



微型机控制技术

PJ2报告

在线应用编程(IAP)

姓名：马逸君

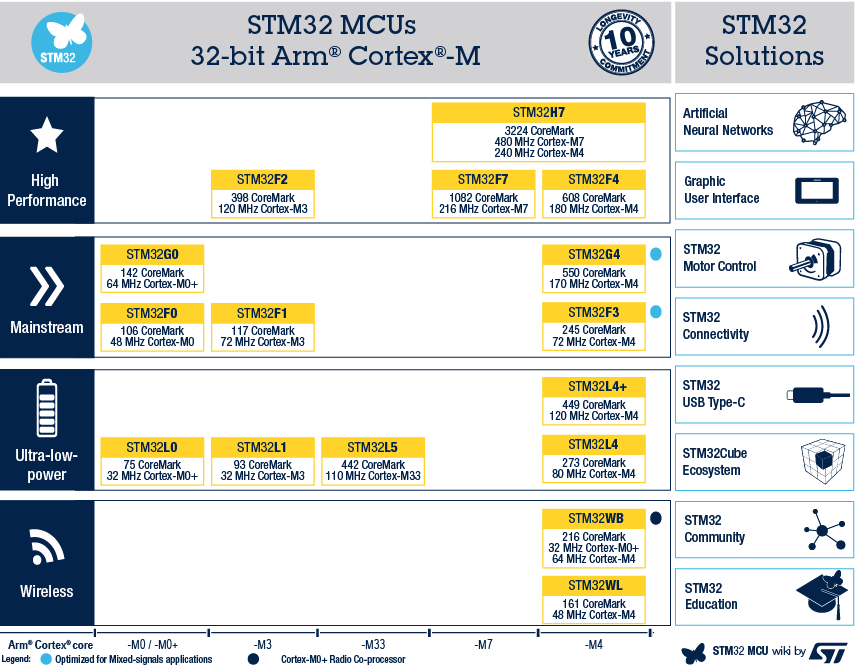
学号：17300180070

2020年6月

平台介绍

1. 芯片功能和资源

基于Arm®Cortex®-M处理器的**STM32系列32位微控制器**旨在为MCU用户提供更高的自由度。它提供的产品结合了超高性能、实时性、数字信号处理、低功耗/低压操作和互联功能，同时集成了完整功能、保持了易于开发的特性。

基于行业标准内核的STM32微控制器无与伦比，提供了广泛的工具和软件选择来支持项目开发，这使得该系列产品非常适合小型项目和端到端平台。

（图片来源：<https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>）

本次我们采用**基于STM32F103芯片的硬石YS-F1Pro开发板**为我们的硬件平台，其中STM32F103是STM32系列的一款主流(mainstream)级产品，频率为72 MHz，高达1 MB的闪存，除一般单片机均有的GPIO、定时器、ADC、UART、SPI、I2C等通信接口外，还拥有电机控制、USB和CAN等资源。硬石YS-F1Pro开发板与同类产品相比，板载硬件主打电机控制功能，适于我们的机器人项目。

1. 软件框架

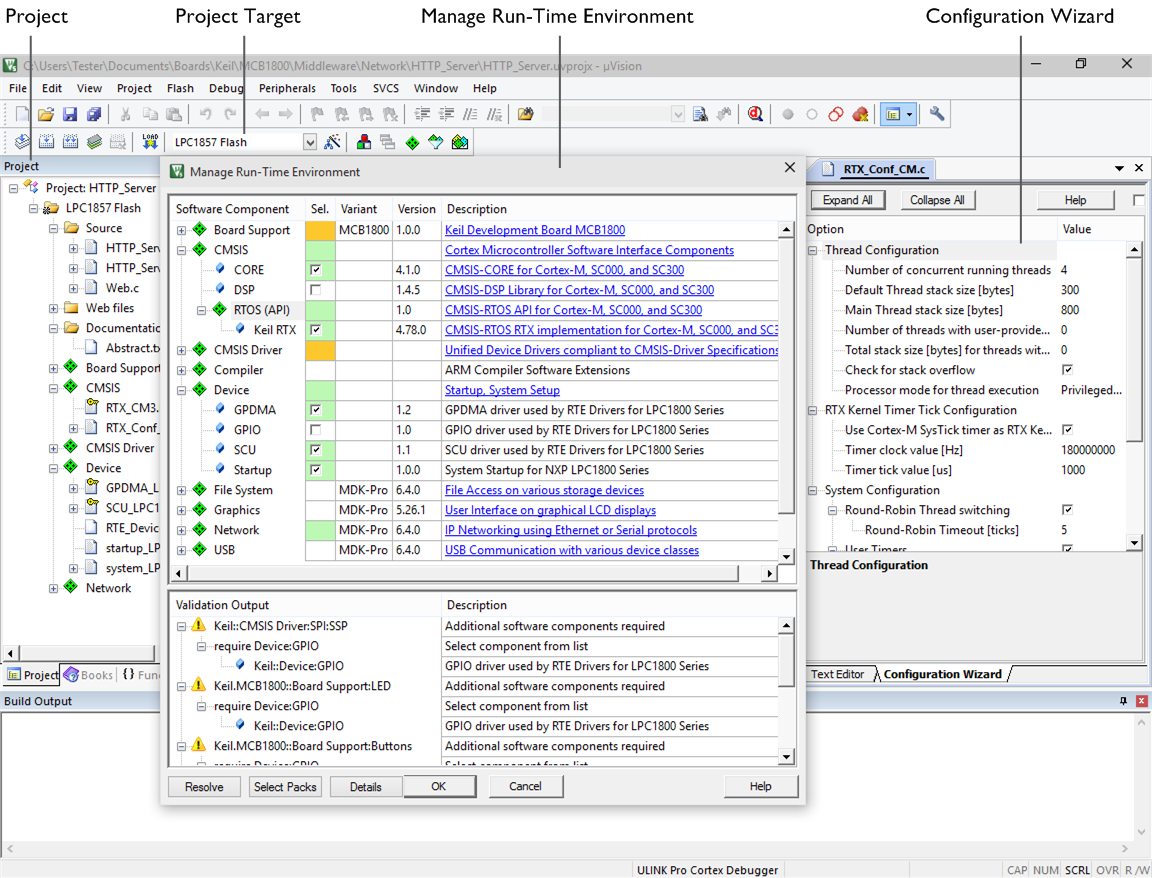
Keil MDK单片机开发套件是用于众多基于Arm Cortex-M处理器的微控制器设备的完整软件开发环境。MDK包括µVision IDE和调试器，Arm C / C ++编译器以及基本的中间件组件。它为所有芯片供应商提供了7000多种设备，并且易于学习和使用。

