

雅鲁 IOT PLATFORM

数据接入指南



Issue 1.03

成都雅鲁科技

About this document

本文为雅鲁 IOT 平台的第三方通信模块接入指南, 提供了乐鑫 ESP8266 芯片或安信可 ESP8266 模块对接雅鲁 IOT 平台的教程。

Revision History

Issue	Date	Author	Descriptions
1.00	2019-02-27	Haibin.Song	创建。
1.01	2019-03-04	Haibin.Song	补充平台数据接入说明; 补充安信可模块的数据查询结果。 补充函数注释
1.02	2019-03-07	Haibin.Song	修改接入例程为使用 TLV 开发包开发
1.03	2019-03-13	Haibin.Song	修复一些 bug 数据帧头 0XFF-0XAA 设置 Tlv 数据包示例函数的修改 补充了一些图文

1. 雅鲁物联网平台-数据接入	1
1.1 登录/注册	1
1.2 创建产品	2
1.3 创建设备	4
1.4 MQTT 及数据帧配置信息	6
2. 安信可 ESP8266.....	8
2.1 准备工作	8
2.1.1 硬件.....	8
2.1.2 开发环境	9
2.1.3 安装环境	10
2.2 测试例-连接 WIFI	11
2.2.1 测试固件 SDK 准备.....	11
2.2.1 SDK 文件的导入.....	11
2.2.2 SDK 开发-连接 wifi	15
2.2.1 固件烧写及测试结果.....	15
2.1 ESP8266 接入雅鲁 IOT 平台	20
2.1.1 SDK 准备	20
2.1.2 导入工程	21
2.1.3 修改 MQTT 配置	21
2.1.4 接入开发	23

2.2	雅鲁 IOT 平台查看数据	39
2.2.1	数据查询	40
2.2.2	关于端口的说明	40

1. 雅鲁物联网平台-数据接入

1.1 登录/注册

进入登录界面，点击注册



填写资料

注册账号

* 昵称

平台接入测试账号

* 用户名

platformtest

* 密码

* 确认密码

邮箱

(选填)

手机号

(选填)

☒我已阅读并接受

用户协议

注册

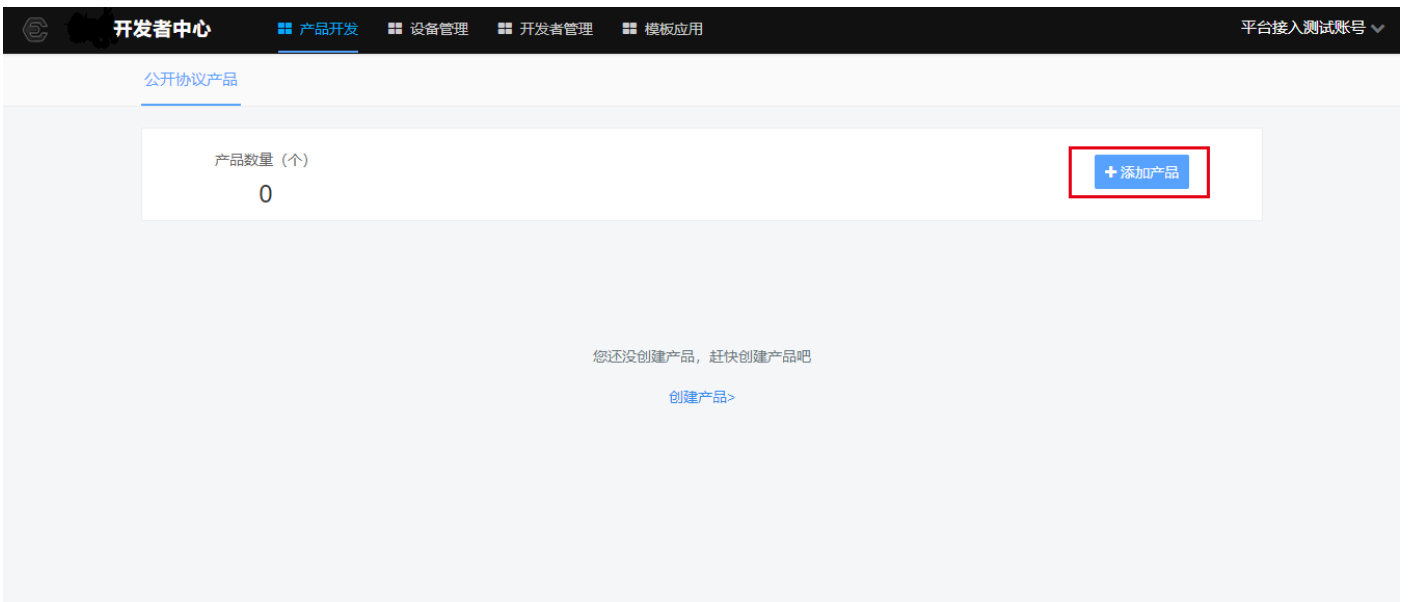
已有账号,去[登录](#) →

使用刚注册的账号密码登录



1.2 创建产品

在产品开发一栏下选择添加产品



根据产品信息选择/填写明细项

* 产品名称

测试产品

* 产品行业

智能泵房

* 产品厂家

安信可

* 行业模板版本

智慧泵房

* 数据解析版本

TlvParser

* 产品类别

电表

技术参数

* 设备接入协议

MQTT

* 联网方式

☐ 有线
 ☐ 移动蜂窝网络
 ☐ NB-IoT
 ☒ WIFI无线

* 操作系统

☐ 雅鲁DthingOS
 ☐ Freertos
 ☐ Mbed
 ☒ 其他
 ☐ Threadx
 ☐ Ucos

* 网络运营商

☒ 中国移动
 ☐ 中国电信
 ☐ 中国联通
 ☐ 其它
 ☐ 中华电信

确定

取消

填写完成后选择确定即完成产品创建，并在主页面下看到新建的产品

公开协议产品			
产品数量 (个)		+ 添加产品	
1			
雅鲁测试产品 编辑 删除 DTU配置	协议 MQTT	设备数 0	创建时间 2019-03-04
共 1 条 上一页 1 下一页			

点击 DTU 配置可查看 MQTT 配置信息



即 MQTT 主题 topic 为: dev001/v1/001/

MQ IP: 47.99.93.72

MQ 端口: 1883

端口协议: MQTT

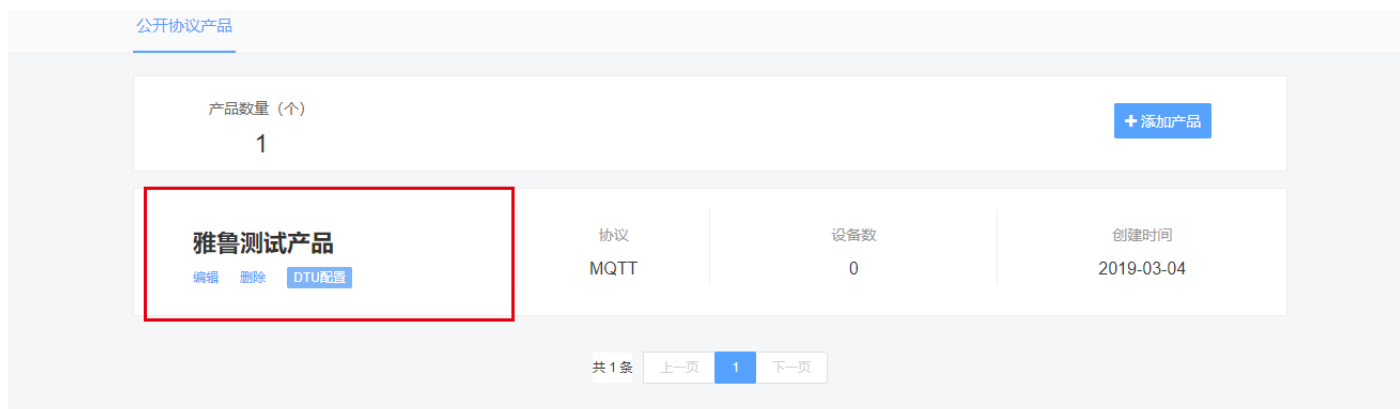
MQ 账号: 68506694937743360

MQ 密码: 5c88b5f9e4b0c8534cd93d24

MQ 密码点击查看 MQ 密码来获取

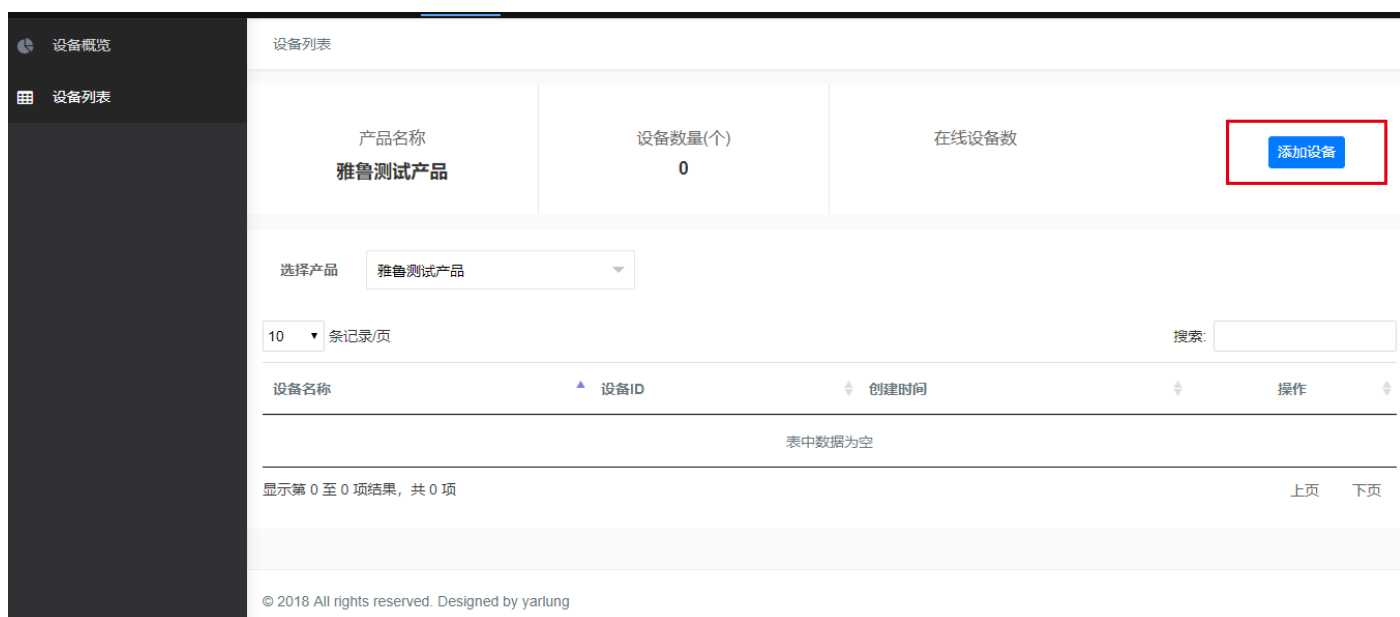
1.3 创建设备

点击产品进入创建设备界面





选择产品列表，添加设备



填写相关信息

注:

设备标识若使用 ip 为标识时前三位保留，最后一位将被端口号替换，如下图填写即为 1.0.2.1，若使用其他标识时可自定义不受影响。

IO 口编号即对应数据帧内的通道号

添加新设备

* 设备名称

测试设备1#

* 设备标识

1.0.2.0

设备序列号

1

设备标签

0

设备描述

设备描述(选填)

设备经纬度

北京市

确定

取消

设备经纬度



经度

116.435045

纬度

39.880459

查看地图点

详细地址

请输入地址

确定

取消

完成设备创建

设备概览

设备列表

设备列表

产品名称

雅鲁测试产品

设备数量(个)

1

在线设备数

XXXXX

添加设备

选择产品

雅鲁测试产品

10

条记录/页

搜索:

设备名称	设备ID	创建时间	操作
测试设备1#	5c7cd31ee4b0578f807ab81b	2019-03-04	<div>详情</div> <div>删除</div>

显示第 1 至 1 项结果, 共 1 项

上页

1

下页

© 2018 All rights reserved. Designed by yarlung

1.4 MQTT 及数据帧配置信息

综合以上 2 章节总结出配置信息。

MQ 配置信息	MQTT 主题	dev001/v1/000/
	MQ IP	47.99.93.72

	MQ 端口	1883
	MQ 账号	DEVPadmin
	MQ 密码	yarlungsoft
数据帧配置信息	设备地址	在平台查看时： 1.0.1.x 被设备端口替换即 1.0.1.1 在配置数据帧时： 1.0.1.x 最后一位默认补 0 即为 1.0.1.0
	设备端口	1
	设备通道	0

