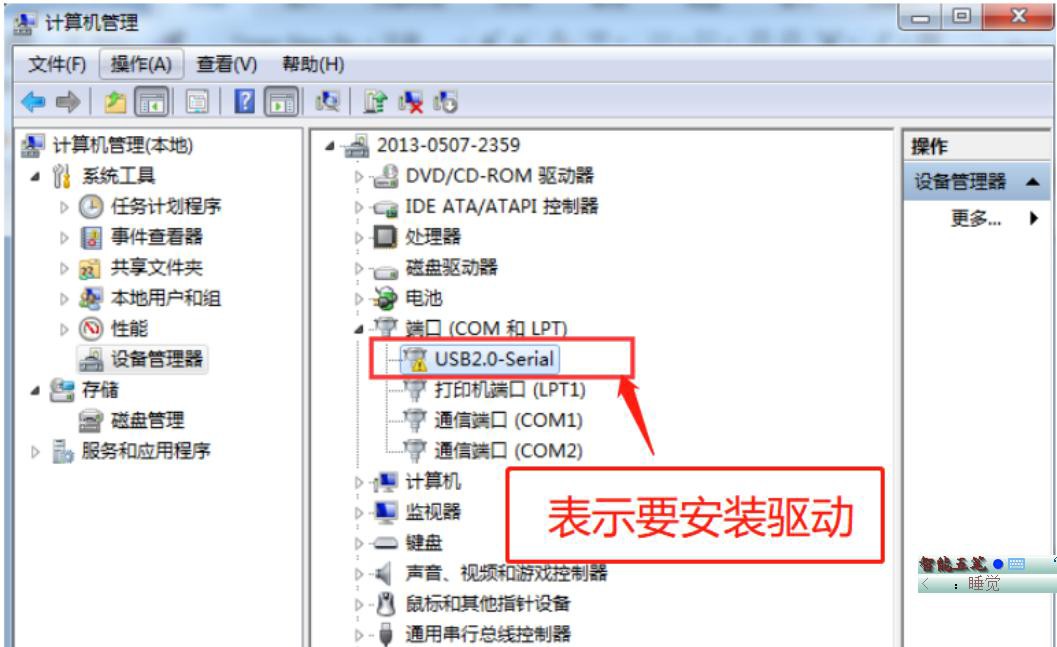
## 2.1. 串口驱动的安装

我们的开发板使用了 CH340G 的 USB 转串口芯片。通过 USB 线把开发板接到电脑上：

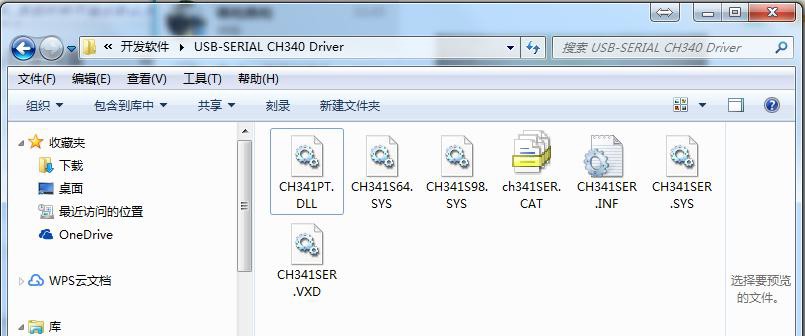
接着查看电脑的设备管理器，如下图表示驱动已经正确安装，可以跳过这一节：



如下图，表示要安装驱动：



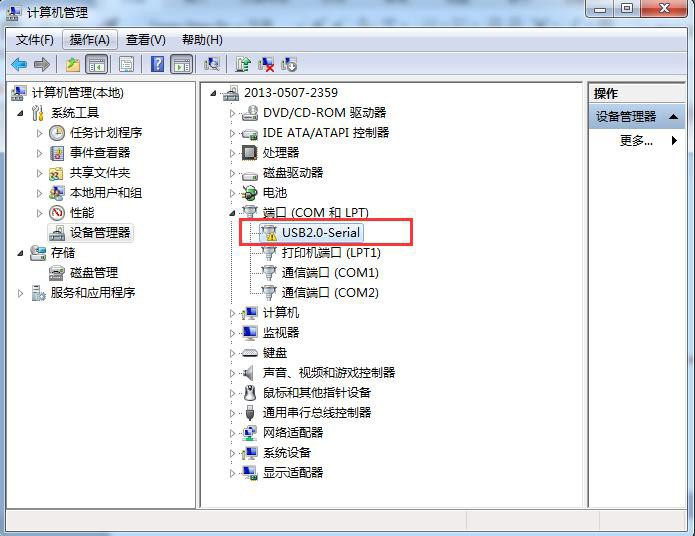
**安装驱动步骤一**:CH340 串口驱动位置：.\开发软件\USB-SERIAL CH340 Driver.rar，解压后如下：



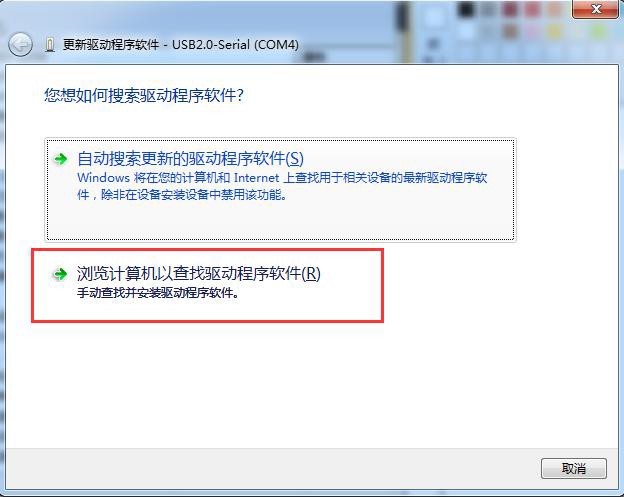
**安装驱动步骤二**:把开发板通过 USB 线接到电脑上（要打开开发板电源），提示安装驱动如下：



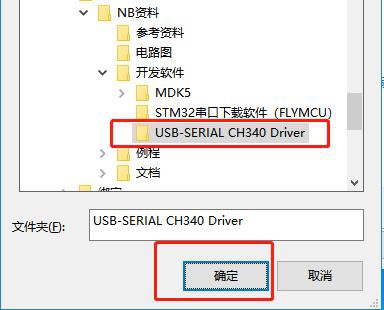
**安装驱动步骤三**:打开电脑的设备管理器，查看串口的驱动是否已自动安装，如下图是未安装的。



**安装驱动步骤四**:右键更新驱动，如下图：



**安装驱动步骤五**:选择第一步解压好的目录：



**安装驱动步骤六**:选择确定后，有可能会出现以下的提示，选择“始终安装此驱动程序软件”。



**安装驱动步骤七**:安装完成后，设备管理器，如下图：



串口驱动程序安装好之后，关于串口程序下载和调试，后面会陆续讲解。

## 2.2.编译环境搭建

BL602&BL604的编译环境为linux环境，WIN10用户可选linux子系统进行开发

1. 准备一台Linux主机，win10用户可使用Liunx子系统
2. 打开终端，安装make，命令sudo apt-get install make国内用户可更改镜像源，加快安装速度
3. 安装git，命令sudo apt-get install git
4. 克隆仓库git clone https://github.com/SmartArduino/Doiting\_BL.git
5. 修改权限，运行以下两条命令
6. chmod -R 777 ./Doiting\_BL/bl\_iot\_sdk/toolchain/

find ./Doiting\_BL/bl\_iot\_sdk/customer\_app -name "genromap"|xargs chmod 777

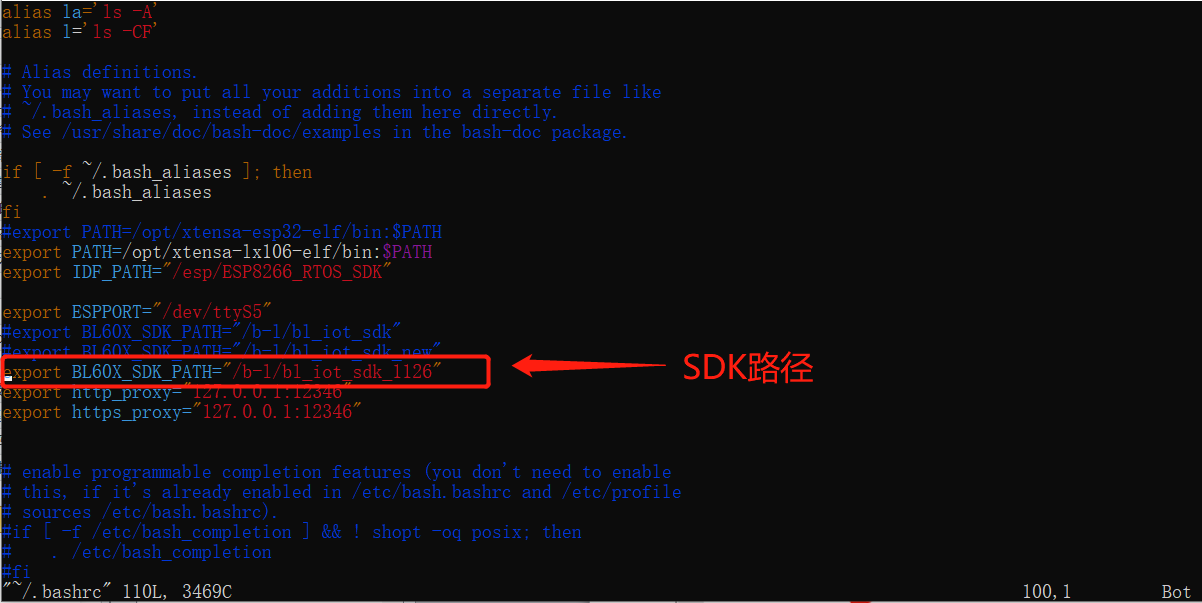
GitHub下载速度较慢，可通过网盘下载完整的SDK：

链接: <https://pan.baidu.com/s/123dQAG2lBEDT8VbtzN0TFQ>

提取码: d4qw

下载好SDK后，配置系统环境变量

1. 打开Linux终端，配置profile文件，命令vim ~/.profile
2. 添加环境变量，填写真实的路径export BL60X\_SDK\_PATH="$HOME/Doiting\_BL/bl\_iot\_sdk"



## 3.3.测试编译环境

以下使用hello-world为例

由于工具链以放在SDK中，故无需再下载工具链，可直接编译

进入例程目录，命令cd Doiting\_BL/bl\_iot\_sdk/customer\_app/sdk\_app\_helloworld/

使用项目提供的sh脚本进行编译，命令./genromap

编译完成提示：

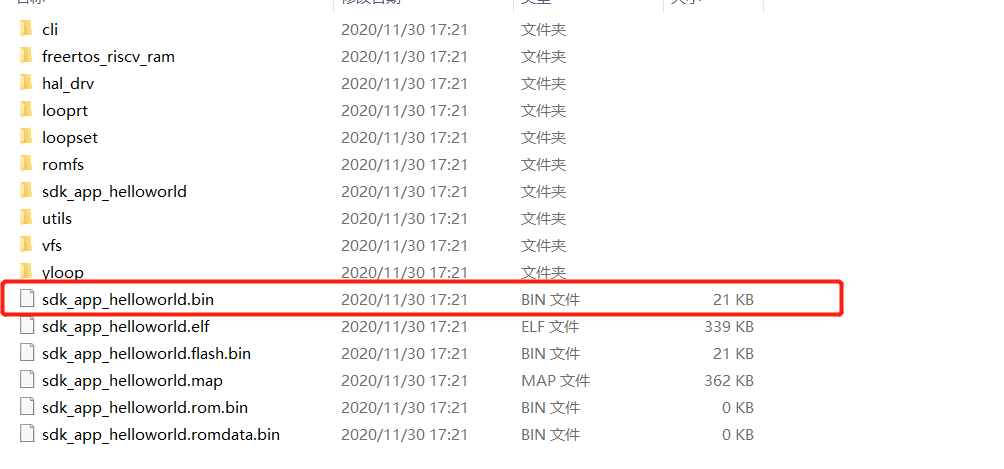
Generating BIN File to /home/hogc/Doiting\_BL/bl/bl\_iot\_sdk/ customer\_app/sdk\_app\_helloworld/build\_out/sdk\_app\_helloworld.bin

Building Finish. To flash build output.

## 3.4.程序烧录

程序编译好后，会在项目目录下生成build\_out文件夹，生成的.bin文件就是我们烧录的固件。





在SDK中已经包含烧录工具，位于....../bl\_iot\_sdk/tools/flash\_tool。

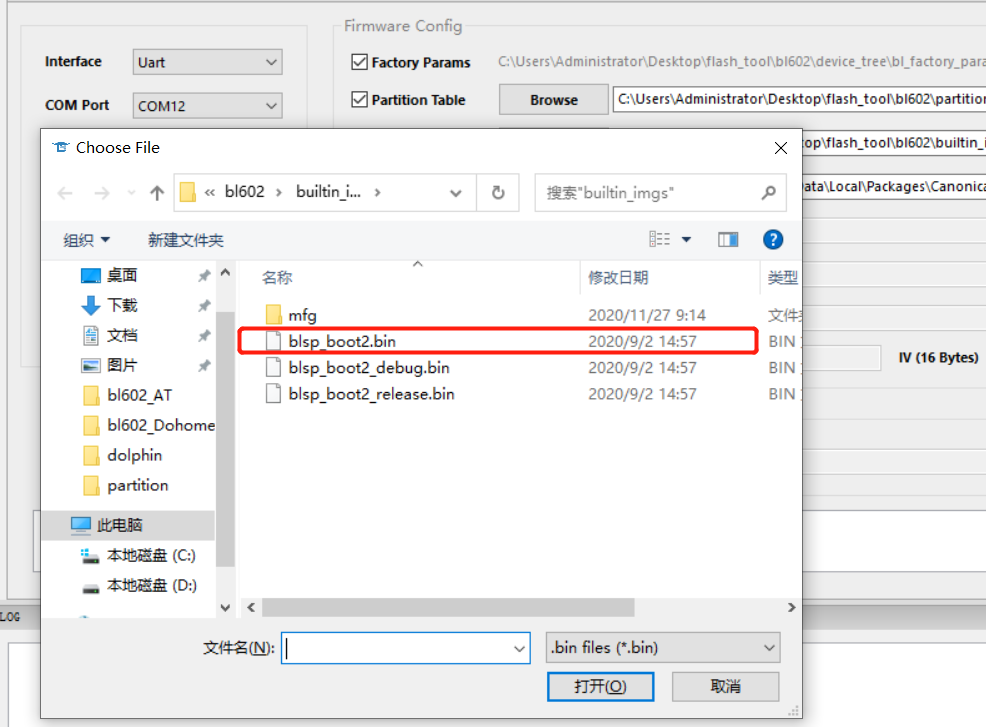


**打开烧录工具**

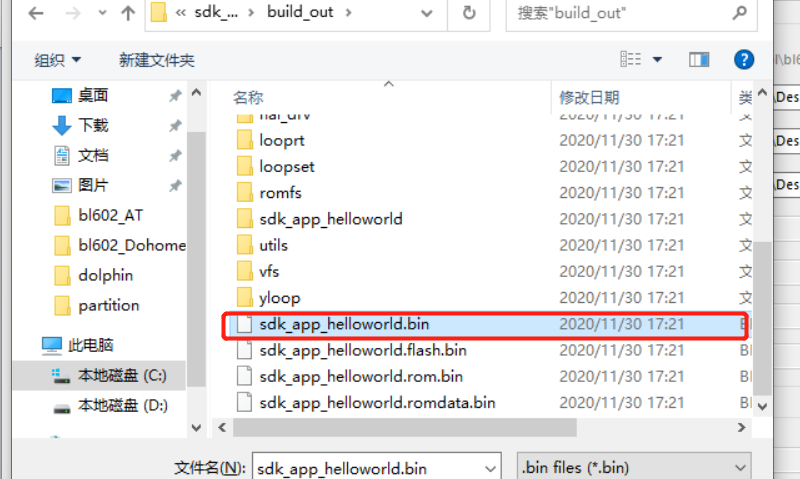
1.选择Partition Table, 选择partition\_cfg\_2M.toml