**µVision IDE**是最受欢迎的单片机开发环境之一。它在一个强大的环境中结合了项目管理、运行时环境、构建工具、源代码编辑和程序调试。µVision易于使用，可加快嵌入式软件的开发速度。µVision支持多个屏幕，并允许您在可视表面上的任何位置创建单独的窗口布局。

μVision调试器提供了在同一环境中测试、验证和优化应用程序代码的能力。该调试器包括传统功能（例如简单和复杂的断点）、监视窗口和执行控制，并提供对设备外围设备的完全可见性。

**C#** 是一种最新的、面向对象的编程语言。它使得程序员可以快速地编写各种基于Microsoft .NET平台的应用程序，Microsoft .NET提供了一系列的工具和服务来最大程度地开发利用计算与通讯领域。正是由于C#面向对象的卓越设计，使它成为构建各类组件的理想之选——无论是高级的商业对象还是系统级的应用程序。

本次实验我们选用µVision IDE进行嵌入式开发，并使用C#创建前端窗体。



目标及原理介绍

控制系统需要用到**在线升级**功能，此处我们选用在线应用编程IAP (in application programming) 这种技术来实现，从而可以在控制系统不停机的同时更新应用程序。事实上，一种更加常见的选择是在线系统编程ISP (in system programming)，ISP作为一种在线编程技术，不用将电路板上的元器件取下，就可以直接进行编程，目前市面上的Flash储存器几乎都是采用这种编程方式；但是它仍然需要直接连接电路板上引脚，有时候在工程项目中会出现电路板被安装在机械结构的特定位置，遇到升级程序的时候，往往需要将机器拆解开，才可以更新程序，非常不方便，为此诞生了IAP 技术，可以极大的降低工作量，无需拆卸设备，只需要通过预留的外设接口配合上位机下载就可以了。

所谓IAP就是在线应用编程，IAP 是用户自己的程序在运行过程中对UserFlash的部分区域进行烧写，方便程序升级，可以不用打开产品，直接通过串口、以太网接口或CAN升级。需要一个引导程序(bootloader)，另外还需要一个APP程序(就是待写入开发板的应用程序)。也就是两块存储区，一个一般称为BOOT区，另一个是存储区，引导程序在BOOT区，APP程序放到存储器，每次程序更新，都更新存放APP的存储器的内容，然后设置程序指针跳到存储器执行。

根据各种可能的使用场景，本次我们拟支持3种方式的IAP：

|  |  |  |
| --- | --- | --- |
| **串口** | **CAN** | **TCP** |
| 上电后等待15s内 通过串口软件导入bin文件（文件导入步骤详见硬石的参考文档） 且点击发送，发送后等待1s串口会显示“收到数据” 同时进入倒计时，如果6s后未发送新的数据 则写入并启动APP | 内容同串口下载，区别是CAN上位机一次发送2字节数据 | 板子没有外扩SRAM ，所以TCP一次只能发送1024个数据，一般发送4-5次即可，数据的划分理论上需要手动进行，可以用C++写一个软件来完成；但在改进版的IAP中可以自动完成。 |