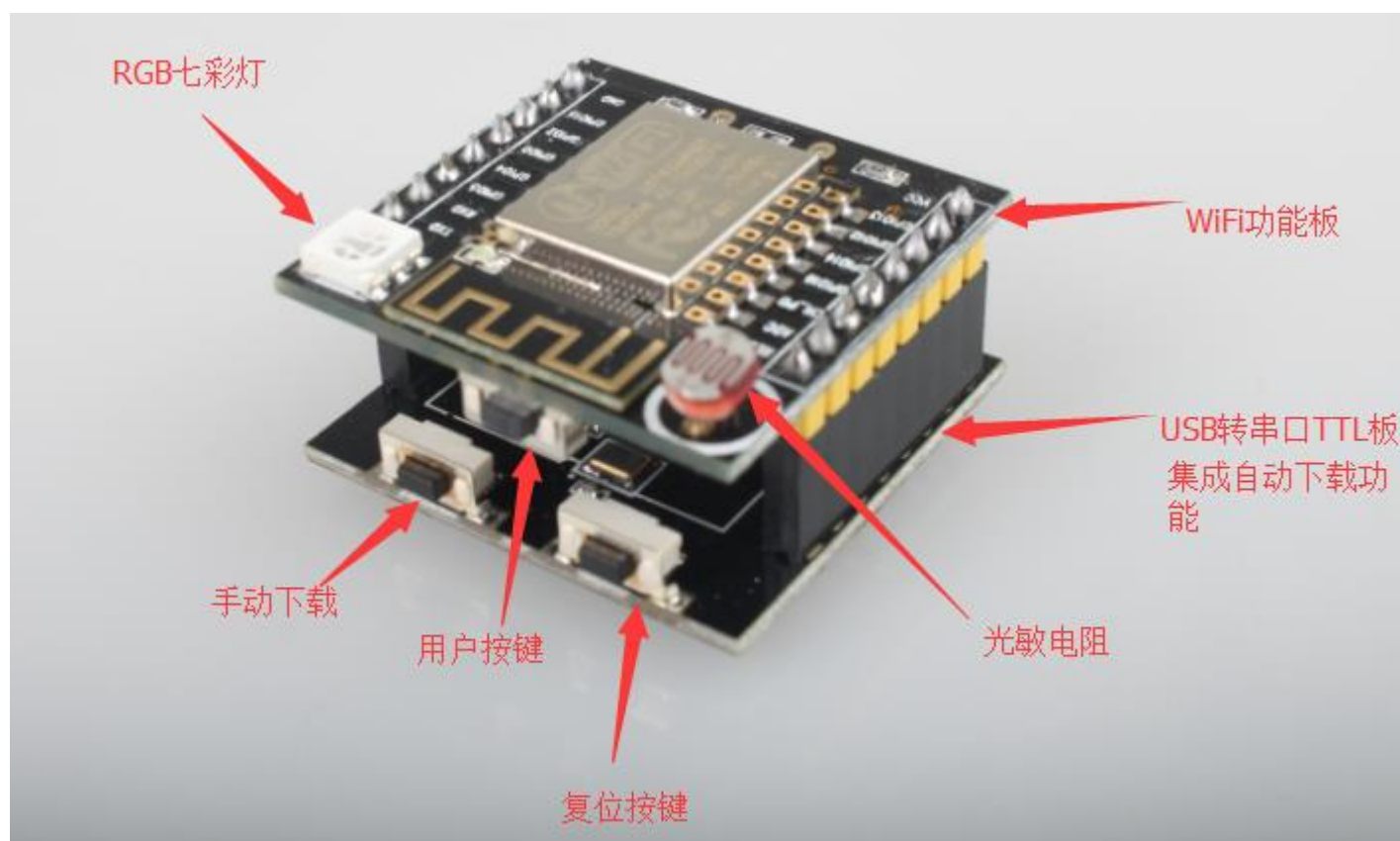
2. 安信可 ESP8266

2.1 准备工作

2.1.1 硬件

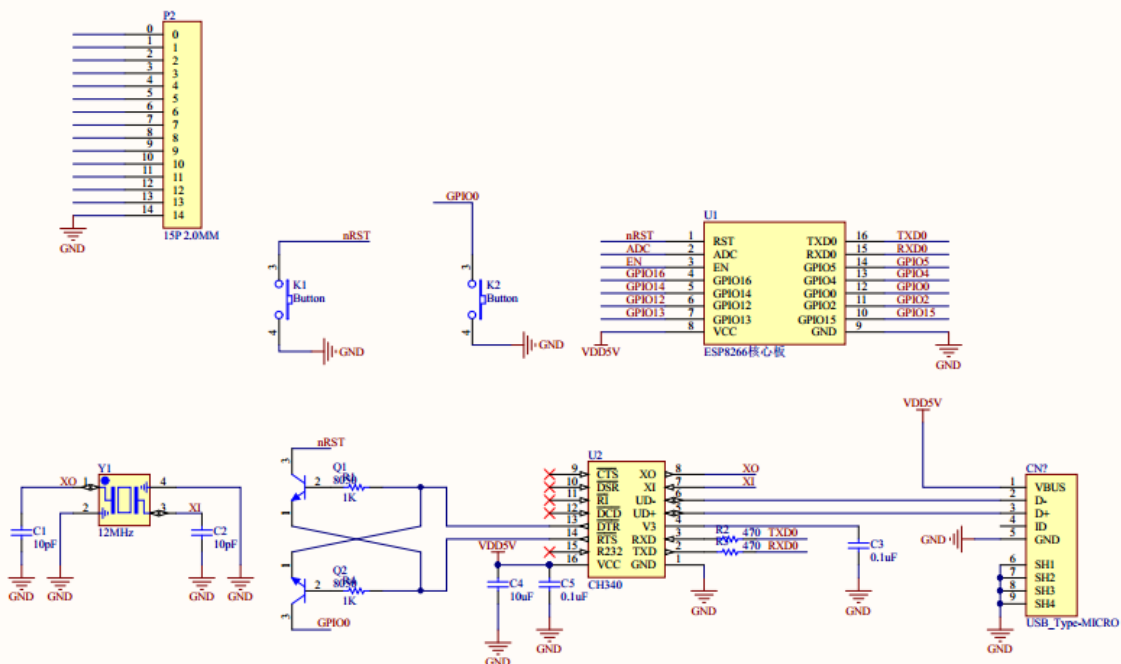
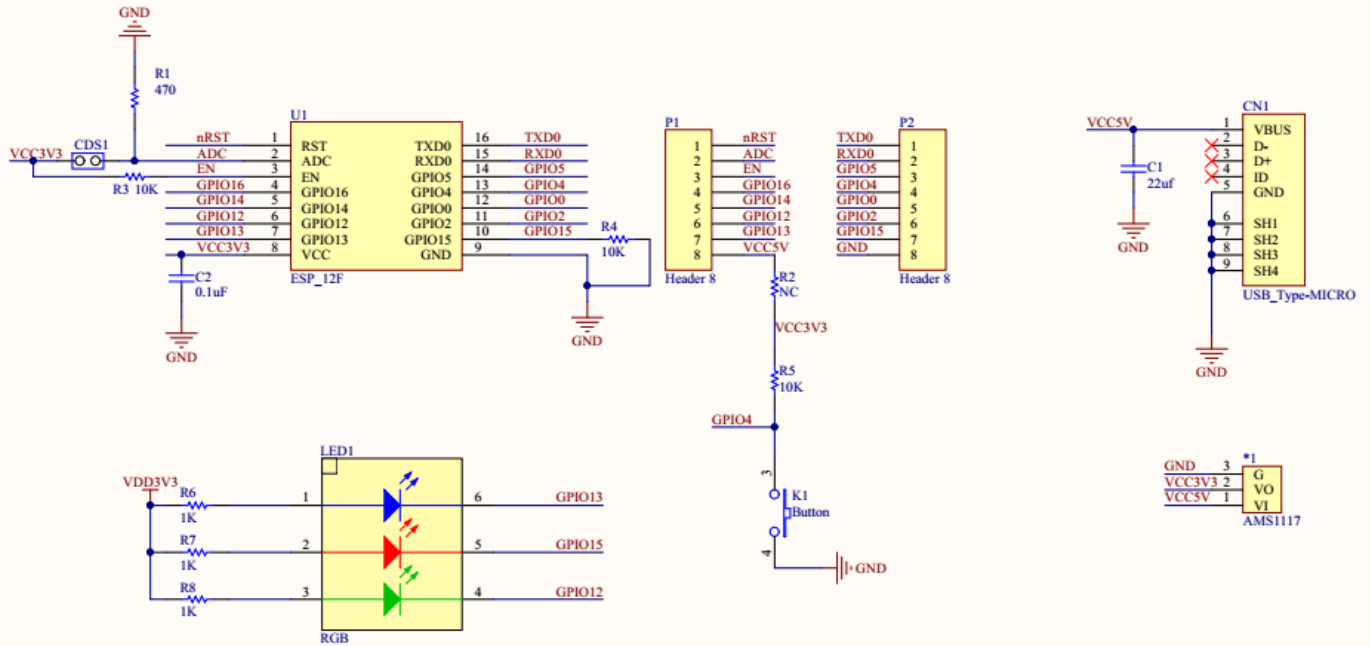
安信可 ESP8266 系列开发板

本人使用的是如下开发板



参考资料: [http://wiki.aithinker.com/esp8266/boards/gizwits/draft?s\[\]=mqtt](http://wiki.aithinker.com/esp8266/boards/gizwits/draft?s[]=mqtt)

电路图:



2.1.2 开发环境

下载安信可开发套件

Cygwin.exe
 Cygwin_Eclipse_IDE.exe
 ESP8266IDE.exe

下载安信可固件烧写软件

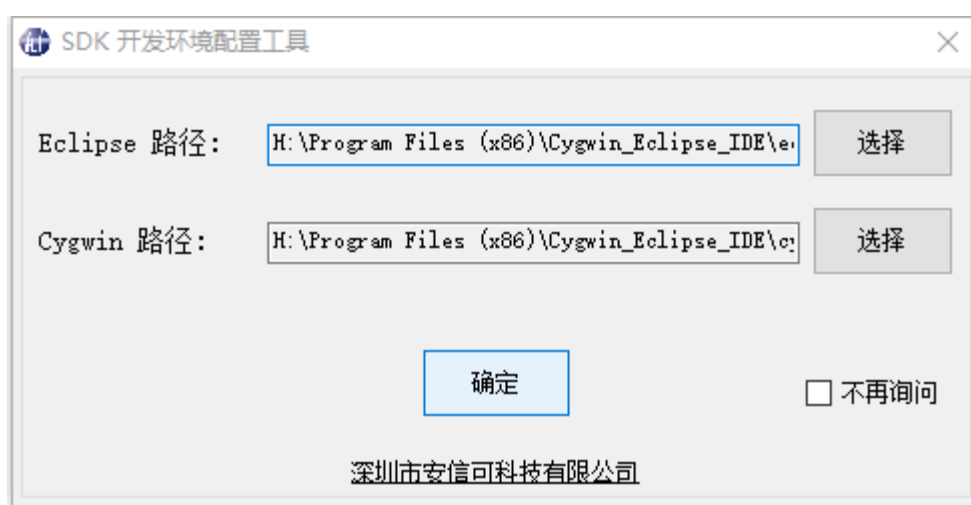
http://wiki.ai-thinker.com/media/esp8266/flash_download_tools_v3.6.4.rar

2.1.3 安装环境

打开 Cygwin.exe 进行自解压

打开 ESP8266IDE.exe 进行安装

请勿在路径中包含中文、空格、中文字符等，建议直接安装在根目录下，如下路径(含空格)可能造成无法打开的情况。建议直接放磁盘根目录下。



打开 Cygwin_Eclipse_IDE.exe



可参考: http://wiki.aithinker.com/ai_ide_install

2.2 测试例-连接 wifi

本测试例提供了用户硬件、开发环境及简单 SDK 二次开发的测试。

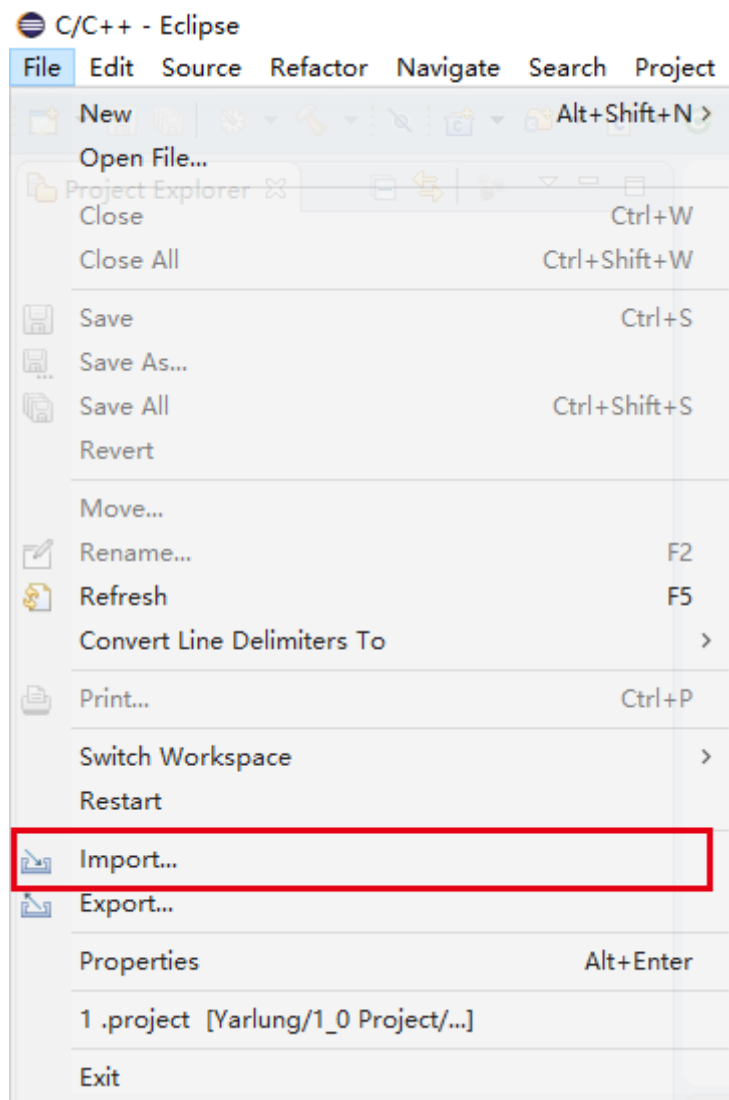
2.2.1 测试固件 SDK 准备

安信可提供的固件

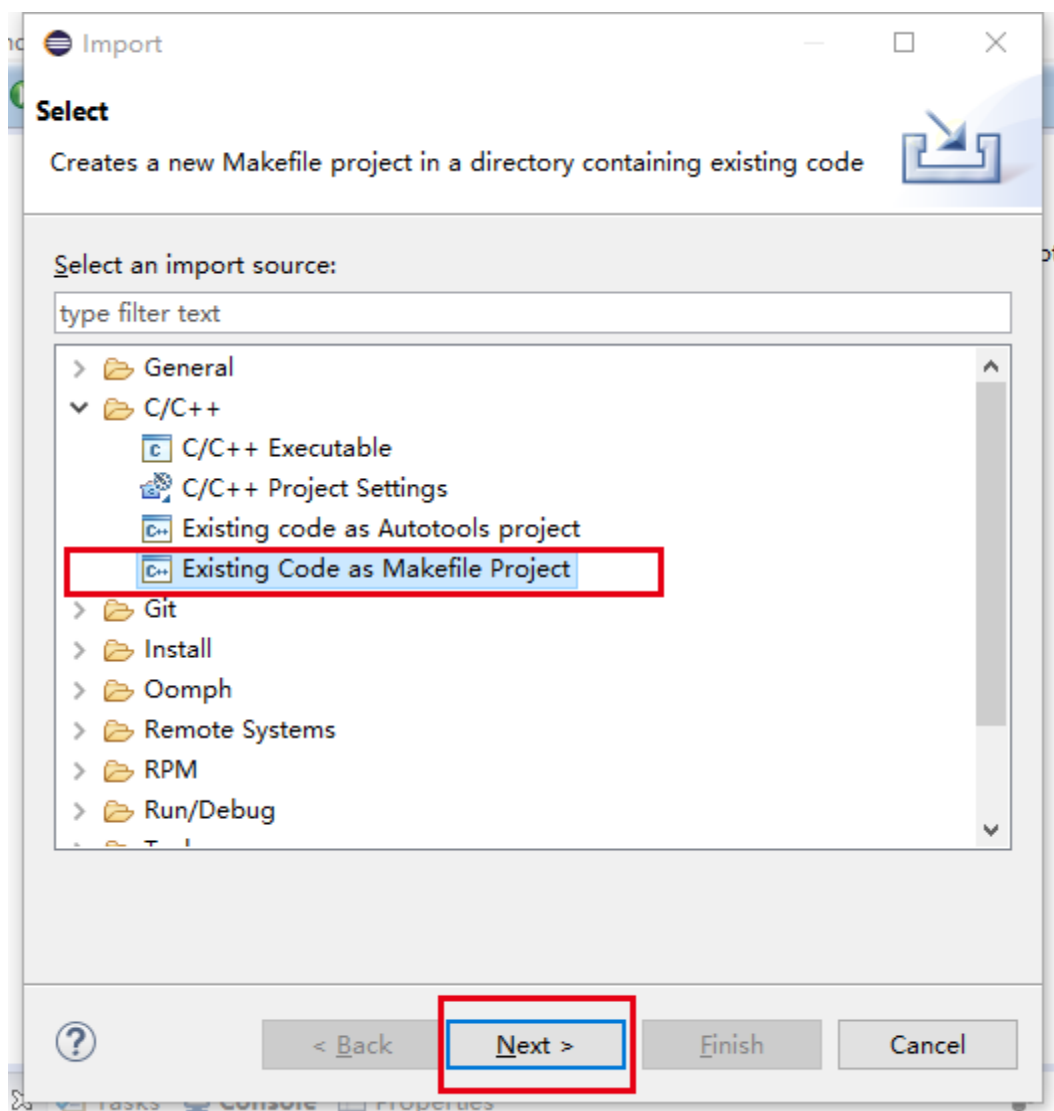
<http://wiki.ai-thinker.com/esp8266/sdk>

