ESP8266 LUA脚本语言开发(13)
ESP8266 LUA开发基础入门篇备份(22)
ESP8266 SDK开发(33)
ESP8266 SDK开发基础入门篇备份(30)
GPRS Air202 LUA开发(11)
HC32F460(华大单片机)物联网开发(17)
HC32F460(华大单片机)学习开发(8)
NB-IOT Air302 AT指令和LUA脚本语言开发(27)
PLC(三菱PLC)基础入门篇(2)
STM32+Air724UG(4G模组)物联网开发(43)
STM32+BC26/260Y物联网开发(37)
STM32+CH395Q(以太网)物联网开发(24)
STM32+ESP8266(ZLESP8266A)物联网开发(1)
STM32+ESP8266+AIR202/302远程升级方案(16)
STM32+ESP8266+AIR202/302终端管理方案(6)
STM32+ESP8266+Air302物联网开发(65)
STM32+W5500+AIR202/302远程升级方案(6)
STM32+W5500物联网开发(12)
UCOSii操作系统(1)
W5500 学习开发(8)
编程语言C#(11)
编程语言Lua脚本语言基础入门篇(6)
编程语言Python(1)
单片机(LPC1778)LPC1778(2)
单片机(MSP430)开发基础入门篇(4)
单片机(STC89C51)单片机开发板学习入门篇(3)
单片机(STM32)基础入门篇(3)
更多

说明

这节提供给用户一份使用串口实现更新STM32的程序(兼容STM32f103全系列)

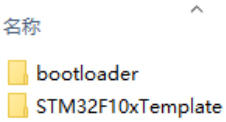
主要说明STM32是如何实现的升级程序.后面的章节都是在这节的基础上进行优化.

该节源码开源: https://gitee.com/yang456/STM32_IAP_Learn.git

请用户认真学习此节!该代码只使用了5字节数组接收程序文件!

测试

1.说明



BootLoader作为引导程序,负责把接收的程序文件写入flash,然后加载执行.

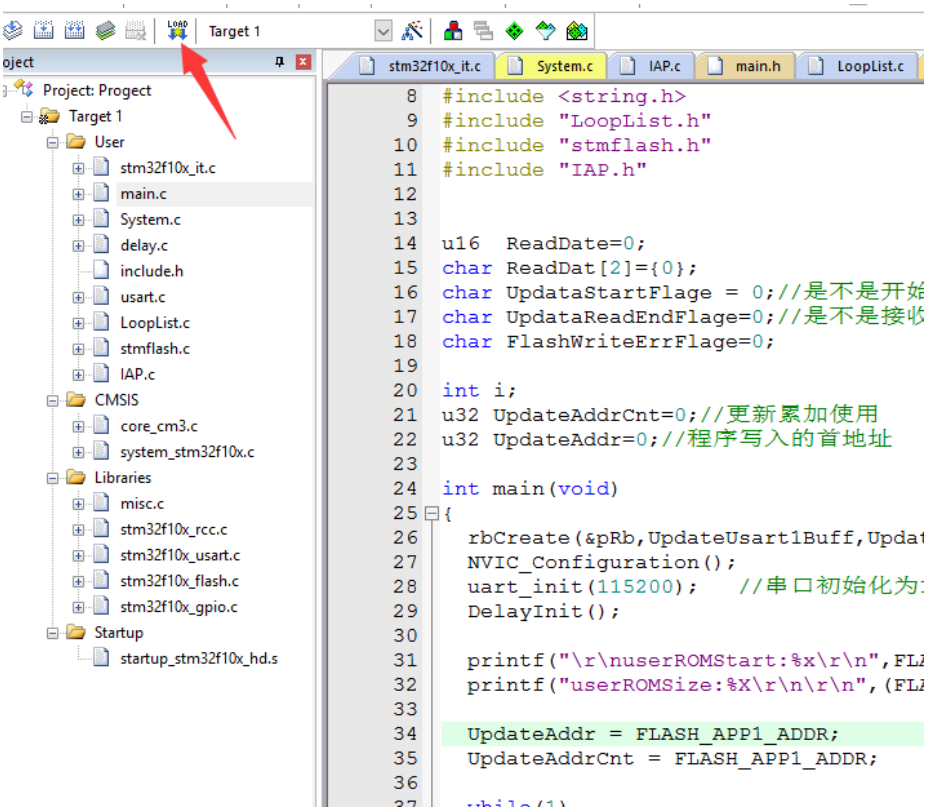
STM32F10xTemplate 是用户程序,这套程序采用串口升级进去.然后执行

2.下载BootLoader程序到单片机(自行下载)

1. C#委托+回调详解(10)
2. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(9)
3. 我的大学四年(7)
4. 用ESP8266+android,制作自己的WIFI小车(Android 软件)(6)
5. 关于stm32的正交解码(6)