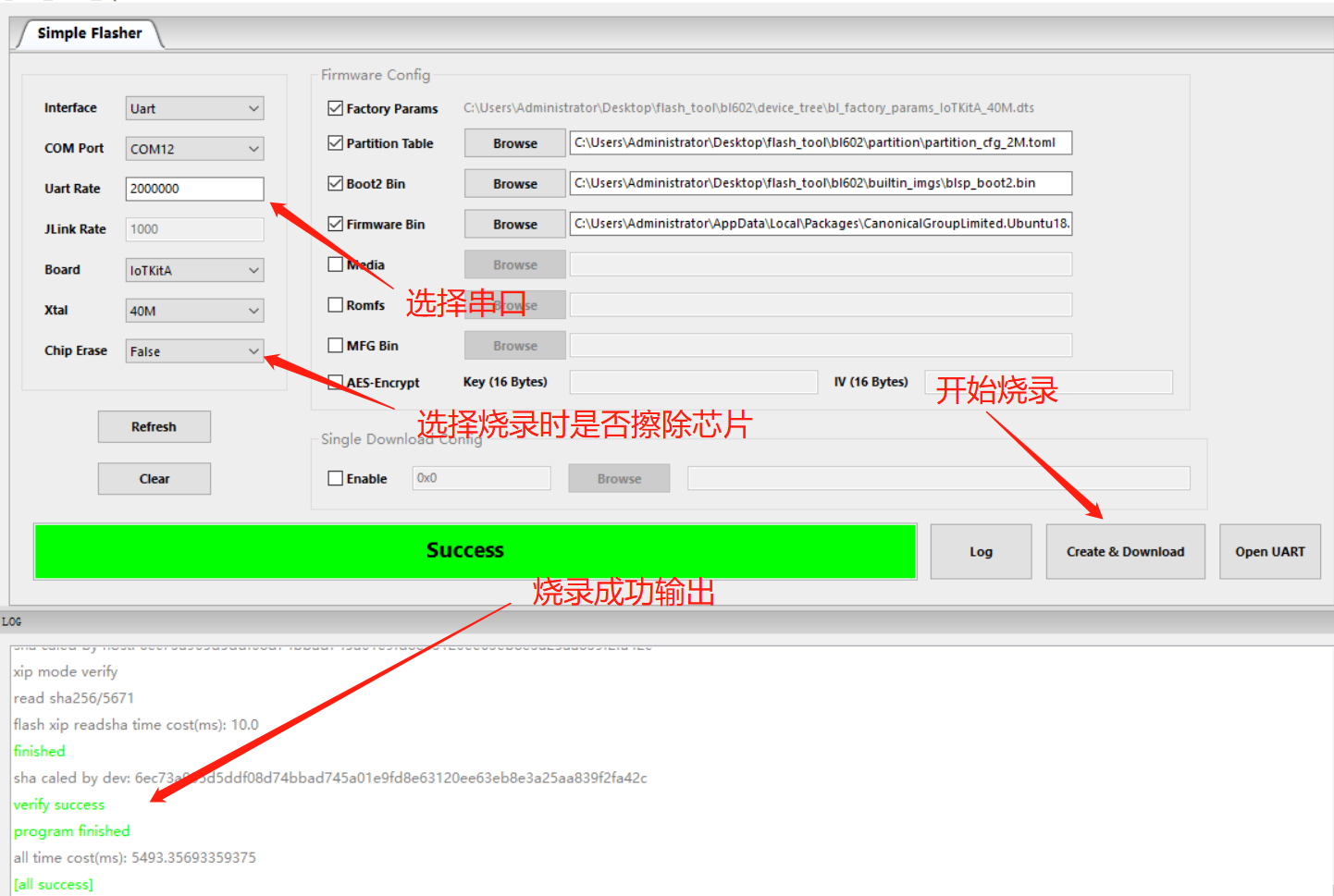
2.选择Boot2 Bin,选择blsp\_boot2.bin



1. 选择Firmware Bin，根据自己编译出的固件位置选择编译好的固件



4．开始烧录，按住D8后按一下EN进入烧录模式，

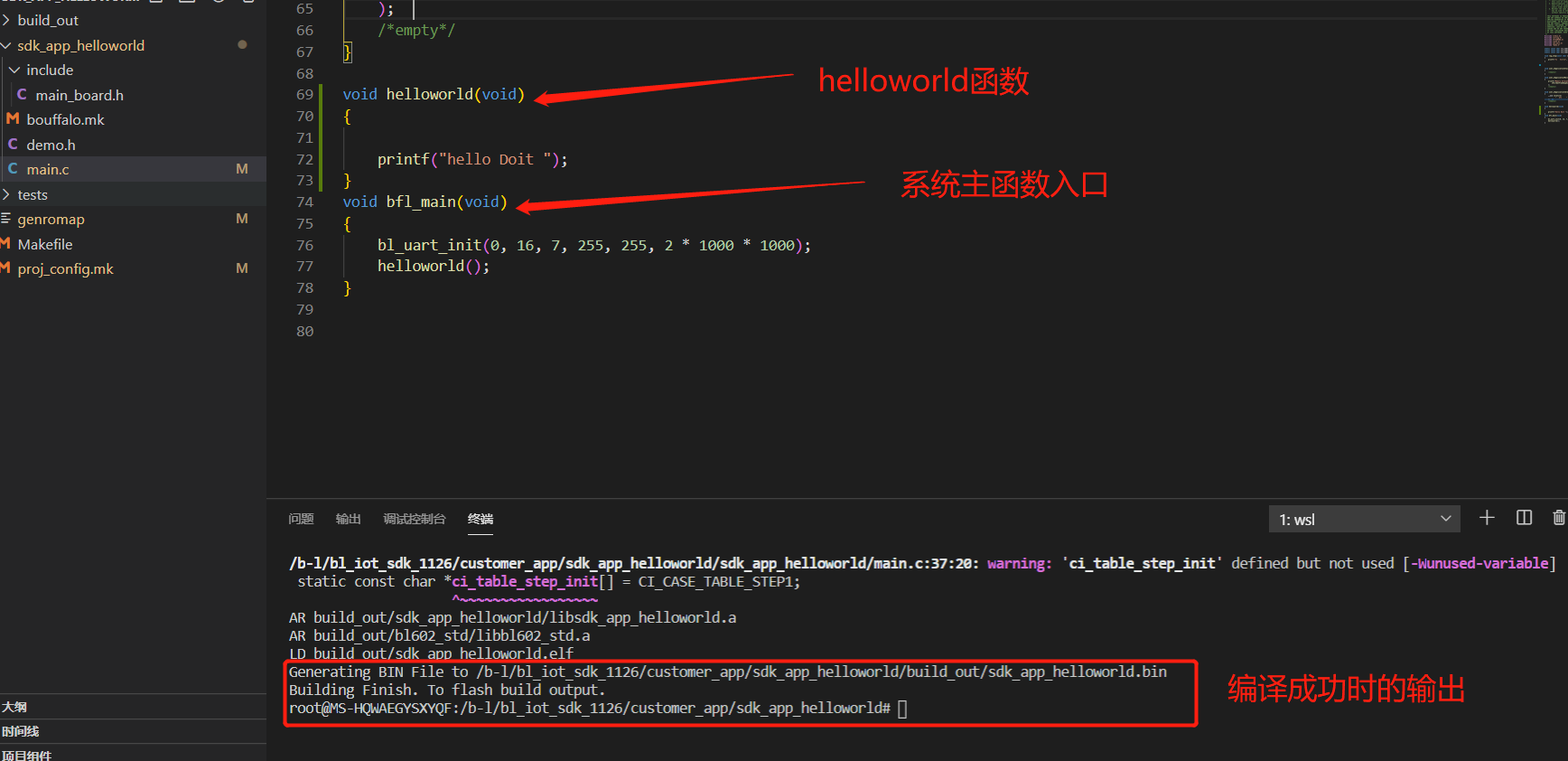


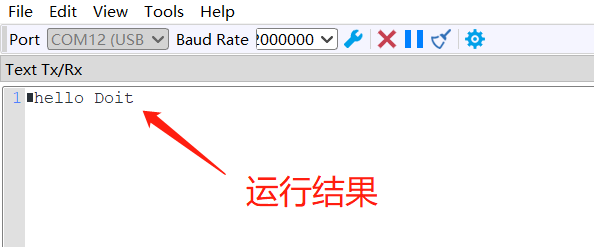
**注意存放固件的路径不能带有中文！**

# 三.外设基础实验

## 3.1. Hello world

BL602/BL604的函数入口在main.c文件中，系统的函数入口为bfl\_main(void)函数，编译成功后，烧录程序。





## 3.2.GPIO控制LED灯

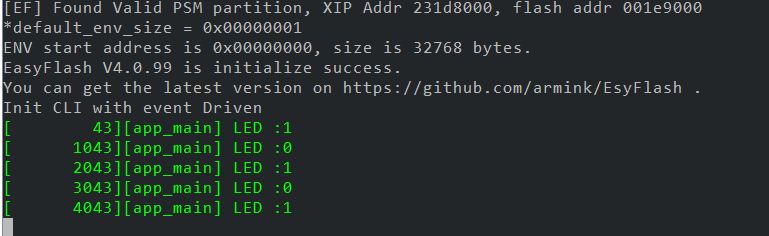
为了方便开发，在bfl\_main(void)系统主函数中已经将一些常用的系统函数初始化，后续的开发只需要通过app\_main.c文件的void user\_main(void)作为函数入口即可。



为方便调试，示例程序已通过处理使串口打印输出能输出红色和绿色两种颜色，白色部分为系统输出，LOGI(TAG, "LED :%d",value)输出为绿色，LOGE(TAG, "LED :%d",value)，输出为红色。

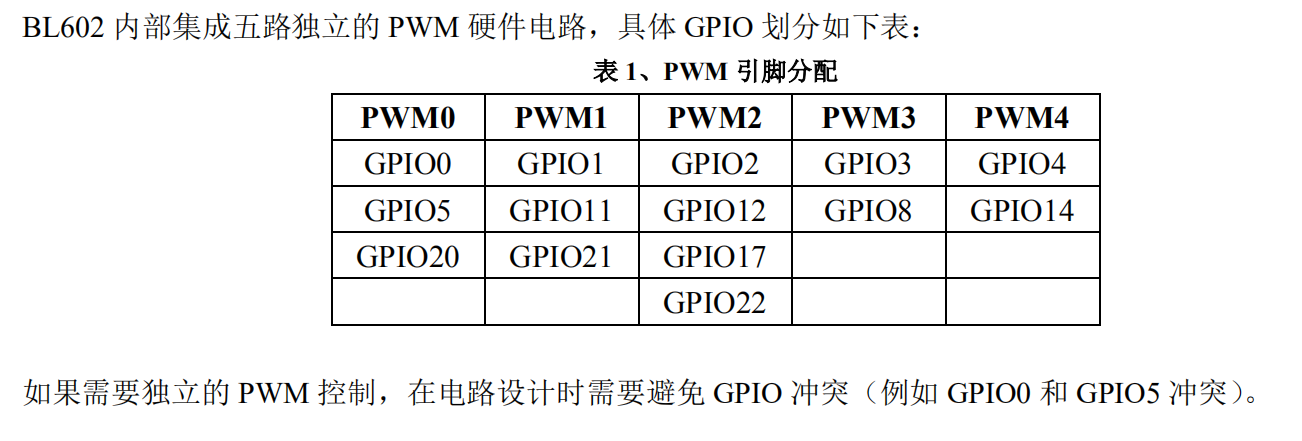


编译完成后，烧录程序，将LED灯接到GPIO1，便可以看到LED灯以一秒的频率闪烁。

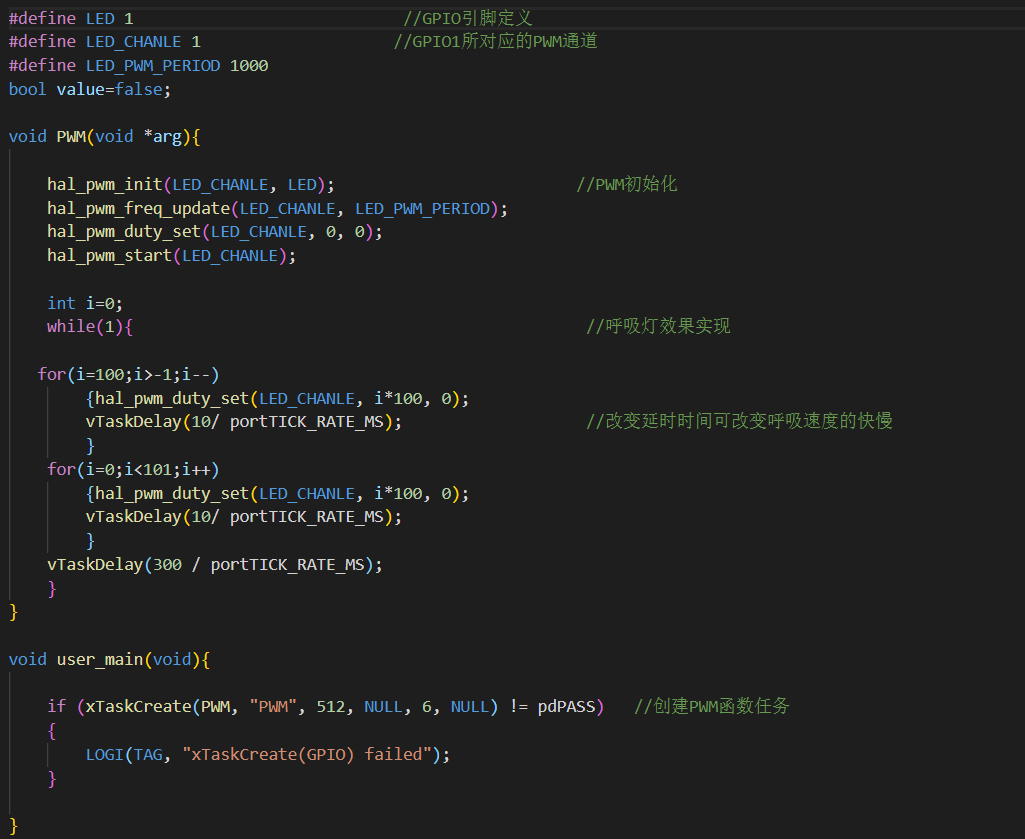


## 3.3.PWM呼吸灯

接着上节的GPIO实现LED闪烁效果，本节通过PWM控制来实现LED灯有暗变亮由亮变暗的呼吸效果，BL602/BL604共有五组PWM，其中GPIO与PWM对应的关系如下图：



选用PWM1控制GPIO1来实现呼吸灯功能



编译完成后，烧录程序，将LED灯接到GPIO1，便可以看到LED灯的呼吸效果

## 3.4. ADC采集电压值实验

本例程用到了两个ADC相关函数

ADC初始化函数：

int hal\_adc\_init(int mode, int freq, int data\_num, int gpio\_num);

mode:高速，慢速

freq：采样频率

data\_num:采样的buffer

gpio\_num:gpio

ADC读取函数：

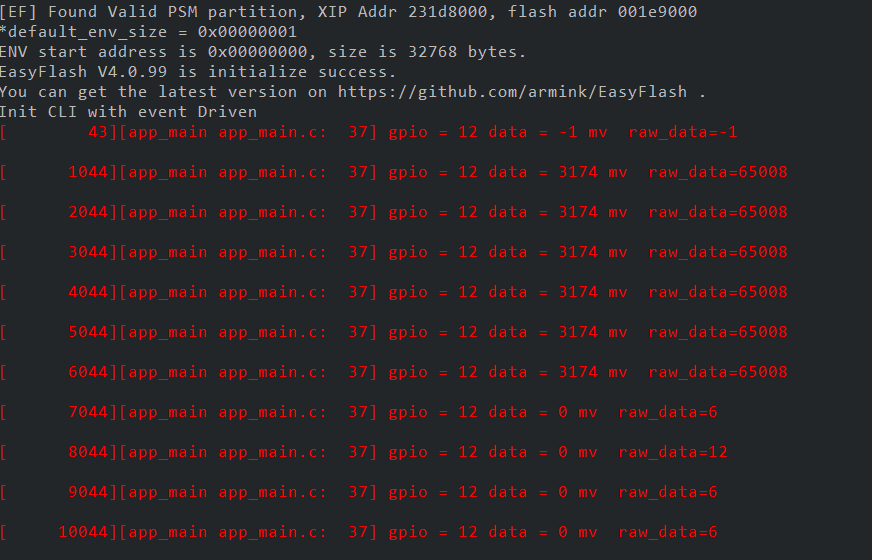
int32\_t hal\_adc\_get\_data(int gpio\_num, int raw\_flag);

gpio\_num：gpio

raw\_flag：原始数据还是转换成voltage的值



编译完成后，烧录程序，将GPIO12接到VCC或GND，可以看到串口输出测量的电压数值和原始数值。



## 3.5.FLASH读写操作

BL602/BL604可以很便捷地对FLASH进行读写操作，下面以存储wifi信息为例进行FLASH读写操作，使用easyflash库可以很方便地对FLASH进行读写操作。

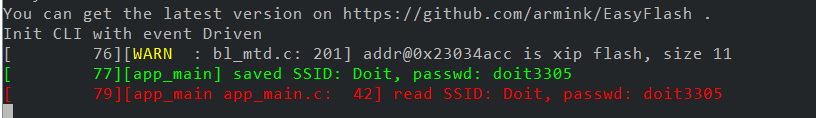
这里用到了FLASH读和写两个函数：

EfErrCode ef\_set\_env\_blob(const char \*key, const void \*value\_buf, size\_t buf\_len);

size\_t ef\_get\_env\_blob(const char \*key, void \*value\_buf, size\_t buf\_len, size\_t \*saved\_value\_len);

