4G EC20 模块开发板

MZH008 使用手册

（板载 STM32F407 单片机）

Revision 1.01

1.GSM/GPRS 模块及开发系统介绍

1.1 EC20 模块

EC20 Mini PCIe 采用标准的 Mini PCIe 封装，同时支持 LTE，UMTS 和 GSM/GPRS 网
络，最大上行速率为 50Mbps，最大下行速率为 100Mbps。EC20 Mini PCIe 包含 EC20 Mini PCIe-A 和 EC20 Mini PCIe-E 版本，使其能够向后兼容现存的 EDGE 和 GSM/GPRS 网络，以确保在缺乏 3G 和 4G 网络的偏远地区也能正常工作。

EC20 Mini PCIe 支持接收分集技术，在终端设备上安装 2 个不同的蜂窝天线，从而实现优质可靠的无线连接。它通过多输入多输出技术（MIMO）降低误码率，改善通信质量。同时，EC20

Mini PCIe 结合了高速无线连接与内置多星座高精度定位 SPS+GLONASS 接收器。

EC20 Mini PCie 内置丰富的网络协议，集成多个工业标准接口，多种操作系统和软件功能(Windows XP/Windows Vista/Windows7/Windows 8/8.1/Linux/Android/eCall)，极大地拓展了 EC20 Mini PCIe 在 M2M 领域的应用范围，如 CPE，路由器，数据卡，平板电脑，车载，安全以及工业级 PDA。

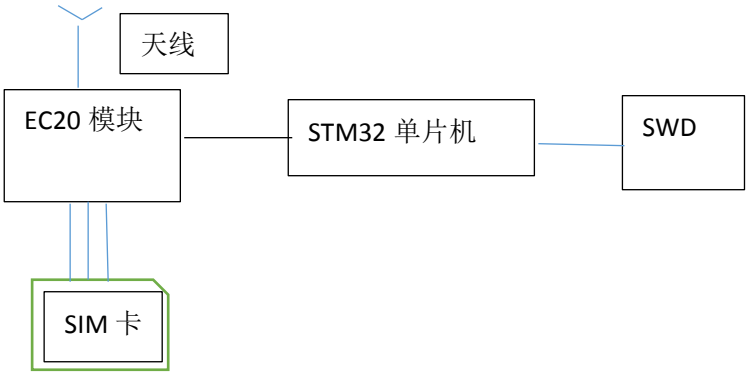
优势

- 支持 LTE，UMTS/HSPA+和 GSM/GPRS/EDGE 网络制式
- 标准的 Mini PCIe 封装，为客户设计及使用提供最大便利
- MIMO 技术满足无线通信系统对数据速率和链接的可靠性要求
- GNSS 接收器实现在任何环境下快速准确定位
- 提供参考设计、评估板和及时的技术支持满足客户产品快速上市的需求

1.2 MZH008 GSM/GPRS 模块开发系统简介

MZH001 GSM/GPRS 模块开发套件是上海移远科技公司用于调试应用 GSM/GPRS 模块 EC20 功能而专门设计一套开发系统。开发板以 STM32 单片机微处理器（型号：STM32F103C8T6）为核心，以 EC20 GPRS 模块为通讯渠道，并且引出了 STM32 单片机的大部分 IO 口资源。用户可在此基础上根据自己的需求开发出完成的系统。如远程控制，远程数据传输等等。

系统框图如下：



开发板包含附件:

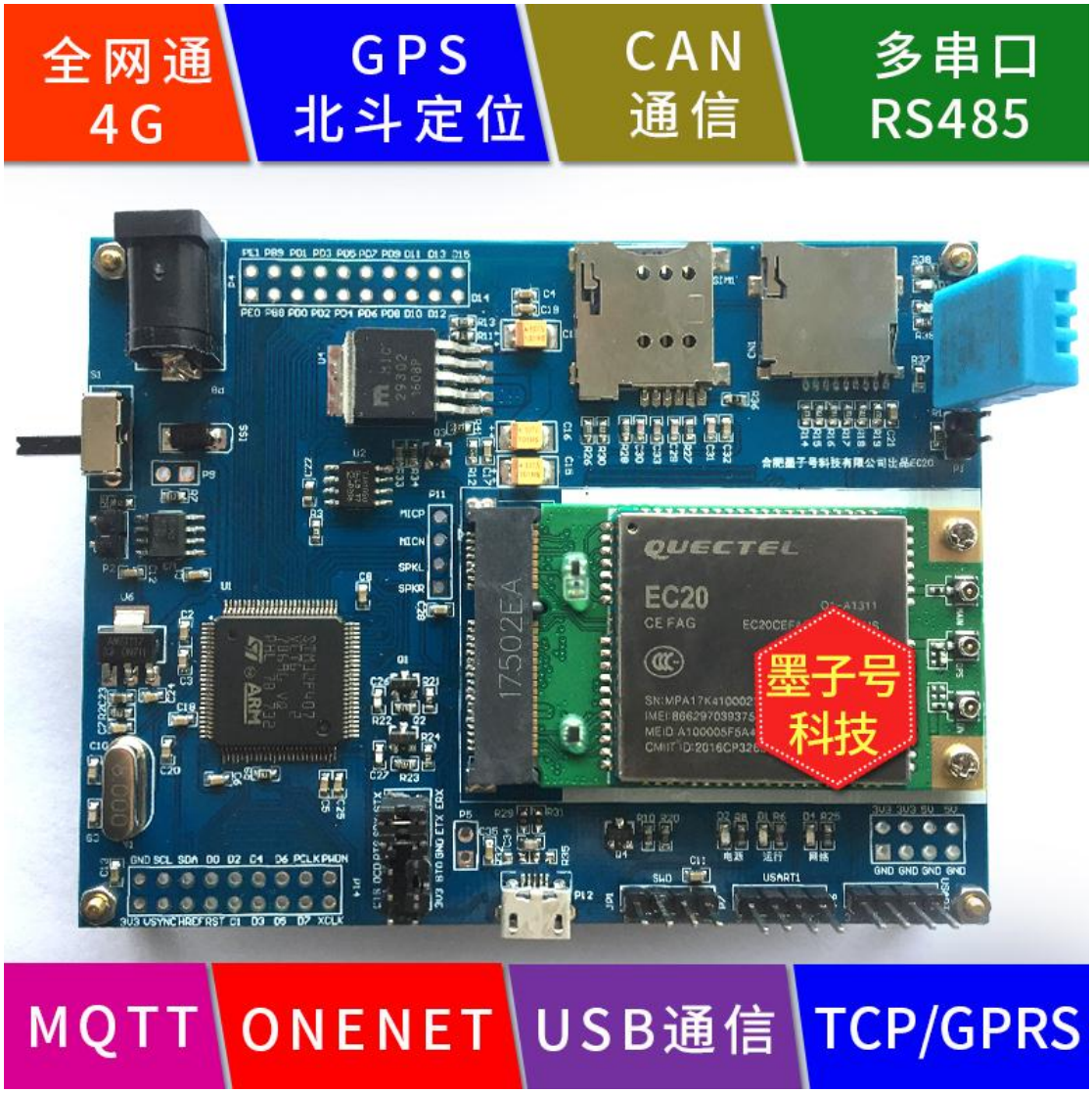
1. 5V/2A 电源适配器一个。
2. 有源 GPS/BD 天线
3. USB 数据通信线，支持用户用 USB 给 EC20 通讯。
4. USB TTL 转接线，支持给单片机下载程序。
5. ST-LINK（需另购，根据客户需求而配）

在 MZH008 开发板上，用户可以非常方便地验证，调试通过 AT 指令集实现 EC20 4G 全网通的各种功能。板子对外引出了安卓 USB 接口，方便用户进行模拟串口对模块进行调试。官方也给出了基于安卓以及 Linux 操作系统的驱动。方便用户之间通过 USB 接口来调试 EC20 模块。同样也给出了 Windows 的驱动。所以用户在使用 USB 调试的时候，需要先安装好对应的驱动之后才可以正常的进行使用。

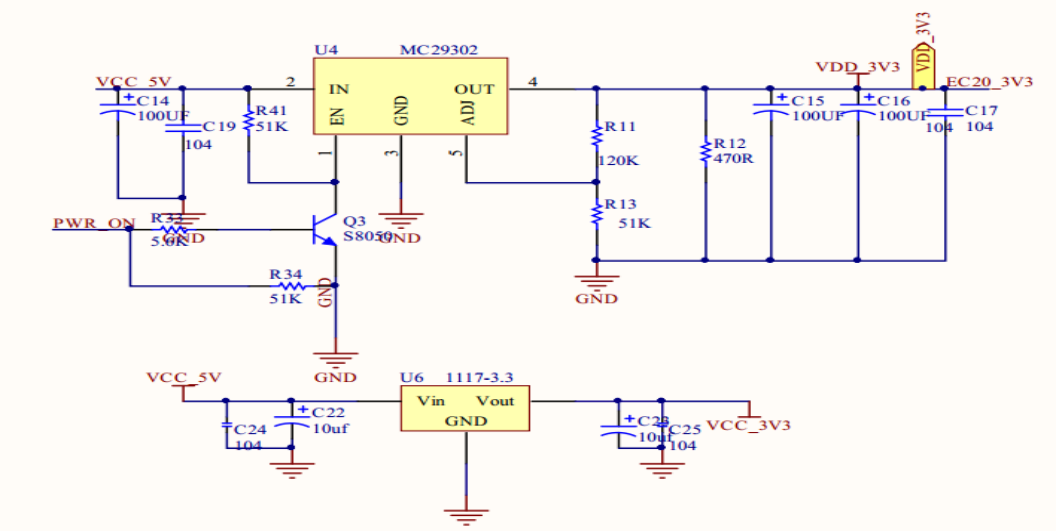
2. MZH008 开发板功能电路介绍

2.1 硬件说明

MZH008 EC20_4G 全网通模块开发板如下图所示：

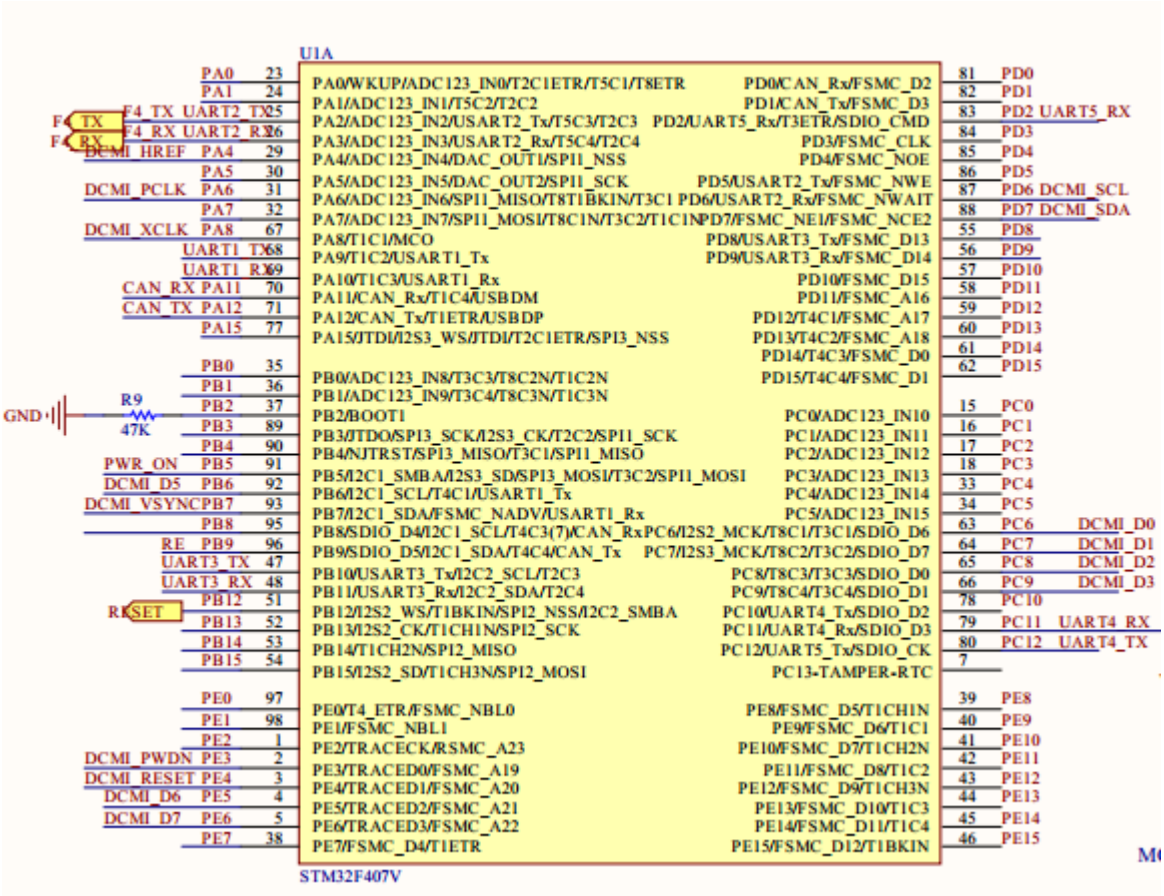


1）电源输入：



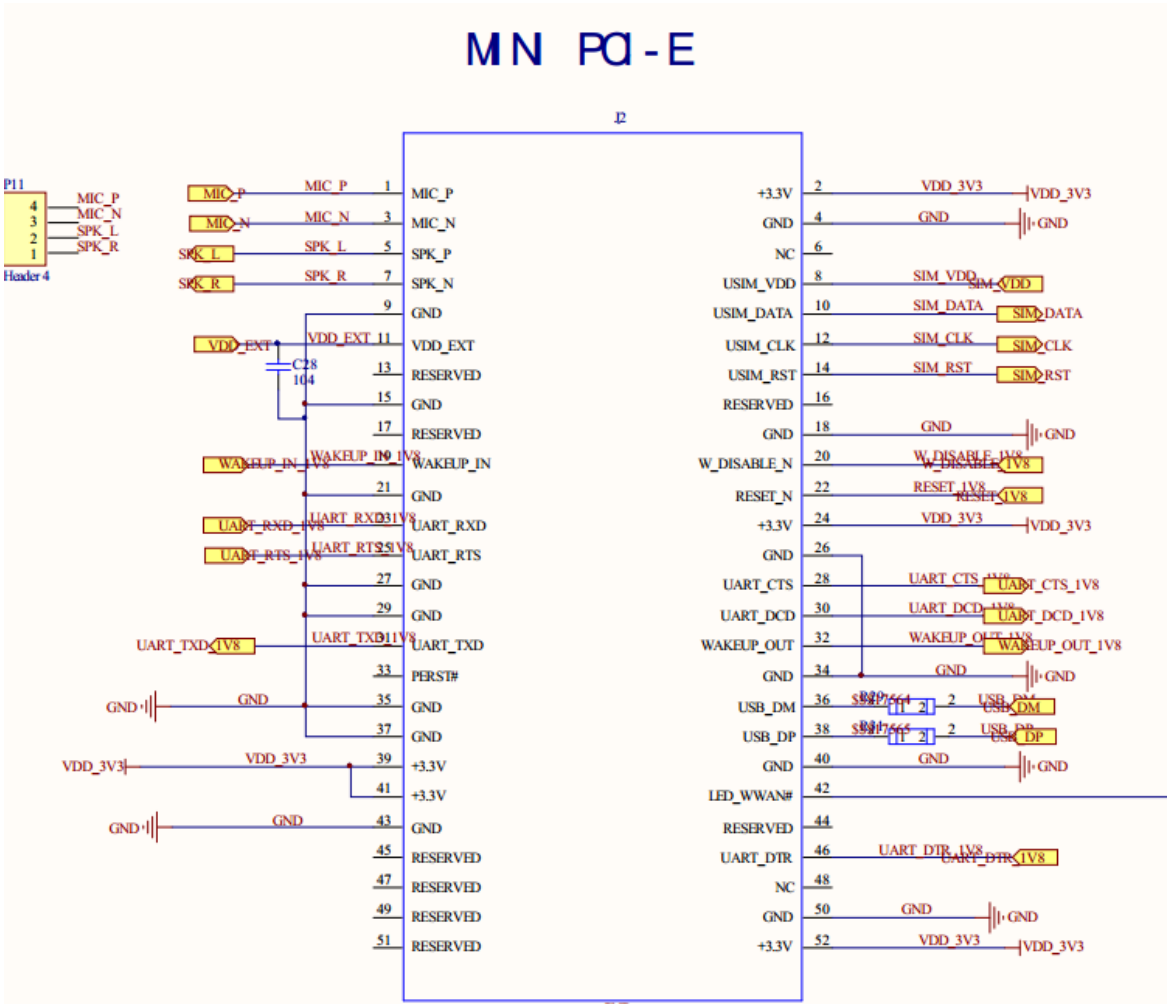
输入电压 5V。由于 MC20 的 GSM 功能在组网的瞬间电流需要达到 2A 左右，从电压输入的地方至少需要保证 2A 的提供，才可以让模块正常工作。如果出现 EC20 组网不成功的问题，可以查看供电是否不足导致的。所以我们在电路上设计完全参照官方给出的规格书采用 MC29302 的 LDO 来实现产生 2A 的电流，并且 PCB 走线也要至少保持 2mm 的线宽才可以。

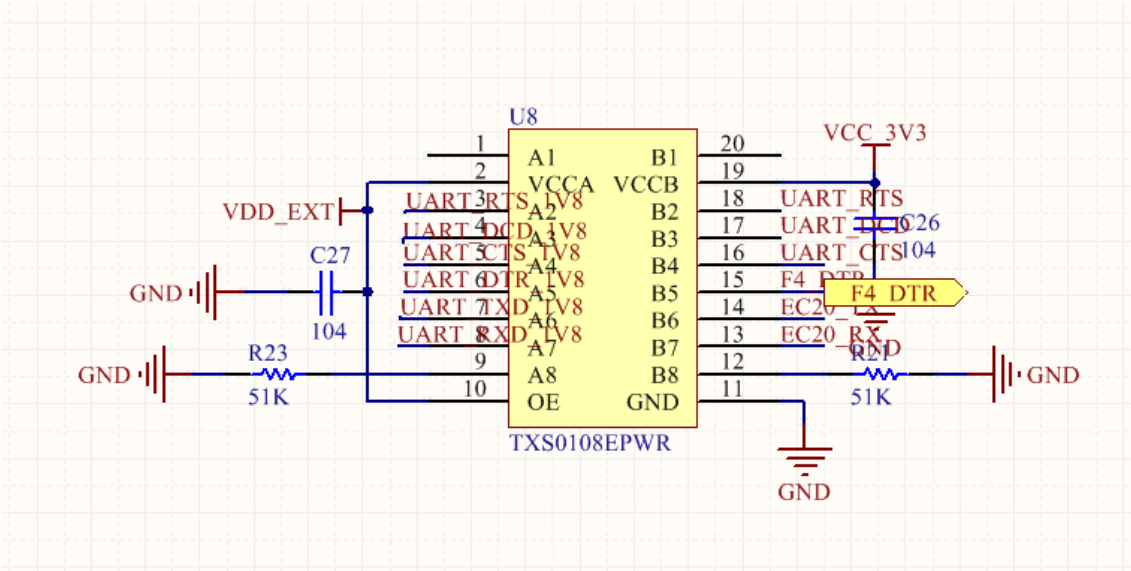
2）**STM32 单片机：STM32F407VET6** 整个系统的控制核心，用户可任意编程。Coretx-M4 内核单片机，主频可达 168Mhz，也支持 DSP 功能，极其强大的单片机。配合着 4G 模块可以实现多功能的实现。并且也引出了 GPIO 口。方便用户编程。



3）4G 全网通 模块 EC20：

上海移远科技推出的 EC20 模块，内置 TCP/IP 协议等。这个模块最大的优点就是在于全网通功能。不管移动，联通，电信 2G 3G 4G 卡都通通吃下。我们使用的是 7 模模块。如果采用的是 5 模，是不支持电信的 2G 3G 卡的。如果大家自行购买模块的时候一定要注意。支持 BD/GPS 双模定位。我们的开发板采用的是 MINIPCI 方式，EC20 已经有底板了。用户可以随意替换模块。单独使用底板都可以对模块进行编程使用。