最新评论

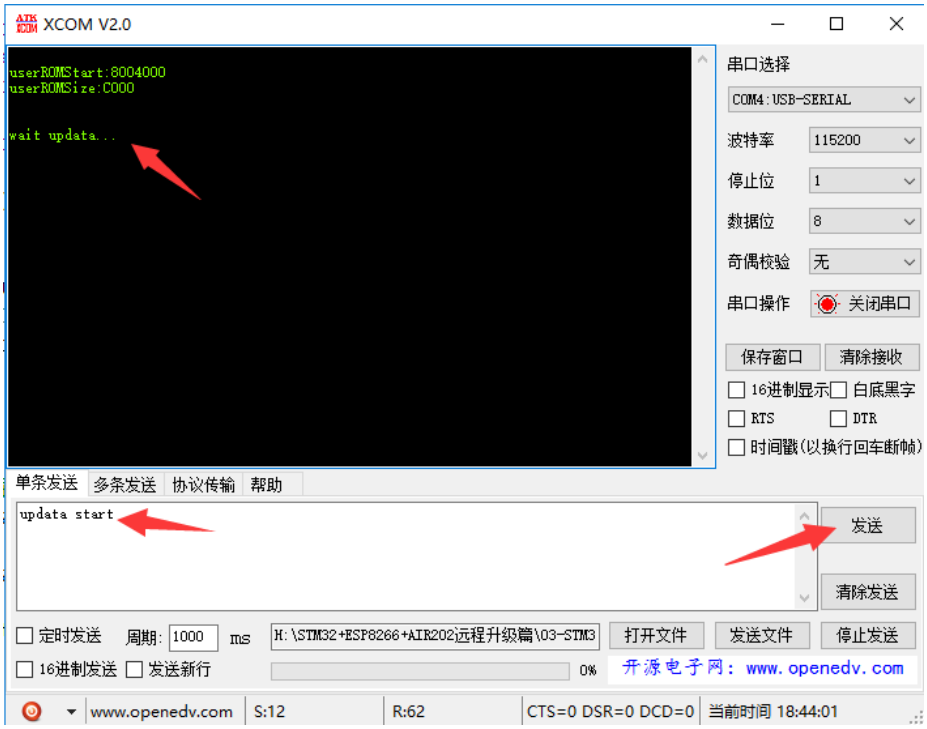
1. Re:0-STM32+W5500基本控制篇(自建物联网平台)-整体运行测试-STM32+W5500连接MQTT服务器
您好 您这个的样例代码有开源吗？
--dqwe56q4wd4
2. Re:用ESP8266+android,制作自己的WIFI小车(Android 软件)
可以重新发一遍源代码吗
--evakzxx