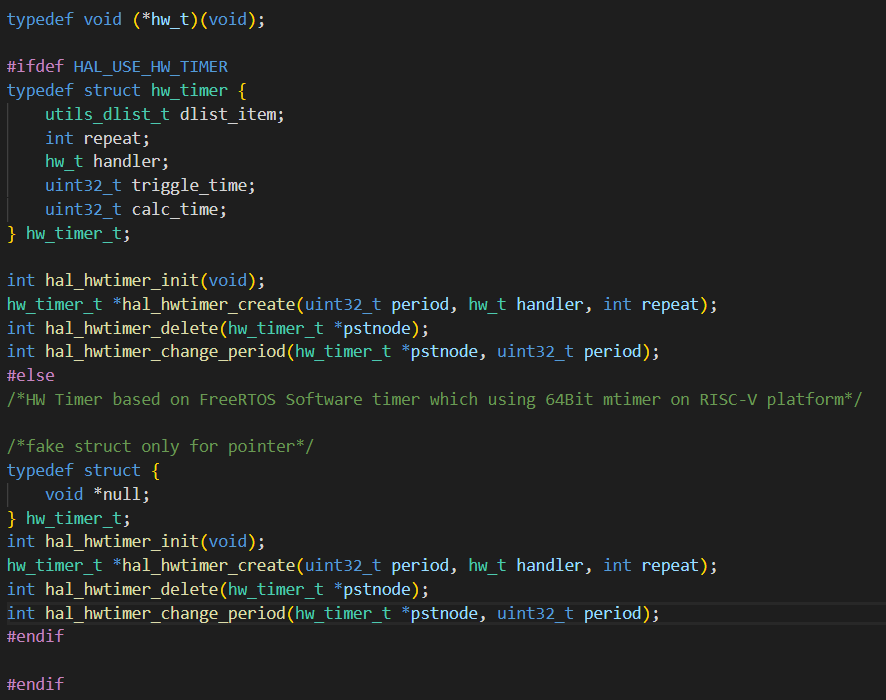


编译完成后，烧录程序，可以看到串口输出两组wifi信息，绿色打印输出的为存储到FALSH的信息，红色打印输出为从FLASH读取到的信息



## 3.6.定时器控制LED闪烁

本例程通过BL602的定时器实现LED灯的闪烁，Timer库相关的函数如下：



**功能实现：**



编译完成后，烧录程序，每2秒输出一个times，LED灯以2秒的频率闪烁

## 3.7.串口收发

例程在SDK目录下的...\customer\_app\sdk\_app\_uart\_echo,直接到SDK下的例程进行编译。

//串口任务

static void uart\_echo\_task(void \*arg)

{

    int length = 0, total = 0;

    uint8\_t buf\_recv[128];

    uint8\_t \*pbuf = buf\_recv;

    const char \*name = arg;

    const char \*send\_recv\_log = "1234567890abcdefg";

    int fd = aos\_open(name, 0);

    log\_info("%s-> fd = %d\r\n", name, fd);

    if (fd < 0) {

        return;

    }

    memset(buf\_recv, 0, sizeof(buf\_recv));

    log\_step(ci\_table\_step\_init);

    aos\_write(fd, send\_recv\_log, strlen(send\_recv\_log));

    log\_step(ci\_table\_step\_send);

    vTaskDelay(1000);

    length = 0;

    while (1) {

        length = aos\_read(fd, pbuf + total, strlen(send\_recv\_log));

        total += length;

        if (total != strlen(send\_recv\_log)) {

            continue;

        }

        if (memcmp(buf\_recv, send\_recv\_log, strlen(send\_recv\_log)) == 0) {

            printf("recvbuff:%s\r\n", send\_recv\_log);

            aos\_write(fd, send\_recv\_log, strlen(send\_recv\_log));

            log\_step(ci\_table\_step\_recv);

            break;

        }

        vTaskDelay(10);

    }

    aos\_close(fd);

    log\_step(ci\_table\_step\_end);

}

//创建任务

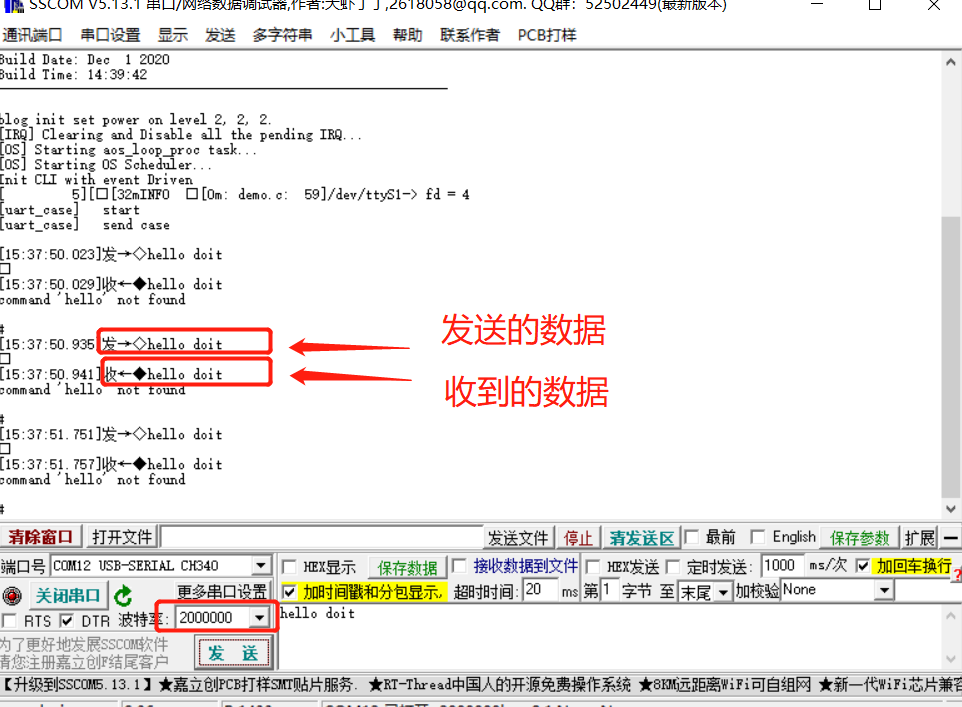
void ci\_loop\_proc()

{

    aos\_task\_new("uart\_echo", uart\_echo\_task, "/dev/ttyS1", 2048);

}

编译完成后烧录程序，打开串口测试工具，发送数据。



# 四.蓝牙通讯

## 4.1.蓝牙通讯

本例程通过蓝牙控制GPIO1上的LED的亮灭

**系统初始化：**

void user\_main(void){

    bl\_gpio\_enable\_output(LED, 0, 0);                                           //初始化GPIO

    do\_ble\_init();                                                              //蓝牙初始化

    do\_ble\_set\_reve\_cb(ble\_reve\_cb);                                            //蓝牙接收回调

    do\_ble\_set\_connect\_cb(ble\_connect\_cb);                                      //蓝牙连接回调

}

**蓝牙初始化函数：**

void do\_ble\_init(void){

    // Initialize BLE controller

    LOGI(TAG, "do\_ble\_init ");

    ble\_controller\_init(configMAX\_PRIORITIES - 1);

    //Initialize BLE Host stack

    hci\_driver\_init();

    bt\_enable(bt\_ready\_cb);

    bt\_set\_name("BL602\_BLE");

    bt\_gatt\_service\_register((struct bt\_gatt\_service \*)&dis);

    notify\_attrs = &ble\_attrs[1];

    bt\_conn\_cb\_register(&conn\_callbacks);

    ble\_start\_advertise();

}

**蓝牙接收回调函数：**

static void ble\_reve\_cb(struct bt\_conn \*conn, const char \*buf, u16\_t len){     //蓝牙处理函数，可在此对接收到的蓝牙数据进行处理

    char ret\_str[10]="";

    LOGE(TAG, "ble\_reve\_buf: %s", buf);

    if(CDM\_CMP(buf,"on"))                                                       //当接收到“on”,拉高GPIO1点亮LED

    {

    bl\_gpio\_output\_set(LED, 1);

    strcpy(ret\_str,"on");

    }

    if(CDM\_CMP(buf,"off"))                                                      //当接收到“off”,拉低GPIO1熄灭LED

    {

    bl\_gpio\_output\_set(LED, 0);

    strcpy(ret\_str,"off");

    }

    do\_ble\_notify(conn, ret\_str, strlen(ret\_str));                              //蓝牙回复状态

    memset(buf, 0, sizeof(buf));

}

**蓝牙连接状态函数：**

static void ble\_connect\_cb(uint8\_t status, char \*addr){                     //蓝牙连接状态回调函数

    if(status == DO\_BLE\_DEVICE\_CONNECT){

        LOGI(TAG, "ble\_connect: %s", addr);

    }else if(status == DO\_BLE\_DEVICE\_DISCONNECT){

        LOGI(TAG, "ble\_disconnect: %s", addr);

        ble\_start\_advertise();                                              //蓝牙断开，开始广播

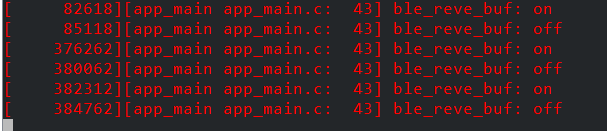
    }

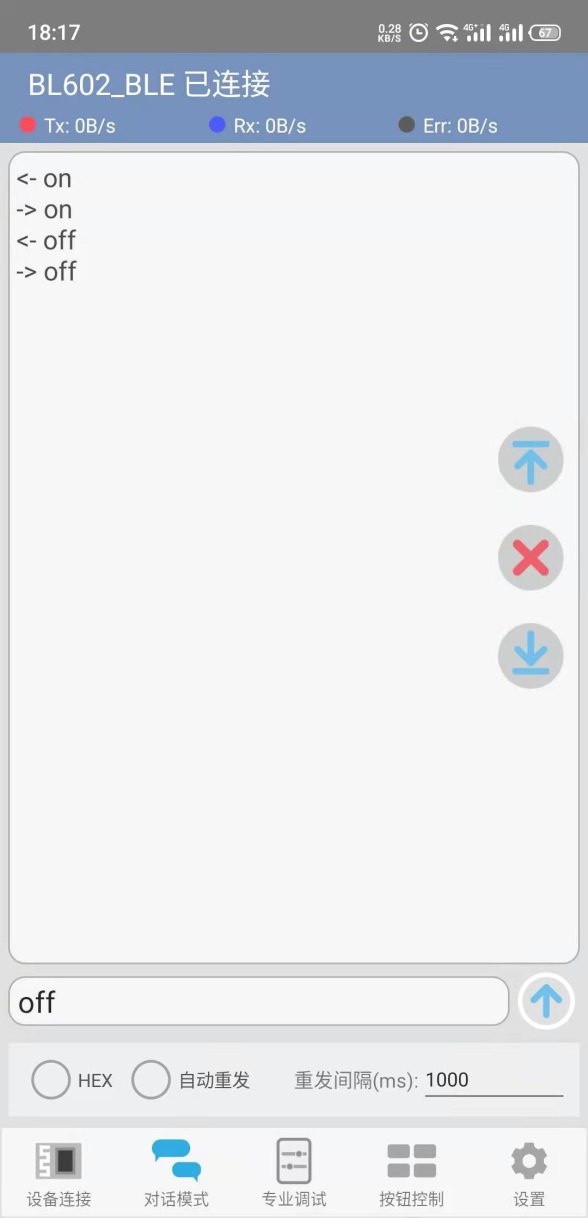
}

**打开“蓝牙调试器”APP，找到模块发射出的蓝牙，点击设置UUID**

****

**连接成功后发送字符“ok”便会收到模块回复一个“ok”，且LED灯会点亮，手机发送“off”,同意也会收到“off”，此时LED灯会熄灭。串口会输出接收到的蓝牙数据**

****

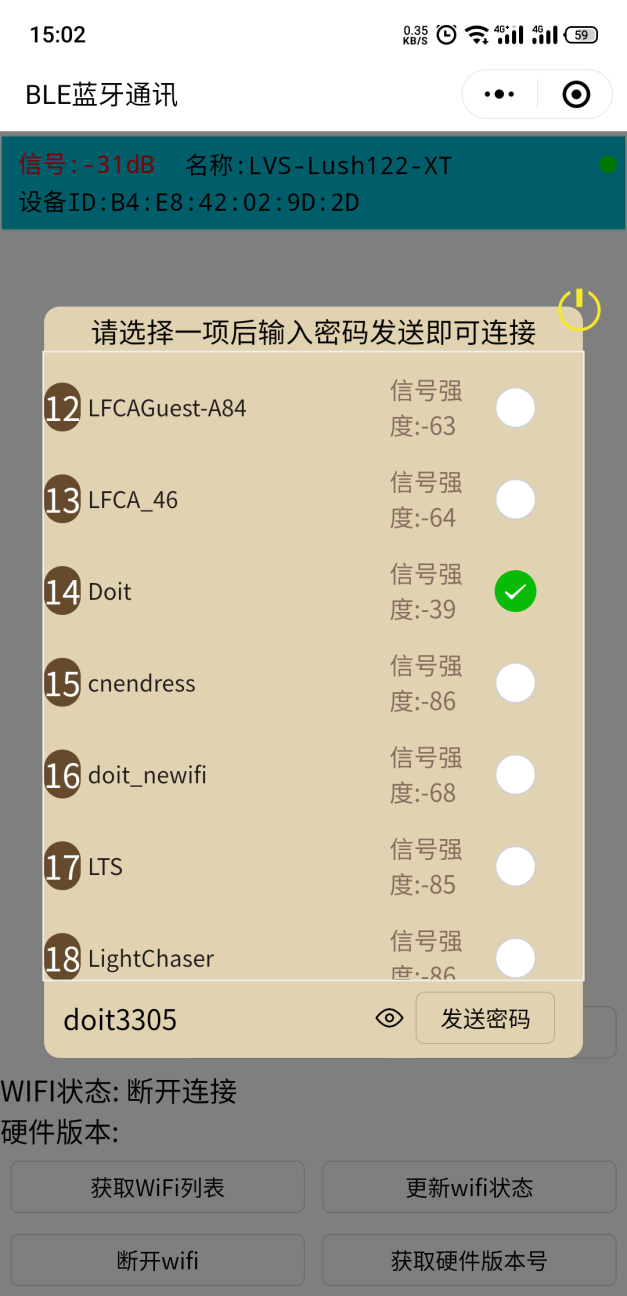
****

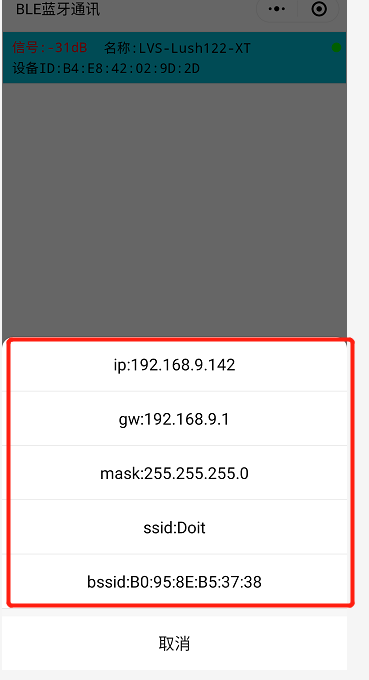
## 4.2.蓝牙配网

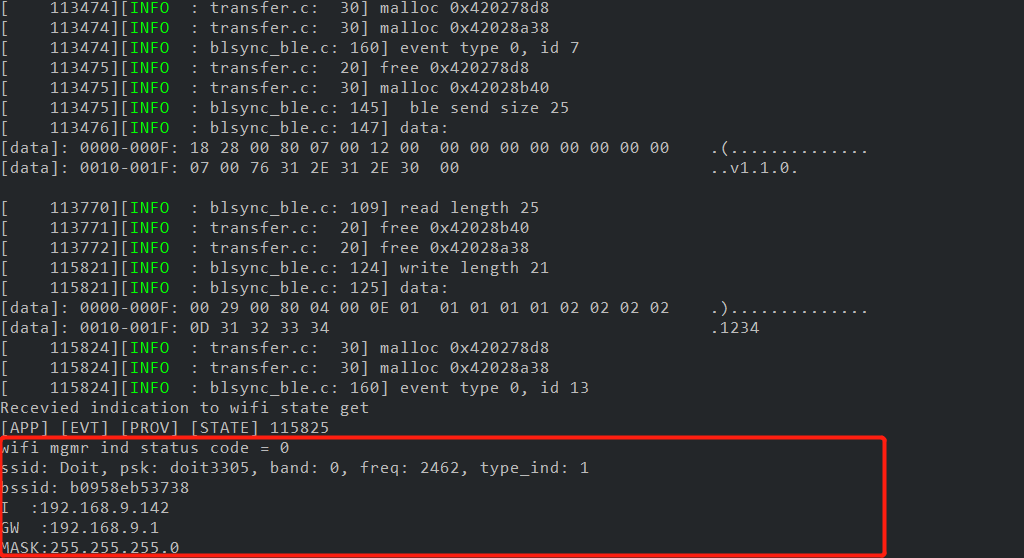
在SDK下的路径...\customer\_app\sdk\_app\_ble\_sync有蓝牙配网的例程，编译例程后烧录程序，用微信扫一扫扫描此二维码，进入配网小程序。