2.2.1 SDK 文件的导入

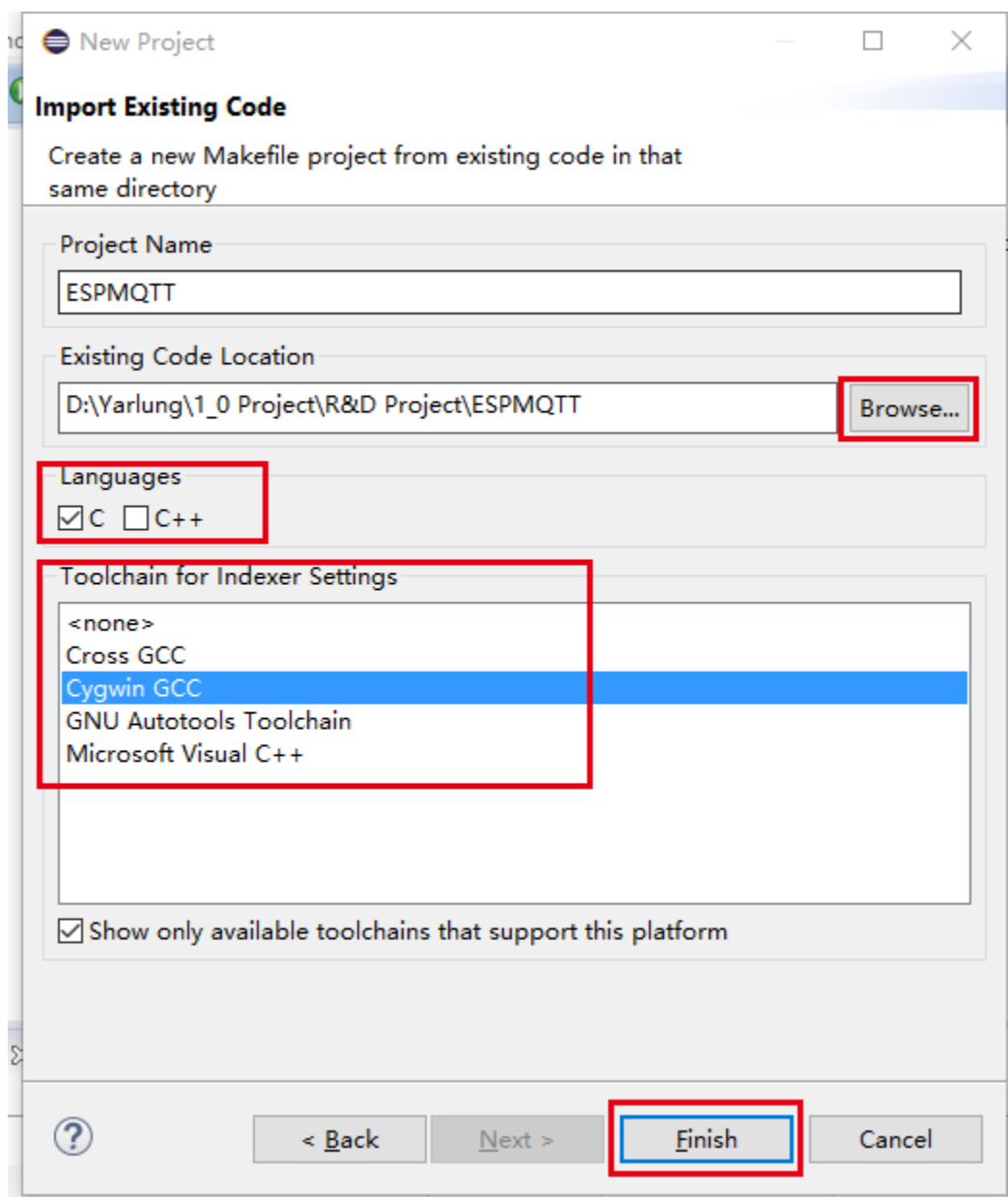
打开 Cygwin_Eclipse_IDE.exe-进入 Eclipse-选择 file- import



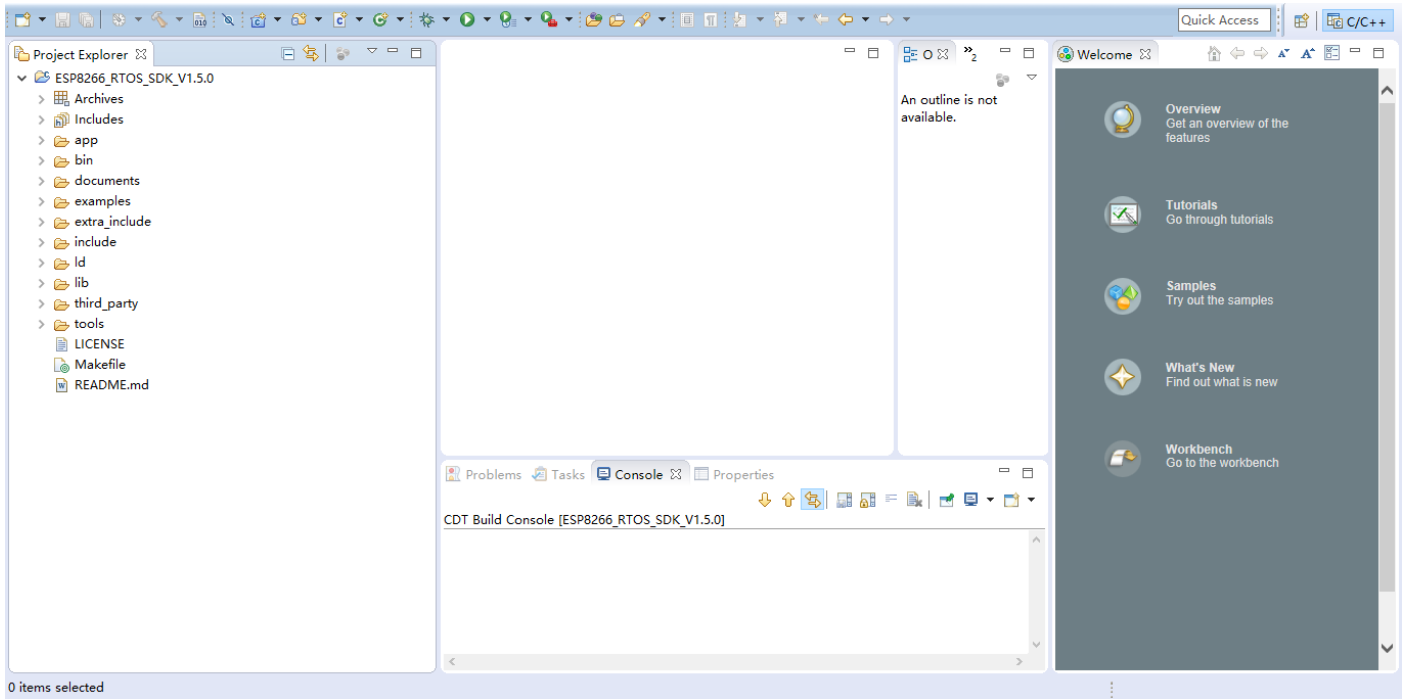
选择 C/C++目录下 Existing Code as Makefile Project



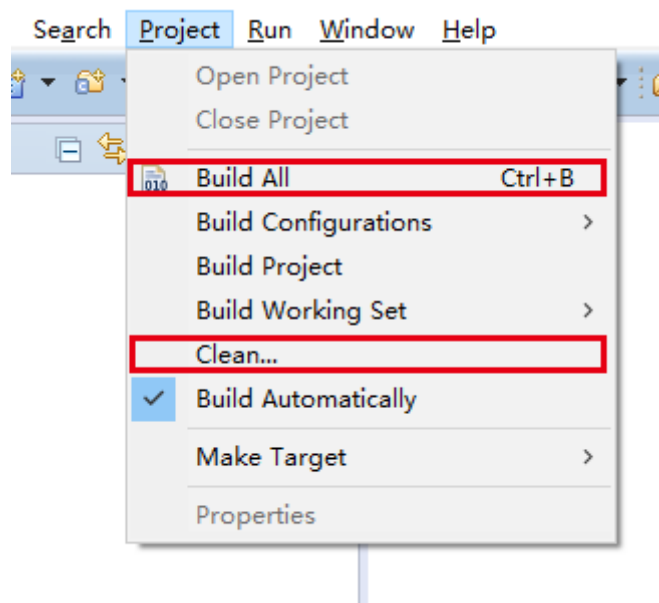
Existing Code Location 处选择 makefile 文件路径



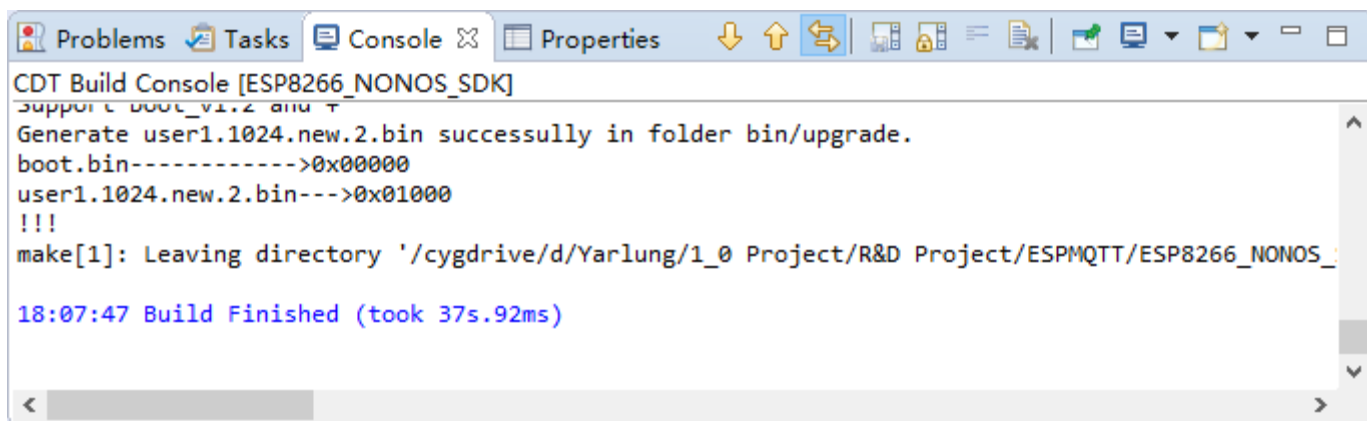
完成导入



清除编译文件并编译



编译完成生成输出文件,在本人使用时使用的一些 sdk 可能存在有的版本 SDK 无法编译通过的情况。因此在使用 SDK 前建议先编译一遍。



```

CDT Build Console [ESP8266_NONOS_SDK]
support boot_v1.2 and
Generate user1.1024.new.2.bin successfully in folder bin/upgrade.
boot.bin----->0x00000
user1.1024.new.2.bin--->0x01000
!!!
make[1]: Leaving directory '/cygdrive/d/Yarlung/1_0 Project/R&D Project/ESPMQTT/ESP8266_NONOS_'

18:07:47 Build Finished (took 37s.92ms)

```

2.2.2 SDK 开发-连接 wifi

在 app\user\user_main.c 下编写连接 wifi 函数

```

struct station_config station_cfg;
uint8 ssid[] = "whatever";
uint8 password[] = "admin123";

void ICACHE_FLASH_ATTR
user_wifi_station_connect(void)
{
    wifi_set_opmode(STATION_MODE);           //设置为 STATION MODE
    os_strcpy(station_cfg.ssid, ssid);        //ssid 名称
    os_strcpy(station_cfg.password, password); //密码
    os_printf("wifi ssid:%s\n password:%s\n", station_cfg.ssid, station_cfg.password); //打印 wifi 信息
    wifi_station_set_config(&station_cfg);    //设置 WIFI 帐号和密码
}