注意 EC20 串口的电平是 1.8V 的，所以直接跟外部设备进行通讯是容易出现问题的。所以需要加入电平转换电路将设备信号转成成外部单片机可以识别的，并将单片机的信号转成 1.8V 给到 EC20。这里给的是 3.3V 电平的参考电路,这里采用电平转换芯片实现电平匹配问题。采用 TI 公司的 TXS0108EPWR 电平转换芯片来讲 1.8V 与 3.3V 之间做信号匹配。

4）4G LTE 天线接口： 4G 天线，不再是普通的 2G 天线，将全面支持 2G 3G 4G 卡。

5）GPS 北斗天线接口：提供用户 GPS 北斗天线接口,采用的是有源天线的接口。我们配对的也是有源天线，如果自己有对应的天线，那要注意看是否是有源的。防止不能正常使用。

6）EC20 模块指示灯：

正常通电之后，开关闭合，电源指示灯会被点亮，过一段时间 GSM 模块的网络指示灯也会进入寻网状态，寻网过程中，网络指示灯会 1s 闪一次，当附着网络之后，就会 2s 闪一次，表明此时模块一切正常。当然前提是需要插入 SIM 卡,SIM 卡全网通所以任意一家运营商的卡都可以正常使用的。下表是网络灯的状态。用户拿到板子如果对网络灯有什么疑问，可以参考下表来对照当前模块处于什么状态下。

表 20：网络指示管脚的工作状态

名称	管脚工作状态	所指示的网络状态
NET_MODE	高电平	注册 LTE 网络状态
	低电平	其他
NET_STATUS	慢闪（200ms 高/1800ms 低）	找网状态
	慢闪（1800ms 高/200ms 低）	待机状态
	快闪（125ms 高/125ms 低）	数据传输模式
	高电平	通话中

7）单片机预留 IO 口，EINT 中断，SPI 通讯口,UART 通讯口。Caerma 接口。OLED 接口。

用户自定义编程时，可以用作外部各种连接资源通讯控制口。引出了 Caerma 接口，用户可以来做图传功能，也可以接 OLED 做相关的显示。丰富设计。

8）SIM 卡座

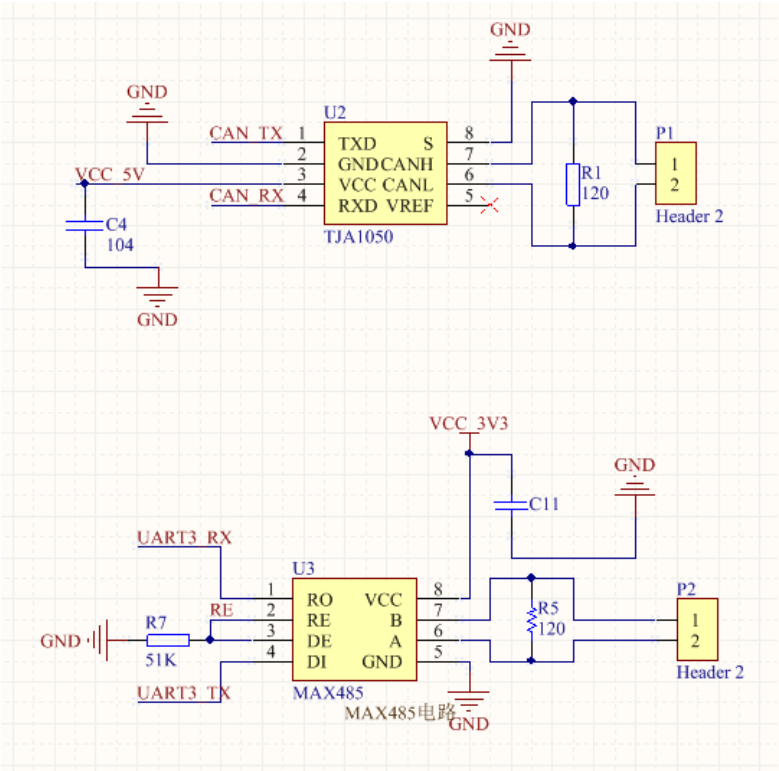
注意：安装或取卸 SIM 卡时，请先断开开发板电源，以免损毁开发板部件。并且手尽可能避免碰触到 SIM 卡座的金属部分，以免静电损坏模块部件。插卡方式一定要注意，可以参照下图所示，反过来也可以插进来。但是不能正常工作的。一定要缺口朝外的插法。

9）SD/TF 卡卡座

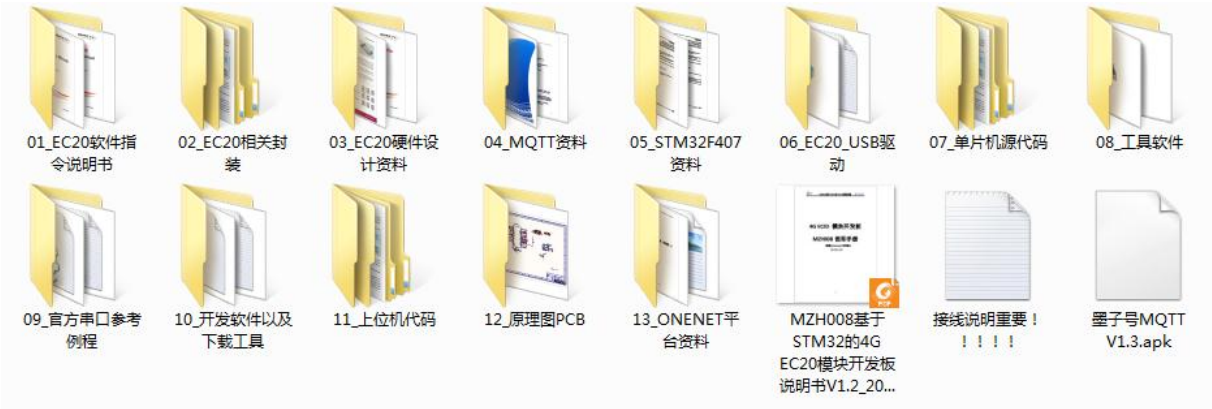
主板从单片机端引出了 SD/TF 卡座，方便用户做一些数据存储。采用的是从单片机端引出的方式，利用 SDIO 接口，如果需要做 TF 数据存储，需要自行移植 FAFT 系统，当然我们会提供相关的例程进行参考。

10）RS485 和 CAN 总线接口

板子也引出了 RS485 模块电路以及 CAN 总线电路，从多角度考虑用户使用工业总线控制，并会提供参考示例给到用户，此外板子也引出了 2 路单独 TTL 串口。方便用户做扩展。



2.2 软件资料说明



如上图我们会给用户提供相关的资料，方便大家来学习使用此开发板。

01 是 EC20 的相关指令说明资料。主要是一些 4G 的网络 AT 指令，GNSS 软件指令，HTTP,PPP 等相关资料说明。用户在使用板子的时候，遇到一些疑惑的指令可以对应的进行查看。

02 是 EC20 的开发板封装，为了让用户更方便的能够自己来设计开发板，这里提供了开发的所有包含原理图的封装以及 PCB 的封装。方便用户来进行设计，采用 AD10 画图软件下设计的。

03 是 EC20 的硬件设计说明，为了更好的了解 EC20 的硬件特性。这里给出 EC20 的硬件相关设计说明书。开发板用到的是 MINIPCI 接口的。他还有 LCC 封装的，对应两款不同的模块。在电路设计上也存在比较大的区别，所以用户根据这些资料来更好的设计电路也可以对开发板的电路硬件做充分的认识。

04 是 MQTT 协议的相关资料说明。包含了 MQTT 协议中文说明书，强烈推荐大家学习 MQTT 是需要对其底层协议规范说明做一个充分的认识。这样才好学习代码。此外有一些模拟软件以及抓包软件。

05 是单片机 STM32F407 的相关资料。因为单片机是核心，掌握单片机的相关使用才可以更好的利用程序来实现相关的功能。所以如果对 STM32 不熟悉或者一些寄存器不了解，可以看看 STM32 的基础资料进行了解。

06 是 EC20 模块的 USB 驱动，主要是方便用户通过 USB 的方式去控制设备，插入 windows 电脑会虚拟出多路串口，从而通过串口调试助手来控制设备。

07 是单片机的源代码，就是 STM32F407 单片机的相关代码，这个也是核心，通过单片机来控制 EC20 实现相关的 TCP 连接，MQTT，透传等多功能。

08 是一些工具软件，比如有串口调试助手，GNSS 定位软件，网络调试助手等。方便用户进行开发。

09 是官方给出的一些基础的串口操作例程。利用 QCOM 软件来对模块的串口进行操作然后实现相关的功能，用户有需要可以进行参考使用。

10 是单片机开发的环境软件，有 KEIL4 下载软件连接，AD10 的画图软件，STLINK 驱动，串口下载程序软件等。

11 是上位机代码，上位机采用 VB 进行设计的。主要有 GPS 地图显示以及 TCP 控制 LED 开关灯。用户需要的话，可以进行使用，都是源码开放。

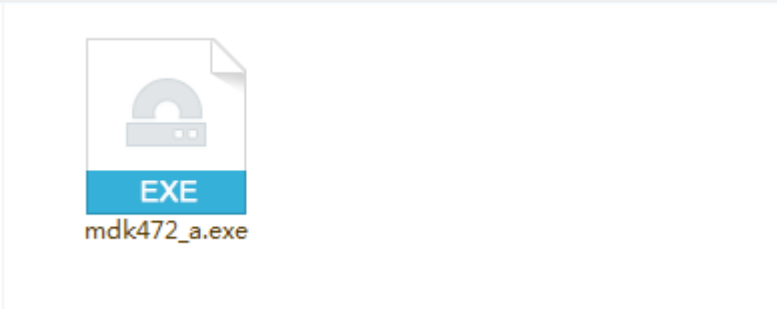
12 是原理图 PCB，这里提供的是 PDF 格式的。

13 是整理的一些 ONENET 平台的资料。如果需要登录 ONENET，需要遵循 ONENET 的相关协议规范。所以必要的参考资料是需要的。

2.3 软件的使用说明

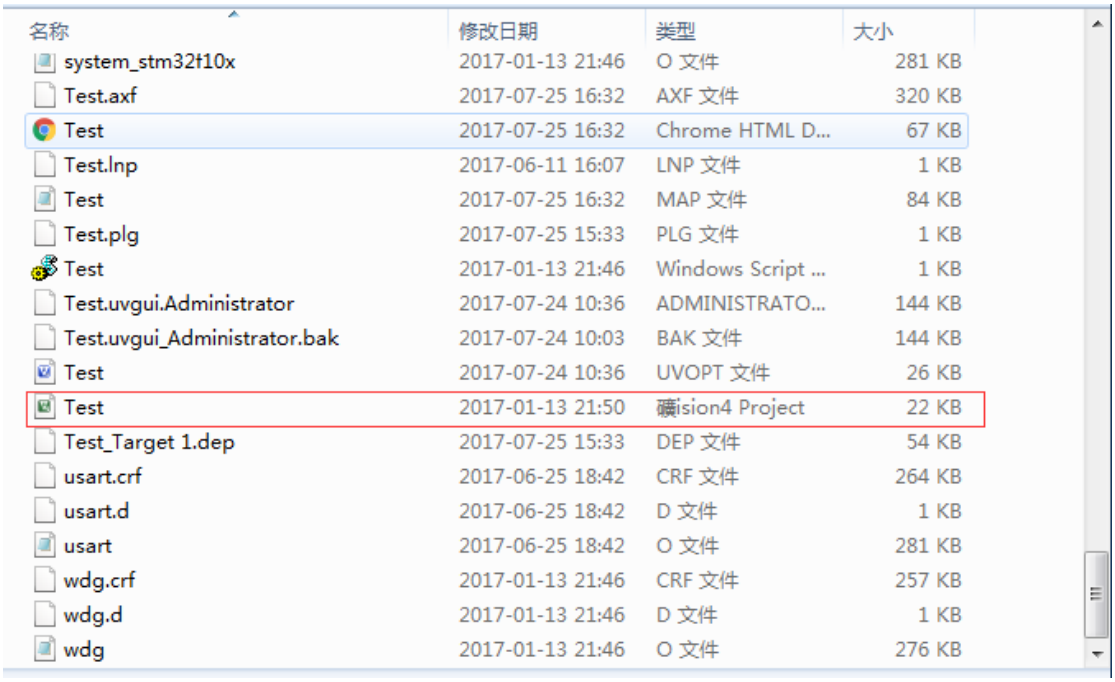
2.3.1 KEIL4 软件使用说明

KEIL4 软件的使用，这里我们使用并提供的 keil4 是基于 4.72 版本的，如果低于此版本以下的，使用当中可能会出现错误，所以推荐各位升级自己的 keil 版本。以符合开发需求，KEIL5 是可以正常使用的。



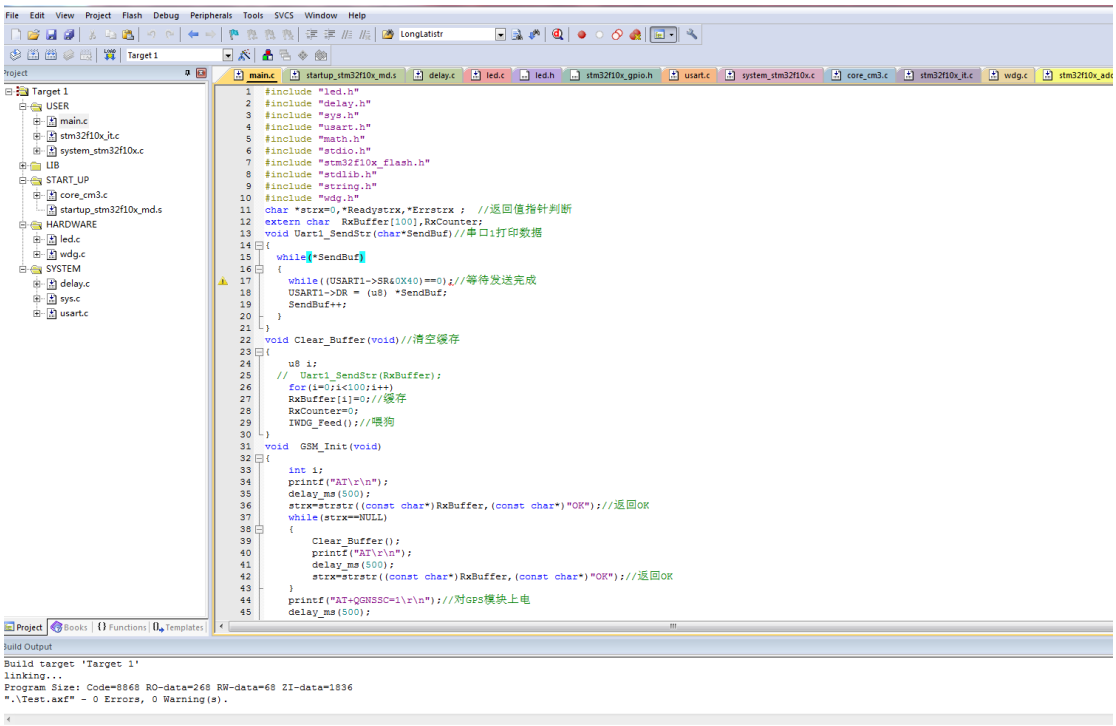
这个是我们百度云网盘所提供的 MDK472 的版本软件，就是 KEIL4 FOR ARM。下载完成之后，按照 NEXT 的方法一步步的将软件进行安装完毕，然后打开我们的工程文件，任意打开我们的代码即可。

我们的所有代码路径类似这个“源代码更新\GPRS_GPS 代码_MC20\USER”。工程文件在最终的 USER 里面，所以我们打开 USER 之后，找到工程文件进行打开。



名称	修改日期	类型	大小
system_stm32f10x	2017-01-13 21:46	O 文件	281 KB
Test.axf	2017-07-25 16:32	AXF 文件	320 KB
Test	2017-07-25 16:32	Chrome HTML D...	67 KB
Test.lnp	2017-06-11 16:07	LNP 文件	1 KB
Test	2017-07-25 16:32	MAP 文件	84 KB
Test.plg	2017-07-25 15:33	PLG 文件	1 KB
Test	2017-01-13 21:46	Windows Script ...	1 KB
Test.uvgui.Administrator	2017-07-24 10:36	ADMINISTRATO...	144 KB
Test.uvgui_Administrator.bak	2017-07-24 10:03	BAK 文件	144 KB
Test	2017-07-24 10:36	UVOPT 文件	26 KB
Test	2017-01-13 21:50	礦ision4 Project	22 KB
Test_Target 1.dep	2017-07-25 15:33	DEP 文件	54 KB
usart.crf	2017-06-25 18:42	CRF 文件	264 KB
usart.d	2017-06-25 18:42	D 文件	1 KB
usart	2017-06-25 18:42	O 文件	281 KB
wdg.crf	2017-01-13 21:46	CRF 文件	257 KB
wdg.d	2017-01-13 21:46	D 文件	1 KB
wdg	2017-01-13 21:46	O 文件	276 KB

我们选择这个 Test 版本为 4 的 Project 进行打开，打开之后，应该就可以看到下面的代码状态图：



```
#include "led.h"
#include "delay.h"
#include "sys.h"
#include "usart.h"
#include "math.h"
#include "stdio.h"
#include "stm32f10x_flash.h"
#include "stdlib.h"
#include "string.h"
#include "wdg.h"
char *strx=0,*Readystrx,*Errstrx ;  //返回值指针判断
extern char  RxBuffer[100],RxCounter;
void Uart1_SendStr(char*SendBuf)//串口1打印数据
{
  while(*SendBuf)
  {
    while((USART1->SR&0X40)==0);//等待发送完成
    USART1->DR = (u8) *SendBuf;
    SendBuf++;
  }
}
void Clear_Buffer(void)//清空缓存
{
    u8 i;
  //  Uart1_SendStr(RxBuffer);
    for(i=0;i<100;i++)
    RxBuffer[i]=0;//缓存
    RxCounter=0;
    IWDG_Feed();//喂狗
}
void  GSM_Init(void)
{
    int i;
    printf("AT\r\n");
    delay_ms(500);
    strx=strstr((const char*)RxBuffer,(const char*)"OK");//返回OK
    while(strx==NULL)
    {
        Clear_Buffer();
        printf("AT\r\n");
        delay_ms(500);
        strx=strstr((const char*)RxBuffer,(const char*)"OK");//返回OK
    }
    printf("AT+QGNSSC=1\r\n");//对GPS模块上电
    delay_ms(500);
```

```
Build target 'Target 1'
linking...
Program Size: Code=8868 RO-data=268 RW-data=68 ZI-data=1836
".\Test.axf" - 0 Errors, 0 Warning(s).
```