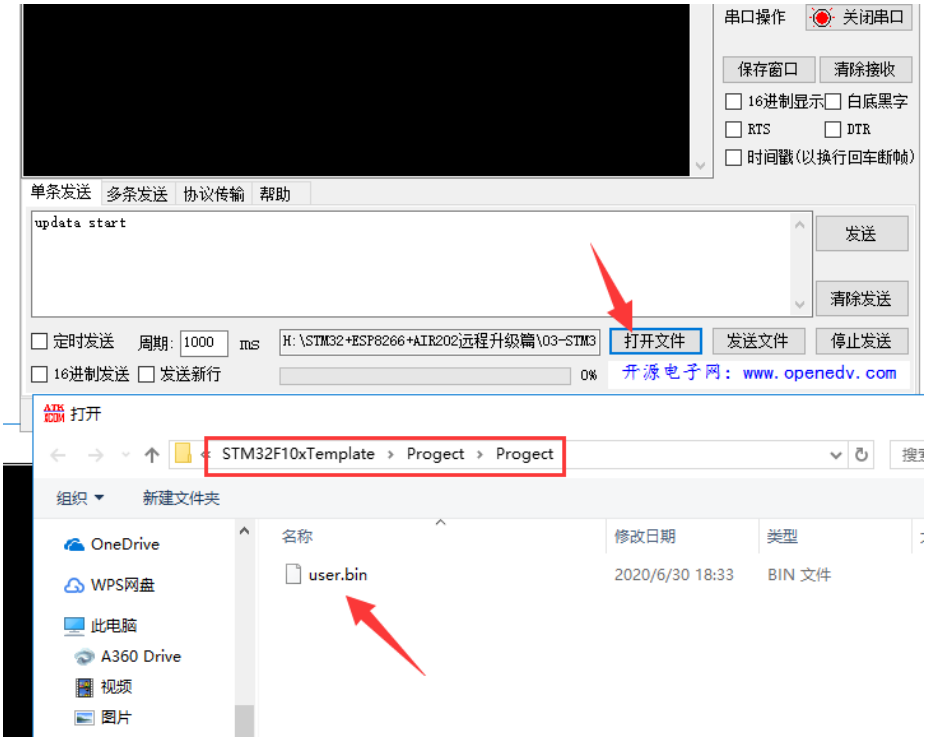
4.打开串口调试助手

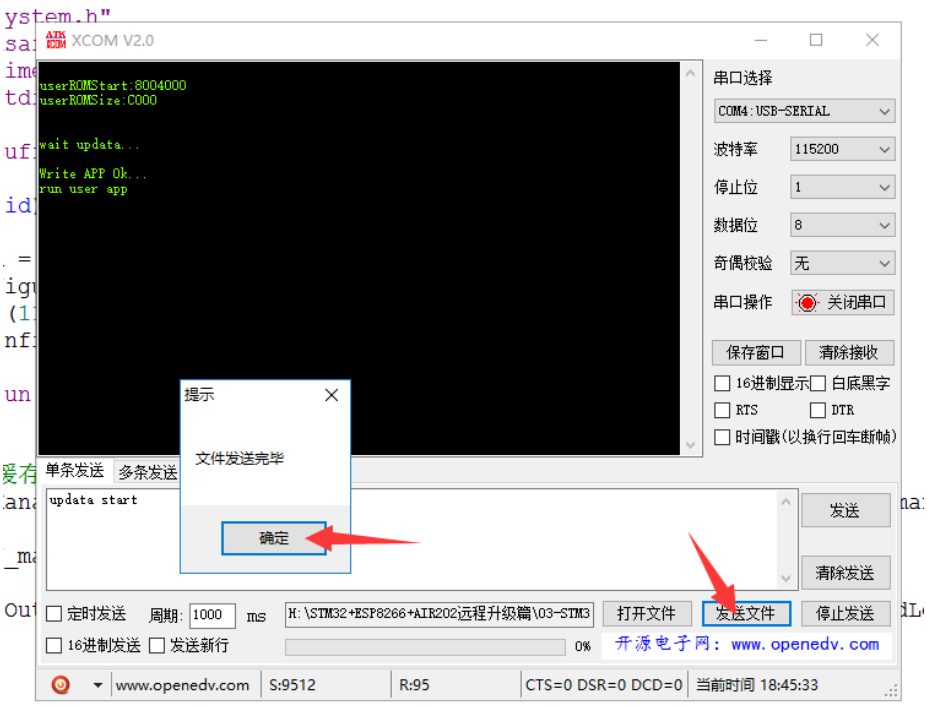
发送 updata start

单片机擦除flash以后返回 wait updata...