```

在主函数中调用连接 wifi 函数

```

void user_init(void)
{
    os_printf("SDK version:%s\n", system_get_sdk_version());
    user_wifi_station_connect();
}

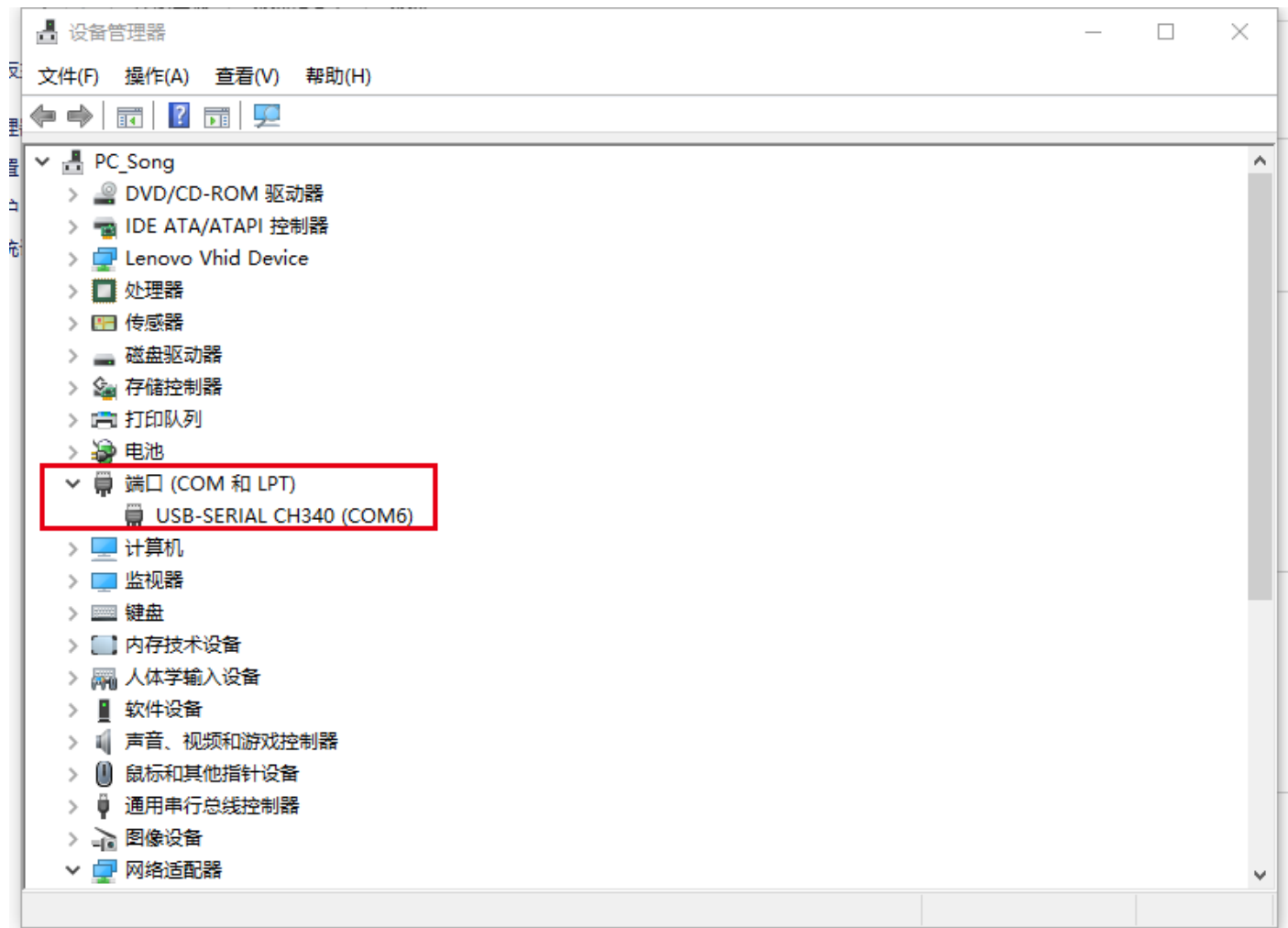
```

保存、编译、烧写固件后重启开发板

提示 os_printf 未定义时可改成 printf 并包含头文件 string。

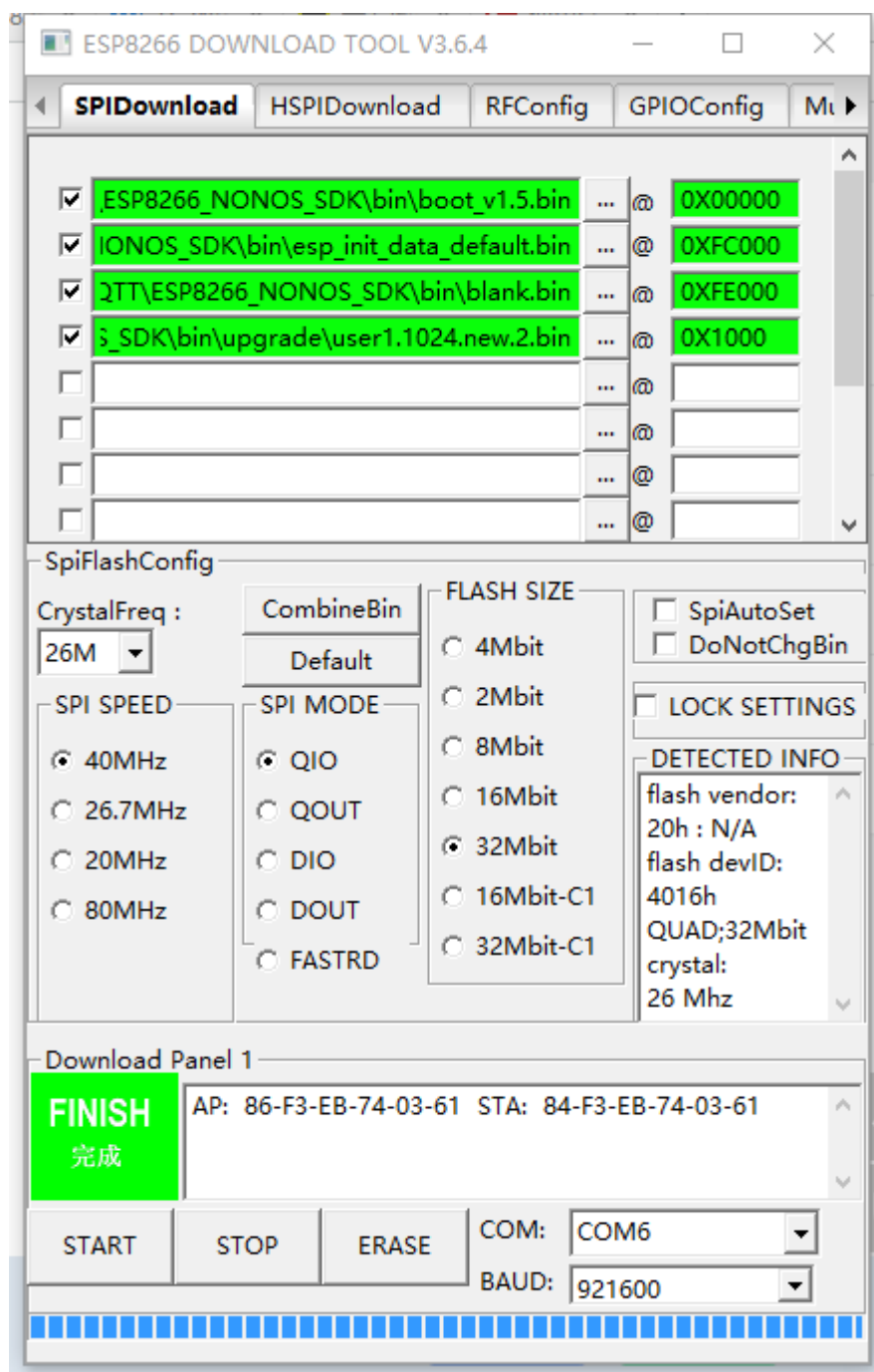
2.2.1 固件烧写及测试结果

找到开发板连接的 COM 口

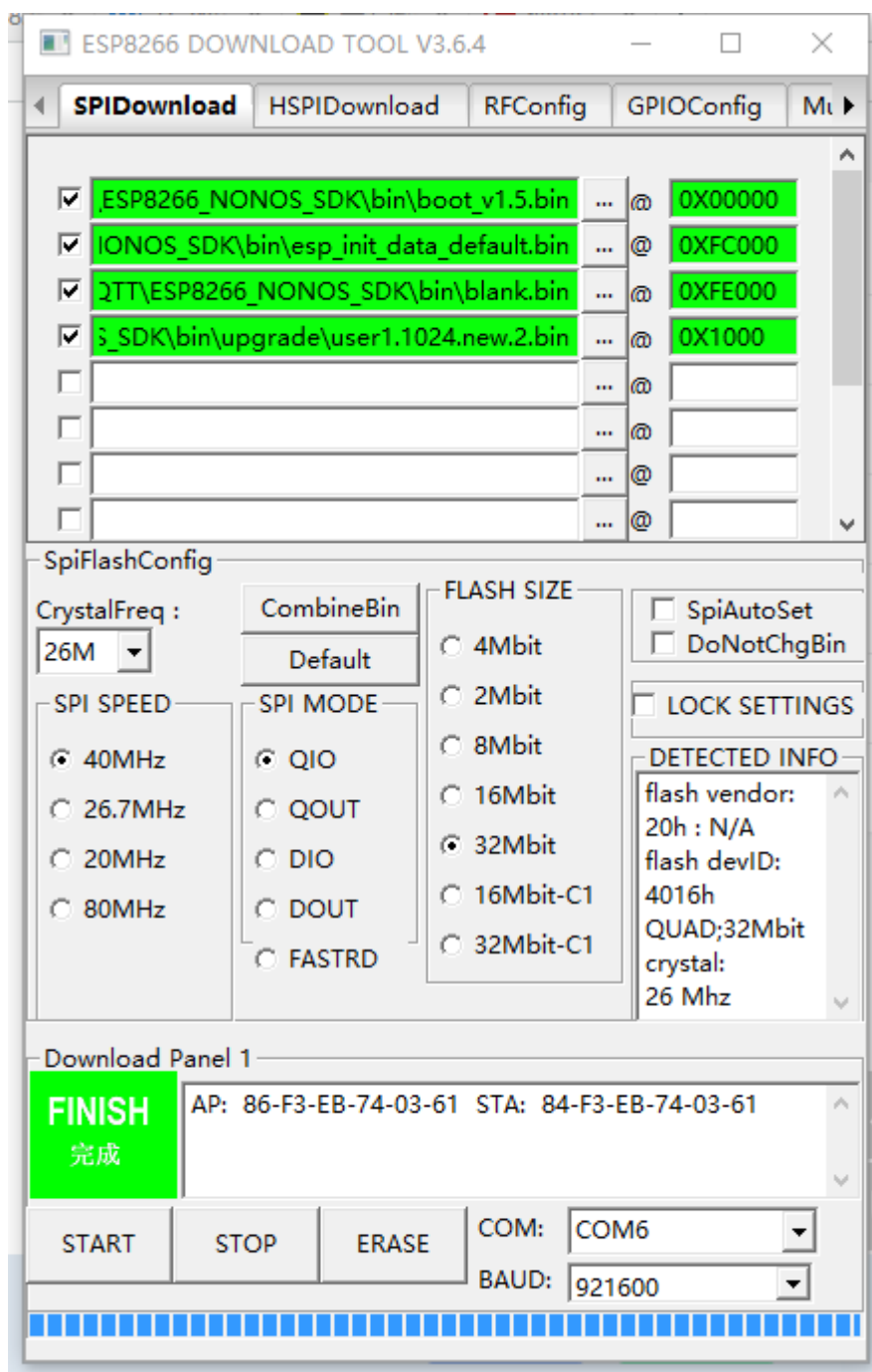


打开烧写软件 ESPFlashDownloadTool_v3.6.4.exe 并定位 bin 路径及烧写扇区

注：不同的硬件及 SDK 烧写的扇区可能存在不同地址的情况请根据实际情况并根据乐鑫/安信的文档进行填写



等待烧写完成

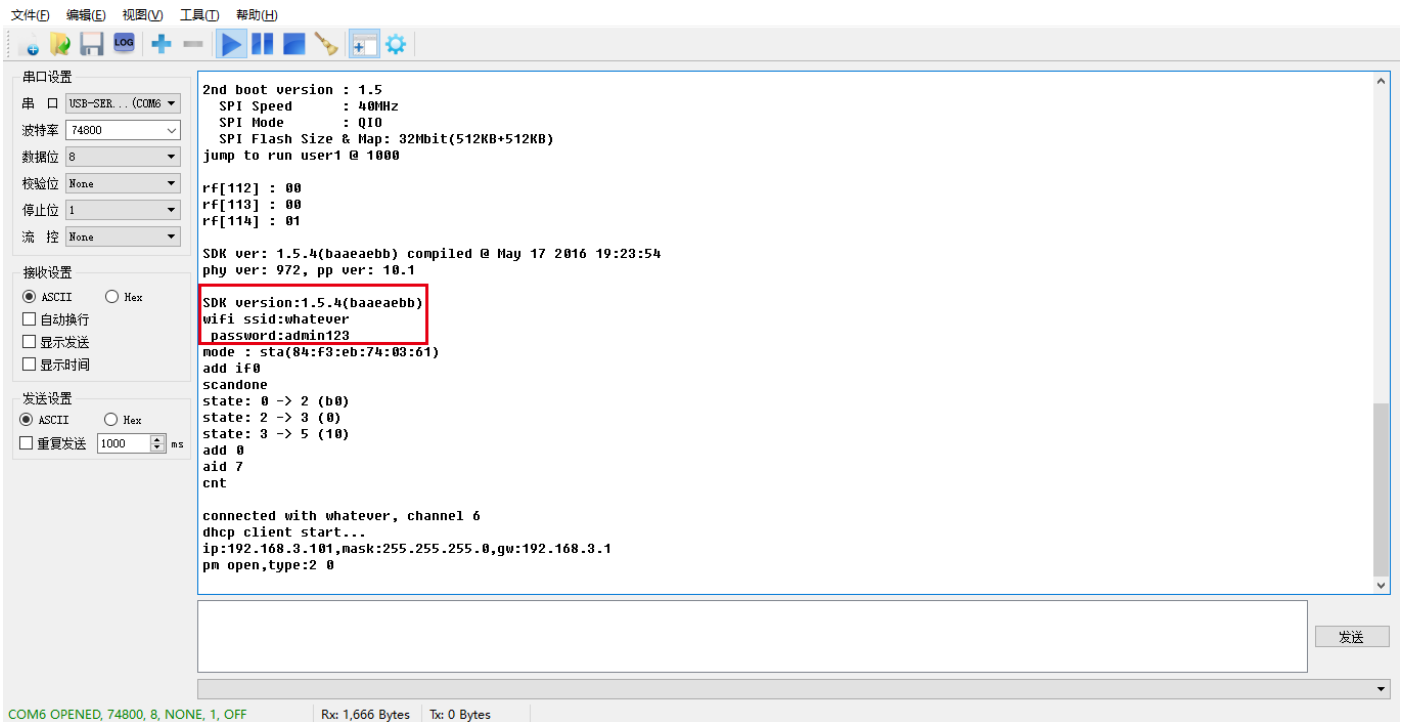


复位开发板使其复位开始工作

在路由器管理面板内可看到模块已连接 wifi



串口工具查看打印信息



注：ESP8266 串口波特率默认为 74800。参照下表：

表 6-2. 串口调试配置

配置项	配置说明
协议类型	串口
端口号	根据实际连入的设备所在的端口号设置。
波特率	设备运行时的波特率，与设备晶振有关。 <ul style="list-style-type: none">• 69120 （晶振 24 MHz）• 74880 （晶振 26 MHz）• 115200 （晶振 40 MHz） ESP8266 AT 示例默认支持 115200 波特率，用户不可修改。 ESP8266 IOT_Demo 及其他示例默认为 74880 波特率，用户可以修改。
数据位	8
校验	无
流控	无

2.1 ESP8266 接入雅鲁 IOT 平台

本章提供了 ESP8266 通过 MQTT 协议接入雅鲁 IOT 平台的案例。

实现过程为设备根据平台提供的配置信息连接 MQ 后，MQTT_Publish 一串 TLV 数据，并在发布成功的回调函数中调用 MQTT_Publish 从而实现不间断发布（实际项目中请慎重考虑这种方式，详细请参考安信可/乐鑫 SDK 开发文档）。平台将根据用户创建的产品及设备信息来订阅这些数据。实现设备发送的数据在平台进行查询。