这就是我们所对应的代码图了，可以看到我们所写的代码，并有对应的注释，各位根据自己的需要，对代码进行编译下载即可。

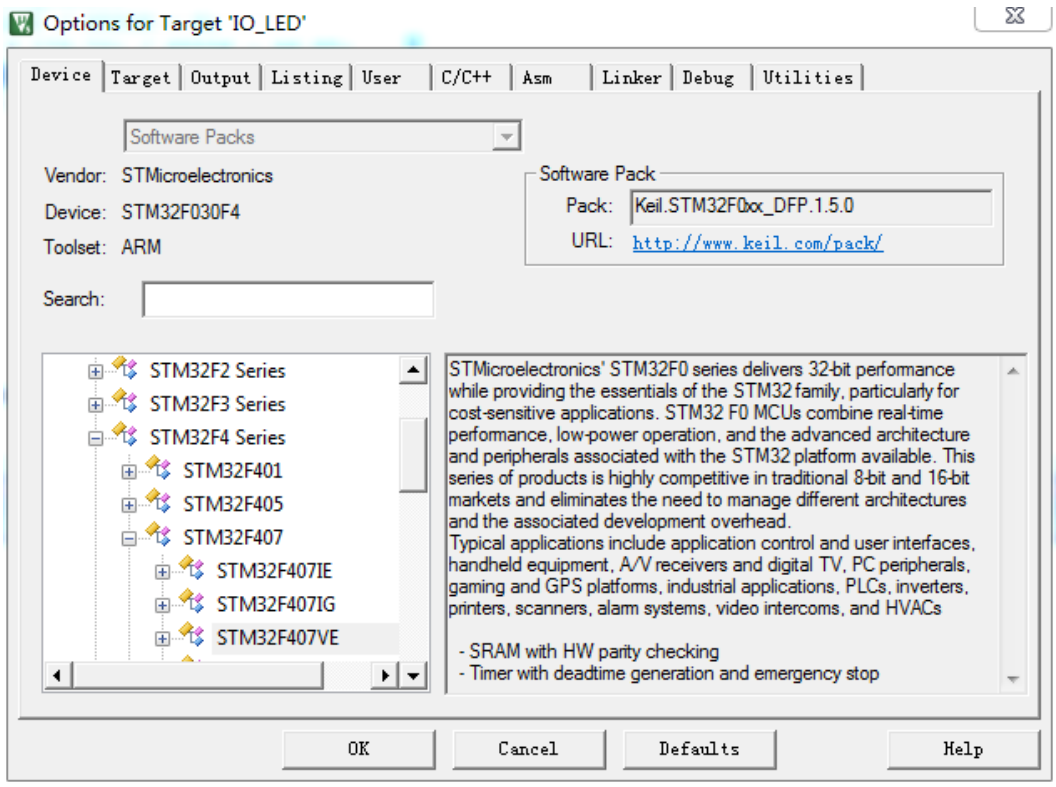
上面分别标注了 1，2，3.这里来解释一下，1 代表的是对代码进行编译并生成 hex 文件，编译如果通过了，下面会提供应该是 0 错误方可，如果出现大于 0 的错误，那么表明此时代码是有问题的，需要查找问题所在。

```
Build target 'Target 1'
linking...
Program Size: Code=8868 RO-data=268 RW-data=68 ZI-data=1836
".\Test.axf" - 0 Errors, 0 Warning(s).
```

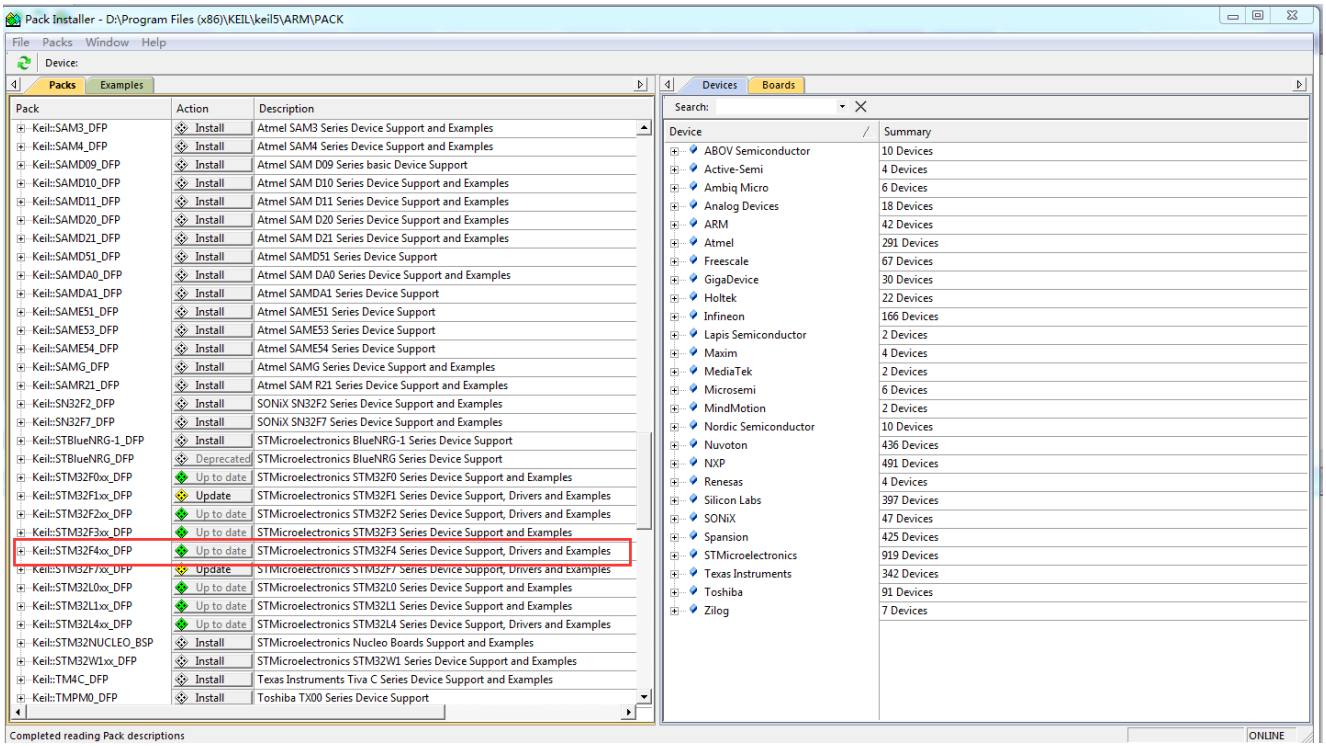
这给出的是 0 错误和 0 警告，表明程序完美，可以正常被使用。在实际使用的时候，一定要注意错误的产生，不然不能正常使用。2 代表的是程序下载，keil 不但具备程序编译的功能，也具备程序下载的功能，通过 2 的按钮可以对程序进行正常的下载，并让其正常的运行，这样我们就通过 STLINK 正常进行下载程序即可，3 是仿真，这一步非常关键，很多时候，大家会忽略掉这个仿真，感觉程序只要下载下去看现象即可，何必看仿真，其实这个思路是完全错误的，有了仿真，我们可以直观的看到现在串口打印以及返回的任何数据，并在一些错误的地方，他会停下来，因为程序当中做了 while 循环判断，这样就可以把每一步的流程把握的很清楚。当然我们的程序也提供了串口 1 打印模块返回的数据，如果有 USB 转 TTL 接在电脑端也可以看对应的工作情况。当然极力推荐大家使用 STLINK 进行仿真程序，这个也是为了以后大家在学习其他程序的时候，所养成的一个良好的习惯。

2.3.2 KEIL5 编译

考虑到有很多客户采用 KEIL5 编译程序，但是由于我们是用 4 编译的。直接打开通常会有一些报错。那么对于 KEIL5 编译首先需要 KEIL5 支持 F407 的库，因为 KEIL5 默认是不带芯片库，需要用户在线下载的。



比如用户使用 KEIL5 在设置单片机选项当中，一定要有支持 STM32F407VE 的选项才可以正常的工作。否则是没有办法正常工作。如果用户的 KEIL5 里面没有这个单片机选项，那么用户需要在线去安装。



上面的截图标志所示，需要找到这个安装目录，然后点击安装进行在线安装，由于我已经安装了，所以就显示 update 了。如果没有安装，那就会出现 Install 的界面，然后进行安装即可。

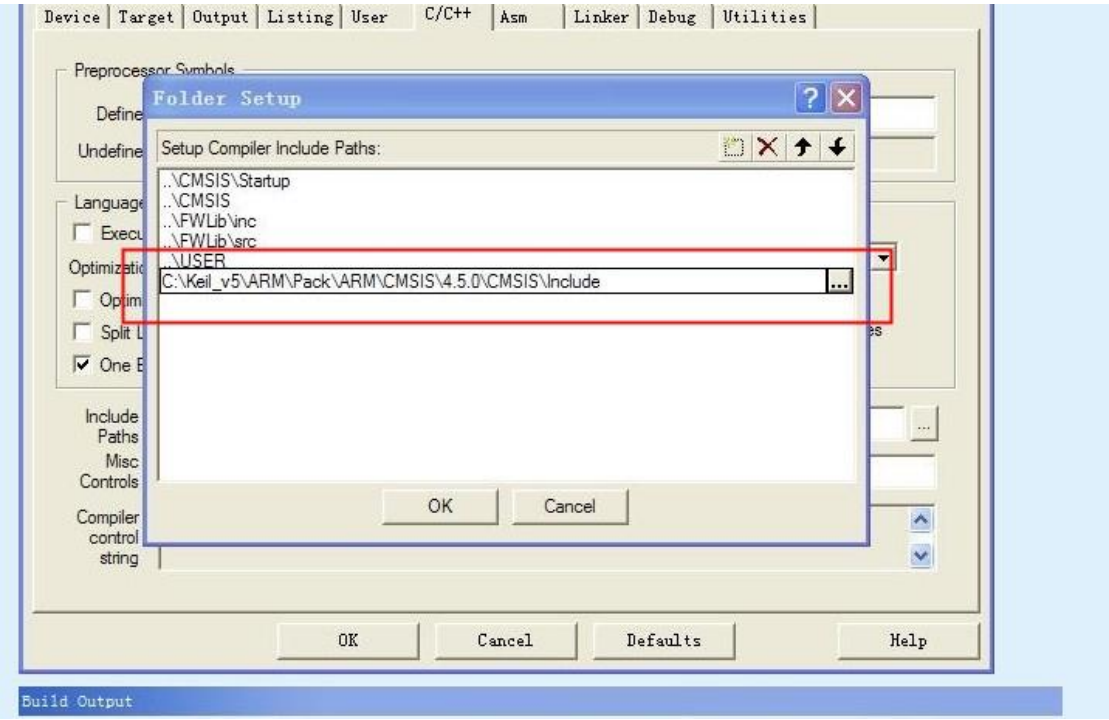
此外由于 KEIL4 和 KEIL5 还存在一些区别。尤其是内核文件的包含路径区别。比如经常会出现如下图的错误。找不到内核文件。

```
#5: cannot open source input file "core_cm3.h": No such file or directory
```

```
#5: cannot open source input file "core_cm3.h": No such file or directory
```

```
#5: cannot open source input file "core_cm3.h": No such file or directory
```

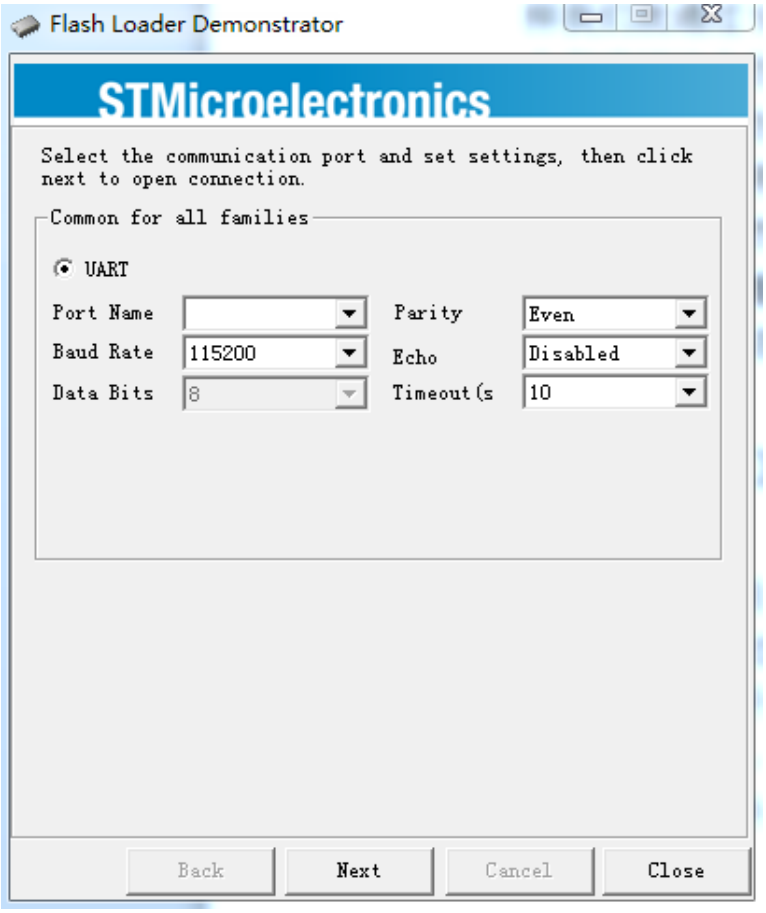
解决的方法就是需要将 KEIL5 的这个文件的路径添加进来。因为 KEIL5 他不能使用自带的内核文件，需要手动添加。



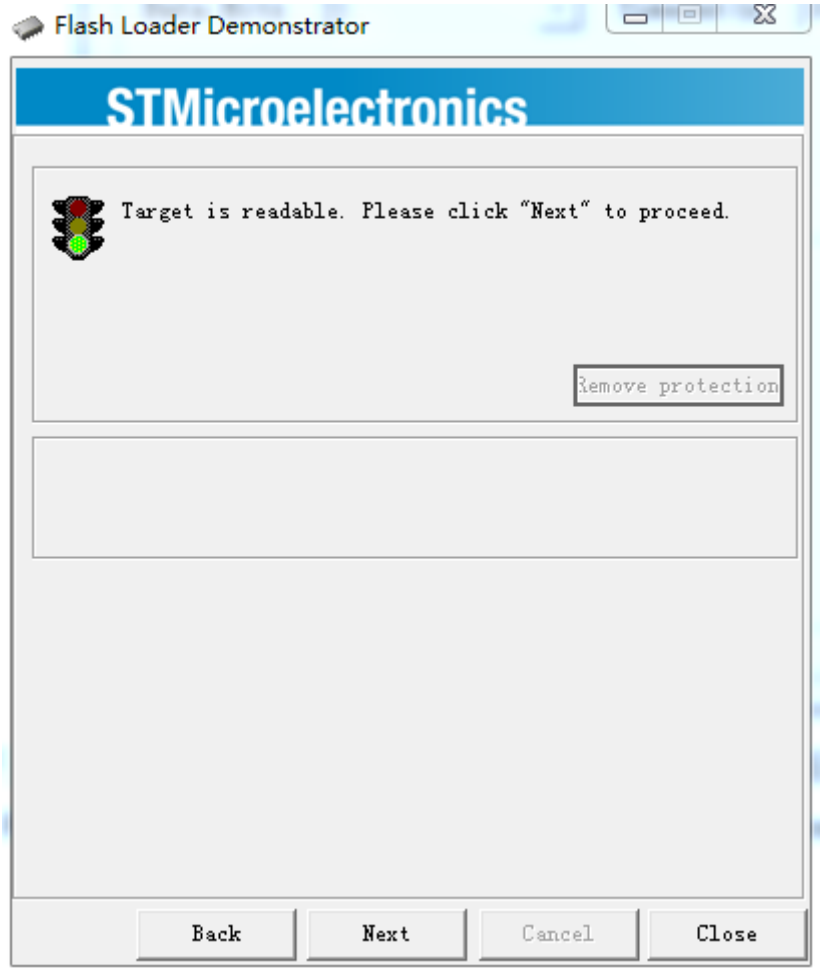
2.3.3 串口 ISP 下载程序

为了方便用户进行程序下载，不使用调试器 ST-LINK 或者 JLINK 给板子下载程序，那么这里就给出了通过串口 1 进行程序下载的方式，只是需要将 **BOOT0 从 GND 的连接地方接入到 3V3**，并重新开机，这样就为了方便重启的时候，用户进入 bootloader 程序，然后进行串口程序下载了。

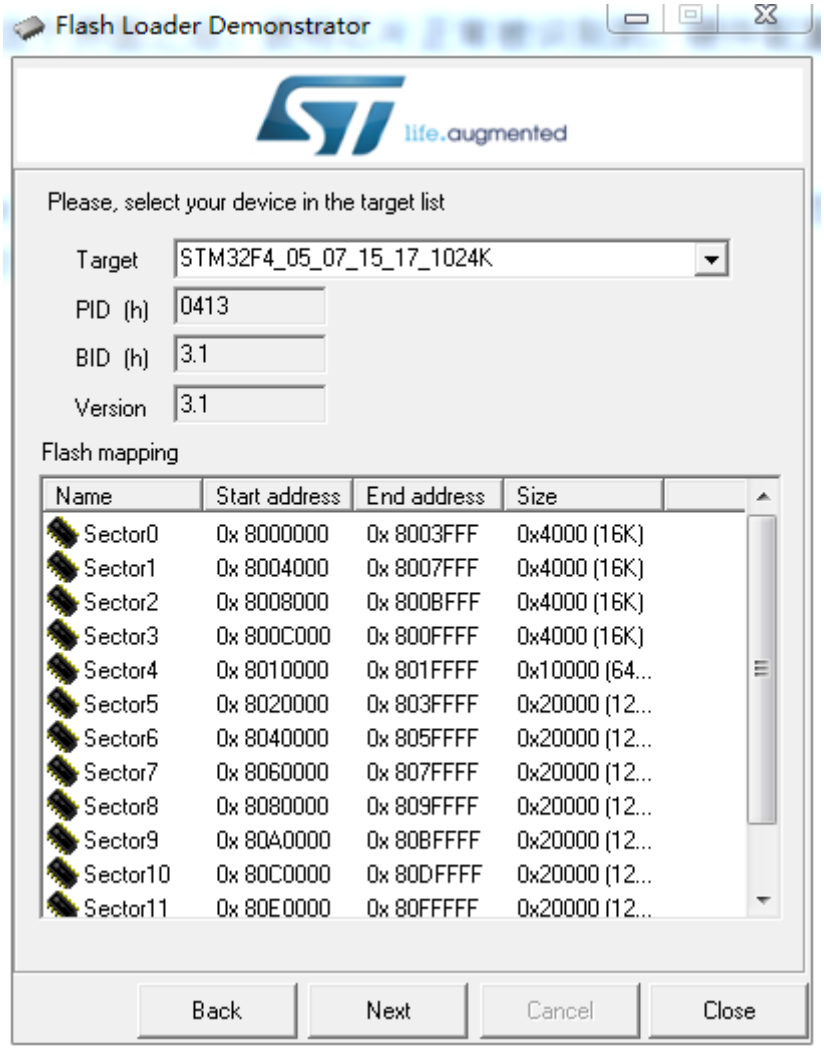
用到的软件是“Flash Loader Demonstrator”



将板子正常上电，并把串口 1 接上 USB_TTL 模块。然后插到电脑端，并对板子上电，然后界面上 Port Name 选择对应的端口就可以实现进行 Next 了。

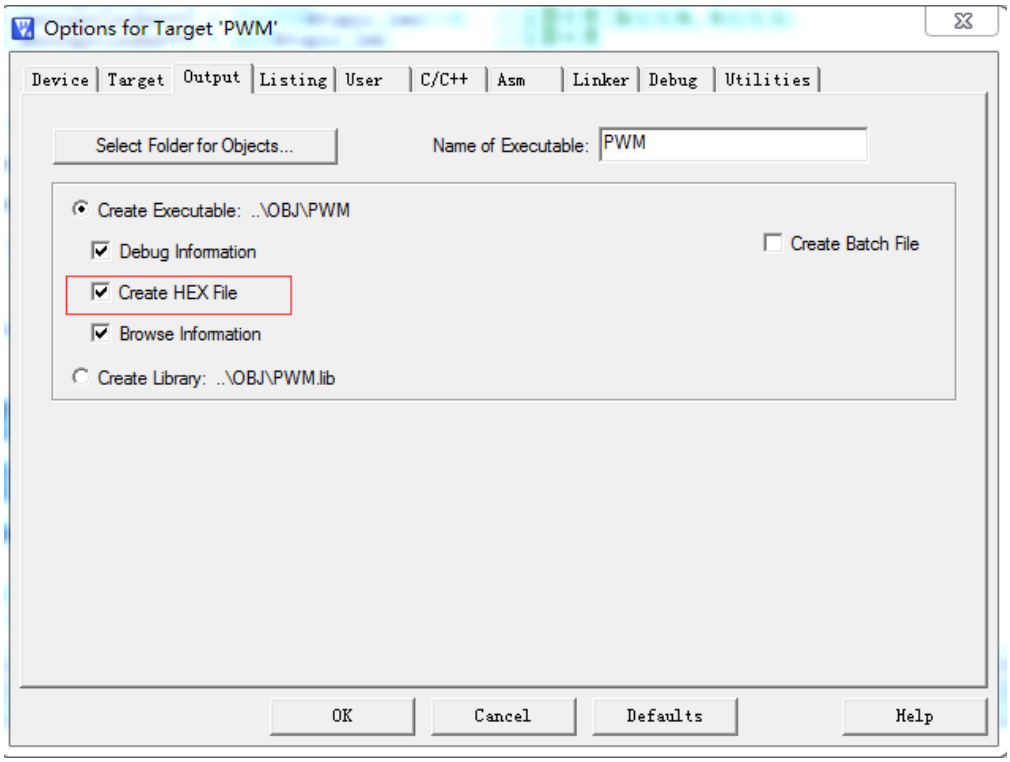


进入这个界面之后，表明芯片正常被识别到，硬件配置正常，可以进行下一步的擦写操作了。



Target 如果默认不是 STM32F4_1024K,那可以人为手动的选择一下进入此界面。然后继续 Next 即可。

这里就有 Erase 选择按钮，就是对单片机进行擦除。Download 就是选择程序下载，用户需要选择一个 file 文件进行添加进来。支持 s19,hex 文件。用户可以用 KEIL 生成一个.hex 文件进行烧录。