实现步骤（含操作截图）

**第一步：嵌入式程序设计**

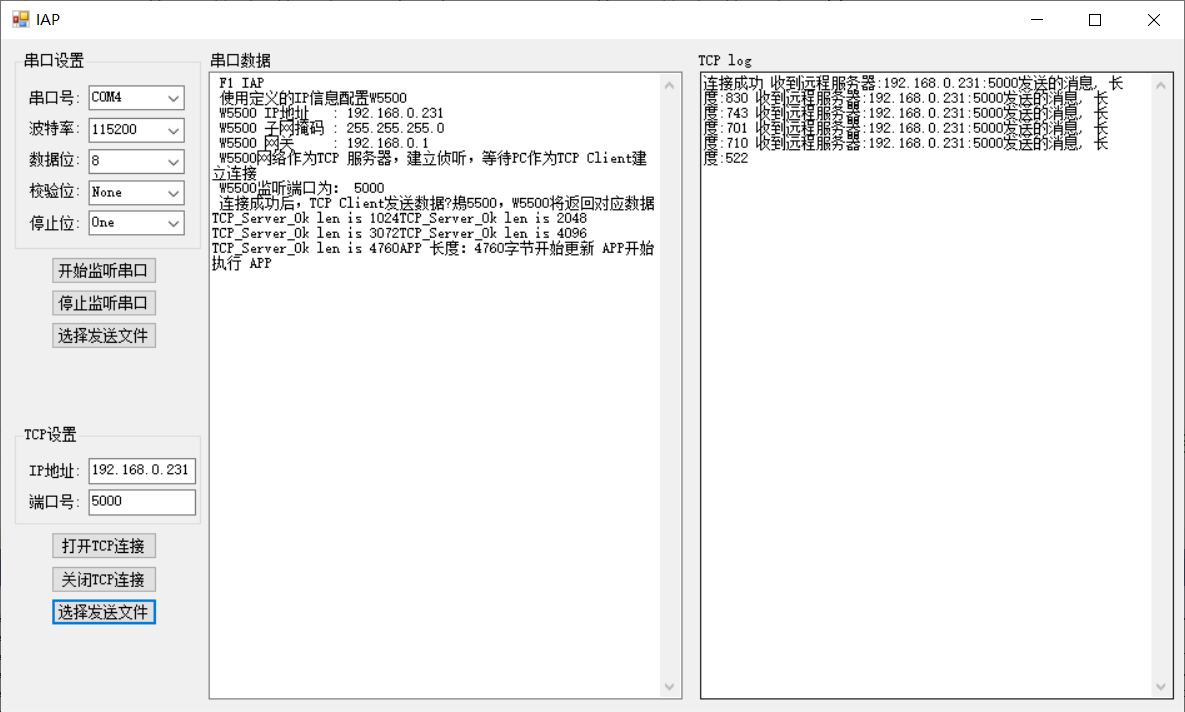
用C语言在 μVision中设计嵌入式程序，使开发板在启动之初监听串口、TCP、CAN，接收用户传入的二进制代码并写入Flash执行。

* 首先进行初始化工作，并打开串口、CAN、TCP服务器。
* 不停（**while(1)**）尝试接收数据，如三个接口之一有数据到来，则写入缓冲区并持续接收那个接口的数据。
* 超时（串口）或收到结束符（TCP）后，将缓冲区的数据写入Flash，然后执行。
* 若第二步中超时未接收到任何数据，则直接执行Flash中现存的代码。

此步的效果通过串口助手工具验证。这样就实现了一个基本的IAP程序。

**第二步：前端窗体设计**

用C#设计前端程序，在PC端供用户选择并发送二进制代码文件。前端提供稳定性优化（错误重发）和操作流程优化（TCP自动分段）。

* 窗体显示从串口接收到的数据和TCP连接日志。因为TCP只发不收（开发板侧会通过串口反馈结果），故不设TCP数据区。
* 用户在窗体上设置串口连接参数，点击按钮即可监听串口、发送二进制代码文件。TCP方式类似。（考虑到CAN不常见，就只实现了串口和TCP。）
* 修改开发板侧程序，将接收到的数据原样发回，供前端校验，待前端反馈校验结果方可写缓冲区。
* C#程序会接收开发板侧回传的数据，与本地记录的发送数据比较，若不等，则向开发板侧发送“失败重试”并重发，或“失败取消”并直接执行既有程序（若重试次数过多）。

核心代码展示

**在第一步的嵌入式程序中，主程序中通过这样一个无限循环接收IAP数据：**

while(1)

{

do\_tcp\_server(); // 尝试接收TCP数据

if (strAppBin.usLength>0) // 收到数据（任一途径）

{

int suc = 1;

if (usart\_have==1)

{

delay\_ms(1000);

printf ( "串口收到数据\n" );

}

while (over\_flag==0) // 数据尚未发送完毕

{

if (usart\_have==1) // 串口收到数据

{

delay\_ms(1000);

printf("%d ",6-usart\_flag);

usart\_flag++;

if(usart\_flag>6) // 倒计时6s

break;

}

else

{

if (do\_tcp\_server() == -1) {suc = 0; break;} // 从TCP接收数据，若前端给的反馈是“失败取消”，则跳到无下载执行

}

}

if (suc)

{

printf("APP 长度: %d字节\n", strAppBin.usLength );

printf("开始更新 APP\n");

IAP\_Write\_App\_Bin(APP\_START\_ADDR, strAppBin .ucDataBuf, strAppBin .usLength ); // 更新Flash代码

delay\_ms(1000);

delay\_ms(1000);

delay\_ms(1000);

delay\_ms(1000);

}

printf ( "开始执行 APP\n" );