进入小程序后点击搜索，选择搜索到的蓝牙设备，点击获取wifi列表，选择需要连接的wifi，输入密码点击发送密码，等待连接，连接成功后能看到wifi的相关信息。





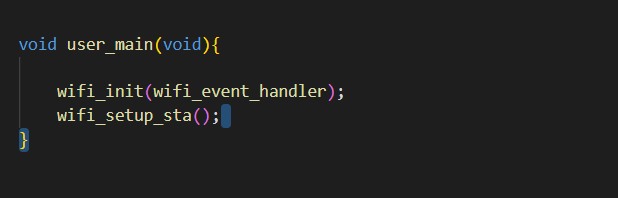


连接成功后串口会输出wifi相关信息

# 五．WIFI通讯

## 5.1.WIFI连接

系统初始化：



WIFI初始化相关函数：

/\*

    wifi初始化

 \*/

void wifi\_init(wifi\_event\_cb\_t user\_wifi\_event\_cb) {

    LOGI(TAG, "wifi init");

    cmd\_stack\_wifi(NULL, 0, 0, NULL);

    static\_wifi\_cb = user\_wifi\_event\_cb;

void wifi\_set\_event\_cb(void (\*user\_wifi\_cb)(input\_event\_t \*event, void \*private\_data));

wifi\_set\_event\_cb(event\_cb\_wifi\_event);

}

static void cmd\_stack\_wifi(char \*buf, int len, int argc, char \*\*argv)

{

    /\*wifi fw stack and thread stuff\*/

    static uint8\_t stack\_wifi\_init  = 0;

    if (1 == stack\_wifi\_init) {

        puts("Wi-Fi Stack Started already!!!\r\n");

        return;

    }

    stack\_wifi\_init = 1;

    printf("Start Wi-Fi fw @%lums\r\n", bl\_timer\_now\_us()/1000);

    hal\_wifi\_start\_firmware\_task();

    /\*Trigger to start Wi-Fi\*/

    printf("Start Wi-Fi fw is Done @%lums\r\n", bl\_timer\_now\_us()/1000);

    aos\_post\_event(EV\_WIFI, CODE\_WIFI\_ON\_INIT\_DONE, 0);

}

/\*

    wifi连接状态

 \*/

void wifi\_event\_handler(wifi\_event\_indicate\_t event){

    switch (event)

    {

        case WIFI\_EVENT\_CONNECT:

            LOGE(TAG, "wifi\_connect");

            break;

        case WIFI\_EVENT\_DISCONNECT:

            LOGE(TAG, "wifi\_disconnect");

            break;

        default:

            LOGE(TAG, "wifi\_event: %d",event);

            break;

    }

}

static void event\_cb\_wifi\_event(input\_event\_t \*event, void \*private\_data)

{

    switch (event->code) {

        case CODE\_WIFI\_ON\_DISCONNECT:

        {

            LOGI(TAG, "wifi disconnect");

            if(static\_wifi\_connect\_status != 0){

                static\_wifi\_connect\_status = 0;

                if(static\_wifi\_cb != NULL){

                    static\_wifi\_cb(WIFI\_EVENT\_DISCONNECT);

                }

            }

        }

        break;

        case CODE\_WIFI\_ON\_CONNECTED:

        {

            LOGI(TAG, "wifi connect");

        }

        break;

        case CODE\_WIFI\_ON\_GOT\_IP:

        {

            LOGI(TAG, "wifi get ip");

            if(static\_wifi\_connect\_status != 1){

                static\_wifi\_connect\_status = 1;

                if(static\_wifi\_cb != NULL){

                    static\_wifi\_cb(WIFI\_EVENT\_CONNECT);

                }

            }

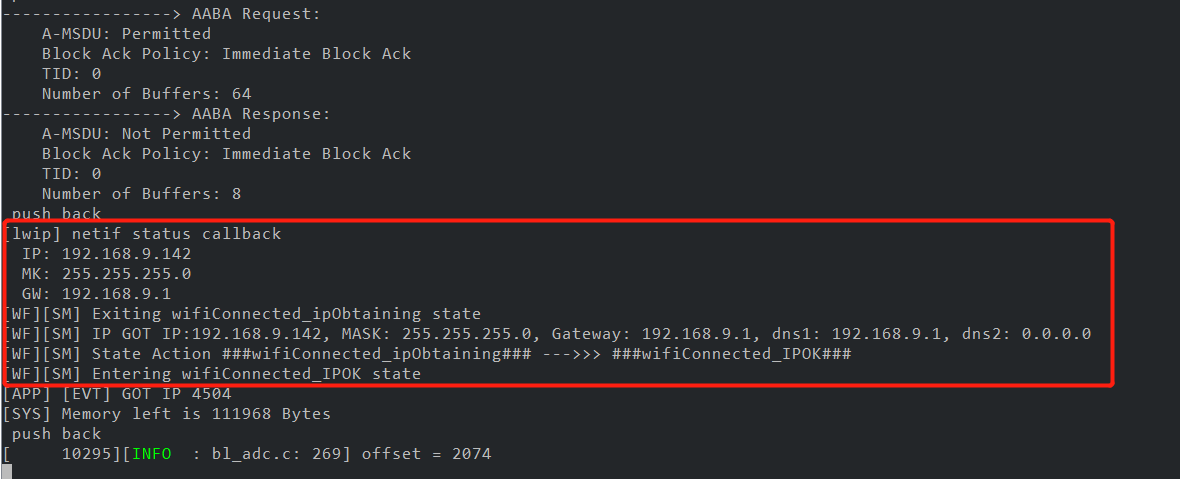
        }

        break;

    }

}

Wifi连接成功后会输出连接的相关信息



## 5.2.TCP CLIENT

在上一节我们已经可以成功连接上WIFI，连接上WIFI后就可以进行TCP通讯，下面来看看bl602作为TCP client与TCP server的通讯。

bl\_iot sdk的socket就是bsd socket，常见的linux socket编程直接可以使用。但目前暂不支持poll api。

//TCP任务函数

static void tcp\_client\_conn(void \*pvParameters){

    LOGI(TAG, "tcp\_client\_conn init");

    char re\_hostbyname\_cnt = 0;

    char databuff[512];

    memset(databuff, 0x00, sizeof(databuff));

    char sub\_buf[200];

    memset(sub\_buf, 0x00, sizeof(sub\_buf));

    struct sockaddr\_in server\_addr;

    int client\_fd = 0;

    struct hostent \*server\_host = NULL;

    while(1){

    if (!wifi\_get\_connect\_status()) {

            // LOGI(TAG, "wifi disconnected!");

    vTaskDelay(1000 / portTICK\_RATE\_MS);

    continue;

    }

    client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (client\_fd < 0) {

    LOGI(TAG, "create socket fail: %d", client\_fd);

    vTaskDelay(1000 / portTICK\_RATE\_MS);

    continue;

    }

    memset(&server\_addr, 0, sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

//修改为你的服务器端口号

server\_addr.sin\_port = htons(80);

//修改为你的连接IP地址

    uint8\_t upload\_srv\_ip[4] = {192, 168, 9, 146};

    memcpy(&server\_addr.sin\_addr.s\_addr, upload\_srv\_ip, 4);

    re\_hostbyname\_cnt = 0;

    uint8\_t sip[4];

    memcpy(sip, (void \*)&server\_addr.sin\_addr.s\_addr, 4);

    LOGI(TAG, "connectting server:(%d.%d.%d.%d)", sip[0], sip[1], sip[2], sip[3]);

    for (;;) {

                    //连接服务器

    if (connect(client\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

    LOGE(TAG, "client connect %d", client\_fd);

    LOGI(TAG, "connect failed!");

    vTaskDelay(5000 / portTICK\_RATE\_MS);

    close(client\_fd);

    break;

    }

    LOGI(TAG, "connect success!");

    g\_client\_fd = client\_fd;

    const struct timeval timeout = { 10, 0 };

    setsockopt(client\_fd, SOL\_SOCKET, SO\_RCVTIMEO, &timeout, sizeof(timeout));

    memset(sub\_buf, 0x00, sizeof(sub\_buf));

            //发送注册包

    sprintf(sub\_buf, "hello");

    LOGI(TAG, "\nsubscribe buff: %s", sub\_buf);

    send(client\_fd, sub\_buf, strlen(sub\_buf), 0);

    b\_start\_keep\_alive = true;

    //处理服务器发送的消息

    while (1) {

    memset(databuff, 0x00, sizeof(databuff));

    int len = recv(client\_fd, databuff, sizeof(databuff), 0);

    if (len > 0) {

    LOGE(TAG, "recv: %s", databuff);

    }

    if (len == 0) {

    LOGE(TAG, "close fd %d", client\_fd);

    break;

    }

    if (len < 0) {

    if (errno == EAGAIN || errno == EWOULDBLOCK || errno == EINTR) {

        ;

    } else {

    LOGE(TAG, "len < 0");

    break;

    }

    }

    close(client\_fd);

    b\_start\_keep\_alive = false;

    break;

    } //recv

    }

    while (1)

    {

    vTaskDelay(10000 / portTICK\_RATE\_MS);

    }

        }

在上节的wifi连接状态函数中添加创建TCP任务创建即可

/\*

    wifi连接状态

 \*/

void wifi\_event\_handler(wifi\_event\_indicate\_t event){

    switch (event)

    {

        case WIFI\_EVENT\_CONNECT:

            LOGE(TAG, "wifi\_connect");

    //WIFI连接成功创建TCP任务

    if (!tcpc\_task\_handle) {

        if (xTaskCreate(tcp\_client\_conn, "tcp\_client\_conn", TCPC\_TASK\_SIZE, NULL, 2, &tcpc\_task\_handle) != pdPASS) {

            LOGE(TAG, "create tcp\_client\_conn fail");

        }

    }

            break;

        case WIFI\_EVENT\_DISCONNECT:

            LOGE(TAG, "wifi\_disconnect");

            break;

        default:

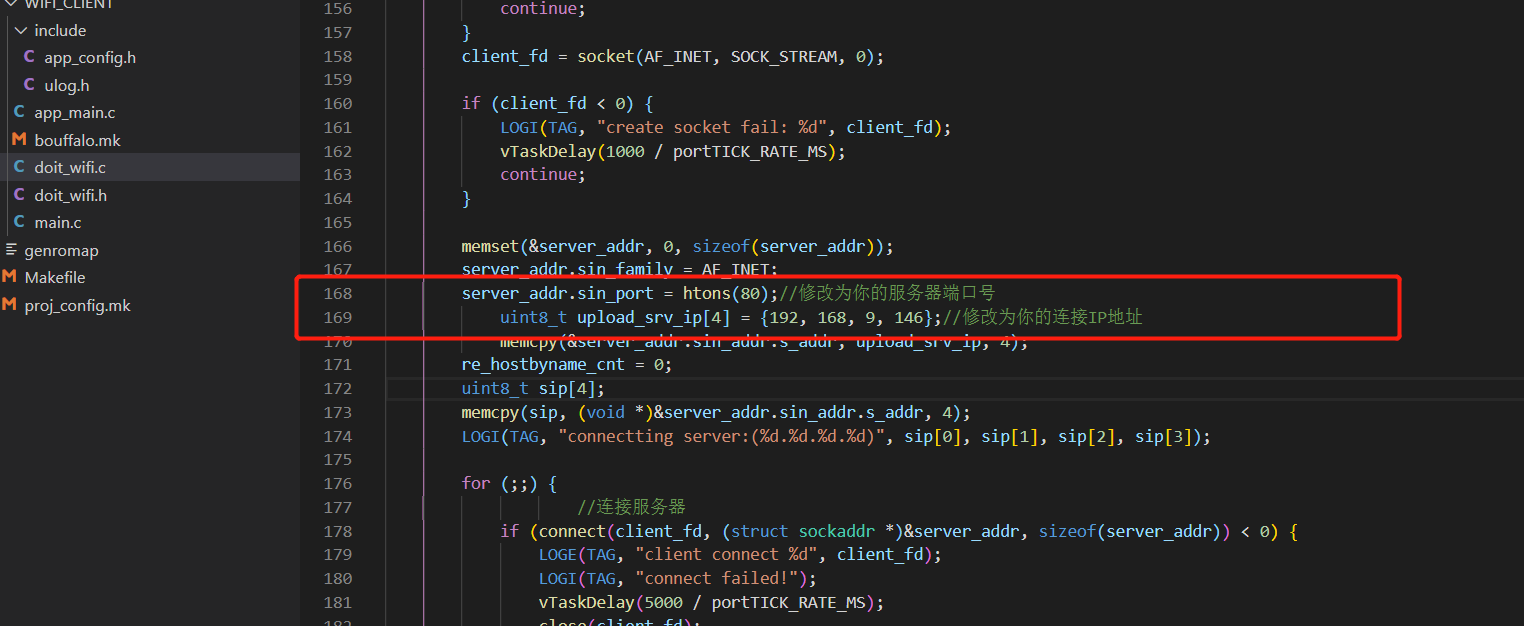
            LOGE(TAG, "wifi\_event: %d",event);

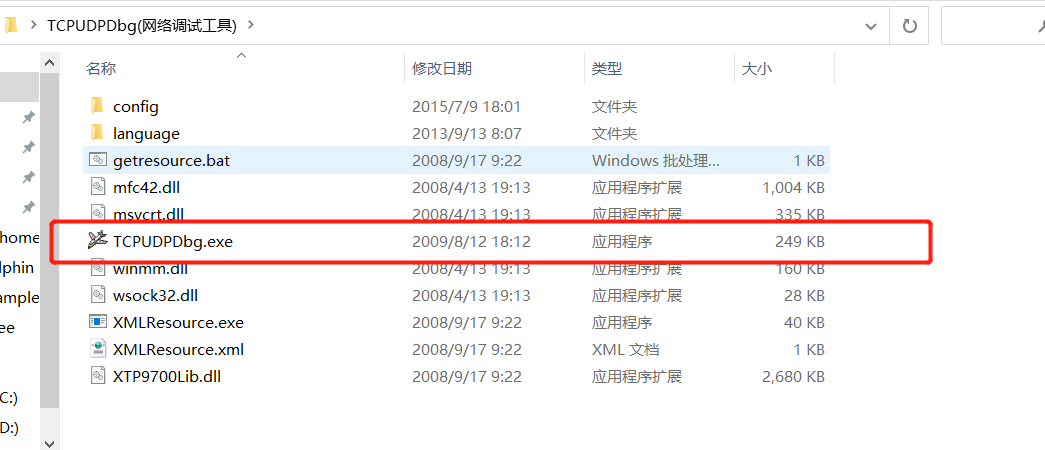
            break;