2.1.1 SDK 准备

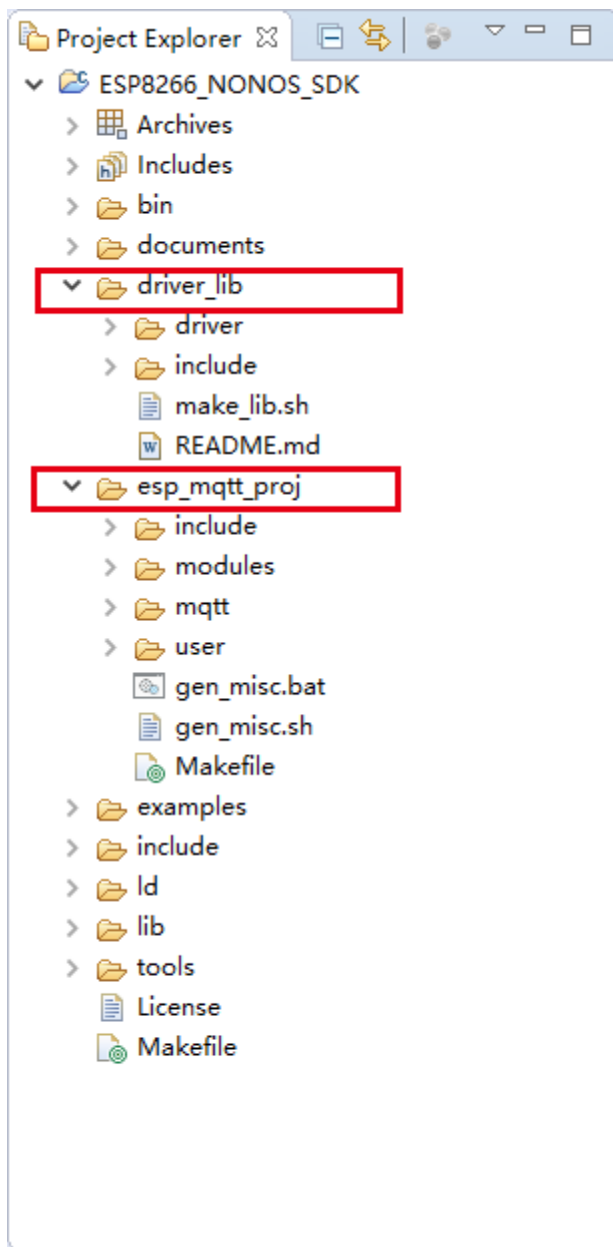
使用乐鑫提供的 ESP8266 NONOS SDK V2.0.0 20160810 版本 SDK，这个版本增加了 MQTT 的示例下载地址：

https://www.espressif.com/zh-hans/support/download/sdks-demos?keys=&field_type_tid%5B%5D=14

2.1.2 导入工程

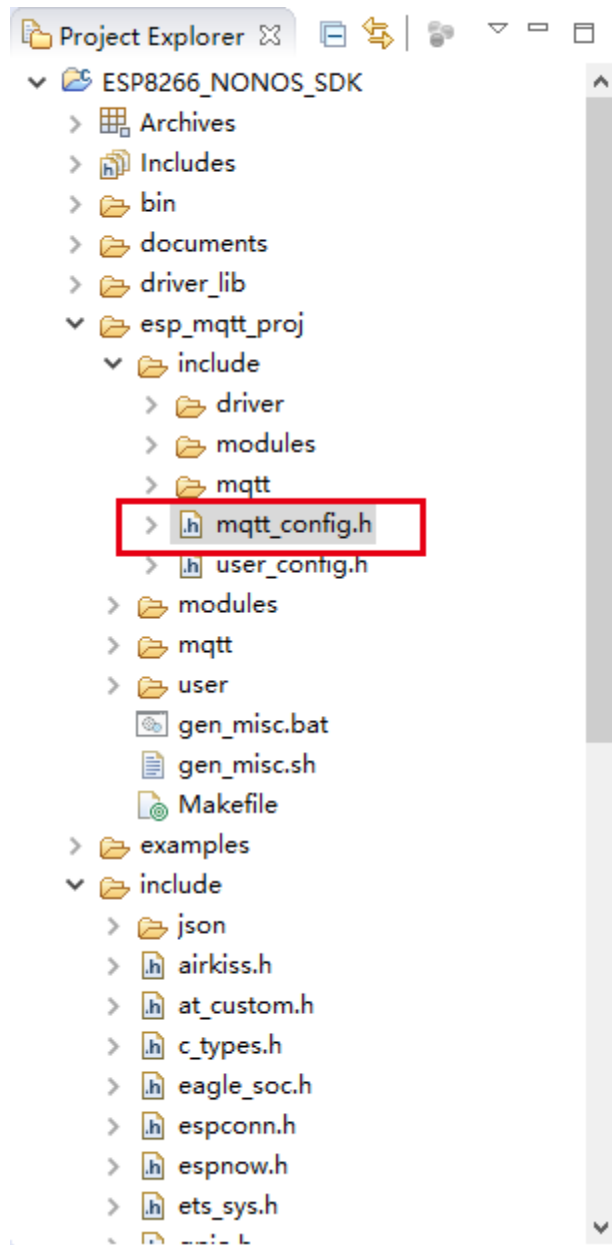
打开 Cygwin_Eclipse_IDE.exe-进入 Eclipse。

导入下载好的 SDK，并删除 driver_lib 下的 makefile 文件避免编译出错，复制示例 examples 目录下的 esp_mqtt_proj 到工程主目录，clean 工程并编译。如不明白的请参照 2.2 章节测试例内的操作。



2.1.3 修改 MQTT 配置

在 mqtt_config.h 内修改 MQTT 配置信息。



根据 MQTT 服务器配置，MQTT 版本。这里的配置信息再 1.4 章节有提供。

配置 wifi 账户密码。

其余部分请根据实际情况配置。

请注意 **CFG HOLDER** 这个宏定义在每次配置信息修改时需要修改成其他数值，否则不会存储配置。

```
#ifndef __MQTT_CONFIG_H__
#define __MQTT_CONFIG_H__

#define CFG HOLDER 0x00FF55A4 /* Change this value to load default configurations */
#define CFG_LOCATION 0x79/* Please don't change or if you know what you doing */
// #define MQTT_SSL_ENABLE

/*DEFAULT CONFIGURATIONS*/

#define MQTT_HOST "47.99.93.72" //MQ 服务器地址
#define MQTT_PORT 1883//MQ 服务器端口
#define MQTT_BUF_SIZE 1024
```

```
#define MQTT_KEEPALIVE      120  /*second*/

#define MQTT_CLIENT_ID      "DVES_%08X"
#define MQTT_USER           "DEVAdmin"//平台 MQ 账户
#define MQTT_PASS           "yarlungsoft"//平台 MQ 密码

#define MQTT_TOPIC " dev001/v1/001/devaxk" //MQ 主题前缀与平台对应

#define STA_SSID "whatever"//wifi 账号 SSID
#define STA_PASS "admin123"//wifi 密码
#define STA_TYPE AUTH_WPA2_PSK

#define MQTT_RECONNECT_TIMEOUT  5  /*second*/

#define DEFAULT_SECURITY  0
#define QUEUE_BUFFER_SIZE      2048

//#define PROTOCOL_NAMEv31  /*MQTT version 3.1 compatible with Mosquitto v0.15*/
#define PROTOCOL_NAMEv311 //MQ 版本          /*MQTT          version          3.11          compatible          with
https://eclipse.org/paho/clients/testing/*/

#endif // __MQTT_CONFIG_H__
```

2.1.4 接入开发

可直接下载雅鲁 TLV 协议包文件：YLTlvPac.c 及 YLTlvPac.h 可快速使用 TLV 协议配置

2.1.4.1 YLTlvPac.h

```
/*
 * YLTlvPac.h
 *
 * Created on: 2019 年 3 月 4 日
 * Author: hb
 * version: 1.0
 */

#ifndef YLTLPAC_H_
#define YLTLPAC_H_

#include "c_types.h"
#include "stdio.h" //uint8_t 等类型定义文件

#define BIGEDIAN //根据芯片、机器自行定义
//TLV 特征帧定义
#define TLVHEAD1 (0xAA)//帧头 1
#define TLVHEAD2 (0xFE)//帧头 2
#define TLVTAIL1 (0xDD)//帧尾 1
#define TLVTAIL2 (0xEE)//帧尾 2
#define TLVVERSION (2)//规范版本号

//TLVADDR 设置标识为 ip:1.0.1.0,使用 ip 作为标识时最后一位为 0 即可
#define TLVADDR (0x01000200)
```

//TLV 功能码, 固定为 1

#define TLVFUN (1)

#define TLVBUFFSIZE (200)//MQTT 发布 buff 大小

/******TLV 帧行业定义******/

#define TLVBID_ELECTRIC (0x0100)//电力通用

#define TLVBID_WATER (0x0200)//水务通用

#define TLVBID_WATER_PUMP (0x0201)//水务泵房

#define TLVBID_WATER_WM (0x0202)//水务水表

#define TLVBID_CITY (0x0300)//城市

#define TLVBID_ENV (0x0400)//环境

#define TLVBID_FACTORY (0x0500)//工厂

/******TAG 定义******/

//数据类型高位

//**#define** TLVTAG_TYPEH_BOOL (0x00)//未支持

//**#define** TLVTAG_TYPEH_SCHAR (0x10)//未支持

#define TLVTAG_TYPEH_UCHAR (0x20)

#define TLVTAG_TYPEH_SSHORT (0x30)

#define TLVTAG_TYPEH_USHORT (0x40)

#define TLVTAG_TYPEH_SINT (0x50)

#define TLVTAG_TYPEH_UINT (0x60)

//**#define** TLVTAG_TYPEH_SLONG (0x70)//未支持

//**#define** TLVTAG_TYPEH_ULONG (0x80)//未支持

//**#define** TLVTAG_TYPEH_FLOAT (0x90)//未支持

//**#define** TLVTAG_TYPEH_DOUBLE (0xA0)//未支持

//数据类型低位

#define TLVTAG_TYPEL_NFD0(0x00)

#define TLVTAG_TYPEL_NFD1(0x01)

#define TLVTAG_TYPEL_NFD2(0x02)

#define TLVTAG_TYPEL_NFD3(0x03)

#define TLVTAG_TYPEL_NFD4(0x04)

#define TLVTAG_TYPEL_NFD5(0x05)

#define TLVTAG_TYPEL_NFD6(0x06)

#define TLVTAG_TYPEL_NFD7(0x07)

/******水务行业******/

//泵类

#define TLVTAG_WATER_PUMPTEMP1 (21)//泵表面温度(度)

#define TLVTYPE_WATER_PUMPTEMP1 (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD2)//数据类型 uchar.2

#define TLVTAG_WATER_PUMPVABRATION1 (22)//泵表面震动(um)

#define TLVTYPE_WATER_PUMPVABRATION1 (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD2)//数据类型 uchar.2

//蓄水池

#define TLVTAG_WATER_PH (23)//PH 值(PH)

#define TLVTYPE_WATER_PH (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD2)//数据类型 uchar.2

#define TLVTAG_WATER_TURBIDI (24)//浊度(NTU)

#define TLVTYPE_WATER_TURBIDI (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD3)//数据类型 uchar.3

#define TLVTAG_WATER_CHLORINE (25)//余氯(mg/L)

#define TLVTYPE_WATER_CHLORINE (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD3)//数据类型 uchar.3

#define TLVTAG_WATER_WATERLEVEL (26)//水位高度(m)

#define TLVTYPE_WATER_WATERLEVEL (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD2)//数据类型 uchar.2

#define TLVTAG_WATER_WATERPRESSURE (27)//水压力(Mpa)

#define TLVTYPE_WATER_WATERPRESSURE (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD3)//数据类型 uchar.3

#define TLVTAG_WATER_WATERTEMP (28)//水温度(Mpa)

#define TLVTYPE_WATER_WATERTEMP (TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD2)//数据类型 uchar.2

#define TLVTAG_WATER_WATERSPEED (29)//水流速(m3/h)

#define TLVTYPE_WATER_WATERSPEED (TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD3)//数据类型 uint.3

#define TLVTAG_WATER_WATERFLOW (30)//水流量(m3/h)

#define TLVTYPE_WATER_WATERFLOW //泵房环境	(TLVTAG_TYPEH_UINT TLVTAG_TYPEL_NFD2)//数据类型 <u>uint.3</u>
#define TLVTAG_WATER_ENVTEMP	(31)//环境温度(度)
#define TLVTYPE_WATER_ENVTEMP sshort.2	(TLVTAG_TYPEH_SSHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_ENVHUM	(32)//环境湿度(%)
#define TLVTYPE_WATER_ENVHUM ushort.2	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_ENVNOISE	(33)//环境噪音(db)
#define TLVTYPE_WATER_ENVNOISE ushort.2	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_ENVFIREWORK	(34)//环境烟火 0-正常 1-报警
#define TLVTYPE_WATER_ENVFIREWORK	(TLVTAG_TYPEH_UCHAR TLVTAG_TYPEL_NFD0)//数据类型 <u>uchar.0</u>
#define TLVTAG_WATER_ENVFLOOD	(35)//环境积水 0-正常 1-报警
#define TLVTYPE_WATER_ENVFLOOD	(TLVTAG_TYPEH_UCHAR TLVTAG_TYPEL_NFD0)//数据类型 <u>uchar.0</u>
#define TLVTAG_WATER_ENVACCESSSTA	(36)//门禁状态 0-关 1-开
#define TLVTYPE_WATER_ENVACCESSSTA //电表	(TLVTAG_TYPEH_UCHAR TLVTAG_TYPEL_NFD0)//数据类型 <u>uchar.0</u>
#define TLVTAG_WATER_EMVA	(37)//电压 VA (V)
#define TLVTYPE_WATER_EMVA ushort.2	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_EMVB	(38)//电压 VB (V)
#define TLVTYPE_WATER_EMVB ushort.2	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_EMVC	(39)//电压 VC (V)
#define TLVTYPE_WATER_EMVC ushort.2	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)// 数 据 类 型
#define TLVTAG_WATER_EMIA	(40)//电流 IA (A)
#define TLVTYPE_WATER_EMIA ushort.3	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD3)// 数 据 类 型
#define TLVTAG_WATER_EMIB	(41)//电流 IB (A)
#define TLVTYPE_WATER_EMIB ushort.3	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD3)// 数 据 类 型
#define TLVTAG_WATER_EMIC	(42)//电流 IC (A)
#define TLVTYPE_WATER_EMIC ushort.3	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD3)// 数 据 类 型
#define TLVTAG_WATER_EMCT	(43)//CT
#define TLVTYPE_WATER_EMCT ushort.0	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD0)// 数 据 类 型
#define TLVTAG_WATER_EMPT	(44)//PT
#define TLVTYPE_WATER_EMPT ushort.0	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD0)// 数 据 类 型
#define TLVTAG_WATER_EMFREQ	(45)//频率 (HZ)
#define TLVTYPE_WATER_EMFREQ	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD2)//数据类型 <u>sint.2</u>
#define TLVTAG_WATER_EMACTIVEPOW	(46)//总有功功率 (KW)
#define TLVTYPE_WATER_EMACTIVEPOW	(TLVTAG_TYPEH_SINT TLVTAG_TYPEL_NFD2)//数据类型 <u>sint.2</u>
#define TLVTAG_WATER_EMREACTIVEPOW	(47)//总无功功率 (KV.A)
#define TLVTYPE_WATER_EMREACTIVEPOW	(TLVTAG_TYPEH_SINT TLVTAG_TYPEL_NFD2)//数据类型 <u>sint.2</u>
#define TLVTAG_WATER_EMPOWERFACTOR	(48)//功率因素 ()
#define TLVTYPE_WATER_EMPOWERFACTOR sshort.3	(TLVTAG_TYPEH_SSHORT TLVTAG_TYPEL_NFD3)// 数 据 类 型
#define TLVTAG_WATER_EMWATTHOUR	(49)//有功电度 (KWH)
#define TLVTYPE_WATER_EMWATTHOUR	(TLVTAG_TYPEH_UINT TLVTAG_TYPEL_NFD2)//数据类型 <u>uint.2</u>
#define TLVTAG_WATER_EMVARHOUR	(50)//无功电度 (KV.AH)
#define TLVTYPE_WATER_EMVARHOUR //水表基础	(TLVTAG_TYPEH_UINT TLVTAG_TYPEL_NFD2)//数据类型 <u>uint.2</u>
#define TLVTAG_WATER_WMFLOW	(51)//水表流量 (m3)
#define TLVTYPE_WATER_WMFLOW	(TLVTAG_TYPEH_UINT TLVTAG_TYPEL_NFD3)//数据类型 <u>uint.3</u>
#define TLVTAG_WATER_WMPRESSURE	(52)//水压 (Mpa)
#define TLVTYPE_WATER_WMPRESSURE ushort.3	(TLVTAG_TYPEH_USHORT TLVTAG_TYPEL_NFD3)// 数 据 类 型