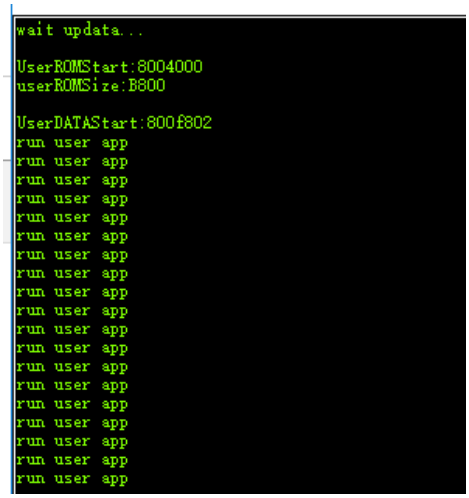


5.发送程序文件STM32F10xTemplate 用户程序的 bin文件





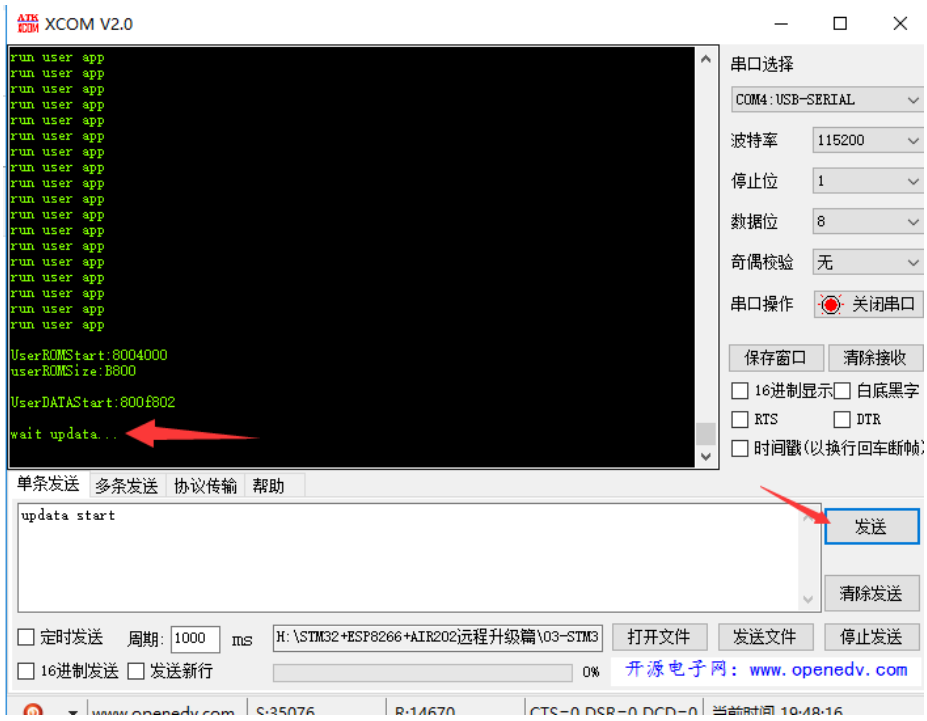
用户程序每隔1S打印 run user app



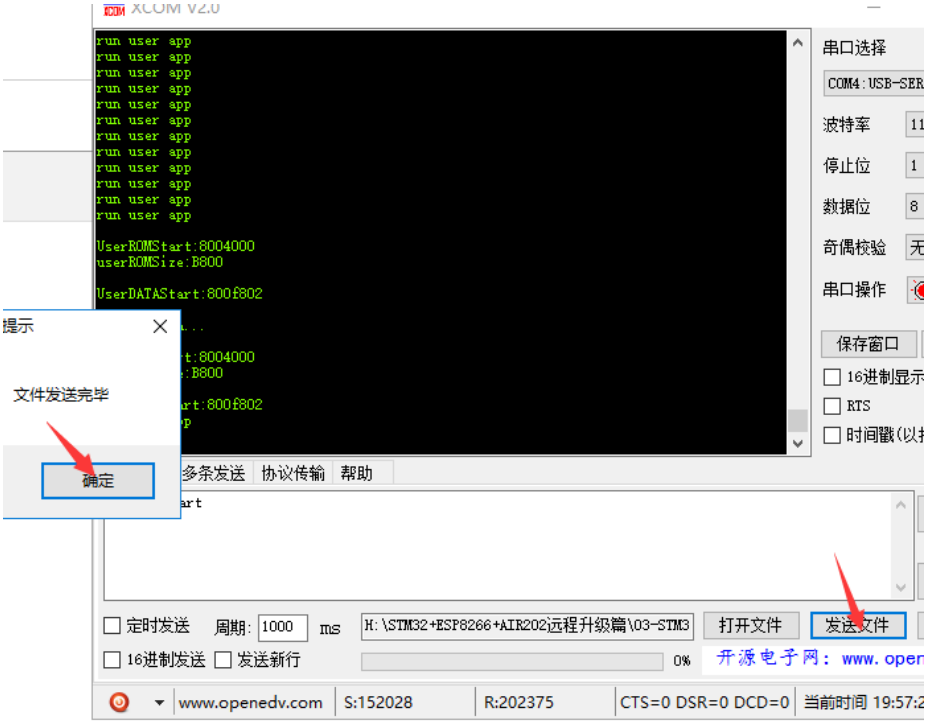
6.在用户程序执行的时候再次发送

updata start

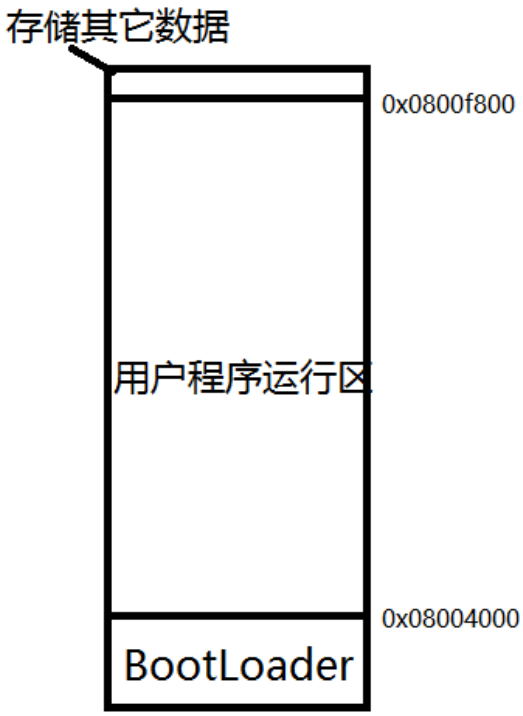
等待返回 wait updata... 可以再次发送程序文件,执行更新.