IAP\_ExecuteApp(APP\_START\_ADDR); //执行FLASH APP代码

}

else

{

i++;

if (i==65534)

{

i=0;

d++;

}

if (d==0x09)

{

printf ( "\n无下载开始执行 APP\n" );

IAP\_ExecuteApp(APP\_START\_ADDR);

}

}

}

**前端TCP部分的代码如下，包含错误重发和自动分段：**

private void btnTCPSend\_Click(object sender, EventArgs e)

{

// “选择文件”对话框

OpenFileDialog dialog = new OpenFileDialog();

dialog.Multiselect = false;//该值确定是否可以选择多个文件

dialog.Title = "IAP：请选择要发送的代码文件";

dialog.Filter = "所有文件(\*.\*)|\*.\*";

if (dialog.ShowDialog() != System.Windows.Forms. dialogResult.OK) return;

// 读取文件内容

string file = dialog.FileName;

byte[] sendData = System.IO.File.ReadAllBytes(file);

try

{

byte[] curSendData = new byte[1025]; // 分段的数据

byte[] sendBackData0, sendBackData = new byte[1025];

int retryCount = 0; // 重试次数

for (int i = 0; i < sendData.Length; i += 1024)

{

curSendData.Initialize();

Array.Copy(sendData, i, curSendData, 0, Math.Min(1024, sendData.Length - i)); // 取出当前段的数据

// curSendData[1] = 0x00; // 这句语句故意改掉要发送的数据的记录，用于测试错误重发功能

socketSend.Send(sendData, i, Math.Min(1024, sendData. Length - i), SocketFlags.None); // 发送当前段数据

Thread.Sleep(200);

sendBackData0 = Receive(Math.Min(1024, sendData. Length - i)); // 接收回传数据

sendBackData.Initialize();

Array.Copy(sendBackData0, 0, sendBackData, 0, Math.Min(1024, sendData.Length - i)); // 在回传数据的后面补0，使其与发送数据长度相同

Thread.Sleep(100);

if (Enumerable.SequenceEqual(curSendData, sendBackData)) // 若相等

{

socketSend.Send(new byte[2] { 0x55, 0xaa }, 0, 2, SocketFlags.None); // 发送“成功”信号（规定为55AA）

retryCount = 0;

}

else

{

retryCount++; i -= 1024; // 修改i，重试发送当前段

if (retryCount == 3) // 已经重试了3次

{

socketSend.Send(new byte[2] { 0xaa, 0xaa }); // 发送“失败取消”信号（规定为AAAA），开发板侧将直接跳转执行既有代码

this.txt\_Log.AppendText("重试次数超限，失败 \r \n");

this.txt\_Log.AppendText("升级取消 \r \n");

return;

}

else

{

socketSend.Send(new byte[2] { 0x55, 0x55 }); // 发送“失败重试”信号（规定为5555），开发板侧将重新接收当前段

}

this.txt\_Log.AppendText(String.Format("第{0:D}段数据发送失败，重试({1:D}/3)... \r \n", 1+(i+1)/1024, retryCount));

}

}

byte[] endSymbol = new byte[2] { 0xaa, 0x55 };

socketSend.Send(endSymbol); // 发送“结束”信号（AA55）

Thread.Sleep(200);