//泵情况-PLC 采集

```
#define TLVTAG_WATER_PUMPRUNSTA
#define TLVTYPE_WATER_PUMPRUNSTA
#define TLVTAG_WATER_PUMPVOLTAGE
#define TLVTYPE_WATER_PUMPVOLTAGE
ushort.2
#define TLVTAG_WATER_PUMPCURRENT
#define TLVTYPE_WATER_PUMPCURRENT
ushort.3
#define TLVTAG_WATER_PUMPRUNPOWER
#define TLVTYPE_WATER_PUMPRUNPOWER
ushort.2
#define TLVTAG_WATER_PUMPFREQ
#define TLVTYPE_WATER_PUMPFREQ
ushort.2
#define TLVTAG_WATER_PUMPTEMP
#define TLVTYPE_WATER_PUMPTEMP
ushort.2
#define TLVTAG_WATER_PUMPVIBRATION
#define TLVTYPE_WATER_PUMPVIBRATION
ushort.2
#define TLVTAG_WATER_PUMPRUNTIME
#define TLVTYPE_WATER_PUMPRUNTIME
#define TLVTAG_WATER_PUMPWATERSPEED
#define TLVTYPE_WATER_PUMPWATERSPEED
#define TLVTAG_WATER_PUMPINFLOW
#define TLVTYPE_WATER_PUMPINFLOW
#define TLVTAG_WATER_PUMPINPRESSURE
#define TLVTYPE_WATER_PUMPINPRESSURE
ushort.3
#define TLVTAG_WATER_PUMPOUTFLOW
#define TLVTYPE_WATER_PUMPOUTFLOW
#define TLVTAG_WATER_PUMPOUTPRESSURE
#define TLVTYPE_WATER_PUMPOUTPRESSURE
ushort.3
```

/*****电行业*****/

//电表

```
#define TLVTAG_ELECTRIC_EMVA
#define TLVTYPE_ELECTRIC_EMVA
ushort.2
#define TLVTAG_ELECTRIC_EMVB
#define TLVTYPE_ELECTRIC_EMVB
ushort.2
#define TLVTAG_ELECTRIC_EMVC
#define TLVTYPE_ELECTRIC_EMVC
ushort.2
#define TLVTAG_ELECTRIC_EMIA
#define TLVTYPE_ELECTRIC_EMIA
ushort.3
#define TLVTAG_ELECTRIC_EMIB
#define TLVTYPE_ELECTRIC_EMIB
ushort.3
#define TLVTAG_ELECTRIC_EMIC
#define TLVTYPE_ELECTRIC_EMIC
ushort.3
#define TLVTAG_ELECTRIC_EMCT
#define TLVTYPE_ELECTRIC_EMCT
ushort.0
#define TLVTAG_ELECTRIC_EMPT
#define TLVTYPE_ELECTRIC_EMPT
ushort.0
```

```
(53)//泵运行情况 ()
(TLVTAG_TYPEH_UCHAR|TLVTAG_TYPEL_NFD0)//数据类型 UCHAR.0
(54)//泵电压 (V)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(55)//泵电流 (A)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(56)//泵运行功率 ()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(57)//泵运行频率 ()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(58)//泵工作温度 ()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(59)//泵振幅 ()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(60)//泵运行时间 ()
(TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD2)//数据类型 ushort.2
(61)//泵水流速 ()
(TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD3)//数据类型 ushort.3
(62)//泵进口累积流量 ()
(TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD3)//数据类型 ushort.3
(63)//泵进口压力()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(64)//泵出口累积流量 ()
(TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD3)//数据类型 ushort.3
(65)//泵出口压力()
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(37)//电压 VA (V)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(38)//电压 VB (V)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(39)//电压 VC (V)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)// 数 据 类 型
(40)//电流 IA (A)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(41)//电流 IB (A)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(42)//电流 IC (A)
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD3)// 数 据 类 型
(43)//CT
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD0)// 数 据 类 型
(44)//PT
(TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD0)// 数 据 类 型
```

```

#define TLVTAG_ELECTRIC_EMFREQ (45)//频率 (HZ)
#define TLVTYPE_ELECTRIC_EMFREQ (TLVTAG_TYPEH_USHORT|TLVTAG_TYPEL_NFD2)//数据类型 sint.2
#define TLVTAG_ELECTRIC_EMACTIVEPOW (46)//总有功功率 (KW)
#define TLVTYPE_ELECTRIC_EMACTIVEPOW (TLVTAG_TYPEH_SINT|TLVTAG_TYPEL_NFD2)//数据类型 sint.2
#define TLVTAG_ELECTRIC_EMREACTIVEPOW (47)//总无功功率 (KV.A)
#define TLVTYPE_ELECTRIC_EMREACTIVEPOW (TLVTAG_TYPEH_SINT|TLVTAG_TYPEL_NFD2)//数据类型 sint.2
#define TLVTAG_ELECTRIC_EMPOWERFACTOR (48)//功率因素 ()
#define TLVTYPE_ELECTRIC_EMPOWERFACTOR (TLVTAG_TYPEH_SSHORT|TLVTAG_TYPEL_NFD3)//数据类型 sshort.3
#define TLVTAG_ELECTRIC_EMWATTHOUR (49)//有功电度 (KWH)
#define TLVTYPE_ELECTRIC_EMWATTHOUR (TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD2)//数据类型 uint.2
#define TLVTAG_ELECTRIC_EMVARHOUR (50)//无功电度 (KV.AH)
#define TLVTYPE_ELECTRIC_EMVARHOUR (TLVTAG_TYPEH_UINT|TLVTAG_TYPEL_NFD2)//数据类型 uint.2

```

typedef struct

```

{
    uint8_t port;
    uint8_t channel;
    uint8_t tag;
    uint8_t numafterdecimal;
}TLV_PROP_DEF;
typedef struct
{
    uint8_t buff[TLVBUFFSIZE];
    uint32_t length;
}TLV_DEF;

```

//CRC8 校验表

```

static const uint8_t CRC8_TAB[] = {
    0x00,0x5e,0xbc,0xe2,0x61,0x3f,0xdd,0x83,//0~7
    0xc2,0x9c,0x7e,0x20,0xa3,0xfd,0x1f,0x41,//8~15
    0x9d,0xc3,0x21,0x7f,0xfc,0xa2,0x40,0x1e,//16~23
    0x5f,0x01,0xe3,0xbd,0x3e,0x60,0x82,0xdc,//24~31
    0x23,0x7d,0x9f,0xc1,0x42,0x1c,0xfe,0xa0,//32~39
    0xe1,0xbf,0x5d,0x03,0x80,0xde,0x3c,0x62,//40~47
    0xbe,0xe0,0x02,0x5c,0xdf,0x81,0x63,0x3d,//48~55
    0x7c,0x22,0xc0,0x9e,0x1d,0x43,0xa1,0xff,//56~63
    0x46,0x18,0xfa,0xa4,0x27,0x79,0x9b,0xc5,//64~71
    0x84,0xda,0x38,0x66,0xe5,0xbb,0x59,0x07,//72~79
    0xdb,0x85,0x67,0x39,0xba,0xe4,0x06,0x58,//80~87
    0x19,0x47,0xa5,0xfb,0x78,0x26,0xc4,0x9a,//88~95
    0x65,0x3b,0xd9,0x87,0x04,0x5a,0xb8,0xe6,//96~103
    0xa7,0xf9,0x1b,0x45,0xc6,0x98,0x7a,0x24,//104~111
    0xf8,0xa6,0x44,0x1a,0x99,0xc7,0x25,0x7b,//112~119
    0x3a,0x64,0x86,0xd8,0x5b,0x05,0xe7,0xb9,//120~127
    0x8c,0xd2,0x30,0x6e,0xed,0xb3,0x51,0x0f,//128~135
    0x4e,0x10,0xf2,0xac,0x2f,0x71,0x93,0xcd,//136~143
    0x11,0x4f,0xad,0xf3,0x70,0x2e,0xcc,0x92,//144~151
    0xd3,0x8d,0x6f,0x31,0xb2,0xec,0x0e,0x50,//152~159
    0xaf,0xf1,0x13,0x4d,0xce,0x90,0x72,0x2c,//160~167
    0x6d,0x33,0xd1,0x8f,0x0c,0x52,0xb0,0xee,//168~175
    0x32,0x6c,0x8e,0xd0,0x53,0x0d,0xef,0xb1,//176~183
    0xf0,0xae,0x4c,0x12,0x91,0xcf,0x2d,0x73,//184~191
    0xca,0x94,0x76,0x28,0xab,0xf5,0x17,0x49,//192~199
    0x08,0x56,0xb4,0xea,0x69,0x37,0xd5,0x8b,//200~207
    0x57,0x09,0xeb,0xb5,0x36,0x68,0x8a,0xd4,//208~215
    0x95,0xcb,0x29,0x77,0xf4,0xaa,0x48,0x16,//216~223
    0xe9,0xb7,0x55,0x0b,0x88,0xd6,0x34,0x6a,//224~231
    0x2b,0x75,0x97,0xc9,0x4a,0x14,0xf6,0xa8,//232~239

```



```

0x74,0x2a,0xc8,0x96,0x15,0x4b,0xa9,0xf7,//240~247
0xb6,0xe8,0x0a,0x54,0xd7,0x89,0x6b,0x35,//248~255
};

#endif

```

2.1.4.2 YLTlvPac.c

```

/*
 * YLTlvPac.c
 *
 * Created on: 2019 年 3 月 4 日
 * Author: hb
 * version: 1.0
 */

#include "stdio.h" //uint8_t 等类型定义文件
#include "YLTlvPac.h"

TLV_DEF g_tlvpac;
/*****
 *CRC8 数据校验函数
 *通过查表的方式实现
 *p 输入数组
 *len 长度
 *返回校验值
 *****/
uint8_t util_crc8(uint8_t *p, uint32_t len)
{
    uint32_t i;
    uint8_t CRC8=0;
    for(i=0;i<len;i++)
    {
        CRC8 = CRC8_TAB[p[i]^CRC8];
    }
    return(CRC8);//返回校验值
}
/*****
 *本函数为设置消息体数据
 *pdata 为数据 buff 指针
 *devport 为数据帧识别时提供的端口号
 *devchannel 为数据帧识别时提供的通道号即 IO 号
 *datatag 参考 TLVTAG_XXXX 如 TLVTAG_ELECTRIC_EMVA
 *datatype 请参考 TLVTYPE_XXX 如 TLVTYPE_ELECTRIC_EMVA
 *val 发送的数值范围为 -2^128 ~ +2^128
 *返回此段数据段长度
 *****/
uint32_t set_tlv_body(uint8_t *pdata,uint8_t devport,uint8_t devchannel,uint8_t datatag,uint8_t datatype,float val)
{
    uint32_t index,ucval,times,i;
    uint8_t type,numafterdecimal;
    type = datatype&0xF0;
    numafterdecimal = datatype&0X0F;
    //目前禁止小数点后位数大于 7
    if(numafterdecimal > 7)
        return 0;
    if(numafterdecimal != 0)
    {

```



```

//根据 numafterdecimal 即小数点后有效数确定放大倍数
times = 1;
for(i=0;i<numafterdecimal;i++)
{
    times *= 10;
}
ucval = val*times;
}
index = 0;
*(pdata+index++) = devport;//端口号
*(pdata+index++) = devchannel;//IO 号
*(pdata+index++) = datatag;//数据 tag
*(pdata+index++) = datatype;//参考文档
switch(type)
{
    case TLVTAG_TYPEH_UCHAR:
        *(pdata+index++) = ucval&0xff;
        break;
    case TLVTAG_TYPEH_SSHORT:
    case TLVTAG_TYPEH_USHORT:
#ifdef BIGEDIAN
        *(pdata+index++) = (ucval>>8)&0xff;
        *(pdata+index++) = ucval&0xff;
#else
        *(pdata+index++) = ucval&0xff;
        *(pdata+index++) = (ucval>>8)&0xff;
#endif
        break;
    case TLVTAG_TYPEH_SINT:
    case TLVTAG_TYPEH_UINT:
#ifdef BIGEDIAN
        *(pdata+index++) = (ucval>>24)&0xff;
        *(pdata+index++) = (ucval>>16)&0xff;
        *(pdata+index++) = (ucval>>8)&0xff;
        *(pdata+index++) = ucval&0xff;
#else
        *(pdata+index++) = ucval&0xff;
        *(pdata+index++) = (ucval>>8)&0xff;
        *(pdata+index++) = (ucval>>16)&0xff;
        *(pdata+index++) = (ucval>>24)&0xff;
#endif
        break;
    default:
        return 0;
}
return index;//返回增加的数据长度
}

/*****
*设置消息包内容
*返回数据包长度
*****/
uint32_t set_tlv_package()
{
    g_tlvpac.length = 0;
    g_tlvpac.buff[g_tlvpac.length++] = TLVHEAD1;//帧头 1
    g_tlvpac.buff[g_tlvpac.length++] = TLVHEAD2;//帧头 2
    g_tlvpac.buff[g_tlvpac.length++] = TLVVERSION;//规范版本号
    g_tlvpac.buff[g_tlvpac.length++] = 0x00;//总长度高 8 位等待消息体填充后在赋值
    g_tlvpac.buff[g_tlvpac.length++] = 0x00;//总长度低 8 位等待消息体填充后在赋值
    g_tlvpac.buff[g_tlvpac.length++] = (TLVBID_WATER_PUMP>>8)&0xFF;//行业编码