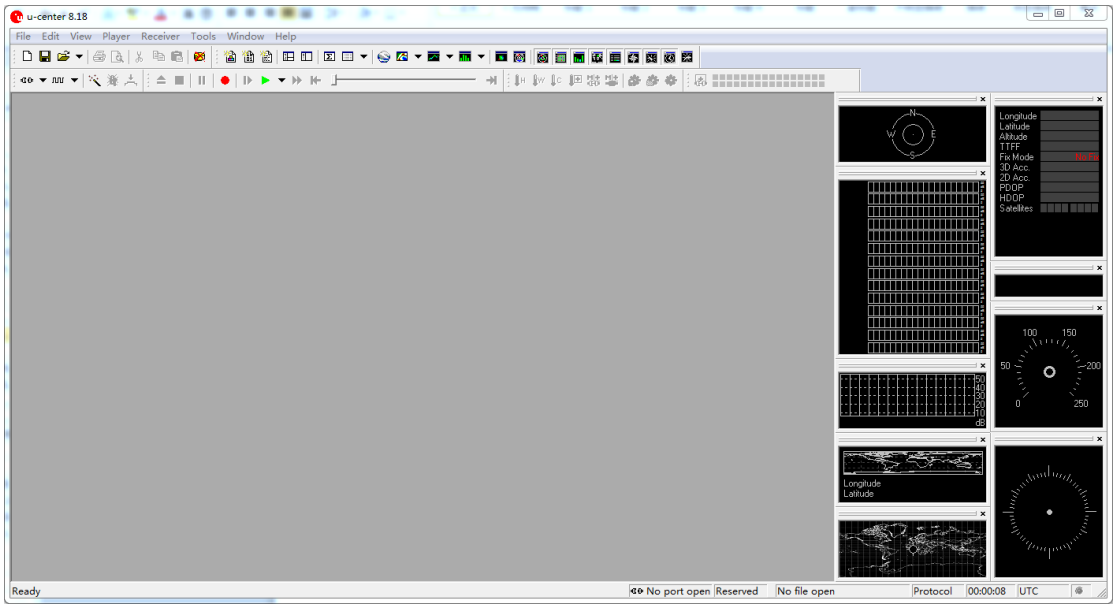


然后在文件路径当中找到.hex 文件进行添加即可。这样软件就会通过 USB_TTL 对单片机的串口 1 下载程序进行程序的更新。

如果出现 successfully 表明下载成功，用户通过 USBTTL 对单片机的程序下载没有问题。

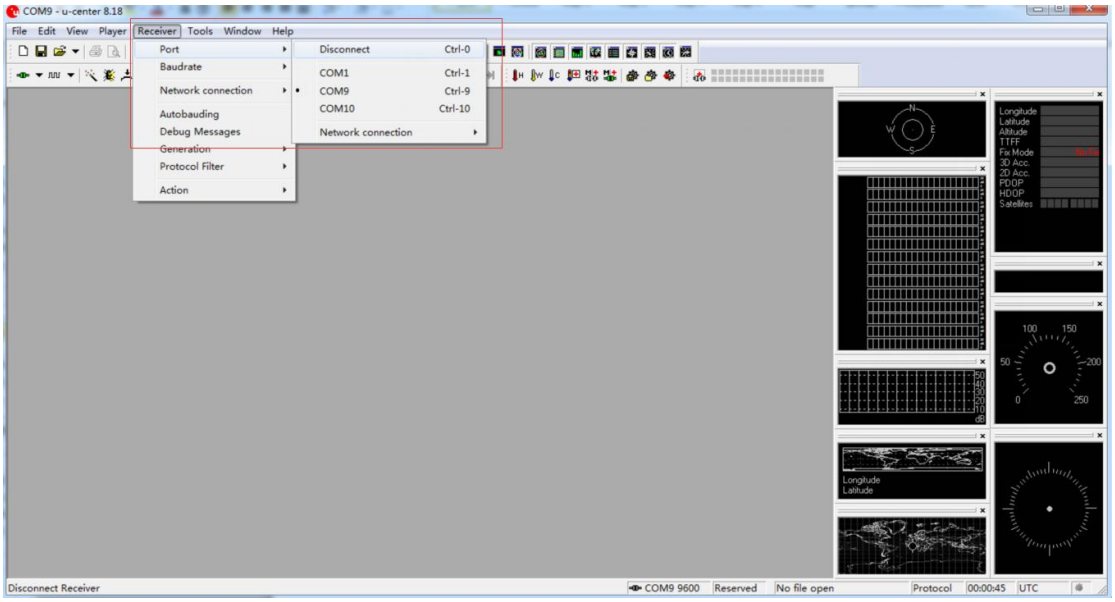
2.3.4 UBLOX 软件使用说明

UBLOX 软件是我们提供给大家进行 GPS 定位中所使用的，使用这个软件主要是方便各位通过 USB 来打印 GPS 返回的数据，注意这里用的是 USB，而不是用串口了。所以可以利用我们提供给大家的 USB 安卓数据线来查看当前的相关定位信息的。打开 GPS 电源之后就可以使用了，GPS 是可以不要插 SIM 卡，直接操作就可以。

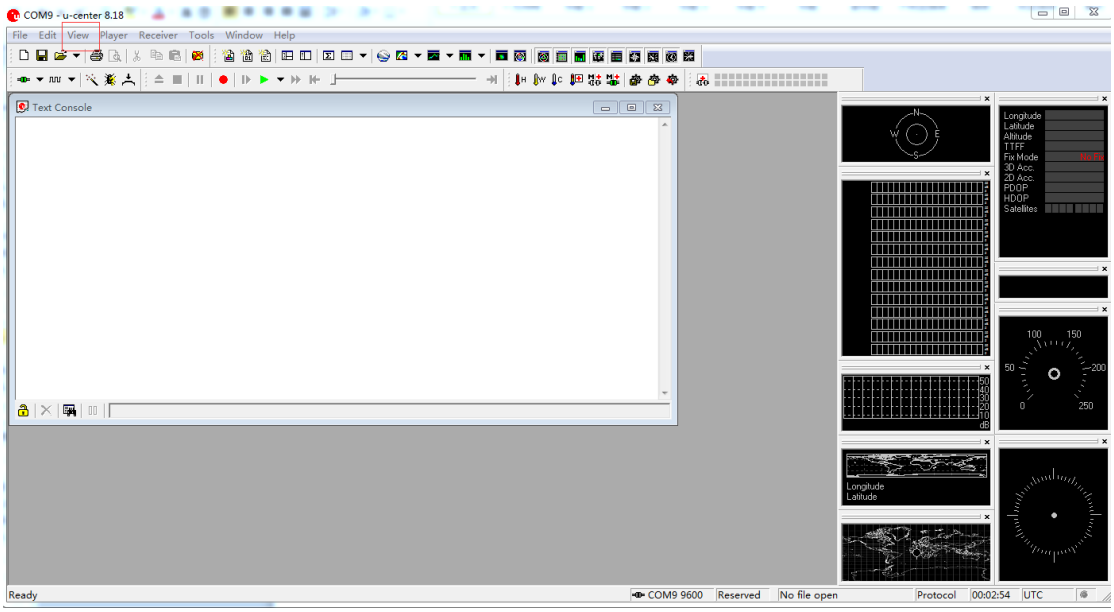


U-center 软件截图

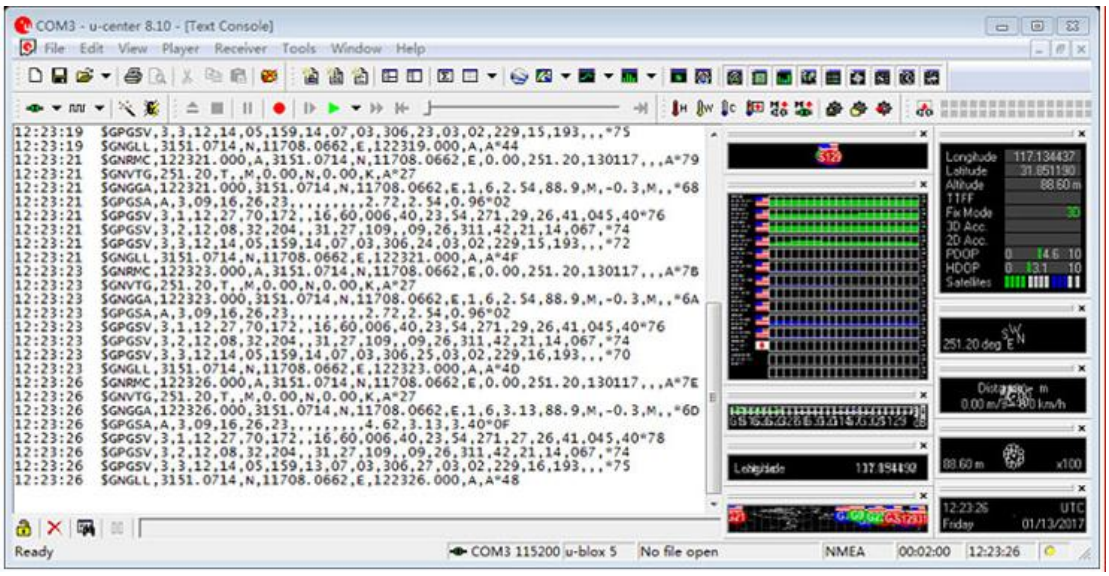
使用之前，需要先打开 USB 所虚拟出来的端口号。先用 AT 端口对 GNSS 模块进行上电，然后 GPS 才能正常工作。并吐出对应的数据。



端口选择好之后，数据默认是看不到的，我们需要选择对应的查看来将数据打印出来进行查看。



VIEW 里面下拉查看 Text Console，然后就会进行正常的数据打印了。如果连接正确。此时就会有对应的数据进行打印，此时的数据是没有定位成功的，关于定位的 GPS 的每段话所代表的含义，各位可以查看我们的 GPS 相关资料来找对应的说明查看即可。



3.主功能代码介绍

3.1 USB 调试模块

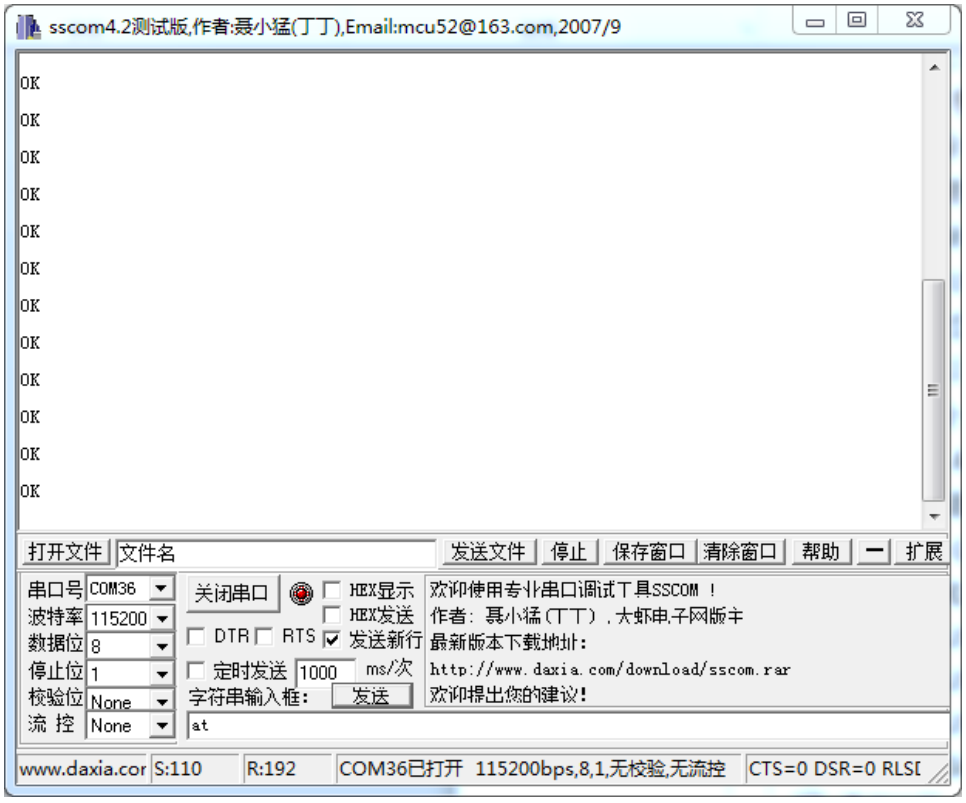
EC20 支持 USB 功能，对于 Windows 客户直接将我们提供的驱动代码安装好之后，当 EC20 正常开机之后，USB 就会在电脑端映射出多个端口出来。这一节也是很重要的。注意 USB 使用和模块的串口是独立开来的。当使用 USB 去调试模块的时候，也支持串口去控制模块。除非是一些特定参数使用 USB 或者是串口去控制模块做一些变化，那么会同时对串口或者 USB 造成同步影响。这个是什么意思呢，比如说 ATE0&W，关闭回显。这个语句使用 USB 写的和串口写的都会同时对 USB 或者串口下一步操作模块造成影响。但是比如说通过 USB 去控制串口的波特率，是没法设置成功的。如果需要控制串口的波特率，一定要通过 EC20 的串口去控制并保存。否则 USB 设置是不起作用。这一点大家在使用过程中可以慢慢的去掌握。

USB 口与驱动配合会映射出 3 个端口：

1. 一个串口用于 AT 指令，
2. 一个调制解调器用于 PPP 拨号，
3. 一个串口用于 GPS 输出。

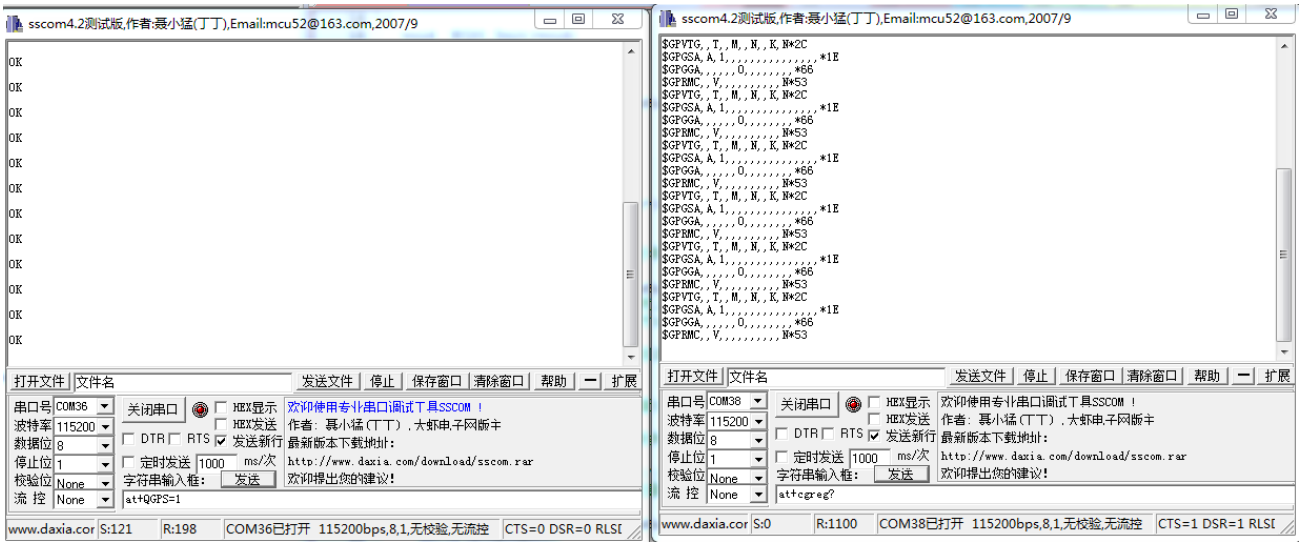
这个是 USB 虚拟出来的串口，分别 COM36,COM37,COM38。COM44 是我们自己的 CH340 以及 COM1 是台式机自带的。

COM36 就是 USB 使用的 AT 端口，就是给到用户发一些 AT 指令去控制 EC20 的。这个调试方法就是通过串口助手去控制模块，与我们用其他模块去控制是一样的。下面打开串口助手然后发 AT 指令去看模块的反馈情况。



看我们发送 at 模块返回了 OK 给到我们，注意这里的波特率是不受限制的。对于 USB 而言，这里的波特率是不存在的，所以用户可以任意设置都不会有任何的影响。

那么此时可以发送将 GPS 电源打开的指令，看看模块反馈的情况。



当 AT 端口发送对 GPS 电源上电之后，NEMA 端口就会立即对外吐出 GPS 数据，这个例子说明 GPS 的数据只要将模块的电源正常上电之后，就可以正常吐出。因为此时我未插天线，所以没有定位。如果需要不进行打印，就在 AT 端口输入 AT+QGPS=0 关闭 GPS 模块的电源就可以了。这样在 NEMA 端口就不会有数据再打印出来了。

上面的调制解调器图片使用的就是通过 EC20 的 USB 进行生成的，对应的端口是

所以通过 USB 就可以实现 PPP 拨号，对计算器笔记本等进行上网。这个好处就在于当在外面需要给安卓设备，笔记本电脑在没有网络的情况下可以通过借用 4G 开发板对其进行上网拨号，其上网速度很流畅。后面会单独对其进行讲解。

USB 这里主要说到这里。有 USB 用户可以完全不需要通过串口再去控制模块了。可以很轻松的实现通过 USB 去获取自己想要的数据，并可以很好的控制 EC20 模块。移远官方提供了 windows 下的驱动，用户需要使用 USB 需要先安装其驱动。也有 Linux 和安卓下的驱动，这些我们会提供资料，但不会提供讲解和支持。需要用户进行自行开发。

3.2 EC20 建立一路 SOCKT 发数据_TCP 发 GPS 定位数据

此代码是通过 GPRS 网络将 EC20 采集的 GPS/BD 数据进行传输到服务器端进行显示。EC20 是 GPRS/GPS/BD 一体的模块。当然也分版本，如果你购买的版本不支持 GPS 功能，那此代码就不能使用，因为 EC20 过于强大，所以购买模块的时候要注意。此处例程将讲解 EC20 的 GPRS 通讯功能将定位数据发送到服务器端进行显示。

这里对于硬件配置是需要将 EC20 的串口 RXTX 与 STM32F4 的串口 2 的 TXRX 对接。当然就是跳线帽对接好就可以了。如果用户想通过串口 1 打印相关的调试信息，可以接入我们提供的 USB TTL 模块接到串口 1 的地方进行打印。注意 USB 目前还是支持发 AT 指令去控制 EC20 的。但是无法打印串口调试信息，这一点要明确。