发送程序文件



整体说明



首先用户需要明白,无论是什么单片机实现更新程序,实质上就是把程序文件写到单片机的存储里面

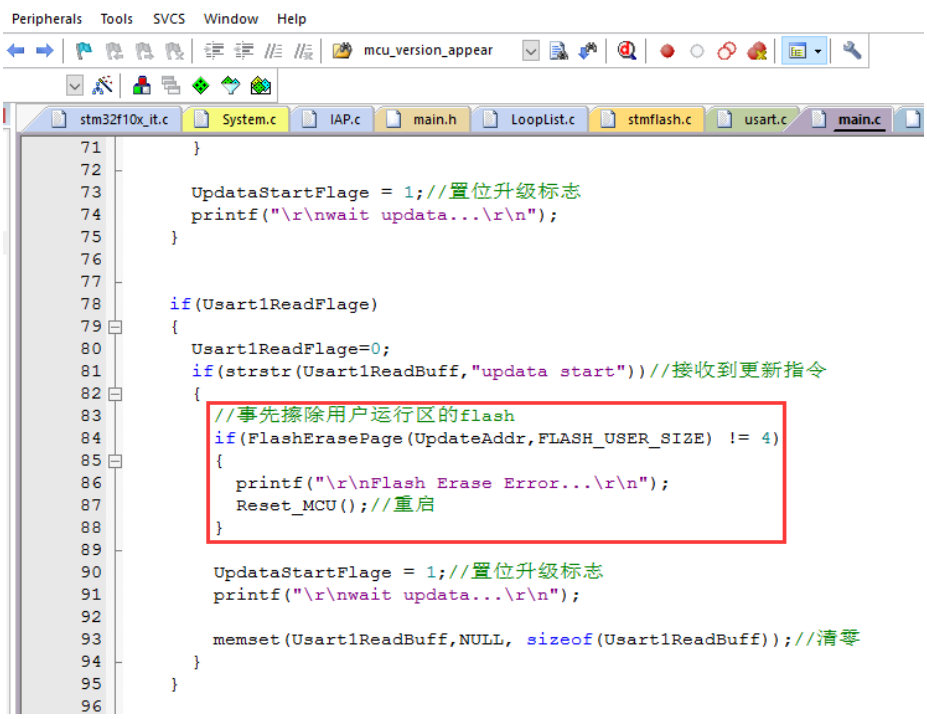
然后调用单片机提供的函数运行而已!!

对于当前的STM32程序而言就是把程序文件从0x08004000这个位置开始,把程序文件写到里面

然后把0x08004000这个地址给一个函数执行.

1.下载完BootLoader以后,当用户发送 updata start

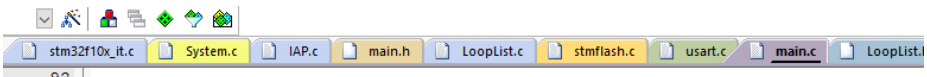
程序擦除用户程序运行区的flash




```
71          }
72
73          UpdataStartFlage = 1;//置位升级标志
74          printf("\r\nwait updata...\r\n");
75        }
76
77
78        if(Usart1ReadFlage)
79        {
80          Usart1ReadFlage=0;
81          if(strstr(Usart1ReadBuff,"updata start"))//接收到更新指令
82          {
83            //事先擦除用户运行区的flash
84            if(FlashErasePage(UpdateAddr,FLASH_USER_SIZE) != 4)
85            {
86              printf("\r\nFlash Erase Error...\r\n");
87              Reset_MCU();//重启
88            }
89
90            UpdataStartFlage = 1;//置位升级标志
91            printf("\r\nwait updata...\r\n");
92
93            memset(Usart1ReadBuff,NULL, sizeof(Usart1ReadBuff));//清零
94          }
95        }
96
```

2.当用户发送程序文件时

把接收的程序从0x08004000这个位置开始,把程序文件写到flash里面

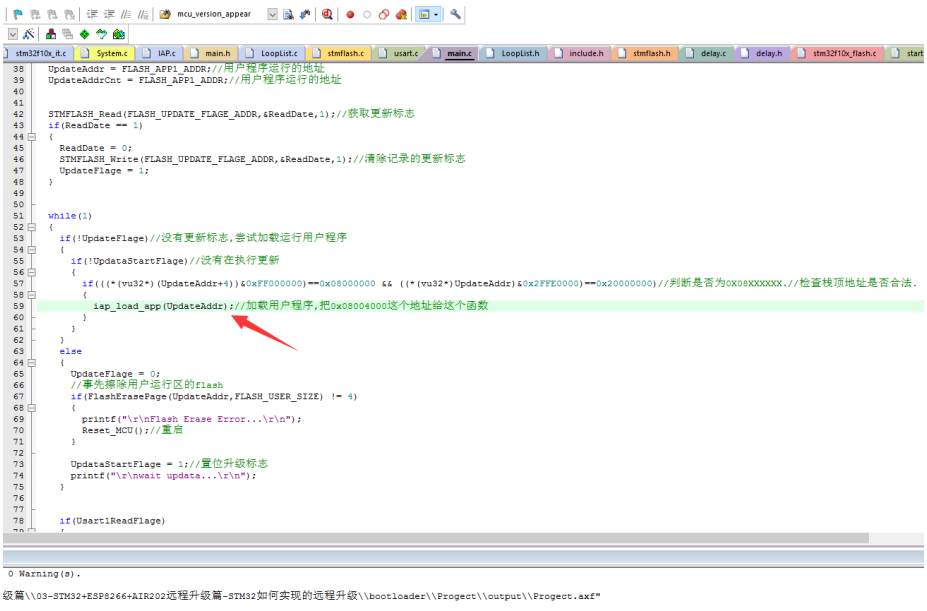



```
92
93            memset(Usart1ReadBuff,NULL, sizeof(Usart1ReadBuff));//清零
94          }
95        }
96
97
98        if(UpdataStartFlage == 1)
99        {
100         if(rbCanRead(&pRb)>1)//接收到更新程序就开始写入
101         {
102           rbRead(&pRb, &ReadDat, 2);//读取两个数据
103           ReadDate = (u16)ReadDat[1]<<8;
104           ReadDate = ReadDate|ReadDat[0];
105
106           if(!FlashWriteErrFlage)//写Flash没有出现过错误
107           {
108             if(FlashWriteHalfWord(UpdateAddrCnt,ReadDate) != 0)//写Flash出错
109             {
110               FlashWriteErrFlage = 1;
111             }
112           }
113           UpdateAddrCnt+=2;
114         }
115         else//写完了
116         {
117           if(UpdataReadEndFlage)//接收完更新程序
118           {
119             UpdataStartFlage=0;
120             if(FlashWriteErrFlage == 1)//Flash写错误
121             {
```