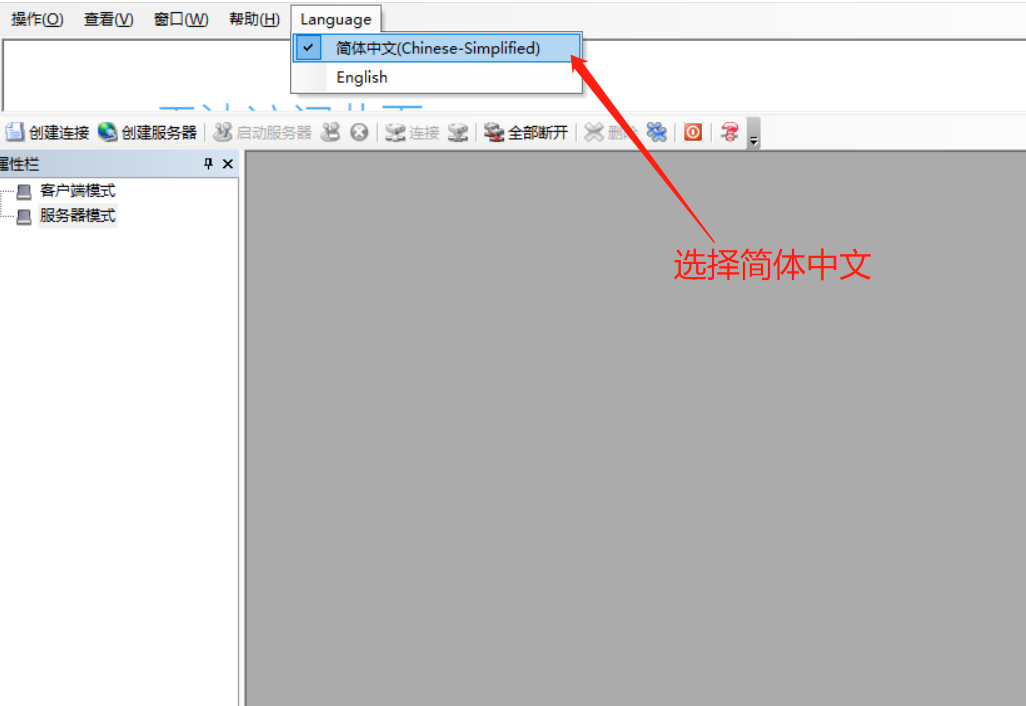
    }

}

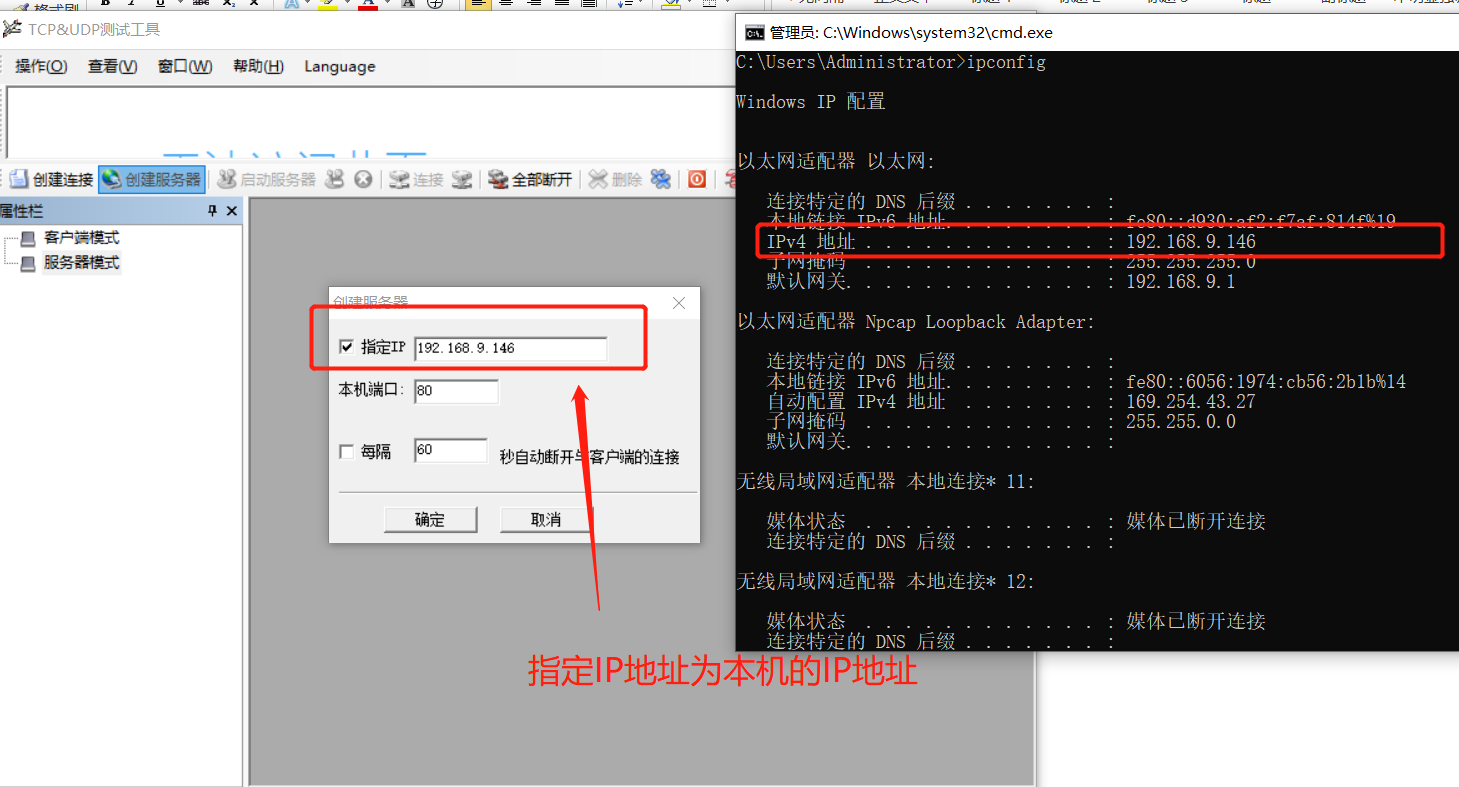
在例程中根据你自己的TCP服务器修改IP地址和端口号，将程序编译好完成后烧录到模块，下面用TCP测试工具进行测试。



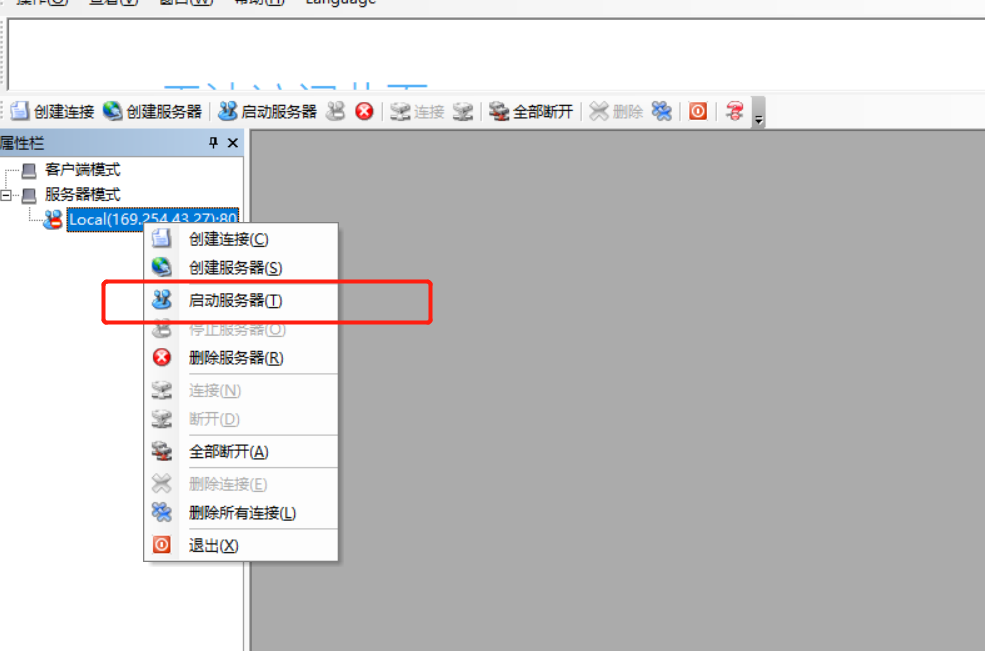
本例程所用到的TCP测试工具，下载地址：[http://bbs.doit.am/forum.php?mod=viewthread&tid=174&highlight=tcp](http://bbs.doit.am/forum.php?mod=viewthread&tid=174&highlight=tcp)也可以用其他TCP调试工具进行测试。



按win+R打开程序搜索，输入cmd按回车打开cmd窗口，输入ipconfig查看本地的IP地址，在调试工具中创建服务器，将IP地址指定为查看到的本机IP地址。

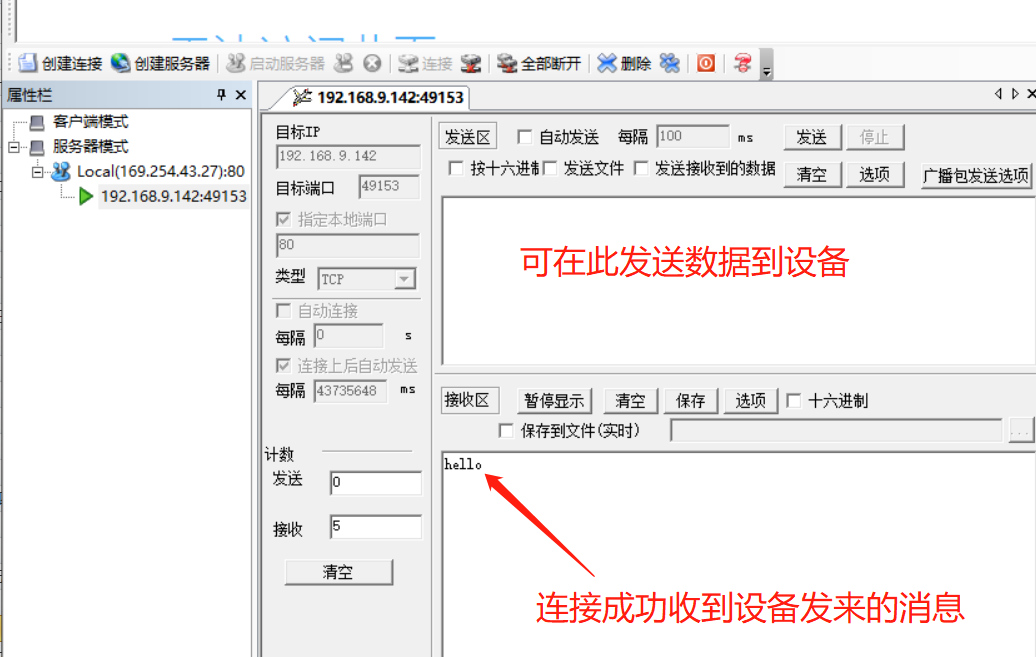


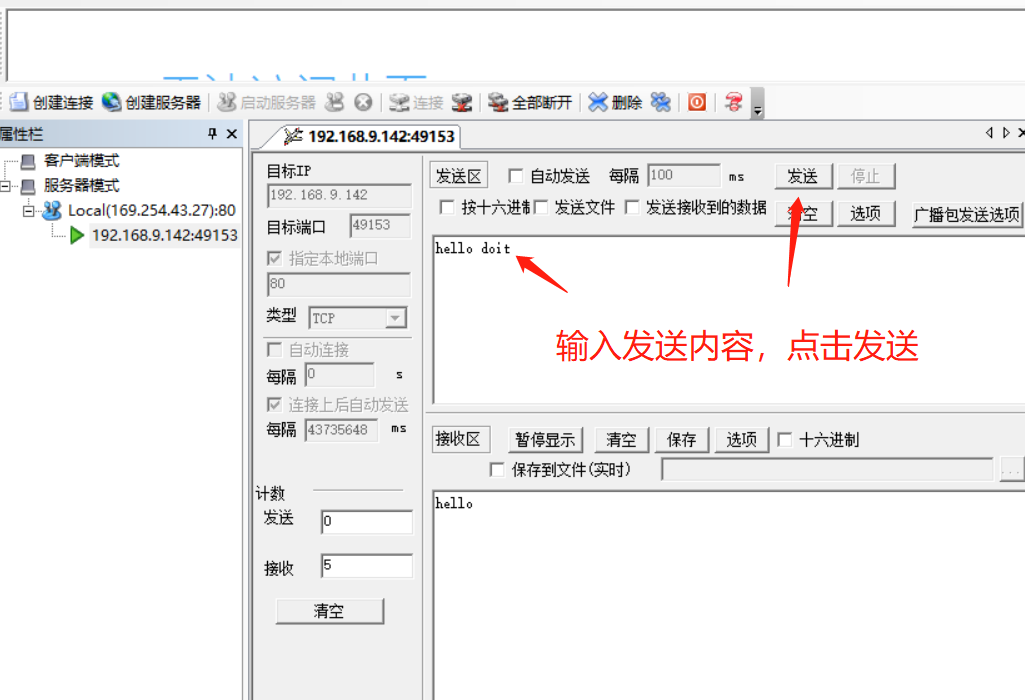
创建服务器后启动服务器



等待模块进行TCP连接，连接成功后会向服务器发送“hello”,串口会输出相关信息，同时在TCP调试工具中会出现一个新的连接，









## 5.3.TCP SERVER

在上一节我们成功实现了模块作为TCP client接入到TCP server实现通讯，这节我们来实现模块作为TCP server,让其他TCP client接入实现通讯。

在5.1节实现wif连接的基础上，在wifi连接成功后创建TCPserver任务。

void wifi\_event\_handler(wifi\_event\_indicate\_t event){

    switch (event)

    {

        case WIFI\_EVENT\_CONNECT:

            LOGE(TAG, "wifi\_connect");

    //WIFI连接成功创建TCP server 任务

    if (!tcpc\_task\_handle) {

    if (xTaskCreate(tcp\_server, "tcp\_server", TCPC\_TASK\_SIZE, NULL, 2, &tcpc\_task\_handle) != pdPASS) {

            LOGE(TAG, "create tcp\_server fail");

        }

    }

            break;

        case WIFI\_EVENT\_DISCONNECT:

            LOGE(TAG, "wifi\_disconnect");

            break;

        default:

            LOGE(TAG, "wifi\_event: %d",event);

            break;

    }

}

//TCP server任务

static void tcp\_server(void \*arg)

{

    char databuff[512];

    uint8\_t \*recv\_data;

    uint32\_t sin\_size;

    int sock = -1, connected, bytes\_received;

    struct sockaddr\_in server\_addr, client\_addr;

    char \*host = (char\*)arg;

    LOGE(TAG," start  tcp\_server ");

    recv\_data = (uint8\_t \*)pvPortMalloc(IPERF\_BUFSZ);

    if (recv\_data == NULL)

    {

        LOGE(TAG,"No memory");

        goto \_\_exit;

    }

    (void) host;

    sock = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sock < 0) {

        LOGE(TAG,"Socket error");

        goto \_\_exit;

    }

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(IPERF\_PORT);

    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;//INADDR\_ANY;

    memset(&(server\_addr.sin\_zero), 0x0, sizeof(server\_addr.sin\_zero));

    if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(struct sockaddr)) == -1) {

        LOGE(TAG,"Unable to bind");

        goto \_\_exit;

    }

    if (listen(sock, 5) == -1) {

        LOGE(TAG,"Listen error");

        goto \_\_exit;

    }

    while (1) {

        sin\_size = sizeof(struct sockaddr\_in);

        connected = accept(sock, (struct sockaddr \*)&client\_addr, (socklen\_t \*)&sin\_size);

        LOGI(TAG,"new client connected from (%s, %d)",

                  inet\_ntoa(client\_addr.sin\_addr),ntohs(client\_addr.sin\_port));

        {

            int flag = 1;

            setsockopt(connected,

                IPPROTO\_TCP,     /\* set option at TCP level \*/

                TCP\_NODELAY,     /\* name of option \*/

                (void \*) &flag,  /\* the cast is historical cruft \*/

                sizeof(int));    /\* length of option value \*/

        }

        while (1) {

            memset(databuff, 0x00, sizeof(databuff));

            bytes\_received= recv(connected, databuff, sizeof(databuff), 0);

            if (bytes\_received <= 0) break;

            send(connected, databuff, strlen(databuff), 0);//将接收到的消息发送回客户端

            LOGE(TAG, "recv: %s", databuff);

        }

        if (connected >= 0) closesocket(connected);

        connected = -1;

    }

\_\_exit:

    if (sock >= 0) closesocket(sock);

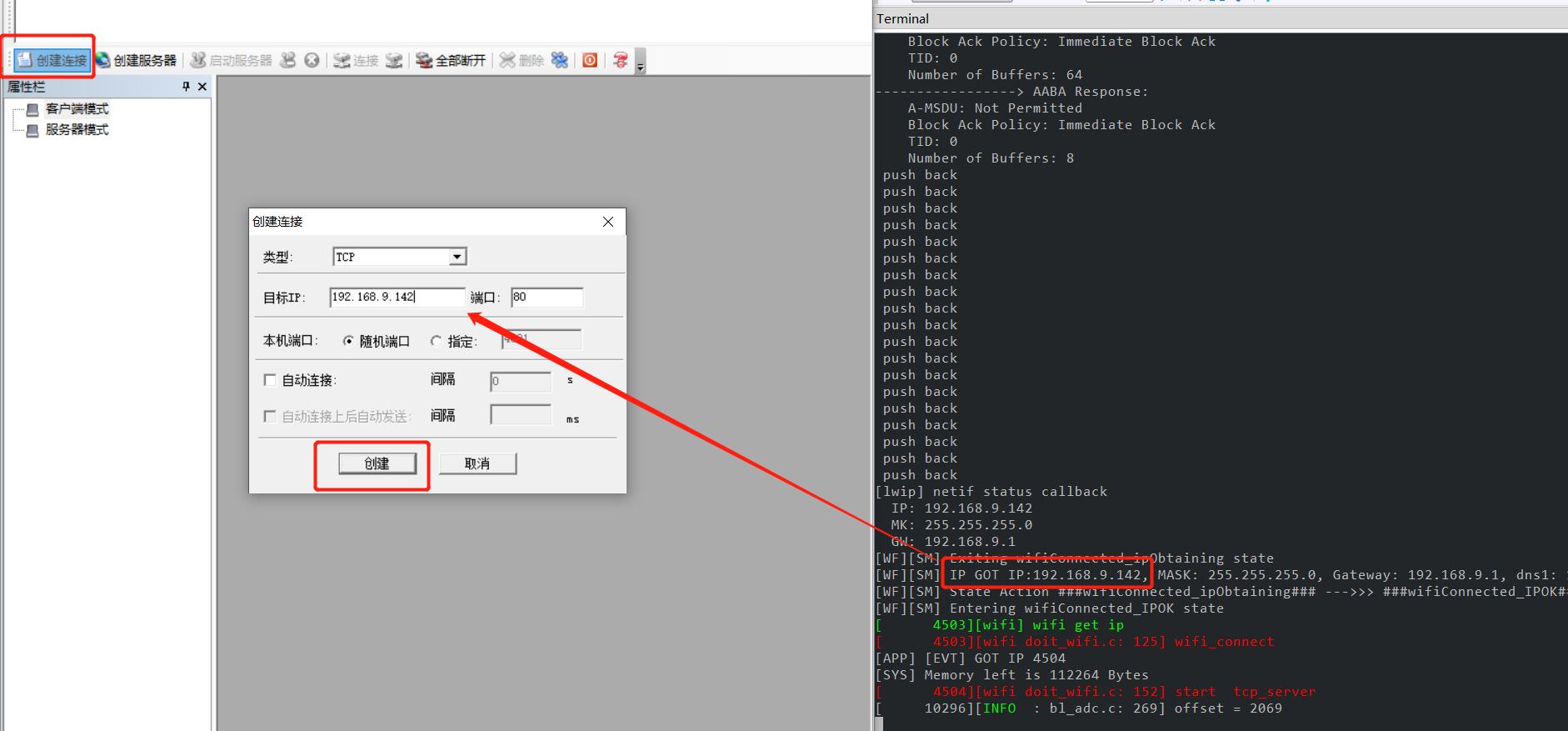
    if (recv\_data) vPortFree(recv\_data);

    if (arg) vPortFree(arg);