下面可以对应看下代码。首先是 GPRS 端的操作，对于 GPRS 的操作流程可以参看我们提供的资料 **Quectel_EC20_TCPIP_AT_Commands_Manual_V1.0**，然后对应着我们的代码进行查看，就会比较清晰的对流程进行了解。

其实 EC20 和 MC20 在一些基本的流程当中有诸多类似，GPRS 大部分的流程都基本相同。只要用户跟着代码走，并按照下面的一些流程说明会很快的对开发进行对应的掌握。

代码内容较多，下面主要对一些关键性代码进行叙述。

（1）通过发送 AT 指令看模块是否返回 OK.来测试模块正常工作这个是第一步，当模块正常返回 OK 之后，那就可以进行下一步操作了。那么之后就需要对 GNSS 模块进行上电，注意虽然 EC20 支持 GPS 功能，但是他的 GPS 是集成在模块内部的，需要单独对 GNSS 部分进行上电才可以。如果重复上电容易返回错误，所以我们给出的例程是可以先判断模块当前是否已经被上电，这样避免重复上电带来指令错误的问题。

printf("AT+QGNSSC=1\r\n");//对 GPS 模块上电

这个语句一定要加上，不然 GPS 没法正常使用。

printf("AT+CPIN?\r\n");//检查 SIM 卡是否在位

这个语句来判断卡是否存在，因为只有卡存在，才可以进行下面的 GPRS 传输，如果只是做 GPS/BD 数据采集，串口打印等功能。是不需要有卡的。卡的存在是主要为了实现远程数据传输来进行准备的。当然打电话发短信的功能卡也是必要的。这个卡是任意的不再受限制，用户用起来也很灵活。

printf(" AT+CGREG?\r\n");//查看是否注册 GPRS 网络

这个是来确认 GPRS 网络是否正常，一般返回是 0，1 或者是 0，5。1 是本地卡，5 是漫游卡。程序判断过程中，需要同步考虑到 2 个返回的问题。

printf("AT+COPS?\r\n");;//查询运营商

当模块正常注网之后，是可以通过指令去查询当前的运营商状态的。运营商是对电信移动联通的返回，用户可以自行查看。

printf("AT+QICLOSE=0\r\n");

推荐用户去关闭一下上一次的连接。这个主要是对调试的时候用的比较多，如果上一次的 Socket 没关闭就重复进行连接。很容易出现报错的现象。这样子关闭的好处保证每次连接只有一次并可靠连接。如果没有连接的情况下是会返回 error 的。所以这里代码将不做判断。只是关闭，不去关心是否关闭出错。

printf("AT+QICSGP=1,1,\042CMNET\042,\042\042,\042\042,0\r\n");//APN 连接

对于手机上网也是一样，都需要告知手机需要连接的 APN,因为移动联通电信的 APN 不尽相同。大家在自己配置 APN 的时候自己根据手机卡的不同自行百度即可。因为是 4G 卡，大家搜索的时候。按照 4G 方式设置即可。

一般移动的 APN 是 **CMNET**, 联通的是 **3GNET**，电信的使用 **CTNET**。如果有一些不清楚的地方，大家可以对照自己的手机查找也可以进行设置。

printf("AT+QIOPEN=1,0,\042TCP\042,\042103.44.145.247\042,28180,0,1\r\n"); /这里就是设置 EC20 连接的服务器端口和 IP 了。其实这个和 MC20 及其相似。只不过多了后面 2 个参数。最后一个参数是设置模块当前是命令模式还是透传模式，1 就是命令。2 就是透传。相对比 MC20 的配置还简单了很多。对于模式切换而言。

AT+QIOPEN 是需要用户输入对应的服务器 IP 和端口号的。这里的 IP 和端口一定是公网的。局域网是不能用的。我们可以通过花生壳软件进行端口映射。或者是自带的公网 IP 来进行操作。**\042 是八进制转义字符“的意思。**可以对应手册当中的 AT 指令说明对应查看就可以了。

printf("AT+QGPSGNMEA=\042RMC\042\r\n");//此语句是打印 NEMA RMC 数据内容。此内容可以参考我们的资料提供的 GPS 相关参数介绍来看。此处打印的是原始数据，对于 GPS 数据是需要进行对应的解析的，才可以显示。如果遇到有的客户拿到原始数据直接在地图上显示，发现有比较大的位置偏移，那就要查看是否将原始数据打印在地图上，而没有做对应的地图转换算法。

采集完成之后，就可以用 **printf("AT+QISEND=0\r\n");**来进行数据打印到服务器端进行显示，因为 EC20 默认是支持多连接的，所以后面的 0 代表的是当前 0 路 sokcet，如果有多个 socket，那后面的数字对应的就是当前的 socket。

因为我们是采用 TCP 连接的，所以可以通过 **printf("AT+QISEND=0,0\r\n");**来查询服务器端是否将数据接收完成。这里需要说明的是有 2 个 0，第一个 0 是当前 socket 后面的 0 代表查询当前剩余的字节数。如果说后面跟的不是 0，而是其他大于 0 的数字，代表的是发送定长数据。这一点一定要注意！

下图是定位地图显示的资料。这个地图软件是我们自行设计的，源码也会开放给客户，采用 VB 编程。软件当中做了对应的算法解析。百度地图存在偏移，所谓的纠偏，如果有的客户测试发现偏移位置很大，请将对应的经纬度数据复制到 Google 地图上进行显示，我们目前没有针对此提出纠偏算法。

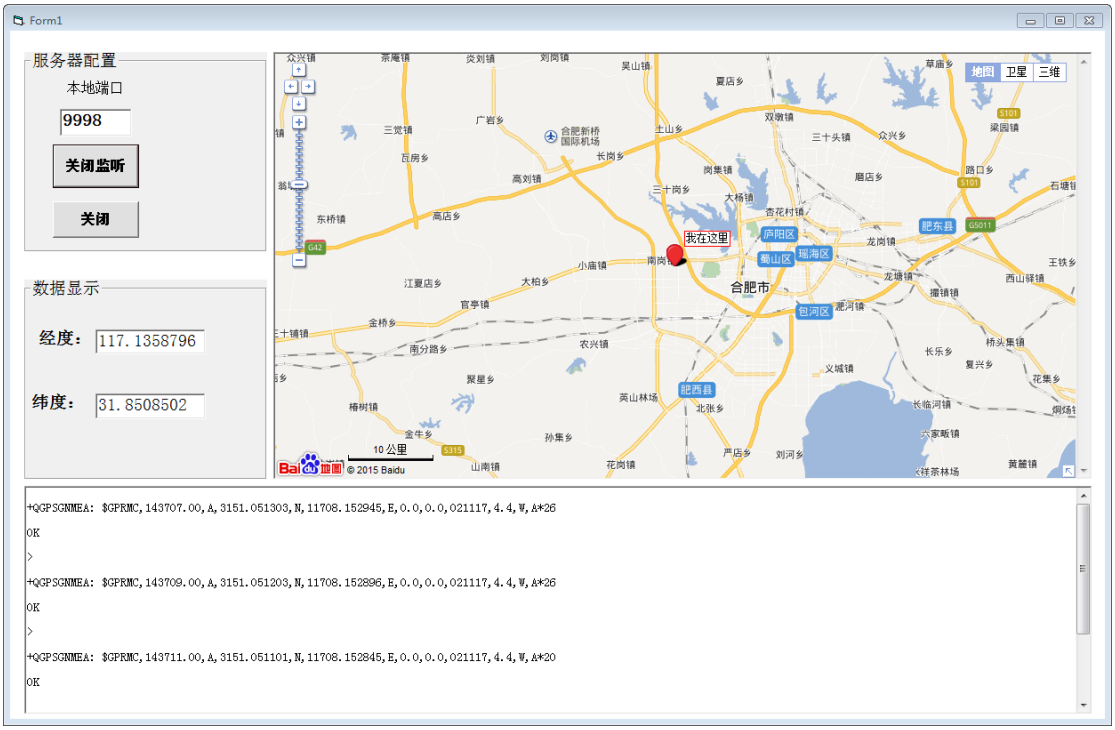


图 3-3 定位地图显示

3.3 EC20 建立多路 SOCKT 发数据_TCP

对于 EC20 而言，功能强大，不像在使用 MC20 的时候，用户经常只用一路 socket 去做 GPRS 数据的传输。EC20 支持 16 路的 socket 的建立，对用户使用编程也比较简单，只要在连接 IP 和端口的地方进行设定即可，选择对应的 Socket 建立即可。对于 TCP 的流程其实上一节有过一些说明，这里主要讲一下多路 Socket 的建立方法。

```
//建立第一组 socket
    printf("AT+QIOPEN=1,0,\042TCP\042,\042103.44.145.247\042,28180,0,1\r\n");// 这里是需要登陆的 IP 号码，采用直接吐出模式
    delay_ms(500);
    strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 0,0");//检查是否登陆成功
    while(strx==NULL)
    {
        strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 0,0");//检查是否登陆成功
        delay_ms(100);
    }
//建立第二组 socket
    printf("AT+QIOPEN=1,1,\042TCP\042,\042114.115.148.172\042,10000,0,1\r\n");// 这里是需要登陆的 IP 号码，采用直接吐出模式
```

```
delay_ms(500);
strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 1,0");//检查是否登陆成功
while(strx==NULL)
{
        strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 1,0");//检查是否登陆成功
        delay_ms(100);
}
```

```
printf("AT+QIOPEN=1,0,\042TCP\042,\042103.44.145.247\042,28180,0,1\r\n");
printf("AT+QIOPEN=1,1,\042TCP\042,\042114.115.148.172\042,10000,0,1\r\n");
```

上面两句就是分别建立 0 和 1 路 socket。其实从程序上很简单，主要就是第二个参数进行下改动。如果是 0 就写 0，如果是 1 就写成 1 即可。然后 IP 和端口号要写成不一样的地址，因为是要连接到 2 处不同位置的服务器，如果写成一样的，其逻辑上就会出现了问题。

```
strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 0,0");//检查是否登陆成功
strx=strstr((const char*)RxBuffer,(const char*)"+QIOPEN: 1,0");//检查是否登陆成功
```

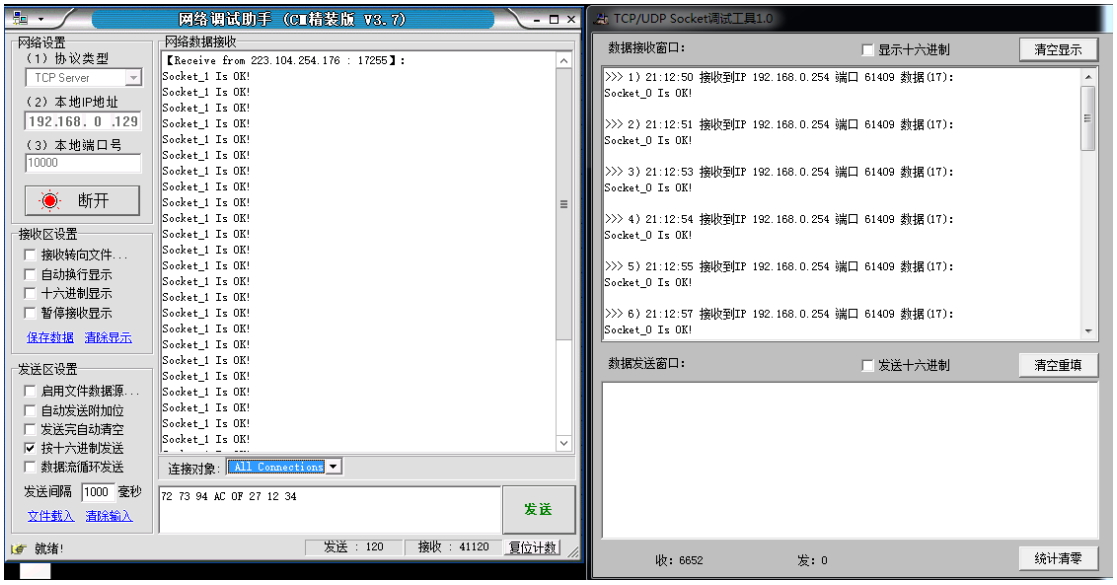
这两句话是对 socket 连接是否正确做一个判断，主要看也是在 QIOPEN 后面的第一个参数不一样，如果是 0 就判断 0，如果是 1 就判断 1，最后一个参数是 0 代表接入成功，如果是非 0 那表示接入是失败的这点需要注意。

```
printf("AT+QISEND=%c\r\n",channel);
```

发送数据方面也有需要注意的地方。首先现在发送数据的 socket 不再固定是 0 了，而是要根据当前的 socket 是几，从而参数写入什么数据才可以。那么这里就写一个变量参数，让用户去进行设置。判断发送是否结束也需要按照参数的模式进行发送了。

```
EC20Send_MultiStrData('0',"Socket_0 Is OK!\r\n");
EC20Send_MultiStrData('1',"Socket_1 Is OK!\r\n");
```

上面两句就是发数据到 2 个不同的服务器端进行数据的显示。这里我们分别在个人电脑以及云服务器。当然个人电脑借助花生壳进行的操作，实现的公网方式。在电脑端分别运行网络调试助手然后进行显示数据。



3.4 GPRS 透传数据例程

透传这里做的是 TCP 透传方式，透传主要针对的是 GPRS 数据传输情况下的操作模式，透传情况下，所有的指令都会当作数据进行传输。

EC20 透传配置较为简单，只需要在配置 IP 和端口提交的模式下选择透传即可。这样连接到服务器之后就可以进行数据透传了。需要将模块先配置为退出透传模式，否则模块一直处于透传模式下，所有的指令将无法进行识别。在配置退出透传情况下，需要注意前后做一定的延迟，否则会出现退出失败的现象。

```
delay_ms(1000);
printf("+++");////退出透传模式，避免模块还处于透传模式中
delay_ms(1000);
```

在退出透传之后，就可以对模块进行对应的配置了，基本的 GPRS 网络配置模式都是一样的，这里需要注意就是配置模块进入透传的地方。

```
printf("AT+QIOPEN=1,0,\042TCP\042,\042103.44.145.247\042,28180,0,2\r\n");//设置为透传模式
delay_ms(500);
strx=strstr((const char*)RxBuffer,(const char*)"CONNECT");//检查是否登陆成功
while(strx==NULL)
{
    strx=strstr((const char*)RxBuffer,(const char*)"CONNECT");//检查是否登陆成功
    delay_ms(100);
}
```

透传只要在连接服务器端的时候，最后一个参数设置为 2 即可。1 代表是命令模式，2 就是透传模式。这样只要模块连接到服务器之后接着发送任何数据都是透传数据了。

透传连接到服务器判断方式是有区别的，采用“CONNECT”来判断，大家在自己写代码的时候，这一点也要注意的。

```
printf("Transparent Access Mode\r\n");//透传
```

这里直接往串口端发送任意数据都可以进行正常的显示了。所以透传对用户而言使用起来非常方便，此时服务器数据收到之后也可以进行数据的下发。

这是透传和指令进行相互切换的服务器端进行的数据显示，只要掌握了透传相关的操作，切换也只要按照+++进行操作即可。

```
///透传模式下接受数据
void EC20Send_RecAccessMode(void)
{
strx=strstr((const char*)RxBuffer,(const char*)"Rec data");//接收到服务器下发数据
                if(strx)
                {
                                printf("Rec data is ok\r\n");//数据发回去
                            delay_ms(500);
                        Clear_Buffer();
          }
      strx=strstr((const char*)RxBuffer,(const char*)"NO CARRER");//服务器主动关闭
                 if(strx)
                 {
                        while(1)
                        {

            }
          }
}
```

透传情况下支持服务器进行数据下发，用户可以在此地方加入自己需要判断的函数

并进行判断是否得到，得到之后就发回给服务器端进行显示，如果服务器端自动关闭了。EC20 会得到响应标志，"NO CARRER"就是服务器端断开了。用户可以在此处加入相关的标记处理做死循环等待看门狗服务重连或者做其他的动作都是可以的。开放给用户进行编程。

3.5 EC20_单片机串口 1 透传 TCP 数据 DTU

本节将给大家介绍关于通过串口 1 下发相关的数据然后给到 EC20，EC20 将数据进行 TCP 透传的方式给到服务器，这个方式就是类似于 DTU 模式的设计思路。

TCP 透传上一节已经说过，这里不做基本的流程做重复。当设备正常连接上服务器之后，此时串口 1 下发对应的数据给到单片机的数组变量里面去。并通过定时器去检测串口 1 数据接收是否结束。如果结束则置位标志接收位为 1。这样串口调试代码，就是希望通过串口调试助手对模块进行调试。我们的板子默认是将模块的串口直接接在了单片机的串口 2 上，所以说直接通过电脑对模块进行调试会较为麻烦一些，所以为了让大家能够体验通过电脑对模块的基本操作，这里提供了一个代码供用户使用。即通过电脑下发数据到串口 1，串口 1 接收到数据之后给单片机通过串口 2 进行数据打印给模块，模块响应到数据之后，会返回对应的状态值。然后把返回的状态值通过串口 1 打印的方式打印给电脑，电脑就可以通过串口调试助手对模块进行操作了。就像是通过串口助手直接调试模块是一个样子了。在代码当中使用定时器来做超时检测，当检测数据 20ms 内没有新数据更新，表明数据接收完成，接收完成之后，将数据发给串口 2 发送。这样的情况下，调试效果与直接接到模块的串口引脚上调试的效果是一样的。注意串口助手的波特率设置为 115200。

```
void USART1_IRQHandler(void)                    //串口 1 中断服务程序
    {
        char Res;
if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)   //接收中断，可以扩展来控制
        {
                Res=USART_ReceiveData(USART1);//接收模块的数据;
                TIM_Cmd(TIM3, ENABLE);
                    Timeout=0;
          RxBuffer1[RxCounter1++] =USART_ReceiveData(USART1);//接收模块的数据

    }
}

void USART2_IRQHandler(void)                    //串口 2 中断服务程序
    {
        char Res;
if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)   //接收模块返回的数据
        {
            RxBuffer[RxCounter++] =USART_ReceiveData(USART2);//接收模块的数据
            if(RxCounter>99)
                RxCounter=0;
                Res=USART_ReceiveData(USART2);//接收模块的数据;
                UART1_send_byte(Res);
```

```
    }

}
```

主要就是注意两个串口中断函数，对两个串口函数进行学习和了解。从而实现这个测试环境的搭建。

图 3-5 串口助手调试模块

3.6 EC20_TCP 透传温湿度 LED 控制交互

EC20_TCP 透传温湿度 LED 控制交互代码的功能是 EC20 与服务器建立连接，通过透传的方式。隔一段时间发送温湿度数据保持长连接。然后利用我们设计的上位机软件可以接收模块的服务器上报的温湿度个数并进行计数显示，同时将温湿度数据在对应的框图中进行对应的显示。服务器端收到对应的数据之后，就可以下发控制指令给到模块，模块收到指令对其数据进行解析，从而识别出 LED 控制状态位对 LED 进行翻转，并将 LED 状态实时的反馈给上位机进行直观的显示。对于此设计而言，主要是需要掌握模块下发数据的格式内容，而后根据内容，解析对应的数据值，从而进行动作。

因为是透传，所以对于程序设计而言，是较为简单轻松的，用户也相对较容易理解。只要模块与服务器建立连接之后，发数据就用 pirntf 函数直接发送，下发的数据直接在串口中断函数当中进行对应接收了。

```
DHT11_Read_Data(&temperature,&humidity);          //读取温湿度值
        senddata[0]='S';
        senddata[1]=':';
        senddata[2]=temperature/10+0x30;
        senddata[3]=temperature%10+0x30;
        senddata[4]=humidity/10+0x30;
        senddata[5]=humidity%10+0x30;
        if(LED0)//关灯
        senddata[6]='0';
        else
        senddata[6]='1';
        if(LED1)//关灯
        senddata[7]='0';
        else
        senddata[7]='1';
        senddata[8]=0;
      printf(senddata);//透传
```