```

```

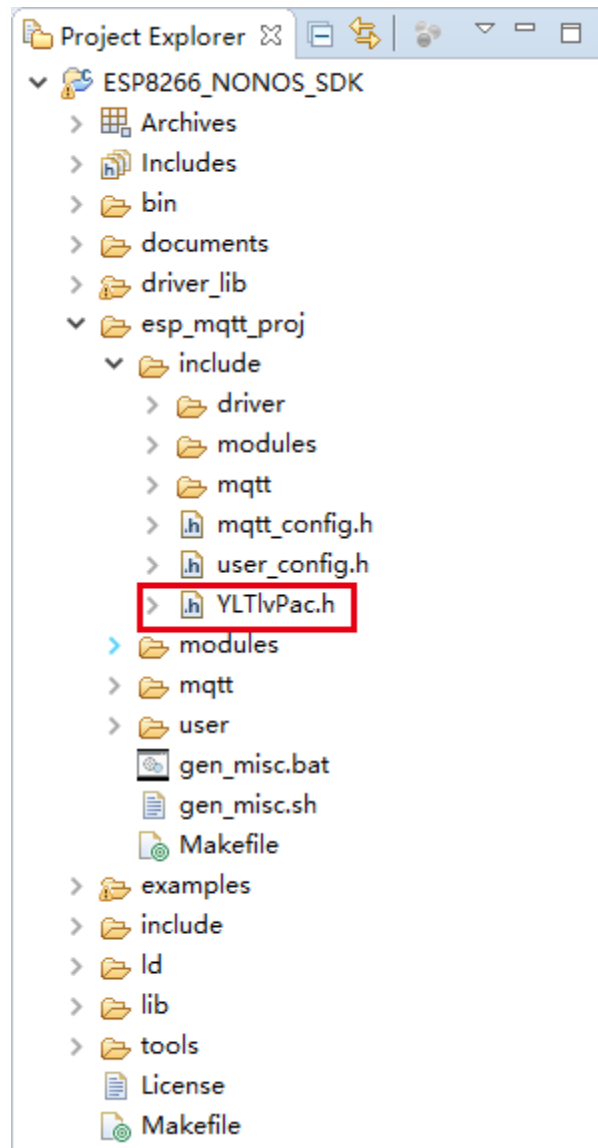
g_tlvpac.buff[g_tlvpac.length++] = TLVBID_WATER_PUMP&0xFF; //子行业编码
g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>24)&0xFF; //地址 1
g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>16)&0xFF; //地址 2
g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>8)&0xFF; //地址 3
g_tlvpac.buff[g_tlvpac.length++] = TLVADDR&0xFF; //地址 4
g_tlvpac.buff[g_tlvpac.length++] = TLVFUN; //功能码
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 YY 10 进制 为 0 时使用服务器时间
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 MM
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 DD
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 HH
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 MM
g_tlvpac.buff[g_tlvpac.length++] = 0; //时间 SS
//填充端口为 1, IO 口为 0, 数据表示 tag 为 0x25(电表 A 相电压值), 小数点后有效位数为 2, 数值为 210.1 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVA,
    TLVTYPE_ELECTRIC_EMVA,
    210.11);
//填充端口为 1, IO 口为 0, 数据表示 tag 为 0x26(电表 B 相电压值), 小数点后有效位数为 2, 数值为 220.2 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVB,
    TLVTYPE_ELECTRIC_EMVB,
    220.22);
//填充端口为 1, IO 口为 0, 数据表示 tag 为 0x27(电表 C 相电压值), 小数点后有效位数为 2, 数值为 230.3 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    230.33);
//填充端口为 1, IO 口为 1, 数据表示 tag 为 0x27(电表 C 相电压值), 小数点后有效位数为 2, 数值为 240.4 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x01,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    240.44);
//填充端口为 2, IO 口为 0, 数据表示 tag 为 0x27(电表 C 相电压值), 小数点后有效位数为 2, 数值为 250.5 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x02,
    0x00,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    250.55);

g_tlvpac.buff[3] = (g_tlvpac.length+3)>>8; //完成消息体填充后填充帧长信息, 高 8 位
g_tlvpac.buff[4] = g_tlvpac.length+3; //完成消息体填充后填充帧长信息, 低 8 位
g_tlvpac.buff[g_tlvpac.length++] = util_crc8(g_tlvpac.buff, g_tlvpac.length - 1); //crc8 校验
g_tlvpac.buff[g_tlvpac.length++] = TLVTAIL1; //帧尾
g_tlvpac.buff[g_tlvpac.length++] = TLVTAIL2; //帧尾
return g_tlvpac.length; //返回数据包长度
}

```

2.1.4.3 TLV 开发包使用

将 YLTlvPac.h 拷贝至工程目录下



复制 YLTlvPac.c 内所有内容至 user_main.c 内(这种方法最快捷)

注意包含头文件#include "YLTlvPac.h"

```
#include "ets_sys.h"
#include "driver/uart.h"
#include "osapi.h"
#include "mqtt.h"
#include "wifi.h"
#include "config.h"
#include "debug.h"
#include "gpio.h"
#include "user_interface.h"
#include "mem.h"
#include "YLTlvPac.h"
```

```

MQTT_Client mqttClient;

TLV_DEF g_tlvpac;
/*****
*CRC8 数据校验函数
*通过查表的方式实现
*p      输入数组
*len 长度
*返回校验值
*****/
uint8_t util_crc8(uint8_t *p, uint32_t len)
{
    uint32_t i;
    uint8_t CRC8=0;
    for(i=0;i<len;i++)
    {
        CRC8 = CRC8_TAB[p[i]^CRC8];
    }
    return(CRC8);//返回校验值
}
/*****
*本函数为设置消息体数据
*pdata 为数据 buff 指针
*devport 为数据帧识别时提供的端口号
*devchannel 为数据帧识别时提供的通道号即 IO 号
*datatag 参考 TLVTAG_XXXX 如 TLVTAG_ELECTRIC_EMVA
*datatype 请参考 TLVTYPE_XXX 如 TLVTYPE_ELECTRIC_EMVA
*val 发送的数值范围为 -2^128 ~ +2^128
*返回此段数据段长度
*****/
uint32_t set_tlv_body(uint8_t *pdata,uint8_t devport,uint8_t devchannel,uint8_t datatag,uint8_t datatype,float val)
{
    uint32_t index,ucval,times,i;
    uint8_t type,numafterdecimal;
    type = datatype&0xF0;
    numafterdecimal = datatype&0X0F;
    //目前禁止小数点后位数大于 7
    if(numafterdecimal > 7)
        return 0;
    if(numafterdecimal != 0)
    {
        //根据 numafterdecimal 即小数点后有效数确定放大倍数
        times = 1;
        for(i=0;i<numafterdecimal;i++)
        {
            times *= 10;
        }
        ucval = val*times;
    }
    index = 0;
    *(pdata+index++) = devport;//端口号
    *(pdata+index++) = devchannel;//IO 号
    *(pdata+index++) = datatag;//数据 tag
    *(pdata+index++) = datatype;//参考文档
    switch(type)
    {
        case TLVTAG_TYPEH_UCHAR:
            *(pdata+index++) = ucval&0xff;
            break;
        case TLVTAG_TYPEH_SSHORT:

```

```

        case TLVTAG_TYPEH_USHORT:
#ifdef BIGEDIAN
            *(pdata+index++) = (ucval>>8)&0xff;
            *(pdata+index++) = ucval&0xff;
#else
            *(pdata+index++) = ucval&0xff;
            *(pdata+index++) = (ucval>>8)&0xff;
#endif
        break;
        case TLVTAG_TYPEH_SINT:
        case TLVTAG_TYPEH_UINT:
#ifdef BIGEDIAN
            *(pdata+index++) = (ucval>>24)&0xff;
            *(pdata+index++) = (ucval>>16)&0xff;
            *(pdata+index++) = (ucval>>8)&0xff;
            *(pdata+index++) = ucval&0xff;
#else
            *(pdata+index++) = ucval&0xff;
            *(pdata+index++) = (ucval>>8)&0xff;
            *(pdata+index++) = (ucval>>16)&0xff;
            *(pdata+index++) = (ucval>>24)&0xff;
#endif
        break;
        default:
            return 0;
    }
    return index;//返回增加的数据长度
}

/*****
*CRC8 数据校验函数
*通过查表的方式实现
*返回数据包长度
*****/
uint32_t set_tlv_package()
{
    g_tlvpac.length = 0;
    g_tlvpac.buff[g_tlvpac.length++] = TLVHEAD1;//帧头 1
    g_tlvpac.buff[g_tlvpac.length++] = TLVHEAD2;//帧头 2
    g_tlvpac.buff[g_tlvpac.length++] = TLVVERSION;//规范版本号
    g_tlvpac.buff[g_tlvpac.length++] = 0x00;//总长度高 8 位等待消息体填充后在赋值
    g_tlvpac.buff[g_tlvpac.length++] = 0x00;//总长度低 8 位等待消息体填充后在赋值
    g_tlvpac.buff[g_tlvpac.length++] = (TLVBID_WATER_PUMP>>8)&0xFF;//行业编码
    g_tlvpac.buff[g_tlvpac.length++] = TLVBID_WATER_PUMP&0xFF;//子行业编码
    g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>24)&0xFF;//地址 1
    g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>16)&0xFF;//地址 2
    g_tlvpac.buff[g_tlvpac.length++] = (TLVADDR>>8)&0xFF;//地址 3
    g_tlvpac.buff[g_tlvpac.length++] = TLVADDR&0xFF;//地址 4
    g_tlvpac.buff[g_tlvpac.length++] = TLVFUN;//功能码
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 YY 10 进制 为 0 时使用服务器时间
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 MM
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 DD
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 HH
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 MM
    g_tlvpac.buff[g_tlvpac.length++] = 0;//时间 SS
    //填充端口为 1, IO 口为 0,数据表示 tag 为 0x25(电表 A 相电压值),小数点后有效位数为 2, 数值为 210.1 的值
    g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
        0x01,
        0x00,
        TLVTAG_ELECTRIC_EMVA,
        TLVTYPE_ELECTRIC_EMVA,

```

```

        210.11);
//填充端口为 1, IO 口为 0,数据表示 tag 为 0x26(电表 B 相电压值),小数点后有效位数为 2, 数值为 220.2 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVB,
    TLVTYPE_ELECTRIC_EMVB,
    220.22);
//填充端口为 1, IO 口为 0,数据表示 tag 为 0x27(电表 C 相电压值),小数点后有效位数为 2, 数值为 230.3 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    230.33);
//填充端口为 1, IO 口为 1,数据表示 tag 为 0x27(电表 C 相电压值),小数点后有效位数为 2, 数值为 240.4 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    240.44);
//填充端口为 2, IO 口为 0,数据表示 tag 为 0x27(电表 C 相电压值),小数点后有效位数为 2, 数值为 250.5 的值
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,
    0x01,
    0x00,
    TLVTAG_ELECTRIC_EMVC,
    TLVTYPE_ELECTRIC_EMVC,
    250.55);

g_tlvpac.buff[3] = (g_tlvpac.length+3)>>8;//完成消息体填充后填充帧长信息, 高 8 位
g_tlvpac.buff[4] = g_tlvpac.length+3;//完成消息体填充后填充帧长信息, 低 8 位
g_tlvpac.buff[g_tlvpac.length++] = util_crc8(g_tlvpac.buff,g_tlvpac.length - 1);//crc8 校验
g_tlvpac.buff[g_tlvpac.length++] = TLVTAIL1;//帧尾
g_tlvpac.buff[g_tlvpac.length++] = TLVTAIL2;//帧尾
return g_tlvpac.length;//返回数据包长度
}

void mqtt_publish_data()
{
    int len;
    len = set_tlv_package();
    MQTT_Publish(&mqttClient, MQTT_TOPIC, g_tlvpac.buff, len, 2, 0);
}

void wifiConnectCb(uint8_t status)
{
    if(status == STATION_GOT_IP){
        MQTT_Connect(&mqttClient);
    } else {
        MQTT_Disconnect(&mqttClient);
    }
}

void mqttConnectedCb(uint32_t *args)
{
    MQTT_Client* client = (MQTT_Client*)args;
    INFO("MQTT: Connected\r\n");
    //MQTT_Subscribe(client, "/mqtt/topic/0", 0);//取消连接时就发布数据
    //MQTT_Subscribe(client, "/mqtt/topic/1", 1);//取消连接时就发布数据
    //MQTT_Subscribe(client, "/mqtt/topic/2", 2);//取消连接时就发布数据