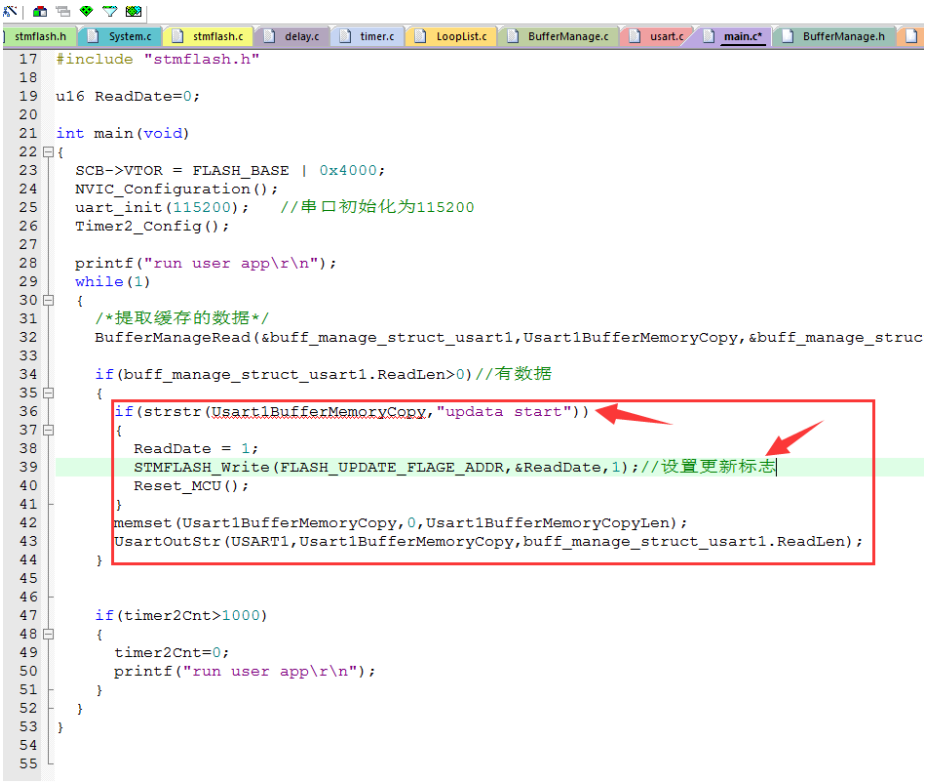
3.当检测到接收到用户程序,打印下有没有错误信息,然后重启

```
111             }
112           }
113           UpdateAddrCnt+=2;
114         }
115         else//写完了
116         {
117           if(UpdataReadEndFlage)//接收完更新程序
118           {
119             UpdataStartFlage=0;
120             if(FlashWriteErrFlage == 1)//Flash写错误
121             {
122               printf("\r\nFlash Write Error...\r\n");
123             }
124             else if(!UpdateOverflow)//没有溢出过
125             {
126               //Flash开始的地址是
127               if(((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000 && ((*(vu
128               {
129
130               }
131               else
132               {
133                 printf("\r\nFlash Data Error...\r\n");
134               }
135             }
136             else//数据溢出
137             {
138               printf("\r\nData Overflow...\r\n");
139             }
140             Reset_MCU();//重启
141           }
142         }
143       }
144     }
145   }
146
```

4.一旦有了用户程序,则加载用户程序



5.在用户程序里面,如果接收到updata start 则设置一个更新标志(存储在flash里面)



```c
#include "stmflash.h"

u16 ReadDate=0;

int main(void)
{
  SCB->VTOR = FLASH_BASE | 0x4000;
  NVIC_Configuration();
  uart_init(115200);   //串口初始化为115200
  Timer2_Config();

  printf("run user app\r\n");
  while(1)
  {
    /*提取缓存的数据*/
    BufferManageRead(&buff_manage_struct_usart1,Usart1BufferMemoryCopy,&buff_manage_struc

    if(buff_manage_struct_usart1.ReadLen>0)//有数据
    {
      if(strstr(Usart1BufferMemoryCopy,"updata start"))
      {
        ReadDate = 1;
        STMFLASH_Write(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//设置更新标志
        Reset_MCU();
      }
      memset(Usart1BufferMemoryCopy,0,Usart1BufferMemoryCopyLen);
      UsartOutStr(USART1,Usart1BufferMemoryCopy,buff_manage_struct_usart1.ReadLen);
    }


    if(timer2Cnt>1000)
    {
      timer2Cnt=0;
      printf("run user app\r\n");
    }
  }
}
```