代码在主循环当中不断的检测获取温湿度数据，然后把数据通过 printf 这个函数直接打印到串口缓存进行输出给 EC20 发送到服务器端，服务器端收到数据之后，就可以显示温湿度数据以及状态，模块下发的指令解析如下：

```
///透传模式下接受数据
void EC20Send_RecAccessMode(void)
{
```

```
strx=strstr((const char*)RxBuffer,(const char*)"A1");//对 LED0 开灯
    if(strx)
    {
        LED0=0;
        Clear_Buffer();
 }
strx=strstr((const char*)RxBuffer,(const char*)"A0");//对 LED0 关灯
    if(strx)
    {
        LED0=1;
        Clear_Buffer();
 }
    strx=strstr((const char*)RxBuffer,(const char*)"B1");//对 LED1 开灯
    if(strx)
    {
        LED1=0;
        Clear_Buffer();
 }
strx=strstr((const char*)RxBuffer,(const char*)"B0");//对 LED1 关灯
    if(strx)
    {
        LED1=1;
        Clear_Buffer();
 }

    strx=strstr((const char*)RxBuffer,(const char*)"NO CARRIER");//服务器主动关闭
     if(strx)
     {
          while(1)
          {
              Uart1_SendStr("Server Is Closed!\r\n");//告知用户服务器被关闭
    }
  }


}
```

上面就是判断服务器端下发数据的命令解析了。从代码上来看，是直接对 Rxbuffer 函数进行的数据解析，用户在实际操作的时候，也可以利用仿真的方式对数据查看，可以查看服务器下发的时候，Rxbuffer 所接收到的数据是否是只是数据而没有任何的命令值，方便用户来理解透传。

这个上位机可以用我们提供的“TCP 控制 LED 灯 “来进行实现。

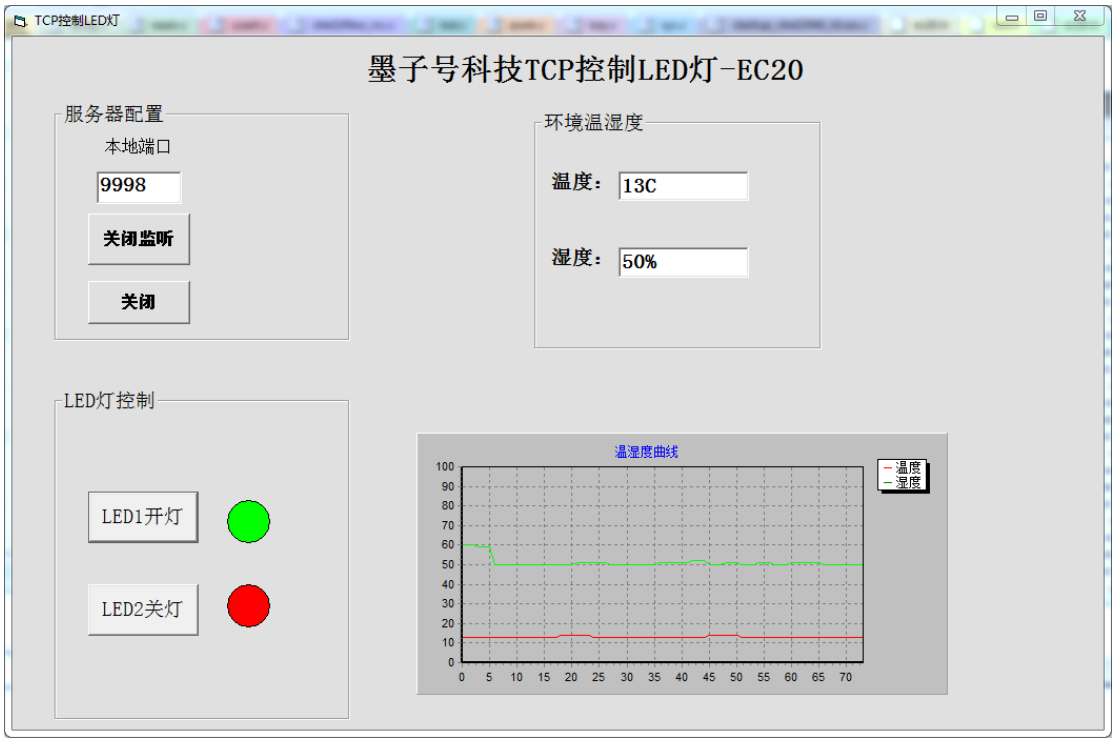


图 3-10 TCP 控制 LED 灯

4.域名介绍

一般而言，服务器端我们采用的是利用申请花生壳的域名来作为服务器端接收数据。如果你自己有服务器，固定独立 IP 那也就不需要申请花生壳的域名了。关于域名的申请，我们会提供一个花生壳申请域名的教程进行介绍，这边就不作多方面介绍。当域名申请成功之后，就可以利用网络调试助手来实现对 TCP 传输的数据进行接收并进行显示，同时也可以利用 TCP 网络进行数据下发来控制单片机。

此外也可以利用相关的可视化界面软件进行编程，实现采集的数据正常显示在自己的服务器软件当中。实现很好的工作。具体的细节会在如下的描述当中进行详细介绍。

4.1 花生壳域名申请

第一步我们首先打开花生壳官网，进入官网进行申请，我们可以在百度先输入“花生壳“，然后弹出如下框：

图 4-1 花生壳申请

我标注的第一个红框里面，进入官网界面，第二个红框是花生壳客户端，只有在电脑端运行花生壳客户端才可以正常的使用花生壳域名解析服务。

图 4-2 花生壳官网

进入花生壳软件之后，我们可以看到很多标题，然后可以进入我箭头标注的端口号位置，点入进去，其实花生壳说白了，他就是个中转站，你把数据发给他的服务器，然后他在把数据发给你电脑端，从而建立了一个模块与个人电脑端的数据传输，对于花生壳而言，他也是有公网 IP 的，所以我们可以把数据直接发到他的 IP 上，然后对应的端口就是我们每个人需

要的，因为只有端口号不一样，那么发过去的数据才可能到达对应的通道，不然每个人 IP 和端口号都是一样，那不是混淆了嘛，所以我们需要先申请一个属于自己的一个端口号，类似 QQ 号的道理。

现在花生壳的端口号已经不再免费，需要 98 元才能买一个了。个人用户推荐买一个 98 块的普通版本端口就可以了。不需要买专业版或者企业版的。

图 4-3 花生壳端口

买完之后，他会需要你注册一个用户名来管理你的端口。因为国家的实名制要求，所以现在注册用户名都要实名制了。如果你不实行实名制，那很可能导致花生壳注册成功，没法使用的情况。

4.2 客户端管理

上面提到了，用户使用的时候，需要下载一个花生壳客户端进行管理，那么安装成功之后，并输入账号和密码之后，会进入下面的界面：

图 4-4 花生壳软件

那此时我们可以点击“**内网穿透**“按钮，进入对应的配置。

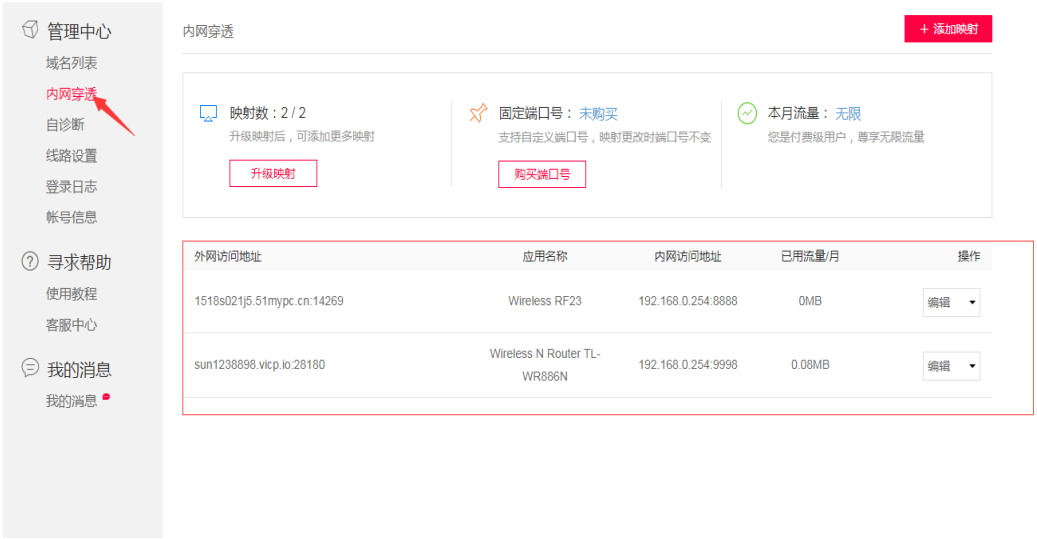


图 4-5 花生壳配置

进入内网穿透之后，就会看到外网访问地址的界面产生，一般而言，购买单个端口的只有一个域名与端口，因为我这个是专业版的，所以有 2 个以上。一般而言，一个就足矣了。

下面开始对内网穿透进行配置，首先进行最右边的编辑按钮进行编辑。

图 4-6 内网映射

这个界面是用来配置内网穿透，就是将外网的数据传到我们个人电脑中。就是内网。内网主机的 IP 号我们可以用 ipconfig 来看，

图 4-7 IP 配置

看到 IPV4 的地址是 192.168.0.254，所以我们内网主机也需要与其设置的一样，并且这个地方建议做静态配置，静态配置如果不熟悉，麻烦百度一下。

然后可以配置内网端口号，内网端口，推荐不要使用默认的 8080，我们可以用超过 10000 号以上的最佳，他最大可以有 65535 的端口号。

然后配置好之后进行提交即可，这样就把花生壳配置好了。然后我们下面就可以用网络调试助手进行配置调试了。

图 4-8 服务器配置

这样就配置 TCP 服务器，然后把花生壳客户端开着，单片机代码里面把 IP 地址和端口改成自己的，这样就可以正常的进行数据传输了。并且也可以把我们提供的上位机软件进行打开，实现对应的数据采集显示了。


网络调试助手 (COM精装版 V3.7)
网络设置
(1) 协议类型
TCP Server
(2) 本地IP地址
192.168. 0 .254
(3) 本地端口号
9998
断开
接收区设置
接收转向文件...
自动换行显示
十六进制显示
暂停接收显示
保存数据 清除显示
发送区设置
启用文件数据源...
自动发送附加位
发送完自动清空
按十六进制发送
数据流循环发送
发送间隔 10000 毫秒
文件载入 清除输入
网络数据接收
OK
>
+QGPSGNMEA:
$GPRMC,141329.00,A,3151.042495,N,11708.151530,E,0.0,0.0,021117,4.4,W,A*2B
OK
>
+QGPSGNMEA:
$GPRMC,141332.00,A,3151.042486,N,11708.151528,E,0.0,0.0,021117,4.4,W,A*2A
OK
>
+QGPSGNMEA:
$GPRMC,141334.00,A,3151.042491,N,11708.151520,E,0.0,0.0,021117,4.4,W,A*22
OK
>
连接对象: All Connections
http://www.cmsoft.cn
发送
就绪!
发送 : 0
接收 : 1287
复位计数

图 4-9 服务器数据显示

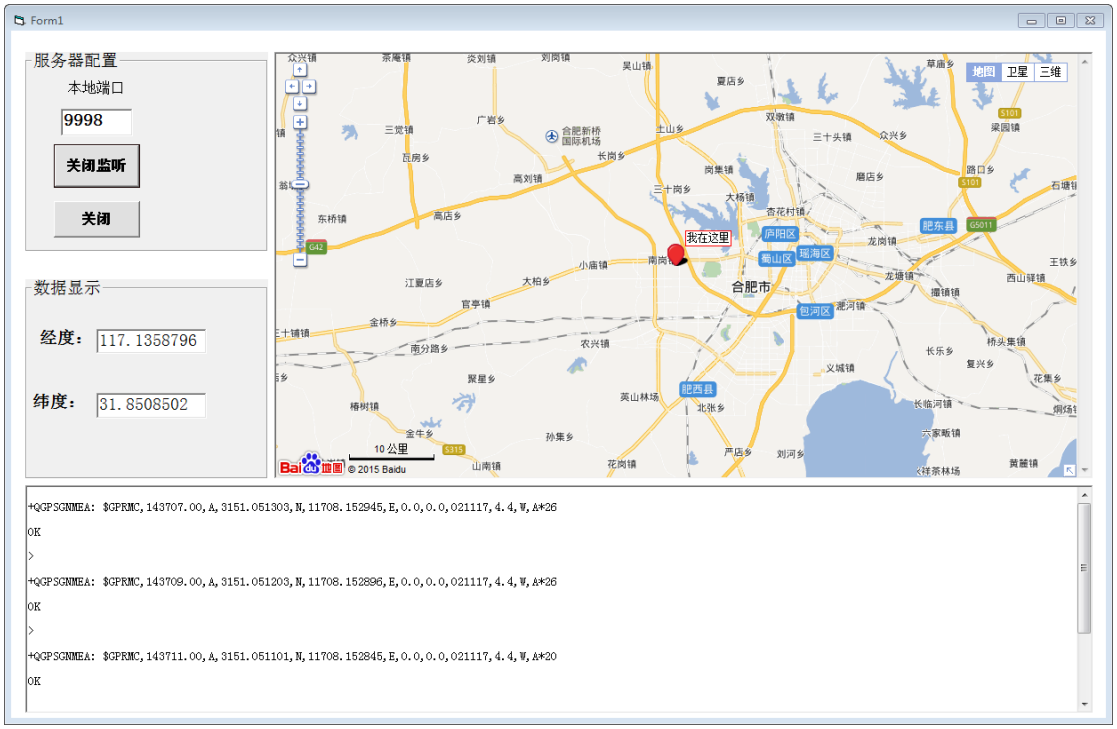


图 4-10 上位机地图显示

5.MQTT 开发介绍

MQTT 应该是很多用户极为关注的也是现在非常热的物联网协议规范。MQTT 网上有很多的名词解释，这里就不做多余的阐述了。对于用户而言，需要掌握的就是 MQTT 协议到底怎么用，应该需要如何把 MQTT 更好的运用到协议规范上来。我们所设计的代码将完全从 MQTT 底层驱动代码来讲解，这样方便用户来学习使用并熟练掌握 MQTT。其 MQTT 的学习推荐大家来看“MQTT-3.1.1-CN”来掌握其底层协议的规范要求，这样对照着代码来看就比较清楚了。对于此章节而言，我们尽量来详细一点的去讲解一下 MQTT，当然这个能力有限，如果有一些讲的不周全或者有错误的地方，还希望大家能够指正相互学习。

5.1 MQTT 有什么内容

对于 MQTT 而言，首先我们需要知道的是 MQTT 其实就是搭建在 TCP 协议上的一个规范。所以用过 TCP 的大家都了解 TCP 有比较严密的握手协议并可以很稳定的进行通讯。所以从而了解到 MQTT 也是如此。对于 TCP 通讯而言，是有客户端以及服务器两个部分组成。

客户端多数说的是我们的 EC20 开发板。而服务器端一般指的是需要有公网 IP。或者利用花生壳等第三方工具实现的公网 IP 的映射。那对于 MQTT 也是一样也需要有服务器。但是

这个服务器运行的软件的标准就是需要支持 MQTT 服务器协议。不能再用我们简单调试用的网络调试助手等被动显示工具了。

一般而言，市面有很多免费的 MQTT 服务器，所以对于用户而言更多需要关心的时候我们的客户端。那么对于客户端而言。需要注意掌握的是 MQTT 主要分为几个部分。

1.TCP 建立连接，用户需要将设备先连接到服务器端，连接方式就是普通的 TCP 方式进行连接即可。

2.当 TCP 建立连接之后，就可以进行 MQTT 登录，对于要使用 MQTT，必须要严格遵守 MQTT 的协议规范，对于客户端而言。需要先登录到服务器端，一般是需要提交登录帐号和密码的。这个根据不同的服务器来决定的，有的要求需要，有的是可以不要，比如登录 ONENET 那就必须要有才可以正常登录。此外为了让登录能够正常就需要给每一个客户端定义一个 ID，这个根据不同的服务器来决定，一般的可以自行进行设定，不做限制。

3.登录成功之后，服务器会返回登录成功的字节标志给到客户端，客户端可以对登录的效果进行检测。如果登录数据无效，服务器会返回“close”给到客户端并自动断开当前的连接，表明当前用户操作上非法。

4.登录成功之后，那么用户就可以进行数据的订阅与发送了。具体的下面会单独详细介绍。发送数据就是客户端将数据发送给服务器。服务器如果收到正确的数据格式的时候，会返回正确的标志给到用户，如果数据格式错误，也是一样会关闭连接。对于订阅而言，就是客户端订阅发送函数发送的主题名一样。这样当客户端发送数据的时候，服务器会自动下发数据给到订阅的客户端进行数据的接收与显示。这样就实现了多个客户端相互通讯的功能，稳定且便利。

5.2 MQTT 服务器登录

对于 MQTT 服务器的登录这一节，要学习的话首先可以先了解一下 MQTT 协议规范要求对于 CONNECT 的格式定义。这里截取一部分协议说明来做讲解。

5.2.1 MQTT 登录请求

3.1 CONNECT – 连接服务端

客户端到服务端的网络连接建立后，客户端发送给服务端的第一个报文**必须**是 CONNECT 报文 [MQTT-3.1.0-1]。

在一个网络连接上，客户端只能发送一次 CONNECT 报文。服务端**必须**将客户端发送的第二个 CONNECT 报文当作协议违规处理并断开客户端的连接 [MQTT-3.1.0-2]。有关错误处理的信息请查看 4.8 节。

有效载荷包含一个或多个编码的字段。包括客户端的唯一标识符，Will 主题，Will 消息，用户名和密码。除了客户端标识之外，其它的字段都是可选的，基于标志位来决定可变报头中是否需要包含这些字段。

3.1.1 固定报头

图例 3.1 –CONNECT 报文的固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 报文类型 (1)				Reserved 保留位			
	0	0	0	1	0	0	0	0
byte 2…	剩余长度值							

上面有描述，客户端连接到服务器端就是采用 TCP 将客户端与服务器建立对应的连接之后，MQTT 第一步要做的就是发送一次 CONNECT 报文，注意此处的说明是只能发送一次，如果多次发送，服务器端就会认为是非法连接，并关闭客户端建立的 TCP 连接。

那么对于 MQTT 发送数据报文的格式他分为固定报头和可变报头。协议中有非常清晰的描述。这里不多介绍。固定报头根据协议规范填入对应的数据值即可。

对于有效载荷而言就是用户提交的数据，是跟在固定报头之后的，那么协议也规范要求需要填入客户端的唯一标识符。这里称之为 ID。这个 ID 协议规范要求每个设备的 ID 唯一。所以对于此处的 ID 一般我们都采用模块的 IMEI，利用 EC20 每个设备不同的 IMEI 来决定就可以保持唯一性了。对于其他的用户名和密码这个根据不同的规范场合需要，可填可不填。但是 ID 是必要的。

3.1.2 可变报头

CONNECT 报文的可变报头按下列次序包含四个字段：协议名（Protocol Name），协议级别（Protocol Level），连接标志（Connect Flags）和保持连接（Keep Alive）。