}

catch (Exception ex) // 异常处理

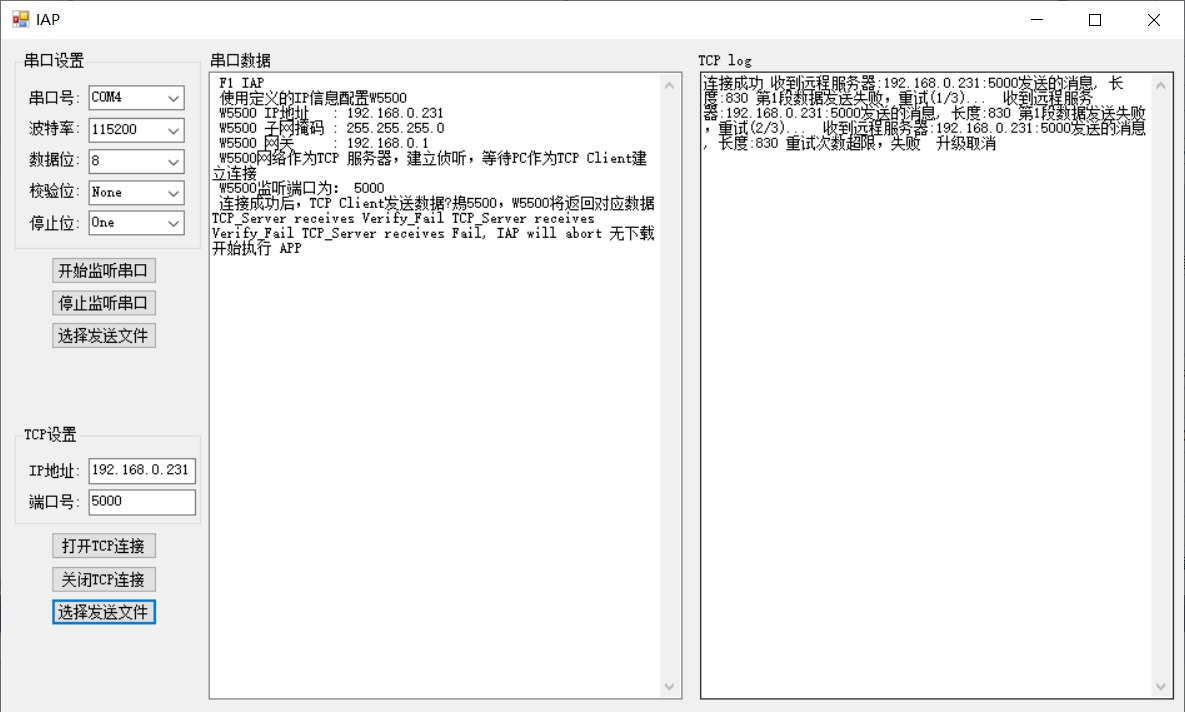
{

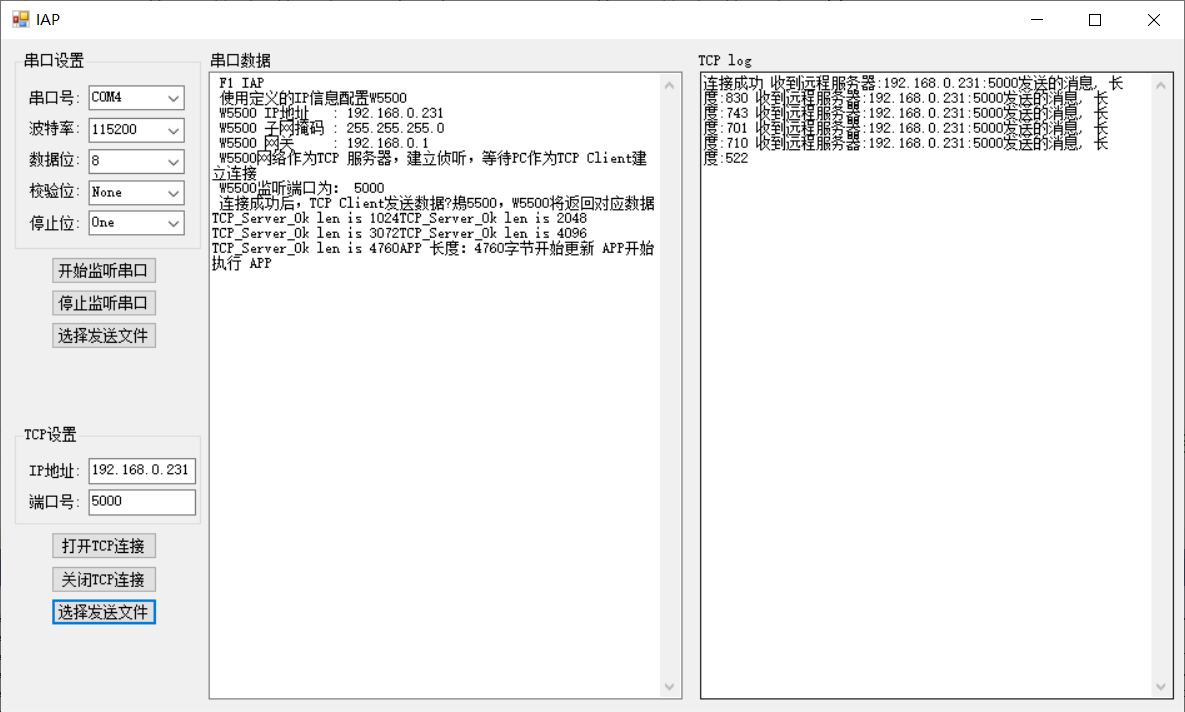
MessageBox.Show("发送消息出错:" + ex.Message);