6.BootLoader 判断有更新标志以后,擦除用户程序运行区的flash

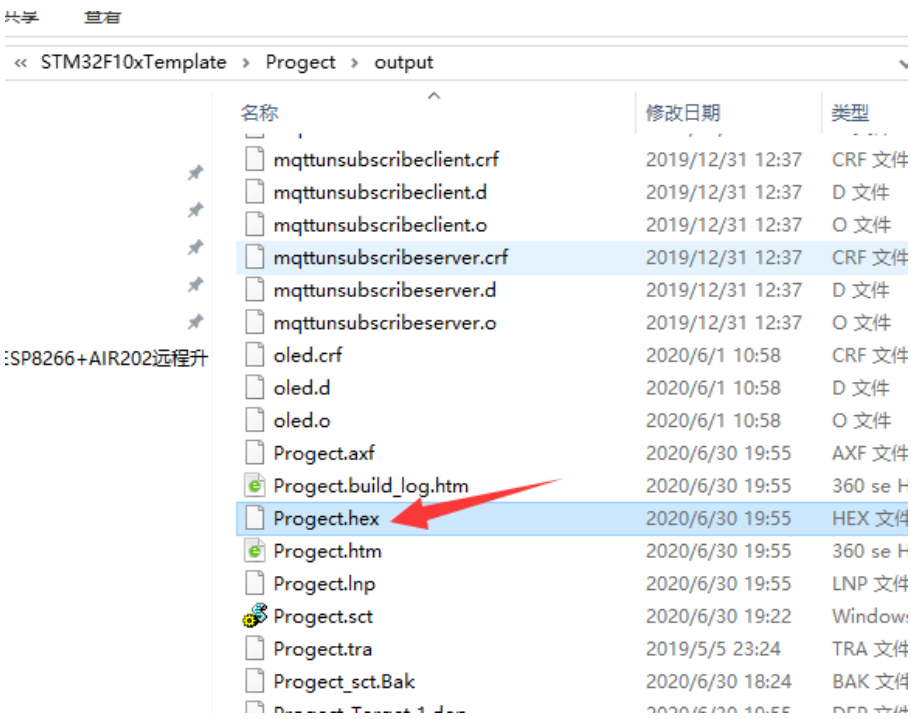
```c
  UpdateAddr = FLASH_APP1_ADDR;//用户程序运行的地址
  UpdateAddrCnt = FLASH_APP1_ADDR;//用户程序运行的地址


  STMFLASH_Read(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//获取更新标志
  if(ReadDate == 1)
  {
    ReadDate = 0;
    STMFLASH_Write(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//清除记录的更新标志
    UpdateFlage = 1;
  }


  while(1)
  {
    if(!UpdateFlage)//没有更新标志,尝试加载运行用户程序
    {
      if(!UpdataStartFlage)//没有在执行更新
      {
        if(((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000 && ((*(vu32*)UpdateAddr)&0x2FFE
        {
          iap_load_app(UpdateAddr);//加载用户程序,把0x08004000这个地址给这个函数
        }
      }
    }
    else
    {
      UpdateFlage = 0;
      //事先擦除用户运行区的flash
      if(FlashErasePage(UpdateAddr,FLASH_USER_SIZE) != 4)
      {
        printf("\r\nFlash Erase Error...\r\n");
        Reset_MCU();//重启
      }

      UpdataStartFlage = 1;//置位升级标志
      printf("\r\nwait updata...\r\n");
    }
```

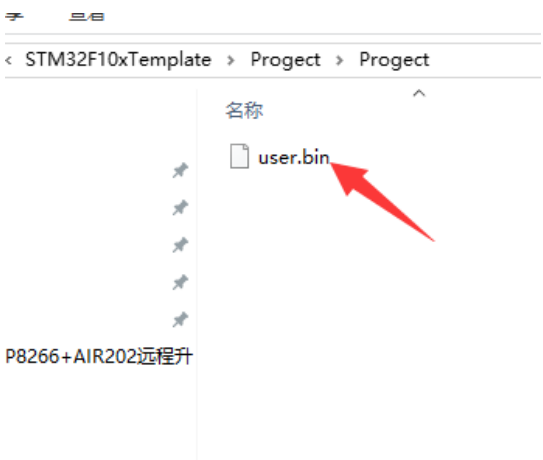
细节说明(bin文件)

1.什么是bin文件?