3.1.2.1 协议名

图例 3.2 -协议名字节构成

	说明	7	6	5	4	3	2	1	0
协议名									
byte 1	长度 MSB (0)	0	0	0	0	0	0	0	0
byte 2	长度 LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'T'	0	1	0	1	0	1	0	0

byte 6	'T'	0	1	0	1	0	1	0	0

可变报头包含了协议名，此处的协议名一定要是 MQTT 才可以，服务器端才可以认为此处的连接是正确协议，否则非法。

此处那么插入一部分协议的底层驱动代码，这样更加方面的让我们来理解 MQTT 的登录。

```
//PROD_ID,SN,id
//构建 MQTT 连接包
u16 mqtt_connect_message(u8 *mqtt_message,char *client_id,char *username,char *password)
{
   u16 client_id_length = strlen(client_id);
   u16 username_length = strlen(username);
   u16 password_length = strlen(password);
   u16 packetLen;
   u16 i,baseIndex;

   packetLen = 12 + 2 + client_id_length;
   if(username_length > 0)
      packetLen = packetLen + 2 + username_length;
   if(password_length > 0)
```

```
            packetLen = packetLen+ 2 + password_length;

        mqtt_message[0] = 16;               //0x10 // MQTT Message Type
CONNECT
        mqtt_message[1] = packetLen - 2;    //剩余长度(不包括固定头部)
        baseIndex = 2;


        if(packetLen > 127)
        {
            mqtt_message[2] = 1;          //packetLen/127;
            baseIndex = 3;
        }
        mqtt_message[baseIndex++] = 0;      // Protocol Name Length MSB
        mqtt_message[baseIndex++] = 4;      // Protocol Name Length LSB
        mqtt_message[baseIndex++] = 77;     // ASCII Code for M
        mqtt_message[baseIndex++] = 81;     // ASCII Code for Q
        mqtt_message[baseIndex++] = 84;     // ASCII Code for T
        mqtt_message[baseIndex++] = 84;     // ASCII Code for T
        mqtt_message[baseIndex++] = 4;      // MQTT Protocol version = 4
        mqtt_message[baseIndex++] = 2;      // conn flags ,要清除掉前面
的连接标志
        mqtt_message[baseIndex++] = 0;      //Keep-alive Time Length MSB
        mqtt_message[baseIndex++] = 120;        // Keep-alive Time Length
LSB    120s
        mqtt_message[baseIndex++] = (0xff00&client_id_length)>>8;//
Client ID length MSB
        mqtt_message[baseIndex++] = 0xff&client_id_length;   // Client
ID length LSB

        // Client ID
        for(i = 0; i < client_id_length; i++)
        {
            mqtt_message[baseIndex + i] = client_id[i];
        }
        baseIndex = baseIndex+client_id_length;

        if(username_length > 0)
        {
            //username
            mqtt_message[baseIndex++] =
(0xff00&username_length)>>8;//username length MSB
            mqtt_message[baseIndex++] = 0xff&username_length;
   //username length LSB
```

```
            for(i = 0; i < username_length ; i++)
            {
                mqtt_message[baseIndex + i] = username[i];
            }
            baseIndex = baseIndex + username_length;
        }

        if(password_length > 0)
        {
            //password
            mqtt_message[baseIndex++] =
(0xff00&password_length)>>8;//password length MSB
            mqtt_message[baseIndex++] = 0xff&password_length;
    //password length LSB
            for(i = 0; i < password_length ; i++)
            {
                mqtt_message[baseIndex + i] = password[i];
            }
            baseIndex += password_length;
        }

        return baseIndex;
    }
```

以上就是 MQTT 登录的底层数据拼接代码。从代码上来看：

```
    mqtt_message[0] = 16;//0x10 // MQTT Message Type CONNECT
   mqtt_message[1] = packetLen - 2;    //剩余长度(不包括固定头部)
```

数据的第 0 位和第 1 位就是固定报文格式，0 位是登录协议类型，1 位是数据剩余长度。

```
        mqtt_message[baseIndex++] = 0;      // Protocol Name Length MSB
        mqtt_message[baseIndex++] = 4;      // Protocol Name Length LSB
        mqtt_message[baseIndex++] = 77;     // ASCII Code for M
        mqtt_message[baseIndex++] = 81;     // ASCII Code for Q
        mqtt_message[baseIndex++] = 84;     // ASCII Code for T
        mqtt_message[baseIndex++] = 84;     // ASCII Code for T
```

这个地方表明是 MQTT 协议，在对应的代码处添加，并告知协议长度。这样整个从代码上来看符合设计需要，然后在对应的地方进行应用即可。

```
    len=mqtt_connect_message(mqtt_msg,"12345678996688","admin","");//
```

构建发送数据包

这里就调用了 CONNECT 函数，那么"12345678996688"他就是设备的 ID 号。后面跟的是用户名和密码。根据需要进行填写，然后调用 TCP 底层发送函数进行数据的发送即可。

5.2.2 MQTT 登录确认连接请求

上面一小节是主要方便用户来学习了 MQTT 登录服务器的数据协议规范并进行发送。那么服务器收到数据之后，会自动对连接的请求进行判断，如果判断的数据是有效的，就会返回如下 3.2 所示，告知用户当前的登录是有效的，否则返回“close”或者其他异常结果。

3.2 CONNACK – 确认连接请求

服务端发送 CONNACK 报文响应从客户端收到的 CONNECT 报文。服务端发送给客户端的第一个报文**必须**是 CONNACK [MQTT-3.2.0-1]。

如果客户端在合理的时间内没有收到服务端的 CONNACK 报文，客户端**应该**关闭网络连接。*合理* 的时间取决于应用的类型和通信基础设施。

3.2.1 固定报头

固定报头的格式见 图例 3.8 – CONNACK 报文固定报头 的描述。

图例 3.8 – CONNACK 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (2)				Reserved 保留位			
	0	0	1	0	0	0	0	0
byte 2	剩余长度 (2)							
	0	0	0	0	0	0	1	0

剩余长度字段

表示可变报头的长度。对于 CONNACK 报文这个值等于 2。

3.2.2 可变报头

可变报头的格式见 图例 3.9 - CONNACK 报文可变报头 的描述。

从结果上来看，数据返回的值 byte1 是 0x20,byte2 是 0x02。然后后面又跟了 2 个可变报头。对应的是反馈结果等。我们在写代码的时候，主要是判断了固定报头，可变报头的数据类型不加以判断。那么对应程序应用而言，只需要判断在发送连接报文之后，返回的数据当中有 0x20,0x02 的时候，表明当前连接是正常了，这样第一步的连接就算顺利完成。

```
if(buffer[0]==0x20&&buffer[1]==0x02)//返回 CONNECT ACK 连接状态
                {
                    printf("Connect ONENET OK!\r\n");
                     LEDStatus=0;//灯点亮
                }
```

5.3 MQTT 数据发布

5.3.1 MQTT 数据发布请求

与 MQTT 服务器建立连接之后，下面就可以进行数据的发送，在 MQTT 里面数据的发送称为发布。类似广播发消息的意思。

那么对于数据的发布也有固定的格式要求，下面是相关协议的截图：

3.3 PUBLISH – 发布消息

PUBLISH 控制报文是指从客户端向服务端或者服务端向客户端传输一个应用消息。

3.3.1 固定报头

图例 3.10 – PUBLISH 报文固定报头描述了固定报头的格式

图例 3.10 – PUBLISH 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (3)				DUP	QoS 等级		RETAIN
	0	0	1	1	X	X	X	X
byte 2	剩余长度							

上面 3.3 描述的就是 Publish 发布消息的协议说明。同样包含固定报头和可变报头，对于 byte1 是报文类型描述以及消息等级，这些用户可以具体去查看协议。那么可以得到的是数据的 byte1 至少是 0x30,所以根据协议要求，在编写代码的时候，要按照此格式来进行编程。发布消息是不需要再重复发 ID，但是需要有发布消息的主题名，这个是为了其他客户端订阅消息而定义的协议。

3.3.2 可变报头

可变报头按顺序包含主题名和报文标识符。

3.3.2.1 主题名

主题名（Topic Name）用于识别有效载荷数据应该被发布到哪一个信息通道。

主题名**必须**是 PUBLISH 报文可变报头的第一个字段。它**必须**是 1.5.3 节定义的 UTF-8 编码的字符串 [MQTT-3.3.2-1]。

PUBLISH 报文中的主题名**不能**包含通配符 [MQTT-3.3.2-2]。

服务端发送给订阅客户端的 PUBLISH 报文的主题名**必须**匹配该订阅的主题过滤器（根据 4.7 节定义的匹配过程）[MQTT-3.3.2-3]。

3.3.2.2 报文标识符

只有当 QoS 等级是 1 或 2 时，报文标识符（Packet Identifier）字段才能出现在 PUBLISH 报文中。2.3.1 节提供了有关报文标识符的更多信息。

3.3.2.3 可变报头非规范示例

图例 3.11 – PUBLISH 报文可变报头非规范示例 举例说明了 表格 3.3 - PUBLISH 报文非规范示例 中简要描述的 PUBLISH 报文的可变报头。

3.3.2 描述的是可变报头，其第一个说明就是主题名，他意思是就是为了识别有效数据载荷应该被发布到哪个信息通道，换言之，客户端可以发布多个不同的主题消息。供其他的客户端进行订阅。此外报文标识符主要是当 Qos 等级为 1，2 的时候，报文标志符才可以用到，他主要用来与服务器端进行数据报文对接的响应的判断，具体用户可以自行了解。

下面结合驱动代码看下内容：

```
    //构建 MQTT_ONENET 发布消息包
    u16 mqtt_onenetpublish_message(u8 *mqtt_message, char * topic, char
* message,u16 len, u8 qos)
    {
       u16 topic_length = strlen(topic);
       u16 message_length;
       u16 i,index=0;
       static u16 id=1;
        u16  datalen;
       message_length =len;
        if(qos)
       mqtt_message[index++] = 0x32;  //0x32  //  MQTT  Message  Type
PUBLISH
        else
       mqtt_message[index++] = 0x30;  //0x30  //  MQTT  Message  Type
PUBLISH
       if(qos)  //16383
        //  mqtt_message[index++]  =  2  +  topic_length  +  2  +
message_length;//加载报文标志符
        datalen= 2 + topic_length + 2 + message_length;//加载报文标志
符
       else
       //mqtt_message[index++] = 2 + topic_length + message_length;   //
Remaining length
        datalen=2 + topic_length + message_length;   // Remaining length
        if(datalen<128)
        mqtt_message[index++]=datalen;
        else

        {

mqtt_message[index++]=(0x80|(datalen-(datalen/128)*128));
               mqtt_message[index++]=datalen/128;
        }
       mqtt_message[index++] = (0xff00&topic_length)>>8;
       mqtt_message[index++] = 0xff&topic_length;

       // Topic
       for(i = 0; i < topic_length; i++)
       {
          mqtt_message[index + i] = topic[i];
       }
       index += topic_length;
```

```
        if(qos)
        {
            mqtt_message[index++] = (0xff00&id)>>8;
            mqtt_message[index++] = 0xff&id;
        id++;//报文标识符可以变进行测试
        }

        // Message
        for(i = 0; i < message_length; i++)
        {
            mqtt_message[index + i] = message[i];
        }
        index += message_length;

        return index;
    }
```

上面是 MQTT 发布消息的底层驱动函数的构造。先看下第一个字节的数据定义。

```
        if(qos)
        mqtt_message[index++] = 0x32;   //0x32  //  MQTT  Message  Type
PUBLISH
         else
        mqtt_message[index++] = 0x30;   //0x30  //  MQTT  Message  Type
PUBLISH
```

这里给出了第一个字节的定义值，这里做了一个消息等级的判断，如果消息等级不为 0 的话，那表明后面的数据内容中会带有消息等级的 id，并置位消息等级的字节位中的地方为 1，那么这里的第一个字节为 0x32,如果为 0 的，表明发布者无需关心发送的消息的等级，第一个字节就为 0x30.那么此时的消息 id 就不需要编码进来。

```
        if(datalen<128)
         mqtt_message[index++]=datalen;
         else
         {
            mqtt_message[index++]=(0x80|(datalen-(datalen/128)*128));
          mqtt_message[index++]=datalen/128;
          }
```

这个函数需要注重介绍一下，因为 MQTT 发布消息数据内容一次性可以发超过 256 个字节，但是表示的剩余长度如果只按照 1 个字节来表示，那么超过 256 个字节之后，数据发送就会出现错误。所以当数据内容过长的时候，就不能按照常规的做法进行长度填充。那么对于此 MQTT 也给出了对应的解决方法。

表格 2.4 剩余长度字段的大小

字节数	最小值	最大值
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

分别表示（每个字节的低 7 位用于编码数据，最高位是标志位）：

1 个字节时，从 0(0x00)到 127(0x7f)

2 个字节时，从 128(0x80,0x01)到 16383(0Xff,0x7f)

3 个字节时，从 16384(0x80,0x80,0x01)到 2097151(0xFF,0xFF,0x7F)

4 个字节时，从 2097152(0x80,0x80,0x80,0x01)到 268435455(0xFF,0xFF,0xFF,0x7F)

从表格 2.4 中可以看到,MQTT 规范上表明剩余长度的字段大小最大可以占用 4 个字节。最长的长度也可以达到很大。实际我们应用的时候，可能不会那么大，但是一般会在 128 个到 16383 之间的长度。所以在设计代码的时候，我们可以判断如果数据长度小于等于 127，就用一个字节来代替。如果是大于 127 就用 2 个字节来代替。上面也给出了 2 个字节的定义规范说明。举例而言，当是 128 个字节的时候，第一个字节应该是 0x80,第二个字节是 0x01.根据这样的规律，设定的代码如下，（仅供参考，或许存在多个方法，只是这样的方法我们验证下来可行）。

```
mqtt_message[index++]=(0x80|(datalen-(datalen/128)*128));
mqtt_message[index++]=datalen/128;
```

有兴趣的用户可以自行去摸索此规律，这里不做深入的展开说明。

5.3.2 MQTT 数据发布确认请求

同登录一样，发布消息到服务器端之后，服务器都会对每一组报文进行识别并确认是否合法，如果是合法将返回相关的 ACK 给到客户端进行确认。

3.4 PUBACK –发布确认

PUBACK 报文是对 QoS 1 等级的 PUBLISH 报文的响应。

3.4.1 固定报头

图例 3.12 - PUBACK 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (4)				保留位			
	0	1	0	0	0	0	0	0

byte 2	剩余长度(2)							
	0	0	0	0	0	0	1	0

剩余长度字段

表示可变报头的长度。对 PUBACK 报文这个值等于 2.

3.4.2 可变报头

包含等待确认的 PUBLISH 报文的报文标识符。

图例 3.13 – PUBACK 报文可变报头

Bit	7	6	5	4	3	2	1	0
byte 1	报文标识符 MSB							
byte 2	报文标识符 LSB							

对于发布确认服务器返回的内容也是极为简单的，他主要是包含了固定报头和可变报头，对于固定报头的 2 个字节数据都是固定的，分别是 byte1 0x40,byte2 0x02;用户需要判断是否返回此两个字节，对于可变报头根据 Qos 等级来确认是否需要加以判断与识别。

```
if(buffer[0]==0x40&&buffer[1]==0x02)//返回 PUB ACK 确认状态
                    {
                        printf("REC OK,PUB NO ACK !\r\n");
                       LEDStatus=!LEDStatus;//灯闪烁
                    }
```

5.4 MQTT 订阅数据

5.4.1 MQTT 订阅数据请求

MQTT 他是一个通讯协议，除了支持往服务器发数据之外，也支持客户端设备从服务器端订阅数据。当然每个客户端都支持数据的发送和订阅。或者只作为单独的发送或者单独的订阅都可以。

订阅数据也是一样，需要先登录 MQTT 服务器，等服务器确认 OK 之后，就执行订阅。在订阅数据的同时，需要先上传订阅的内容主题。就是你需要订阅哪一个名称下面的数据。提交之后，服务器会返回订阅成功的标志给客户端。后面客户端只需要进行数据等待。当服务器端有其他客户端上传订阅主题的时候。服务器自动会将数据下发给对应订阅此主题的客户端，此时订阅客户端只需要去对数据进行解析，实现自己的数据需要即可。

那么如果只是订阅数据包的话，在没有数据下发的时候，需要保持一定的心跳包的发送。否则长时间不上传数据，服务器就会断开设备的连接。

3.8 SUBSCRIBE - 订阅主题

客户端向服务端发送 SUBSCRIBE 报文用于创建一个或多个订阅。每个订阅注册客户端关心的一个或多个主题。为了将应用消息转发给与那些订阅匹配的主题，服务端发送 PUBLISH 报文给客户端。SUBSCRIBE 报文也（为每个订阅）指定了最大的 QoS 等级，服务端根据这个发送应用消息给客户端。

3.8.1 固定报头

图例 3.20 – SUBSCRIBE 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (8)				保留位			
	1	0	0	0	0	0	1	0
byte 2	剩余长度							

SUBSCRIBE 控制报固定报头的第 3,2,1,0 位是保留位，**必须**分别设置为 0,0,1,0。服务端**必须**将其它的任何值都当做是不合法的并关闭网络连接 [MQTT-3.8.1-1]。

剩余长度字段

等于可变报头的长度（2 字节）加上有效载荷的长度。

3.8 就是对订阅主题的相关描述，客户端是可以实现从服务器端创建报文订阅，当然报文订阅可以不限制于仅有一个，可以多个报文订阅。这样服务器端就将订阅的主题收到数据之后直接发布给客户端。从而实现通讯。

3.8.2 可变报头

可变报头包含客户端标识符。2.3.1 提供了有关报文标识符的更多信息。

3.8.2.1 可变报头非规范示例

图例 3.21 - 报文标识符等于 10 的可变报头，非规范示例 展示了报文标识符设置为 10 时的可变报头。

图例 3.21 – 报文标识符等于 10 的可变报头，非规范示例

	描述	7	6	5	4	3	2	1	0
报文标识符									
byte 1	报文标识符 MSB (0)	0	0	0	0	0	0	0	0
byte 2	报文标识符 LSB (10)	0	0	0	0	1	0	1	0

3.8.3 有效载荷

SUBSCRIBE 报文的有效载荷包含了一个主题过滤器列表，它们表示客户端想要订阅的主题。SUBSCRIBE 报文有效载荷中的主题过滤器列表**必须**是 1.5.3 节定义的 UTF-8 字符串 [MQTT-3.8.3-1]。服务端**应该**支持包含通配符（4.7.1 节定义的）的主题过滤器。如果服务端选择不支持包含通配符的主题过滤器，**必须**拒绝任何包含通配符过滤器的订阅请求 [MQTT-3.8.3-2]。每一个过滤器后面跟着一个字节，这个字节被叫做 服务质量要求（Requested QoS）。它给出了服务端向客户端发送应用消息所允许的最大 QoS 等级。

3.8.2 是关于可变报头的相关介绍，对于用户而言，需要在有效载荷当然填写一个主题名称。这样的主题名称就是用户需要订阅的主题。对于 Qos 是服务质量要求，用户可以自行进行研究。

```
//构建 MQTT 订阅请求/取消订阅包
//whether=1,订阅; whether=0,取消订阅
u16 mqtt_subscribe_message(u8 *mqtt_message,char *topic,u8 qos,u8 whether)
{
    u16 topic_len = strlen(topic);
    u16 i,index = 0;
    static u16 id=3;
    if(whether)
        mqtt_message[index++] = 0x82;                  //0x82  // 消 息 类 型 和 标 志 SUBSCRIBE 订阅
    else
        mqtt_message[index++] = 0xA2;                  //0xA2 取消订阅
    mqtt_message[index++] = topic_len + 5;             //剩余长度(不包括固定头部)
    mqtt_message[index++] = (0xff00&id)>>8;            //消息标识符
    mqtt_message[index++] = 0xff&id;                   //消息标识符
    mqtt_message[index++] = (0xff00&topic_len)>>8;     //主题长度(高位在前,低位在后)
    mqtt_message[index++] = 0xff&topic_len;            //主题长度

    for (i = 0;i < topic_len; i++)
    {
        mqtt_message[index + i] = topic[i];
```

```
        }
        index += topic_len;

        if(whether)
        {
            mqtt_message[index] = qos;//QoS 级别
            index++;
        }

        return index;
    }
```

上面是关于数据包订阅的驱动代码。
第一个字节是数据标志位，可以看下：

```
        if(whether)
            mqtt_message[index++] = 0x82;    //0x82 //消息类型和标志 SUBSCRIBE 订阅
        else
            mqtt_message[index++] = 0xA2;                    //0xA2 取消订阅
```

从刚才的 3.8 当中可以看到。对于订阅请求的默认值应该是 0x82。同样，也支持用户对已经订阅的主题采取取消订阅。这样再有主题数据包的时候，服务器就不会下发给到用户了。