}

}

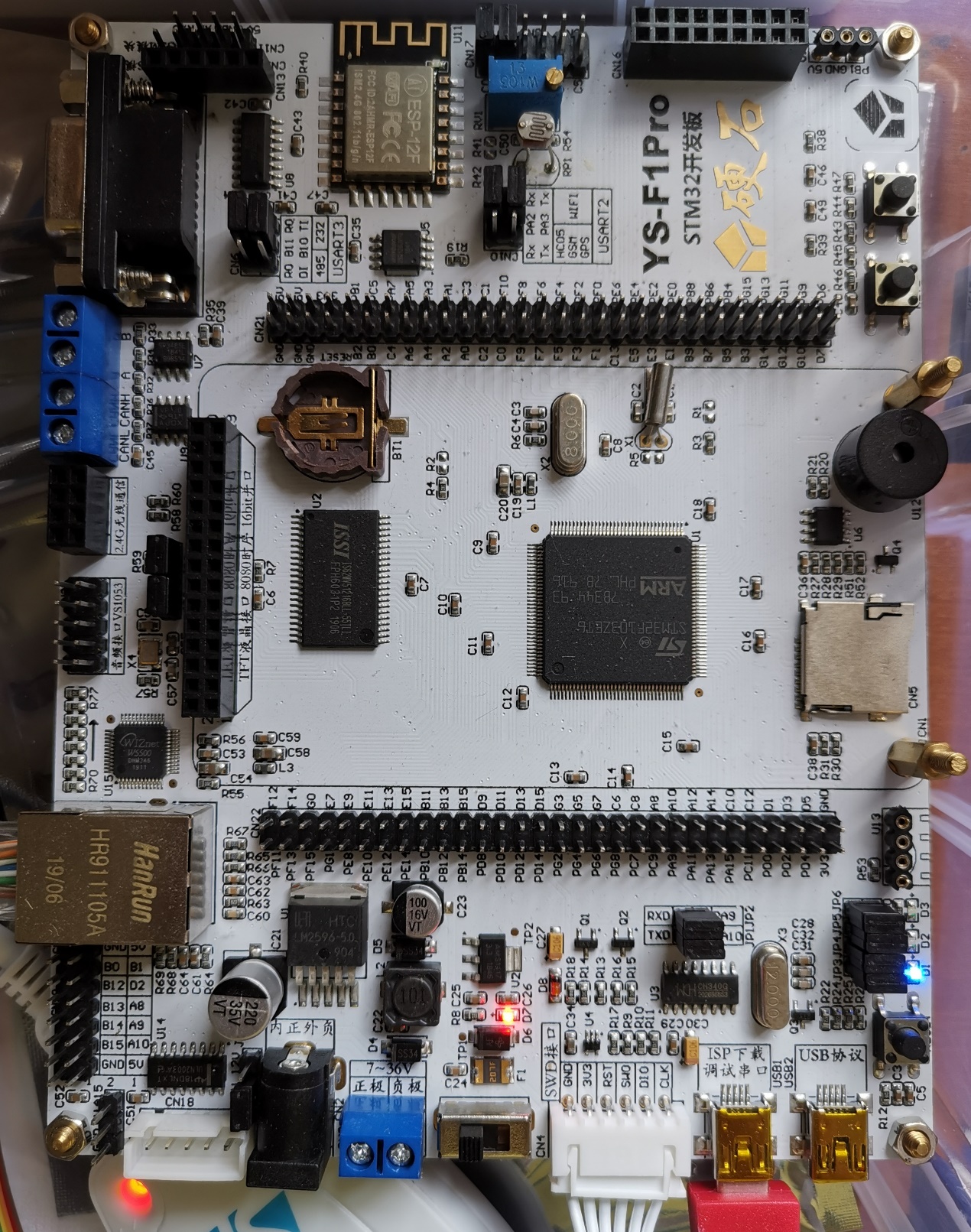