```

```
//MQTT_Publish(client, "/mqtt/topic/0", "hello0", 6, 0, 0);//取消连接时就发布数据
//MQTT_Publish(client, "/mqtt/topic/1", "hello1", 6, 1, 0);//取消连接时就发布数据

}

void mqttDisconnectedCb(uint32_t *args)
{
    MQTT_Client* client = (MQTT_Client*)args;
    INFO("MQTT: Disconnected\r\n");
}

void mqttPublishedCb(uint32_t *args)
{
    MQTT_Client* client = (MQTT_Client*)args;
    INFO("MQTT: Published\r\n");
    os_delay_us(1000000);
    mqtt_publish_data();
}

void mqttDataCb(uint32_t *args, const char* topic, uint32_t topic_len, const char *data, uint32_t data_len)
{
    char *topicBuf = (char*)os_zalloc(topic_len+1),
        *dataBuf = (char*)os_zalloc(data_len+1);

    MQTT_Client* client = (MQTT_Client*)args;

    os_memcpy(topicBuf, topic, topic_len);
    topicBuf[topic_len] = 0;

    os_memcpy(dataBuf, data, data_len);
    dataBuf[data_len] = 0;

    INFO("Receive topic: %s, data: %s \r\n", topicBuf, dataBuf);
    os_free(topicBuf);
    os_free(dataBuf);
}

/*****
 * FunctionName : user_rf_cal_sector_set
 * Description   : SDK just reversed 4 sectors, used for rf init data and paramters.
 *               We add this function to force users to set rf cal sector, since
 *               we don't know which sector is free in user's application.
 *               sector map for last several sectors : ABCCC
 *               A : rf cal
 *               B : rf init data
 *               C : sdk parameters
 * Parameters    : none
 * Returns       : rf cal sector
 *****/
uint32 ICACHE_FLASH_ATTR
user_rf_cal_sector_set(void)
{
    enum flash_size_map size_map = system_get_flash_size_map();
    uint32 rf_cal_sec = 0;

    switch (size_map) {
        case FLASH_SIZE_4M_MAP_256_256:
            rf_cal_sec = 128 - 5;
            break;
    }
}
```

```

        case FLASH_SIZE_8M_MAP_512_512:
            rf_cal_sec = 256 - 5;
            break;

        case FLASH_SIZE_16M_MAP_512_512:
        case FLASH_SIZE_16M_MAP_1024_1024:
            rf_cal_sec = 512 - 5;
            break;

        case FLASH_SIZE_32M_MAP_512_512:
        case FLASH_SIZE_32M_MAP_1024_1024:
            rf_cal_sec = 1024 - 5;
            break;

        default:
            rf_cal_sec = 0;
            break;
    }

    return rf_cal_sec;
}

void user_init(void)
{
    //uart_init(BIT_RATE_115200, BIT_RATE_115200); //沿用 74880 的波特率, 避免串口调试时需要切换因此注释掉这
    句
    os_delay_us(1000000);

    CFG_Load();

    MQTT_InitConnection(&mqttClient, sysCfg.mqtt_host, sysCfg.mqtt_port, sysCfg.security);
    //MQTT_InitConnection(&mqttClient, "192.168.11.122", 1880, 0);

    MQTT_InitClient(&mqttClient, sysCfg.device_id, sysCfg.mqtt_user, sysCfg.mqtt_pass, sysCfg.mqtt_keepalive, 1);
    //MQTT_InitClient(&mqttClient, "client_id", "user", "pass", 120, 1);

    MQTT_InitLWT(&mqttClient, "/lwt", "offline", 0, 0);
    MQTT_OnConnected(&mqttClient, mqttConnectedCb);
    MQTT_OnDisconnected(&mqttClient, mqttDisconnectedCb);
    MQTT_OnPublished(&mqttClient, mqttPublishedCb);
    MQTT_OnData(&mqttClient, mqttDataCb);

    WIFI_Connect(sysCfg.sta_ssid, sysCfg.sta_pwd, wifiConnectCb);

    mqtt_publish_data(); //发布一次
    INFO("\r\nSystem started ...\r\n");
}

```

一些函数的说明:

MQTT 数据发布

```

void mqtt_publish_data()
{
    int len;
    len = set_tlv_package();
    MQTT_Publish(&mqttClient, MQTT_TOPIC, g_tlvpac.buff, len, 2, 0);
}

```


MQTT 连接成功回调函数，注释掉原 SDK 内的消息发布调用

```
void mqttConnectedCb(uint32_t *args)
{
    MQTT_Client* client = (MQTT_Client*)args;
    INFO("MQTT: Connected\r\n");
    //MQTT_Subscribe(client, "/mqtt/topic/0", 0); //取消连接时就发布数据
    //MQTT_Subscribe(client, "/mqtt/topic/1", 1); //取消连接时就发布数据
    //MQTT_Subscribe(client, "/mqtt/topic/2", 2); //取消连接时就发布数据

    //MQTT_Publish(client, "/mqtt/topic/0", "hello0", 6, 0, 0); //取消连接时就发布数据
    //MQTT_Publish(client, "/mqtt/topic/1", "hello1", 6, 1, 0); //取消连接时就发布数据
}
```

发布数据成功回调函数

```
void mqttPublishedCb(uint32_t *args)
{
    MQTT_Client* client = (MQTT_Client*)args;
    INFO("MQTT: Published\r\n");
    os_delay_us(1000000);
    mqtt_publish_data();
}
```

主函数内需要进行一次消息发布才能在回调函数中不断调用消息发布

```
void user_init(void)
{
    //uart_init(BIT_RATE_115200, BIT_RATE_115200); //沿用 74880 的波特率，避免串口调试时需要切换因此注释掉这句
    os_delay_us(1000000);

    CFG_Load();

    MQTT_InitConnection(&mqttClient, sysCfg.mqtt_host, sysCfg.mqtt_port, sysCfg.security);
    //MQTT_InitConnection(&mqttClient, "192.168.11.122", 1880, 0);

    MQTT_InitClient(&mqttClient, sysCfg.device_id, sysCfg.mqtt_user, sysCfg.mqtt_pass, sysCfg.mqtt_keepalive, 1);
    //MQTT_InitClient(&mqttClient, "client_id", "user", "pass", 120, 1);

    MQTT_InitLWT(&mqttClient, "/lwt", "offline", 0, 0);
    MQTT_OnConnected(&mqttClient, mqttConnectedCb);
    MQTT_OnDisconnected(&mqttClient, mqttDisconnectedCb);
    MQTT_OnPublished(&mqttClient, mqttPublishedCb);
    MQTT_OnData(&mqttClient, mqttDataCb);

    WIFI_Connect(sysCfg.sta_ssid, sysCfg.sta_pwd, wifiConnectCb);

    mqtt_publish_data(); //发布一次
    INFO("\r\nSystem started ...\r\n");
}
```

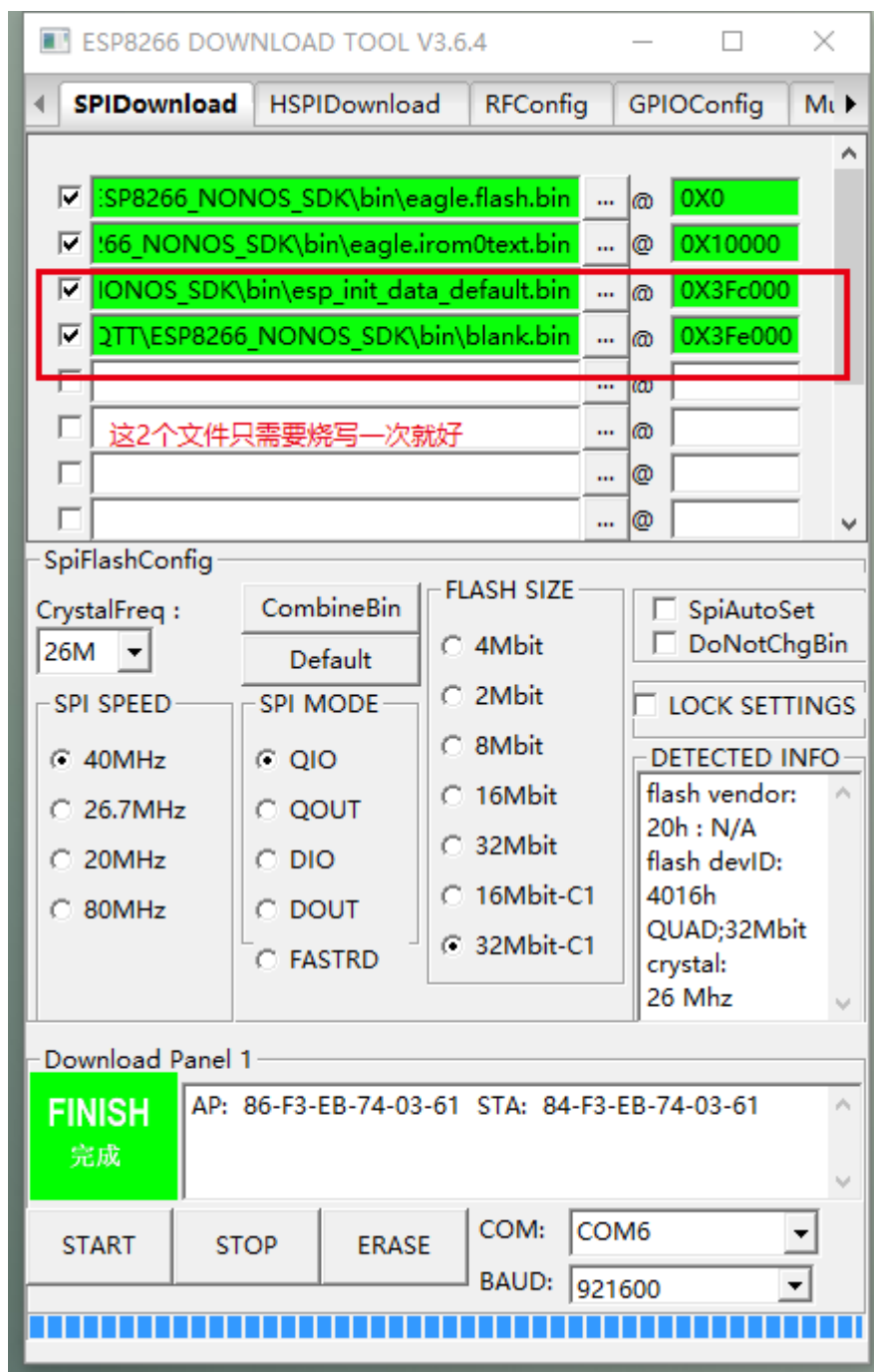
编译成功后得到文件的烧写地址。

```

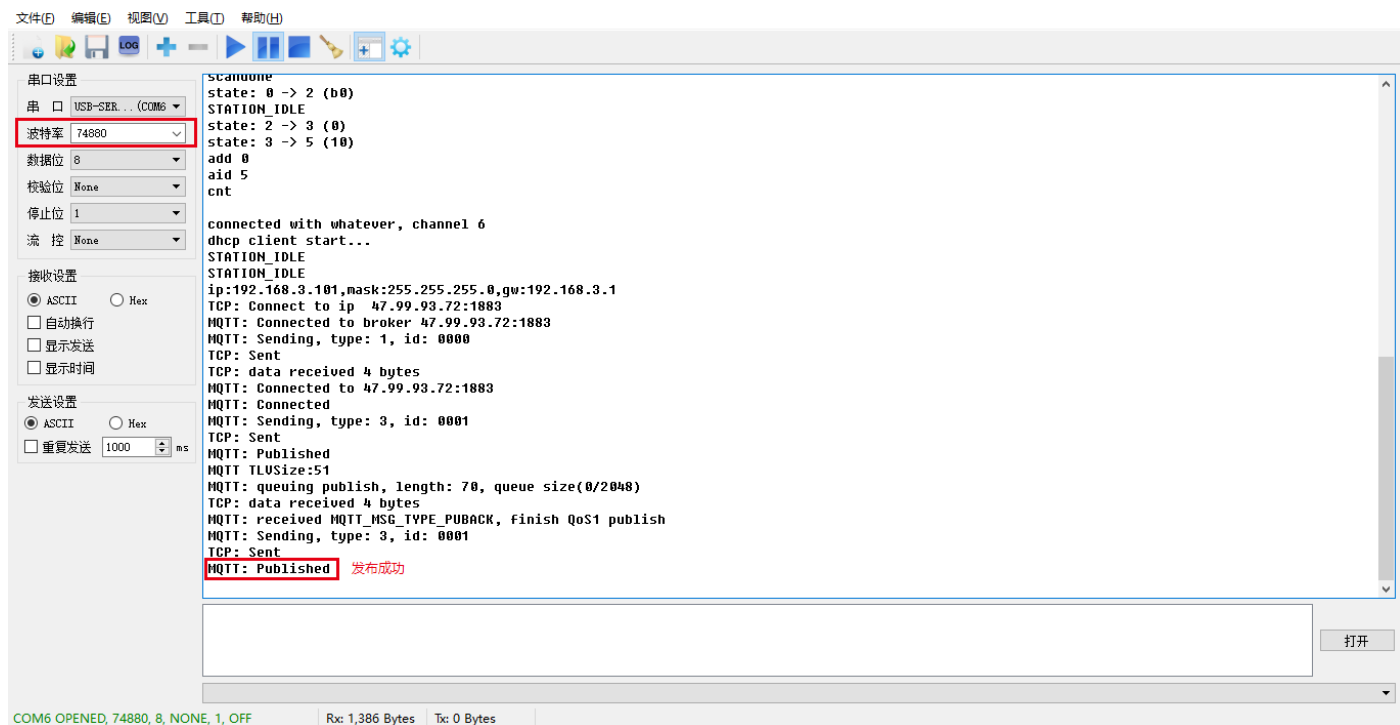
CDT Build Console [ESP8266_NONOS_SDK]
xt-xcc -L../lib -nostdlib -I../lib/eagle.app.v0.10 -Wl,--no-check-sections -Wl,--gc-sections -u call_user_start -Wl,--static -Wl,--start-group
!!!
No boot needed.
Generate eagle.flash.bin and eagle.irom0text.bin successfully in folder bin.
eagle.flash.bin----->0x00000
eagle.irom0text.bin---->0x10000
!!!
make[1]: Leaving directory '/cygdrive/d/Yarlung/1_0 Project/R&D Project/ESP8266_NONOS_SDK/esp_mqtt_proj'
16:38:31 Build Finished (took 9s.369ms)
  
```

使用烧写软件进行烧写。烧写完成后复位使其开始工作。

注意烧写地址变化。



可使用串口助手进行查看或调试，注意程序默认波特率为 74880。



2.2 雅鲁 IOT 平台查看数据

打开已创建的设备列表，点击详情。



2.2.1 数据查询

设备概览

设备列表

设备列表 > 历史数据

设备详情 历史数据 数据趋势 设备在线记录

设备数量总数(个) 0 昨日新增(个) 0 最近7日新增(个) 0

选择产品 测试产品 选择设备 1# 设备属性 所有属性 时间范围 最近1小时 查询

10 条记录/页 搜索:

设备名称	设备ID	数据值	时间
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:29
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:29

10

条记录/页

搜索:

设备名称	设备ID	数据值	时间
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:29
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:29
类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:29
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:28
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:28
类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:28
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:27
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:27
类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:27
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:26

2.2.2 关于端口的说明

由于设置tlv数据帧函数中有一段如下

//填充端口为2, IO口为0,数据表示tag为0x27(电表C相电压值),小数点后有效位数为2,数值为250.5的值

```
g_tlvpac.length += set_tlv_body(g_tlvpac.buff+g_tlvpac.length,  
    0x02,  
    0x00,
```

```
TLVTAG_ELECTRIC_EMVC,  
TLVTYPE_ELECTRIC_EMVC,  
250.55);
```

标识往端口 2 发数据，但是我们创建的设备端口为 1 号所以上图只显示 3 条我们设置的 tlv 数据帧

类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:29
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:28
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:28
类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:28
A相电压	1.0.2.1@dc.emeter.ua	210.11	2019-03-13 15:48:27
B相电压	1.0.2.1@dc.emeter.ub	220.22	2019-03-13 15:48:27
类型uc	1.0.2.1@dc.emeter.uc	230.33	2019-03-13 15:48:27

如想要获取到端口 2 的数据，创建一个端口为 2 的设备即可。