3.10 UNSUBSCRIBE –取消订阅

客户端发送 UNSUBSCRIBE 报文给服务端，用于取消订阅主题。

3.10.1 固定报头

图例 3.28 – UNSUBSCRIBE 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (10)				保留位			
	1	0	1	0	0	0	1	0
byte 2	剩余长度							

UNSUBSCRIBE 报文固定报头的第 3,2,1,0 位是保留位且**必须**分别设置为 0,0,1,0。服务端**必须**认为任何其它的值都是不合法的并关闭网络连接 [MQTT-3.10.1-1]。

3.10.1 就是关于取消订阅报文的固定报头，从数据上来看第一个数据值就是0xA2.所以对应的代码上就是正确的。

5.4.2 MQTT 订阅数据请求确认

客户端订阅成功之后，服务器会下发消息给到客户端，告知用户当前订阅正确。如果说订阅就会收到 error 等内容。用户在订阅数据包的时候，要严格按照协议规范来进行数据的订阅。

3.9 SUBACK – 订阅确认

服务端发送 SUBACK 报文给客户端，用于确认它已收到并且正在处理 SUBSCRIBE 报文。

SUBACK 报文包含一个返回码清单，它们指定了 SUBSCRIBE 请求的每个订阅被授予的最大 QoS 等级。

3.9.1 固定报头

图例 3.24 – SUBACK 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (9)				保留位			
	1	0	0	1	0	0	0	0
byte 2	剩余长度							

剩余长度字段

等于可变报头的长度加上有效载荷的长度。

3.9 给出了服务器端发送 SUBACK 订阅确认给到客户端，用来表明服务器端收到了订阅请求并会对此报文进行相关的处理。首先也是固定数据头，第一位数据值是 0xA0。第二位跟的是剩余长度，此长度不是固定的，他是由下面的可变报头决定的。对照可变报头来看内容。

3.9.2 可变报头

可变报头包含等待确认的 SUBSCRIBE 报文的报文标识符。图例 3.25 - SUBACK 报文可变报头 描述了可变报头的格式。

图例 3.25 – SUBACK 报文可变报头

Bit	7	6	5	4	3	2	1	0
byte 1	报文标识符 MSB							
byte 2	报文标识符 LSB							

3.9.3 有效载荷

有效载荷包含一个返回码清单。每个返回码对应等待确认的 SUBSCRIBE 报文中的一个主题过滤器。返回码的顺序**必须**和 SUBSCRIBE 报文中主题过滤器的顺序相同 [MQTT-3.9.3-1]。

图例 3.26 – SUBACK 报文有效载荷格式 描述了有效载荷中单字节编码的返回码字段。

图例 3.26 – SUBACK 报文有效载荷格式

Bit	7	6	5	4	3	2	1	0
	返回码							
byte 1	X	0	0	0	0	0	X	X

3.9.2 是可变报头的说明。他分为报文标志符和有效载荷。有效载荷当中跟的就是上报的主题名称。所以这个根据用户自己的需要来判断服务器下发的是否是自己上报订阅的主题名称。

5.4.3 MQTT 心跳包

3.12 PINGREQ – 心跳请求

客户端发送 PINGREQ 报文给服务端的。用于：

1. 在没有任何其它控制报文从客户端发给服务的时，告知服务端客户端还活着。
2. 请求服务端发送 响应确认它还活着。
3. 使用网络以确认网络连接没有断开。

保持连接（Keep Alive）处理中用到这个报文，详细信息请查看 3.1.2.10 节。

3.12.1 固定报头

图例 3.33 – PINGREQ 报文固定报头

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT 控制报文类型 (12)				保留位			

	1	1	0	0	0	0	0	0
byte 2	剩余长度 (0)							
	0	0	0	0	0	0	0	0

从上面的协议当中文字描述来看,MQTT 的心跳包无非就是为了维持与服务器端之间的连接。这个主要是为了防止一些客户端只订阅数据，一直等待数据。而长时间的不进行数据交互而出现连接断开的问题。

对于 PING 包数据很简单，只是由固定报头组成，没有可变报头以及数据载荷内容。只有 2 个字节，分别是 0xC0,0x00,0x00 虽然是 0 但是一定要发送。否则数据位不够，心跳包也是非法了。

```
//构建 MQTT PING 请求包
u8 mqtt_ping_message(u8 *mqtt_message)
{
    mqtt_message[0] = 0xC0;     //0xC0 //消息类型和标志 PING
    mqtt_message[1] = 0x00;     //剩余长度(不包括固定头部)

    return 2;
}
```

这个是 PING 包的数据内容底层驱动代码，用户可进行参考。

对于 MQTT 的协议大致上基础的地方上面都做了相关的描述，如果用户需要更加复杂或者其他更加细致的内容，推荐大家可以看 MQTT 的官方协议，其中有很详细的描述了每个报文的数据格式，以及处理的机制方法。这样有助于大家的理解。

6.状态机设计讲解

本设计代码当中重点讲解一下状态机，实际上我们编写的代码没用到操作系统，基本采用循环的方式来进行代码的演示，相对而言代码的执行效率就相对较低，无非更多的是采用中断的方式来触发一些数据中断内容数据。对于一些简单的应用是可以正常用的，如果是较为复杂的应用，执行效率低并且也容易出错。所以在这里我们引入了状态机的概念来设计。

6.1 状态机设计架构

状态机的设计思路其实类似于通常用的操作系统，他也是利用时间片的轮询方式来在不同的时间下应用一些功能。类似于操作系统分配不同的任务并进行同步运行。状态机相对操作系统简单一些，他是利用了一个定时器定时中断比如 100ms 中断一次来分配一个任务，就是说分配一个任务在 100ms 内进行完成。然后下一个 100ms 执行另一个任务，但是缺陷是如果说任务量过多，那么使得每一个任务再次被循环使用的时间就会更久一些。所以这一点也是状态机的一个缺陷。当然如果对于一些相对任务量少并希望简洁的处理一些机制问题，使用状态机将非常的轻松并且可靠。

那么状态机设计的思路是利用一个变量比如 timervalue 不断的在定时器中断下面累加。然后在特定的某个值的时候下面执行相关的任务，从而实现切换。

```
switch(timervalue)
{
   case 0://任务分配 1
break;
   case 1://任务分配 2
break;
   case 2:/任务分配 3
break;

   case 3://任务分配 4
break;
....
//等等其他任务

}
```

上面这一段话就是状态机的核心思路了。其实比较简单，这个就是通常遇到的 switch case 语句的方式，不过是按照这种思路来进行展现的。

有了架构之后，用户就可以在不同的取值条件下安排对应的任务了。比如第一个我们可

以采集温湿度，第二个任务做 CAN 数据传输，第三个任务做 485 发送温湿度数据。第四个做 GPS 远程发送等等。这些任务可以做到相互不干涉，并且可以快速的实现响应。

6.2 MQTT 移植到状态机

对于 MQTT 使用状态机，其实思路上就要根据任务的需求来做相关的定义。比如说通常遇到的 MQTT 三步骤，登录连接，发布消息，订阅消息。那么这三个动作可以理解为三个不同的任务。然后分别嵌入到状态机代码里面去。

根据这样的思路，将 MQTT 的动作添加到任务变量里面去。

```
switch(timervalue)
{
   case 0://任务分配 1
   len=mqtt_connect_message(mqtt_msg,GSMinit.IMEI,"","");//mqtt 连接
  break;
   case 1://任务分配 2

len=mqtt_onenetpublish_message(mqtt_msg,GSMinit.pubtopic,t_payload,json_len,1);// 构建发布
   break;
   case 2:/任务分配 3
   len=mqtt_subscribe_message(mqtt_msg,GSMinit.subtopic,0,1);//mqtt 构建订阅
   break;

   case 3://任务分配 4
   break;
   ….
   //等等其他任务

   }
```

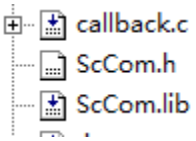
这样就一目了然。根据三个不同任务，分别配置三个任务量。当 MQTT 正常登陆连接之后，发布数据和定语消息是分别独立的，不会影响到发送数据的时候影响数据的订阅。订阅不会影响到消息的发布。这个就可以很好的进行独立运行起来并在有问题的时候可以正常查找。

6.3 状态机的串口数据解析

任意一个函数架构都是离不开对串口数据的解析，这一点毋庸置疑。因为单片机通过与 EC20 数据的交互来获取到相关的信息，所以对于串口数据的处理至关重要。其实串口很简单，无非就是串口中断将所有接收到的数据保存在一个数组变量中然后供用户进行相关的数据解析并对应的处理。

对应状态机而言，我们考虑到用户处理数据的方法（推荐我们的专利代码）提供我们的专有测试方法获取数据解析，用户就不再需要关心数据什么时候来，该如何处理，所有的串口数据都交由我们的代码来进行处理，用户只需要去关心需要什么数据即可。

下面对照说明下：



callback.c 是提供给用户获取对应的串口有效数据进行解析的。ScCom.h 是一些函数定义申明，ScCom.lib 这个是相关的串口数据解析的并提交给 callback，这个函数里面的内容涉及到专利问题，所以不会对用户进行开放。

```
const char ATCMD_Table[] ="$+CPIN: READY$+CGREG: 0,1$+CGREG: 0,5$+QIACT@+QIOPEN: 0,0$+QGPSGNMEA:@>$+QISEND:@closed@+QIURC:$order";
//$代表有\r\n,@是没有\r\n 返回的
```

注意看 ATCMD_Table 这个数组，里面有很多我们熟悉的字符串，比如判断卡是否存在，当前网络注册状态，服务器连接状态等。那么这些是什么意思呢，就是告知用户输入你需要知道的内容，我们的 ScCom.lib 自动会帮你进行解析，当解析到对应的数据之后，会自动将数据结果返回给 callback 函数当中，让用户进行提取。对用户而言，只需要输入想要的内容即可。不再去关心自己怎么去解析，非常的方便。

那么这里前面的数据内容有$和@,这个主要是为了分辨返回的指令和数据当中是否含有\r\n 或者是返回的数据当中不含有\r\n。这样是方便分辨数据的格式。比如**$+CPIN: READY 返回一定是有\r\n 的。但是@+QIURC:TCP 下发数据的响应就不含有\r\n,用户在写代码的时候一定要注意这一点防止写错，得到不同的内容。**

那么还有一点需要注意的时候，每一个$或者@都作为每一个字符串的分割，那么在字符串累计计算的时候，是按照 0 开始计算，所以遇到第一个$那么后面的数据内容就是第 0 位，后面是第一位。依次类推下去。这个很重要，因为在 callback 函数当中会用到。

```
**********
用户需要在 switch 函数里面加入自己需要处理的相关数据信息
ATCMD_Table 是按照$或者@分割数据间隙的，第一个$位置为 0，第二个$为 1，第三个是@为 2，以此类推各个指令所存放的位置就定义到下面的
case x 即可。
**********/
int COM_Get_Callback(unsigned char cmd, unsigned char *buf, unsigned int len)
{

  char *strx,*p;
  unsigned char i,j,k;
  unsigned char untildata;
  if(len > 100)len = 100;
  memcpy(timebuf, buf, len);   //这里是打印上传上来的数据值，供用户查看
  switch(cmd)
        {
        case 0:       //获取到手机卡
          GSMinit.status=1;//status 是我们自己进行定义的，0-7 代表的是登录服务器的，0XFF 是代表等待服务器登录返回 OK,0X0A 代表是可以发送 TCP 数据，0X0B 代表数据发送完
```

成,0X0C 代表是数据采集完成，

//可以准备再次发送数据，具体的可以根据代码每一步的位置进行对应查看即可。0X0F 代表是 SEND 数据之后，进行 QISACK 查询

```
//          LED1=0;
  break;
case 1://本地卡注网
   GSMinit.status=2;

      LED1=0;
  break;
case 2://漫游卡注网

   GSMinit.status=2;
  break;
 case 3://有 IP 存在
   GSMinit.status=4;
  break;
  case 4://登录成功 IP
   GSMinit.status=0x0B;
  break;
  case 5://获取到 GPS 定位数据
  Get_GPSDATA(timebuf);
  Getdata_Change(latdata.getstautus);
  GSMinit.datastatus=2;//计数器后移动。准备采集下一个数据
  GSMinit.status=0x0C;//采集完成
      break;
  case 6://获取到发送数据使能>
        GSMinit.enable=1;
  break;
  case 7:
    strx=strstr((char*)buf,(char*)",");//判断第一个， 号
    strx=strstr((char*)(strx+1),(char*)",");//判断第二个逗号
    untildata=*(strx+1)-0x30;
    if(untildata==0)
    {
                              GSMinit.status=0x09;//
                              GSMinit.datastatus=0;
    }
      break;
    case 8://主机自动掉线
        while(1)
        {}
      case 9://模块的数据返回
      memcpy(timebuf, RxBufferDMA, 100);  //这里是打印上传上来的数据
```

值，供用户查看

```
                    MqttSample_RecvPkt(timebuf);
                //
                    break;

                case 10://订阅请求数据返回
                    MqttSample_SubRecvPkt(timebuf);
                    break;

            }
    return 1;

}
```

上面是截取了 callback.c 文件当中的 COM_Get_Callback 函数，这个函数的存在意义就是扫描程序前面配置的相关接收指令获取。这里也是按照 switch case 语句的方式来进行的定义设计。

```
switch(cmd)//
{
  case 0://CPIN READY
break;
  case 1
break;
.....
}
```

cmd 就是对应前面数组定义的相关指令的位数，就是每一个前面定义的指令所对应的位置，从 0 开始的，这里就是将接收到的指令在这里做解析。

比如我们第 0 位定义的是$+CPIN: READY。当我们程序发送：

```
printf("AT+CPIN?\r\n"); //获取卡的状态
```

那么模块此时会返回对应的状态值给到单片机，是存在卡还是不存在卡，如果存在卡程序就会自动跳入到这个 callback 函数判断命令里面的第 0 位来。那就表明此时卡是存在的，用户就可以做下一步操作，如果哪一步有一些异常都可以从此处看到。或者自己也可以加载一些错误指令的返回等等做仿真调试极大的方便用户测试。

```
switch(GSMinit.status)//判断当前初始化状态
                    {
                        case 0:   printf_CPIN();break;//获取手机卡
                        case 1:   printf_CGREG();break;//查看网络注册状态
                        case 2:   printf_QIACT();break;//激活
                        case 3:   printf_GetQIACT();break;//查询 IP 值
                        case 4:   printf_qiopen();break;//配置连接服务器的 IP
                    }
```

从上面的 switch 函数当中可以看到相关的备注，程序流程按照先判断卡是否存在然后去查询网络注册状态激活场景以及判断是否激活成功获取 IP 并连接到服务器端。这一套流程都是和 callback 函数紧密连接起来。并且用户在操作 EC20 相关注网连接的情况下依然还

可以去做一些其他的工作，相对而言会非常的有效率。

此外为了让设备运行的更有效率，单片机的串口采用 DMA 的方式进行的接收。这样就可以释放掉外部中断而将更多的 CPU 效率集中到其他任务量上。

6.4 任务量编制

以上主要讲解 MQTT 移植到状态机端的使用教程和方法并介绍了，下面主要就是相关的任务量在主函数当中进行相关的编制了。

```
while(1)
{

    if (P1msReq)//10ms 扫描一次
    {

        P1msReq = 0;
       Per1msTask();
        IWDG_Feed();//喂狗
    }
}
```

这个是基本的主函数程序架构，在一个主函数当中。利用 P1msReq 在定时器的不断置位，并在每次判断之后进行清零。这里就保持间隔多久时间进入到 Per1msTask();函数中来。Per1msTask 函数里面就将运行所有的的任务进行配置。

```
void    Per1msTask(void)
{
    static u8 P10msReq = 0;

     if (P10msReq < 10) P10msReq++;//100ms 一次事件
    else P10msReq = 0;
      switch(P10msReq)
            {
                case 0:
                    switch(GSMinit.status)//判断当前初始化状态
                    {
                        case 0:   printf_CPIN();break;//获取手机卡
                        case 1:   printf_CGREG();break;//查看网络注册状态
                        case 2:   printf_QIACT();break;//激活
                        case 3:   printf_GetQIACT();break;//查询 IP 值
                        case  4:   printf_qiopen();break;//配置连接服务器的 IP 和端
口号
                    }
                  break;
                case 1:
                            SCOM_RecieveAT(&m_com);//做数据不断扫描机制
                    break;
```

```
                case 2:
                        if(GSMinit.status==0x0b)//0X0B 代表数据 TCP 数据已经发送完成了，此时可以进行数据采集等其他事宜的工作了。在发送 TCP 数据的时候，建议不要操作其他动作
                        {

len=mqtt_connect_message(mqtt_msg,GSMinit.IMEI,"","");//构建连接数据包 ID 号使用每个模块的 IMEI 来代替
                            GSMinit.status=0x09;
                        }

                        switch(GSMinit.datastatus)//这个地方就是可以根据用户需要从设备获取数据。等所有采集完成之后，再进行数据发送非常方便
                                {
                                    case 1:
                                      printf_GetGPS();//发送获取 GPS 数据
GSMinit.sendcount=0;
                                    break;
                                 }



            break;

        case 3:
                if(GSMinit.status==0x0a)//可以发数据了
                        {

  //printf_sendata(gspdata.senddata);          //发送数据值
                            printf_MQTTsendata(mqtt_msg,len);
                        }
                    if(GSMinit.status==0x0C)  //数据采集完成，准备进行下一次数据发送，
                        GSMinit.status=0x09;//置位发送准备
                            if(GSMinit.status==0x09)//发送命令
                            {
                                //    printf_sendstr();//发送数据命令
                               printf_MQTTsendstrlen(len);

                            }

            break;
           case 4://MQTT 数据的发布
```

```
                    if(MQTT_status.status==0x01)//登录成功
                    {

len=mqtt_subscribe_message(mqtt_msg,GSMinit.subtopic,0,1);//构建 MQTT 订阅数据包
                        GSMinit.status=0x09;
                        MQTT_status.status=0x00;
                    }
                    if(MQTT_status.status==0x02)//订阅正常，准备数据发送
                    {

json_len=MqttOnenet_Savedata(t_payload,latdata.gpsdata[0],latdata.gpsdata[1],DHT11_DATA.temp,DHT11_DATA.humi);//提交 JSON 数据

len=mqtt_onenetpublish_message(mqtt_msg,GSMinit.pubtopic,t_payload,json_len,1);// 构 建 MQTT 发送数据包 stract("app/",GSMinit.IMEI);
                        GSMinit.datastatus=1;//准备采集 GPS 数据
                        MQTT_status.status=0x00;
                    }
                    if(MQTT_status.status==0x03)
                    {

                        MQTT_status.timecount++;
                    }
                    if(MQTT_status.timecount>=10)
                    {
                        MQTT_status.timecount=0;
                        //
                        MQTT_status.status=0x02;//继续进行数据发布
                     }
                break;
                 case 5://温湿度数据的采集
                DHT11_DATA.calcount++;
                 if(DHT11_DATA.calcount>5)
                 {

DHT11_Read_Data(&DHT11_DATA.temp,&DHT11_DATA.humi);//采集温湿度数据
                    DHT11_DATA.calcount=0;
                 }

                    break;


                case 6://对获取到的数据进行解析.
                    if(GSMinit.status==2)
```

```
                                Uart1_SendStr("123\r\n");
                            break;

                        default:

                            break;
                    }


    }
```

上面的代码是“EC20 状态机代码 TCP 传 GPS 数据 - APPMQTTIMEI 定义 3.7”这个文件名下的代码。那么下面的任务分配主要分为几个流程。

（1） EC20 连接服务器任务

（2） SCOM_RecieveAT(&m_com);//做数据不断扫描机制，串口接收数据扫描机制判断任务

（3） MQTT 登录连接服务器任务

（4） MQTT 发布消息任务

（5） MQTT 订阅数据任务

（6） EC20 获取 GPS 任务

（7） 等等可添加任务

从上面的描述来对照代码看将一目了然，非常的清晰可见。用户可以慢慢去理解状态机并配合串口配置代码来实现自己的应用。此代码操作上简单明了，非常适合用户做产品开发以及维护。