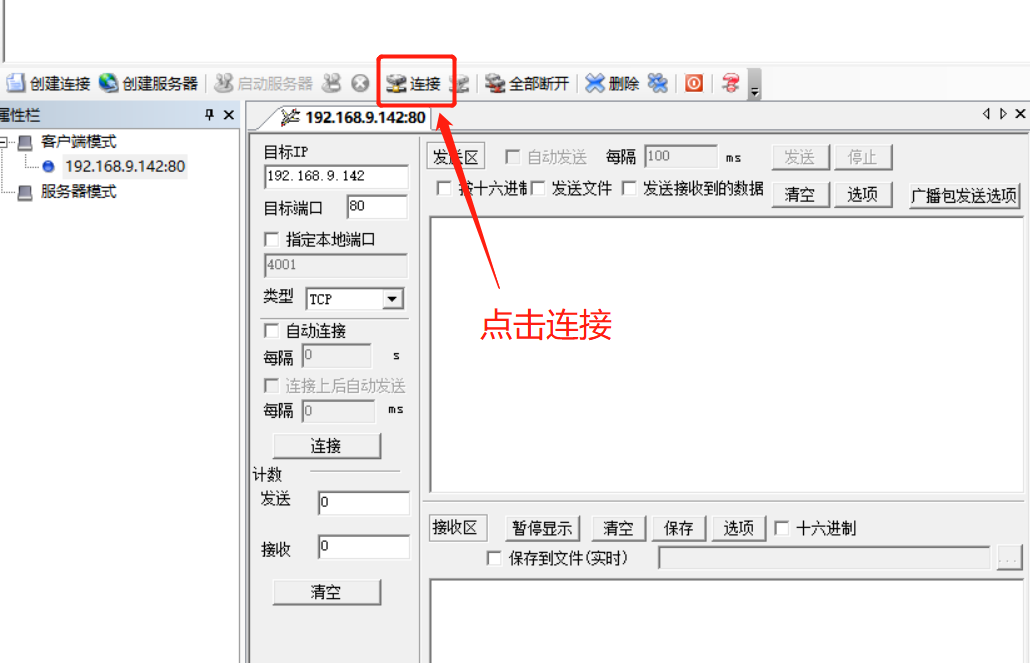
}

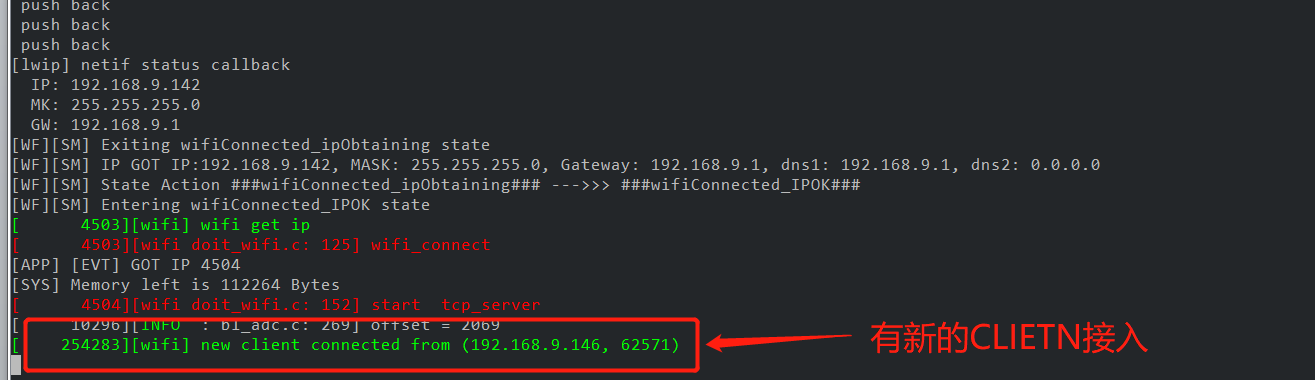
编译好程序后，将固件烧录到模块后复位，观察串口输出，当连接上wifi后，用TCP测试工具建立TCP client连接。



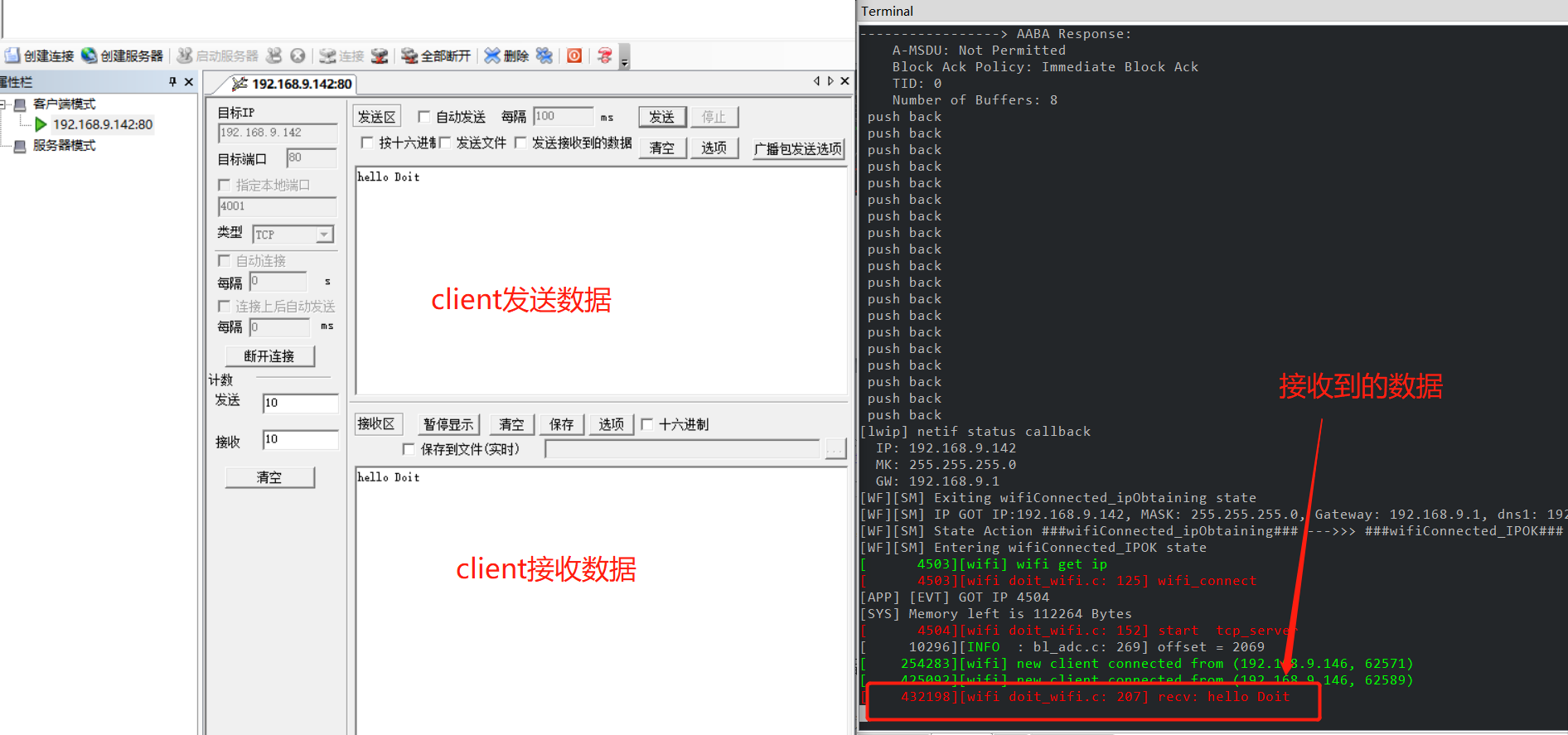


创建成功后点击连接，连接成功后串口会输出该连接的IP和端口





连接成功后进行通讯测试，在测试工具中发送的数据，设备在接收到数据后会将数据发送回客户端，同时串口会打印出数据。



# 六．综合项目

## 6.1.Dohome智能全彩灯具

通过BL602芯片作为系统主控，实现智能全彩灯的控制，可通过wifi接入Dohome平台，同时DoHome APP已经对接了各大智能音箱（小米小爱，百度小度，阿里天猫精灵，Amazon，Google，京东叮咚）。可根据需求通过BL602模组制作智能灯具和其他相关产品。

项目地址：<https://github.com/SmartArduino/Doiting_BL/tree/master/examples/bl602_Dohome_Light_RGB>

在app\_config.h中定义了软件的版本号，开发板产生的热点名，我们需连接通信的DoHome服务器，和我们udp,tcp通信使用的端口号，以及使用到的IO口：

/\*

 \* Log config

\*/

#define USE\_UART\_LOG        1

#define LOG\_COLOR\_ENABLE    0

#define USE\_UDP\_LOG         0

#if USE\_UART\_LOG

#define LOG\_MIAN\_EN       1

#define LOG\_WIFI\_EN       1

#define LOG\_FLASH\_EN      1

#define LOG\_UTILS\_EN      0

#define LOG\_PROTOCOL\_EN   1

#define LOG\_DEVICE\_EN     1

#define LOG\_UDP\_EN        1

#define LOG\_LIGHT\_EN      0

#define LOG\_TCP\_EN        0

#define LOG\_TCP\_CONFIG\_EN 1

#define LOG\_HTTP\_EN       1

#define LOG\_UPLOAD\_EN     1

#define LOG\_SNTP\_EN       1

#define LOG\_TIMER\_EN      1

#define LOG\_PROCES\_EN     0

#define LOG\_LED\_DRIVE\_EN  0

#define LOG\_LED\_BOARD\_EN  0

#define LOG\_OTA\_EN        1

#define LOG\_UDPLOG\_EN     0

#define LOG\_FACTORY\_EN    1

#define LOG\_KEY\_EN        1

#define LOG\_LOG\_EN        1

#define LOG\_IR\_EN         1

#define LOG\_PRODUCT\_EN    1

#endif

/\*

 \* product config

\*/

#define IS\_WYRGB

// #define IS\_WY

// #define IS\_STRIP

#if defined IS\_STRIP

#define HARDWARE\_TYPE           "\_STRIPE"

#elif defined IS\_WYRGB

#define DEVIVE\_NAME\_PREFIX      "Light\_"

#define HARDWARE\_TYPE           "\_DT-WYRGB"

#define PIN\_LED\_R       27

#define PIN\_LED\_G       13

#define PIN\_LED\_B       26

#define PIN\_LED\_W       4

#define PIN\_LED\_Y       16

#elif defined IS\_WY

#define HARDWARE\_TYPE           "\_DT-WY"

#else

#define HARDWARE\_TYPE           "\_LED"

#endif

/\*

 \* doit config

\*/

#define FW\_VERSION              "1.1.0"

#define DEF\_SSID\_PREFIX         "DoHome\_"

#define DEF\_HOSTNAME            "DoHome"

#define ORGANIZATION            "\_DOIT"

#define CHIP\_TYPE               "\_ESP32"

#define REMOTE\_SERVER\_IP        "115.28.78.23"

#define REMOTE\_SRV\_HOST         "led\_iot.doit.am"

#define REMOTE\_SRV\_PORT         8899//6007

#define UPLOAD\_SRV\_HOST         "dohome.doit.am"

#define UPLOAD\_SRV\_PORT         8008

#define NTP\_SERVER\_HOST         "xinfeng.doit.am"

#define NTP\_SERVER\_PORT         80

#define UPLOAD\_SRV\_URL          "http://dohome.doit.am:8008"

#define NTP\_SERVER\_URL          "http://xinfeng.doit.am/iot\_api/get\_iot\_time.php"

#define MAX\_STA\_NUM             3

#define TCP\_SRV\_PORT            5555

#define SPI\_FLASH\_SEC\_SIZE      4096

#define UDP\_BROADCAST\_INFO\_PORT 6095

#define UDP\_SRV\_PORT            6091

#define WEB\_SRV\_PORT            80

#define DNS\_SRV\_PORT            53

#define OTA\_TCP\_PORT            6093

#define OTA\_TCP\_ECHO\_PORT       6094

#define HTTP\_UPO

#define WIFI\_DEF\_MAGIC          0x03

#define KEEPALIVE\_TIME          60\*1000

#define RESTART\_TRIGGER\_COUNT      3

#define EXT\_UPLOAD\_PERIOD           (60)        //second

#define BASE\_UPLOAD\_PERIOD          (50)        //second

#define SYNC\_TIME\_PERIOD            (30\*60)         //second

#define MIN\_TIMEZONE\_OFFSET     0

#define MAX\_TIMEZONE\_OFFSET     23

#define TIMER\_NUM               7       // you can setup at most TIMER\_NUM timer

#define SEQ\_SIZE                (16 \* 5)

#define POWERUP\_COLOR\_MAX\_NUM   16

#define MAX\_CUSTOM\_COLOR        16

#define OTA\_PKG\_SIZE            240

#define HB\_SEND\_MAX\_COUNT       8

#define SSID\_PASS\_LEN           64

#define SSID\_SSID\_LEN           32

#define COLOR\_MAX               1000

#define LIGHT\_GRA\_STEP          10

#define USE\_WM\_HTTP             0

#ifndef IS\_STRIP\_WS2812

#define USE\_RTC\_TIME            1

#endif

#define RESTART\_REASON\_RESET\_NULL                   0

#define RESTART\_REASON\_RESET\_DEV                    1

#define RESTART\_REASON\_REBOOT\_REQ                   2

为了稳定的配网，此方案使用ap配网，通过连接开发板产生的热点，手机app把路由器wifi名称密码发送给开发板完成配网：

int wifi\_setup\_ap(void)

{

    LOGI(TAG, "wifi\_setup\_ap");

    char def\_ap\_ssid[33] = {"BL602\_AP"};

    memset(def\_ap\_ssid, 0x00, sizeof(def\_ap\_ssid));

    get\_default\_ssid(def\_ap\_ssid);

    wifi\_interface\_t wifi\_interface\_ap = wifi\_mgmr\_ap\_enable();

    dhcpd\_set\_server\_gw("192.168.4.1");

    wifi\_mgmr\_ap\_start(wifi\_interface\_ap, def\_ap\_ssid, 0, NULL, 1);

    return 1;

}

连上服务器后开始发送心跳包：

static void vTimerCallback(TimerHandle\_t timer)

{

    //服务器连接上了,开始发送心跳

    if (b\_start\_keep\_alive) {

        char keep\_msg[256];

        memset(keep\_msg, 0x00, sizeof(keep\_msg));

        sprintf(keep\_msg, "cmd=keep&device\_id=%s&device\_key=%s\r\n", dev\_id, dev\_key);

        send(g\_client\_fd, keep\_msg, strlen(keep\_msg), 0);

        LOGI(TAG, "keepalive buff: %s", keep\_msg);

    } else {

        // LOGI(TAG, "tcp client disconnect=====");

    }

}

处理下发的数据进行设备控制：

/\*

    处理tcp服务下发的数据

 \*/

static void parse(int socket, char \*buff, int len)

{

    // LOGI(TAG, "tcp\_package: %s", buff);

    if (strstr(buff, "cmd=subscribe") != NULL) {

        char \*pstart = strstr(buff, "res=");

        if (pstart != NULL) {

            pstart += strlen("res=");

            int r = \*pstart - '0';

            if (r == 1) {

                LOGI(TAG, "register device on remote server OK");