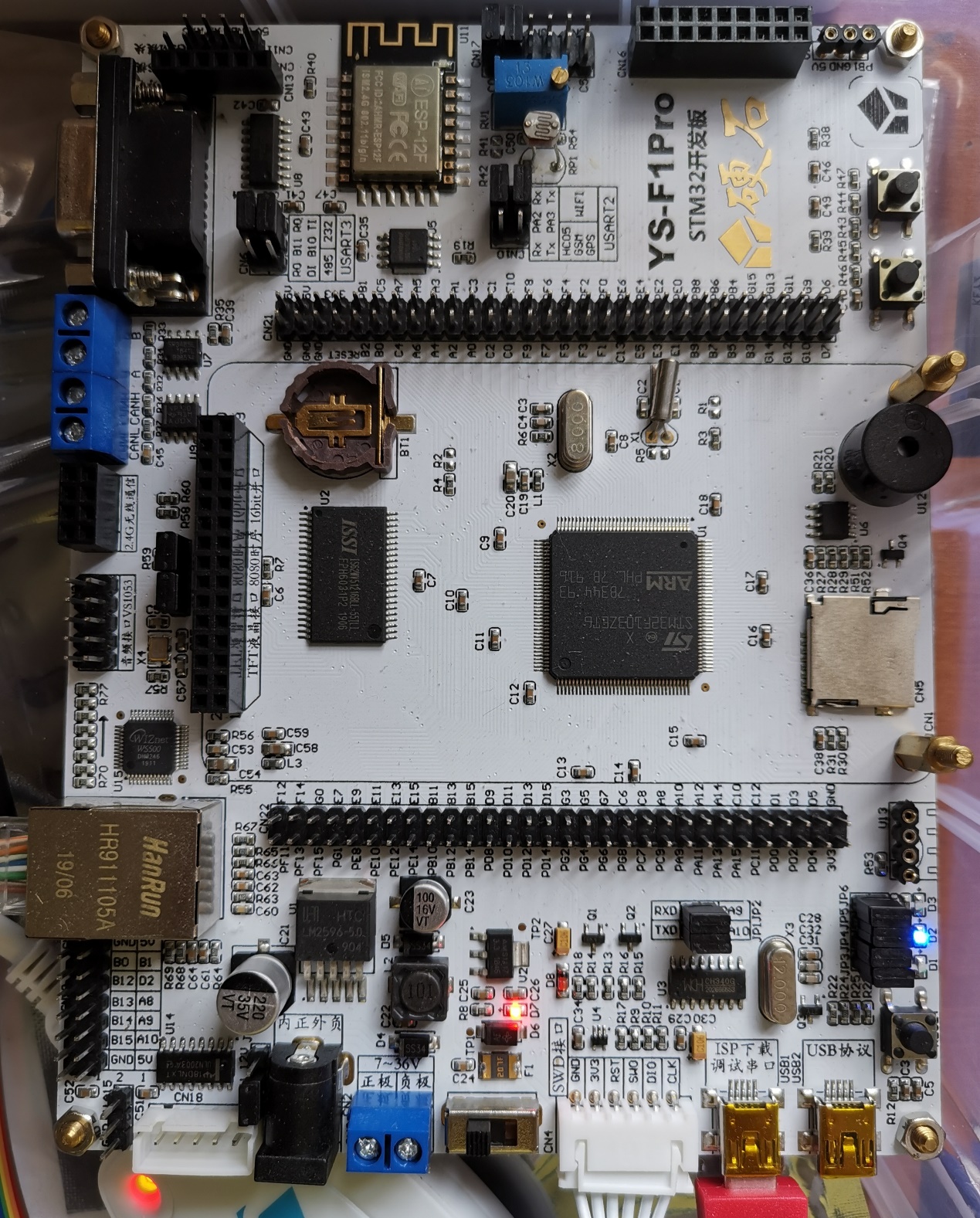
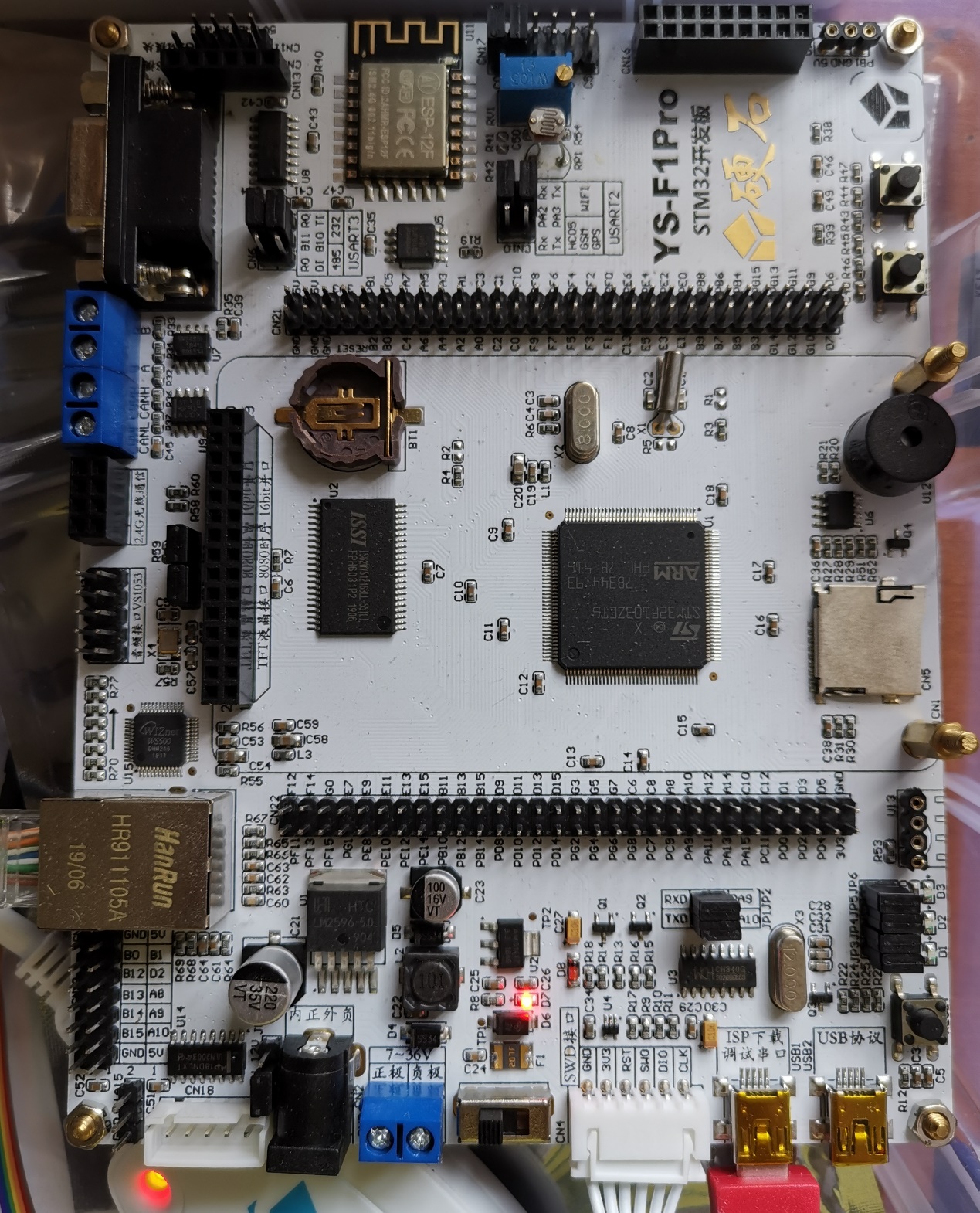
效果图