            } else {

                LOGI(TAG, "register device failed!");

            }

        } else {

            LOGI(TAG, "result not found!");

        }

    } else if (strstr(buff, "cmd=publish") != NULL) {

        char \*pstart = strstr(buff, "message=");

        if (pstart != NULL) {

            pstart += strlen("message=");

            char ret\_buf[256];

            parse\_dohome\_protocol(pstart, len - (pstart - buff), ret\_buf);

        } else {

            LOGI(TAG, "message not found!");

        }

    } else if (strstr(buff, "cmd=keep") != NULL) {

        LOGI(TAG, "send heartbeat OK");

    } else {

        LOGI(TAG, "no such command");

    }

}

将程序编译好后烧录到芯片中，手机下载Dohome APP，添加设备，按提示进行设备的配网

APP测试

1. 扫码下载APP



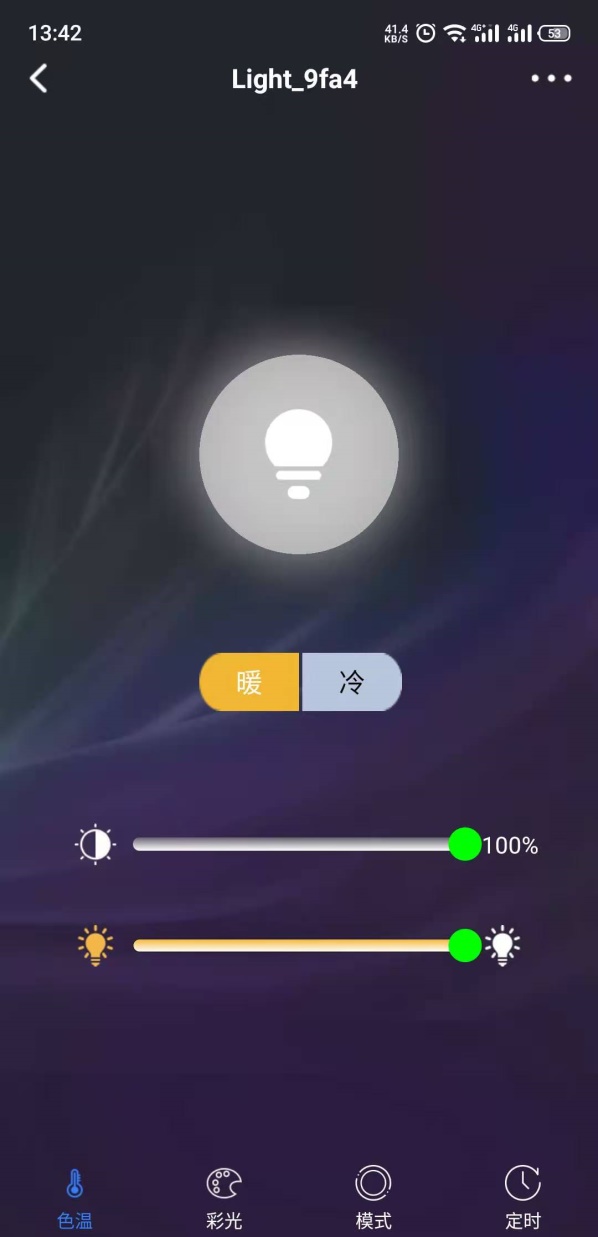
2.进入APP点击右上角添加设备，选择彩色灯



1. 按APP提示进行配网操作

4.添加设备成功，能从Dohome APP控制设备

配网成功后，可使用app控制，也可以绑定智能音箱控制。



绑定完智能音箱的账号后，就可以使用我们自己的智能音箱进行控制。

## 6.2. dolphin蓝牙跳蛋

通过PWM控制震动马达的转速实现不同的震动强度变化，通过蓝牙连接设备，APP下发蓝牙指令实现设备的控制。

**项目地址：**https://github.com/SmartArduino/Doiting\_BL/tree/master/examples/dolphin

蓝牙初始化：

void do\_ble\_init(void){

    // Initialize BLE controller

    LOGI(TAG, "do\_ble\_init ");

    ble\_controller\_init(configMAX\_PRIORITIES - 1);

    //Initialize BLE Host stack

    hci\_driver\_init();

    bt\_enable(bt\_ready\_cb);

    bt\_set\_name("LVS-Lush122-XT");

    bt\_gatt\_service\_register((struct bt\_gatt\_service \*)&dis);

    notify\_attrs = &ble\_attrs[1];

    bt\_conn\_cb\_register(&conn\_callbacks);

    ble\_start\_advertise();

}

蓝牙接收到的数据处理：

//蓝牙处理函数

static void ble\_reve\_cb(struct bt\_conn \*conn, const char \*buf, u16\_t len){

    char cmd[30] = "";

    char data[15] = "";

    char ret\_str[20] = "OK";

    //LOGI(TAG, "ble\_reve\_buf: %s", buf);

    //do\_ble\_notify(conn, ret\_str, strlen(ret\_str));

    char \*dp = strchr(buf,';');

    if(dp == NULL){

        return;

    }else{

        \*dp = '\0';

    }

    LOGI(TAG, "ble\_reve\_buf: %s", buf);

    char \*p = strchr(buf,':');

    if(p != NULL){

        strncpy(cmd, buf, p-buf);

        strncpy(data, p+1, len-(int)(p-buf)-2);

    }else{

        strcpy(cmd, buf);

        cmd[len-1] = '\0';

    }

    LOGI(TAG, "cmd: %s  data: %s", cmd, data);

    if(CDM\_CMP(cmd, "Status")){

        LOGI(TAG, "get Status");

        do\_ble\_notify(conn, ret\_str, strlen(ret\_str));

    }

    else if(CDM\_CMP(cmd, "Vibrate")){           //蓝牙在线模式控制

        LOGI(TAG, "set Vibrate");

        vibrate = atoi(data);

       Preset=0;

       current\_Preset=0;

       //do\_ble\_notify(conn, ret\_str, strlen(ret\_str));

    }

    else if(CDM\_CMP(cmd, "Preset")){            //9种预设模式选择

        LOGI(TAG, "set Vibrate");

        Preset = atoi(data);

    }

    else if(CDM\_CMP(cmd, "Battery")){           //获取电量

        int32\_t percent=get\_battery();           //电池百分比

        if(percent==100) {ret\_str[0]='1';ret\_str[1]='0';ret\_str[2]='0';}

        else{

        int8\_t shiwei= percent/10; int8\_t gewei= percent%10;

        ret\_str[0]='0';

        ret\_str[1]=shiwei+'0';

        ret\_str[2]=gewei+'0';

    }

        ret\_str[3]=';';

        do\_ble\_notify(conn, ret\_str, strlen(ret\_str));//设备蓝牙回复当前电量

    }

    else if(CDM\_CMP(cmd,"Light")){      //灯光闪烁开关

        if(data[0]=='o'&&data[1]=='n'){

        light\_flag=true;

        if(ble\_connect) bl\_gpio\_output\_set(LED\_STATUS, 0);

        }

        if(data[0]=='o'&&data[1]=='f'&&data[2]=='f'){

        light\_flag=false;

        }

    }

    else if(CDM\_CMP(cmd,"PowerOff")){   //关闭设备

        sleep\_start();

    }

    else if(CDM\_CMP(cmd,"OTA")){         //蓝牙发送  OTA;   进入OTA

    LOGI(TAG, "OTA");

    ota\_flag=true;

    flash\_storage\_init();

    wifi\_init(wifi\_event\_handler);

    vTaskDelay(500 / portTICK\_RATE\_MS);

    wifi\_setup\_sta();

    ota\_time= hal\_timer\_now\_s();          //记录OTA开始的时间

    char ota\_ret[20]="OTA\_START";

    do\_ble\_notify(conn, ota\_ret, strlen(ota\_ret));

    }

    else if(CDM\_CMP(cmd,"WIFI")){          //蓝牙配网指令： "WIFI:账号，密码;"

    LOGI(TAG, "SET WIFI ");

    user\_ssid\_t \*wifi\_info = flash\_get\_user\_ssid\_config();

    memset(wifi\_info->ssid, 0, sizeof(wifi\_info->ssid));

    memset(wifi\_info->password, 0, sizeof(wifi\_info->password));

    flash\_ssid\_config\_write();

    char \*p = strchr(data,',');

      if(p != NULL){

        strncpy(wifi\_info->ssid, data, p-data);

        strncpy(wifi\_info->password, p+1, len-(int)(p-data)-2);

        flash\_ssid\_config\_write();

        char ret[20]="SET\_WIFI";

        do\_ble\_notify(conn, ret, strlen(ret));

        do\_ble\_notify(conn, wifi\_info->ssid, strlen(wifi\_info->ssid));

        do\_ble\_notify(conn, wifi\_info->password, strlen(wifi\_info->password));

    LOGI(TAG,"saved SSID: %s, passwd: %s", wifi\_info->ssid, wifi\_info->password);

    }

    }

    vTaskDelay(100 / portTICK\_RATE\_MS);

    //do\_ble\_notify(conn, ret\_str, strlen(ret\_str));

}

void sleep\_task(void \*arg){           //大于1200S无震动，进入休眠模式

static int times=0;

while(1){

//LOGI(TAG, "start sleep");

    if(Grade==0&&vibrate==0&&(charge\_status == DISCHARGE)&&Preset==0) times++;

    else times=0;

    if(ota\_flag) times =0;

   //LOGI(TAG, "times:%d",times);

    while(times>1200) {

        hal\_hbn\_init(&key\_weakup\_pin, 1);

        hal\_hbn\_enter(0);}

    while(ota\_flag&&(hal\_timer\_now\_s()-ota\_time>300)){

        hal\_hbn\_init(&key\_weakup\_pin, 1);

        hal\_hbn\_enter(0);

    }

        vTaskDelay(1000 / portTICK\_RATE\_MS);

}

}

对蓝牙下发的指令进行解析，根据解析后的指令经行震动马达和灯光的控制：

void mada\_task(void \*arg){         //mada模式

uint32\_t duty = 0;

int i=0;

while(1){

    if(!ble\_connect&&(charge\_status == DISCHARGE)){                        //四档离线模式

        if(Grade==0) motor\_g\_set\_vibrate(0);

        if(Grade==1) motor\_g\_set\_vibrate(8);

        if(Grade==2) motor\_g\_set\_vibrate(12);

        if(Grade==3) motor\_g\_set\_vibrate(16);

        if(Grade==4) motor\_g\_set\_vibrate(20);

    }

    else if(ble\_connect&&(charge\_status == DISCHARGE)&&Preset==0){          //蓝牙在线模式

        motor\_g\_set\_vibrate(vibrate);

        if(vibrate==0)led\_status\_flash(0);

    }

    else if(ble\_connect&&(charge\_status == DISCHARGE)&&Preset!=0){

        memcpy(currentPreMod,presetMod[Preset-1],8);

        for(i=0;i<8;i++){

        duty =currentPreMod[i]\*100;

        motor\_g\_set\_duty(duty);

        current\_Preset=currentPreMod[i];

        if(light\_flag){led\_status\_flash(115-current\_Preset);}                   //9种预设模式灯光，在此实现跟马达同时变化

        vTaskDelay(300 / portTICK\_RATE\_MS);

        LOGI(TAG, "presetMod:%d",currentPreMod[i]);

    }

    }

    else {motor\_g\_set\_vibrate(0);Grade=0;vibrate=0;}

    vTaskDelay(100/portTICK\_RATE\_MS);

}

}

9种预设模式变化频率，可以在此调整每个模式下的震动频率变化

const uint8\_t presetMod[9][8] =

{

        { 100, 20, 100, 20, 100, 20, 100, 20 },     //Mod1

        { 50, 100, 50, 100, 50, 100, 50, 100 },     //Mod2

        { 40, 60, 80, 100, 100, 80, 60, 40 },       //Mod3

        { 100, 25, 100, 25, 100, 25, 100, 25 },     //Mod4

        { 75, 100, 75, 100, 75, 100, 75, 100 },     //Mod5

        { 20, 100, 60, 100, 60, 100, 30, 60 },      //Mod6

        { 75, 30, 20, 30, 100, 30, 25, 60 },        //Mod7

        { 100, 100, 100, 100, 100, 100, 100, 100 }, //Mod8

        { 38, 100, 80, 47, 28, 20, 28, 44 },        //Mod9

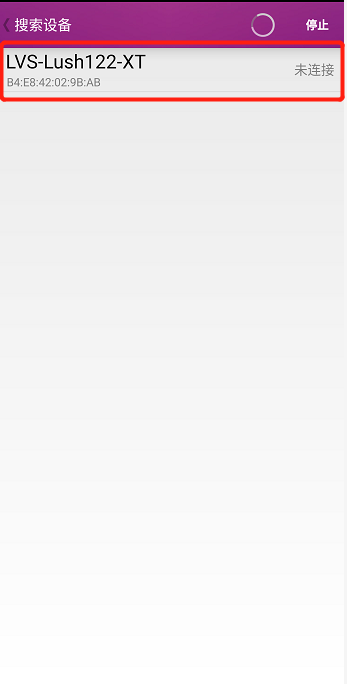
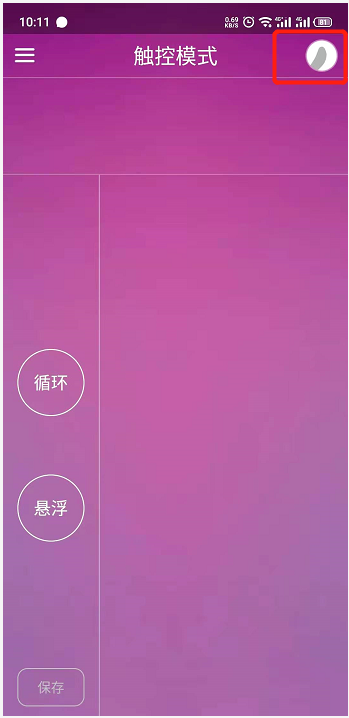
};

将程序编译好后烧录到芯片中，手机下载APP，连接设备蓝牙控制震动马达。

1.扫码下载APP



2.连接设备



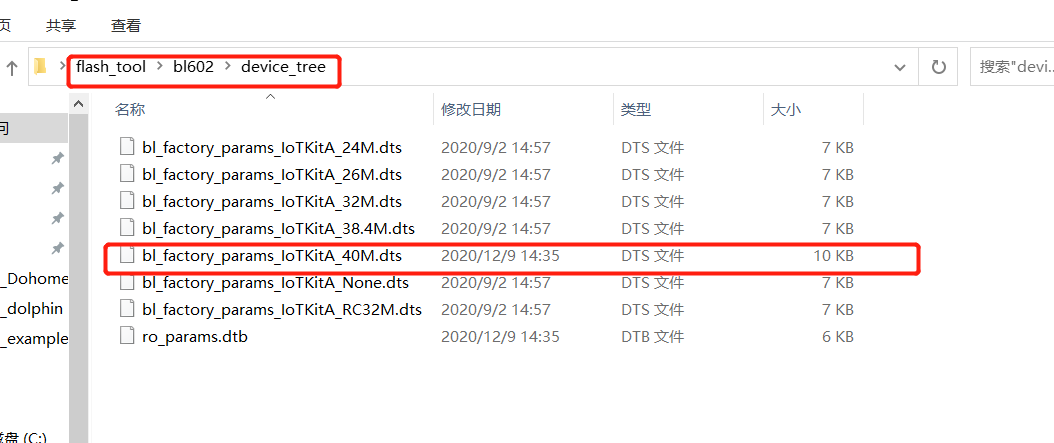
1. APP控制设备，可在APP选择多种震动模式



## 6.3.AT指令

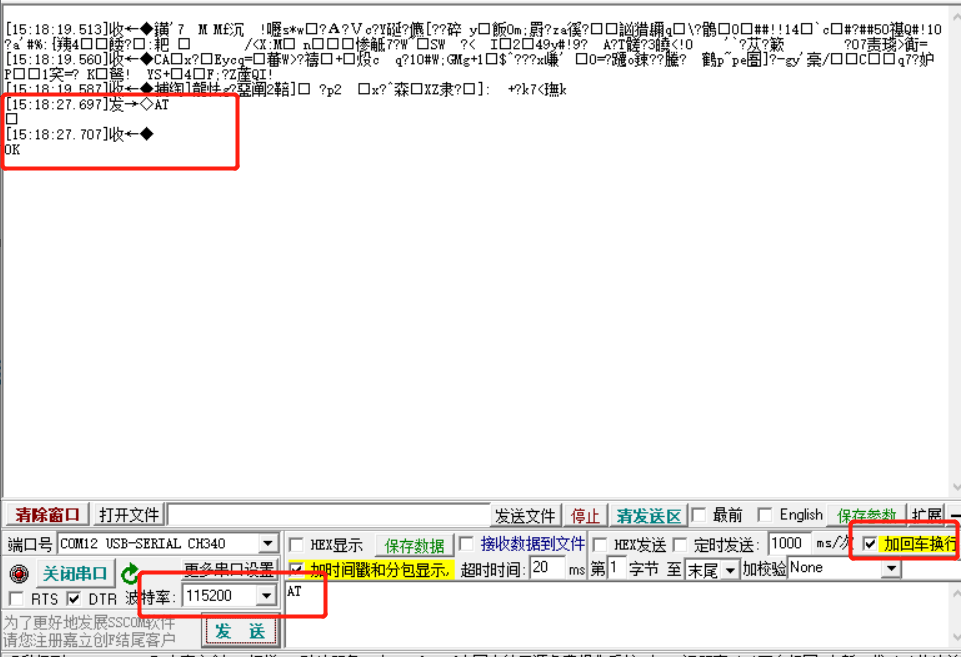
BL602/BL604模组都支持AT固件进行快速开发，AT固件的demo在SDK目录下的\customer\_app\bl602\_demo\_at，本例程通过AT指令实现一些wifi和ble的一些操作。

编译完成后，在烧录前需要最烧录工具下的一个文件就修改，打开该文件，找到uart部分将两组uart所用的引脚互换，修改后保存，保存后烧录固件到模块。



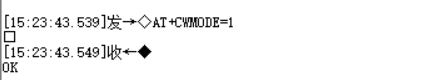


复位模块，选择波特率为115200，发送AT返回OK则说明AT固件运行正常

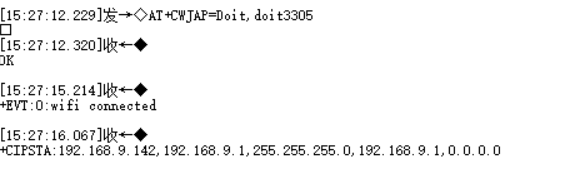


**TCP server测试**

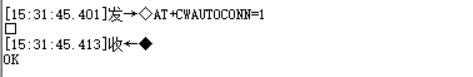
1.设置wifi模式：AT+CWMODE=1



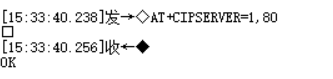
2.连接wifi：AT+CWJAP=Doit,doit3305



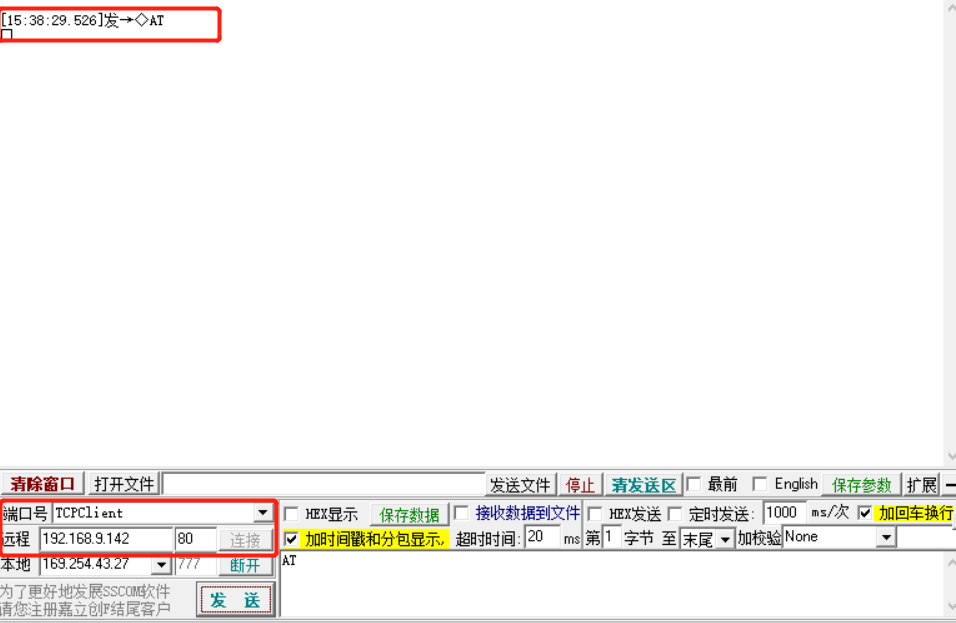
3.设置自动连接: AT+CWAUTOCONN=1



5.建立TCP server: AT+CIPSERVER=1,80



6.用TCP测试工具进行通讯测试，

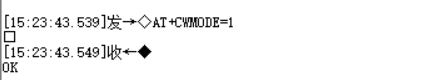


客户端的串口输出：

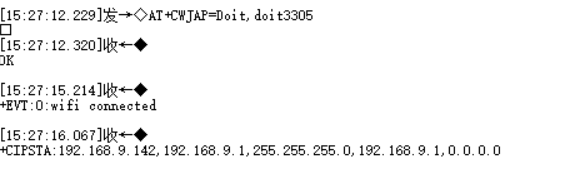


**TCP CLITNT测试**

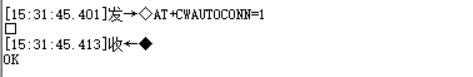
1.设置wifi模式：AT+CWMODE=1



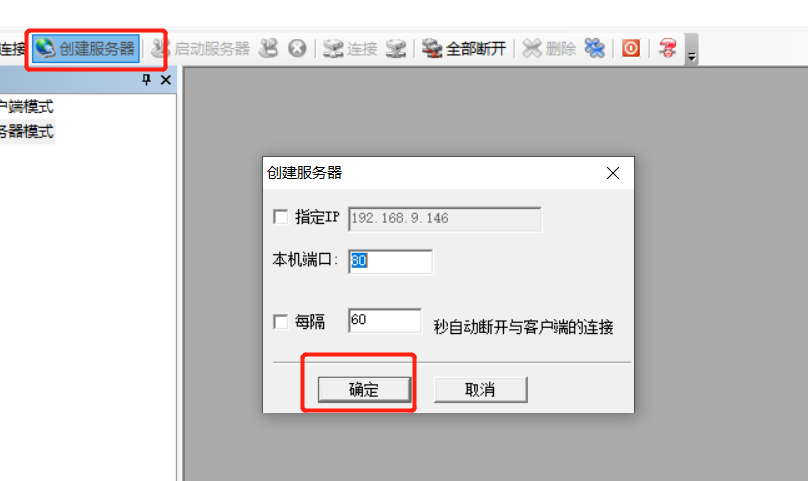
2.连接wifi：AT+CWJAP=Doit,doit3305

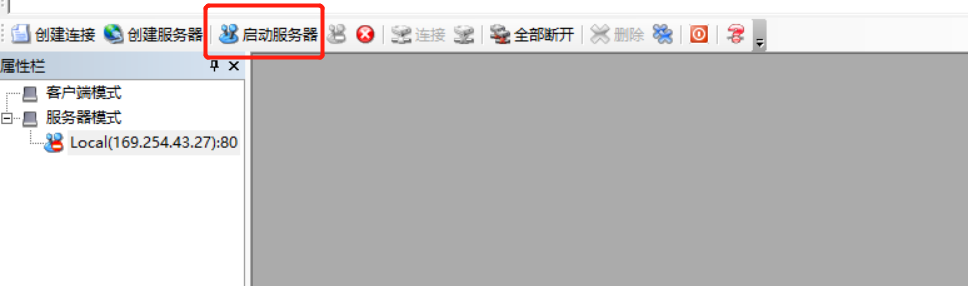


3.设置自动连接: AT+CWAUTOCONN=1

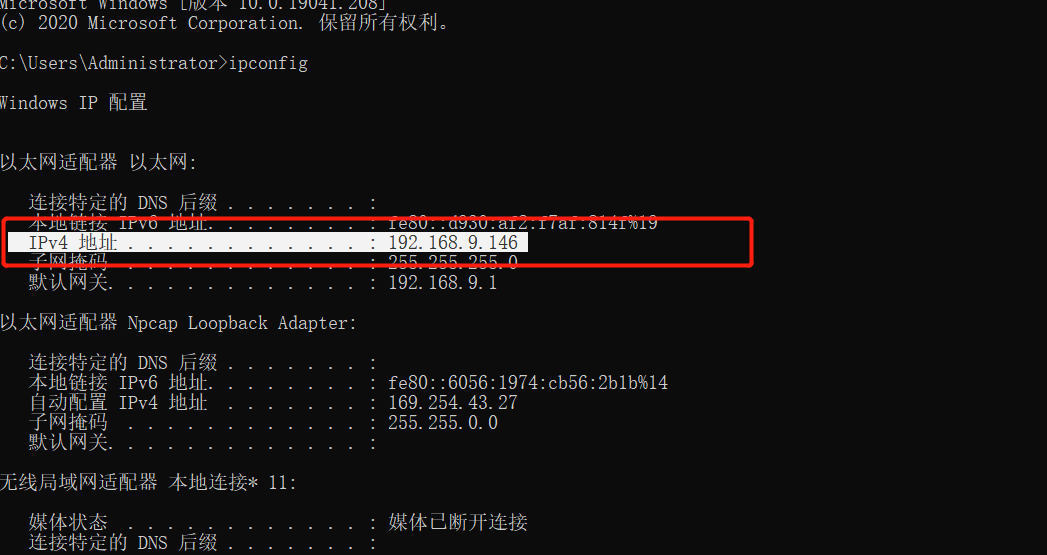


4.在测试工具中新建一个TCP server

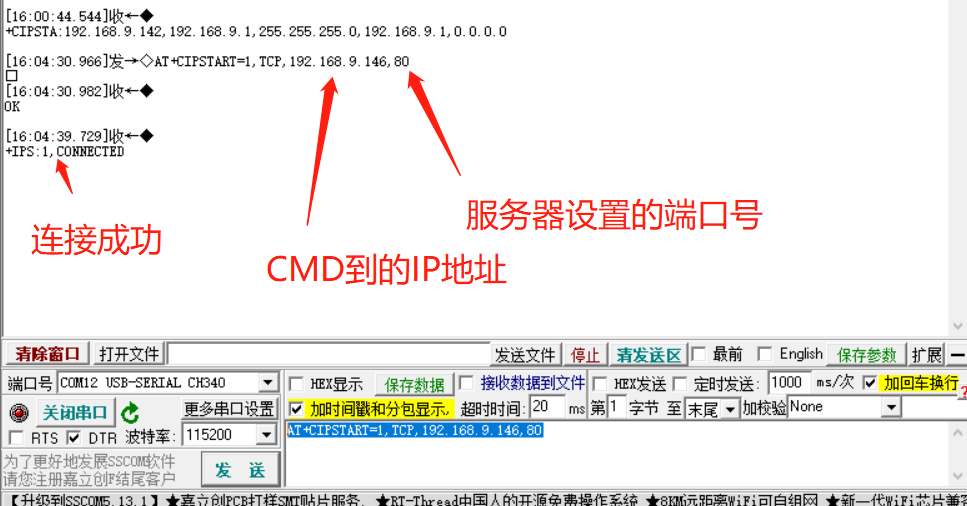




5.查询PC的IP地址，用于连接TCP server,进入CMD窗口，输入ipconfig

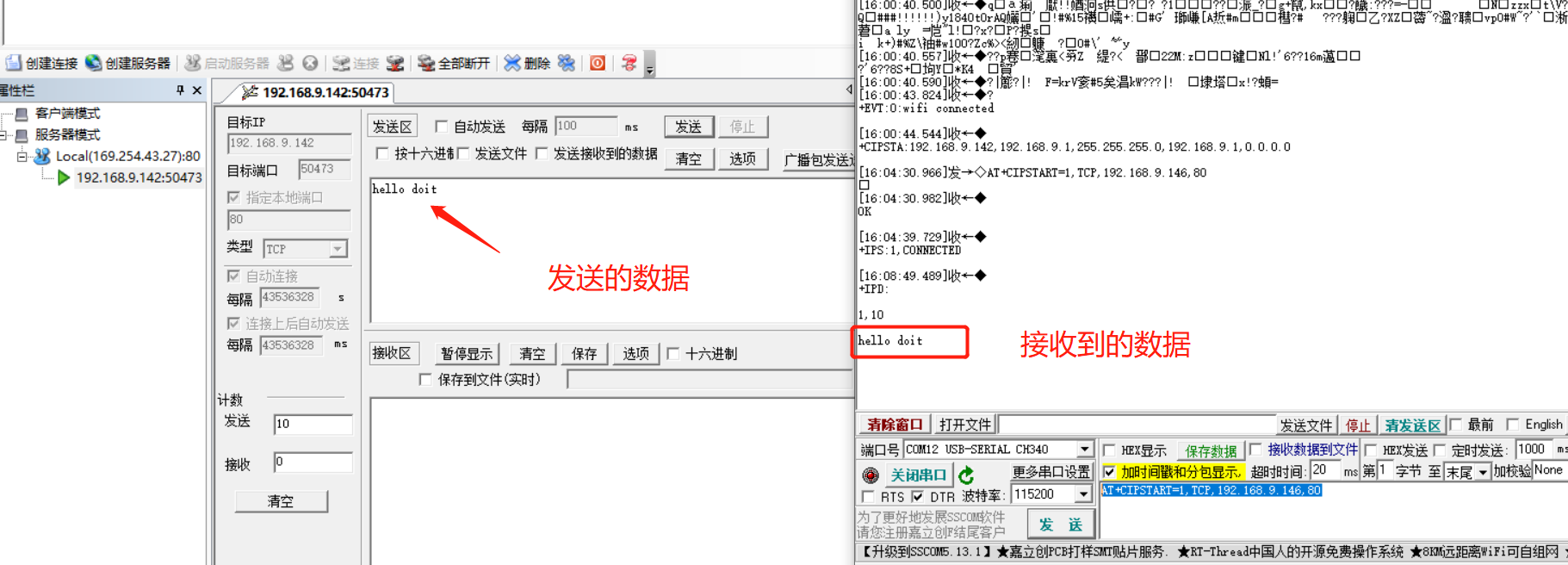


6.连接TCP server: AT+CIPSTART=1,TCP,192.168.9.146,80



7.收据收发测试

服务器发送数据到模块：

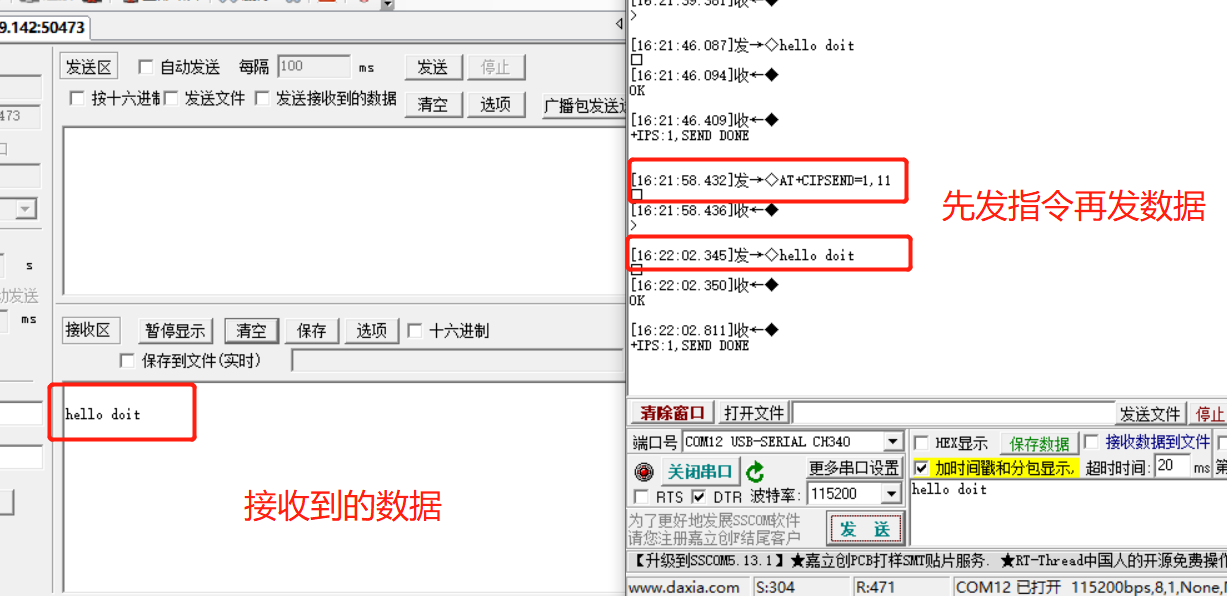


模块发送数据到TCP server:

1.发送数据指令：AT+CIPSEND=1,11

表示即将向 id 为 1 的连接发送 11 字节的数据

2.发送需要发送的数据



AT指令的其他开发，可根据BL602AT指令集进行开发。

# 免责申明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示

许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。



注 意

由于产品升级或其他原因，本手册内容有可能变更。深

圳四博智联科技有限公司保留在没有任何通知或者提示

的情况下对本手册的内容进行修改的权利。本手册仅作

为使用指导，深圳四博智联科技有限公司尽全力在本手

册中提供准确的信息，但是并不确保手册内容完全没有

错误，本手册中的所有陈述、信息和建议也不构成任何

明示或暗示的担保。