****为验证**错误重发**，我们在C#的代码里加入一句：**curSendData[1] = 0x00;** ，即在发送完数据后，故意将发送记录中的二进制代码位改掉。效果如下：

通过TCP正常升级的效果如下：

通过串口正常升级的效果如下：

与此同时，开发板照片：（右下角蓝色LED可见流水灯效果）



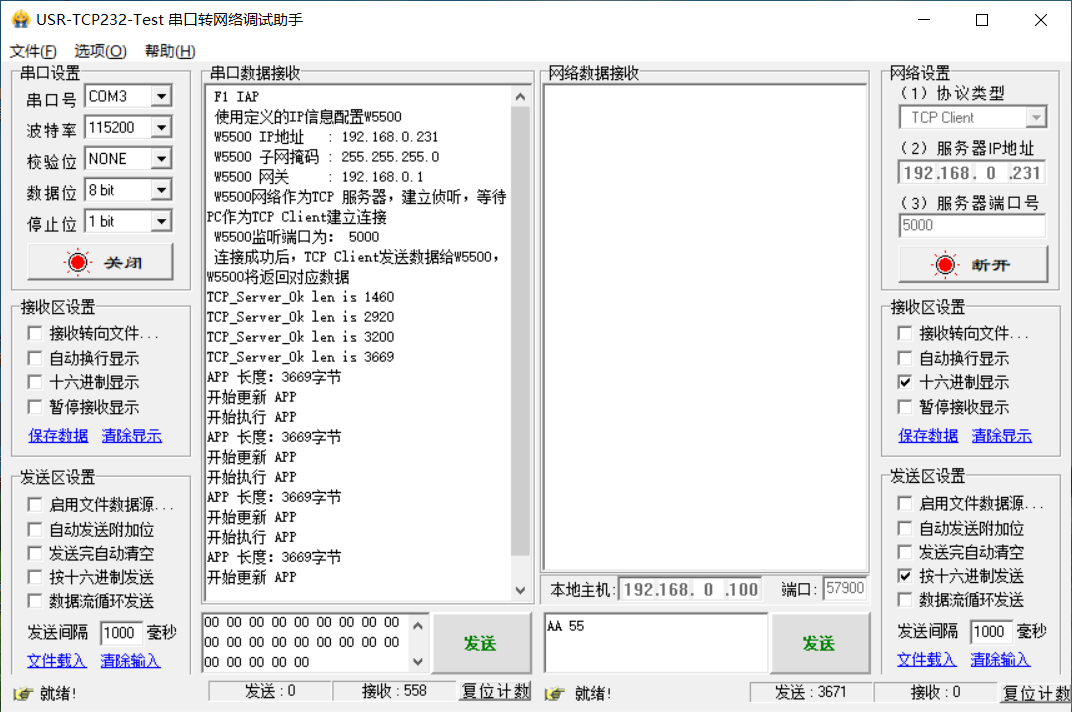
开发过程问题记录

本次开发过程主要有两个值得讨论的问题。

第一个问题是如何判断Flash既有内容为空。事实上一开始，我们拟定的操作流程是：启动后等待5秒，5秒内有IAP请求，则进入数据通信，没有则检查是否已经有用户程序，有则转入用户程序，没有则进入循环等待，并结合指示灯进行状态指示。

但是实验中我们发现，一旦向Flash中写过任何数据，以后Flash就都不为空了，即使断电后重启开发板也是如此。这应该与Flash的性质有关：Flash存储不是易失的。

非常有趣的是，为了模拟Flash为空的情况，我们向Flash中写入多个00，发现IAP程序会进入无限循环状态：

类似地，若IAP中发送的二进制代码为全0，则会进入无限循环更新Flash代码：

对此现象，我们推测原因是，全0的代码会返回（而一个规范的嵌入式程序是永不返回的），导致在调用IAP\_ExecuteApp(APP\_START\_ADDR)开始执行用户程序后，控制又从Flash中程序（是全0的）返回到IAP中，并继续其中的无限循环过程。

考虑到规范的嵌入式程序是永不返回的，所以实际应用中不应出现该种全0代码情况。实际场景下更可能出现的情况是，一旦向Flash中写过任何数据，以后Flash就都不为空了。所以在最终实现中我们移除了用户程序为空指示功能。

第二个问题是，在第二步开发过程中，需要将接收TCP数据的函数改为在每收到一段数据后紧接着接收2bit的前端反馈数据（成功、失败重试、失败取消3种）。运行中我们发现，前端总在接收完第一段数据后陷入卡顿，后调试发现是卡在下一段数据回传的接收函数上。亦即，开发板侧那时还没运行到下一段数据的接收函数。尝试注释掉第二次接收，发现程序正常运行，显然这就是第二次接收函数导致的问题了。

我想到的解决方案是延迟接收，即，开发板侧接收时循环等待，直至传入的数据长度达到2bit为止：

**while ((len2 = getSn\_RX\_RSR(SOCK\_TCPS)) == 0) ;**

recv(SOCK\_TCPS, buff2, len2); // 2 bytes of confirm bits

这样就完美解决了该问题。

附录

硬件原理图（STM32微控制器、

模拟串口、以太网、CAN、Flash）