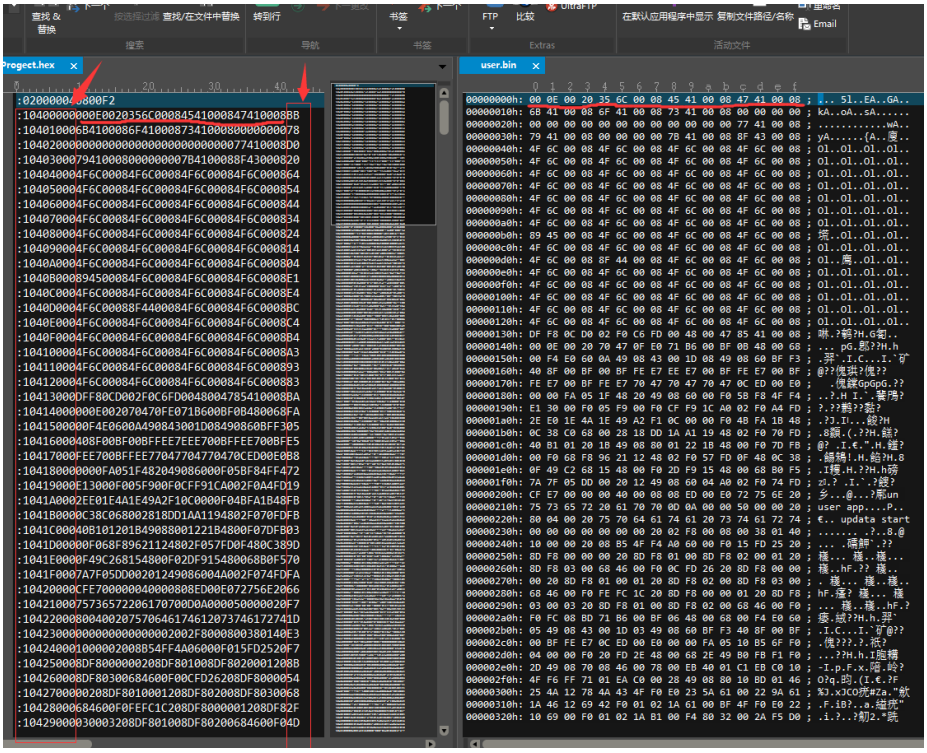
大家肯定知道hex文件

打开这节的hex文件和bin文件





(我使用的UltraEdit这个软件)



注意看hex文件和bin文件的区别

hex比bin文件多了前面一部分,和后面一部分

大家下载单片机程序应该都知道是下载hex文件

但是大家了解整个的下载过程不

其实咱用软件下载的时候首先单片机需要知道下载的这段程序下载到哪个地址上(把程序数据写到哪个Flash地址)

所以hex文件的前面部分就是地址信息,就是告诉芯片我后面的代码段存储到哪个地址上

当然为了保险起见,数据需要加校验,其实hex文件的最后一位就是校验位

像51单片机,STM32的串口下载,下载的时候需要断电上电,或者复位一下,其实咱的单片机里面有一段程序(接收单片机程序,写入Flash) 就是咱所说的bootloader

记不记得都是先点一下下载软件的下载按钮,然后再复位单片机

其实点一下下载软件,下载软件就一直串口输出下载命令呢!单片机一启动是先执行里面内嵌的bootloader,

bootloader一检测到下载命令,就开始执行下载操作了,接收下载软件过来的数据,然后写到Flash里面.....写完了,有的单片机重启下才运行,有的就直接运行了

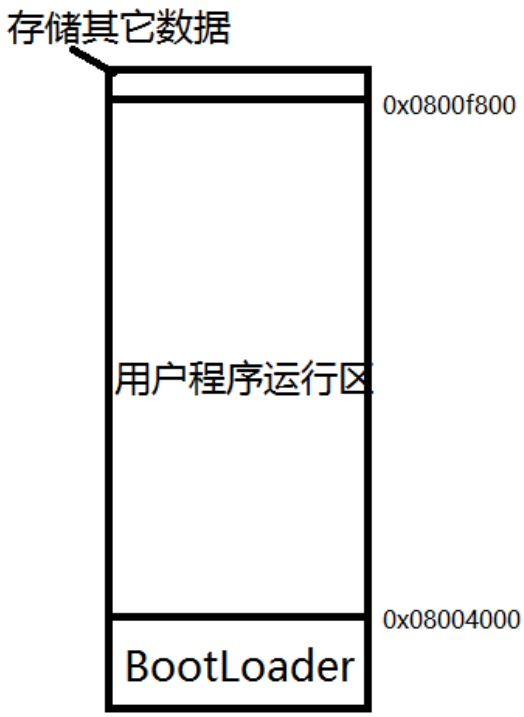
好,现在说bin文件为啥去掉了前面的地址信息

记住,咱自己更新的时候咱就规定好了程序的运行位置

咱们是直接把程序文件写入了相应的地址里面.

2.如何制作bin文件

2.1 概述



在做升级之前,上面的flash存储位置是事先规定好的

stm32的flash地址是从0x08000000开始,默认下载程序的时候都是把程序文件从0x08000000开始写入

这节规定了从0x08000000地址到0x08004000这段flash留给BootLoader使用(16KB)

这个具体留多少要看BootLoader程序bin文件大小,后面有具体说明.

2.2 设置中断偏移地址,程序文件从哪开始执行,就设置偏移多少.

SCB->VTOR = FLASH_BASE | 0x4000;

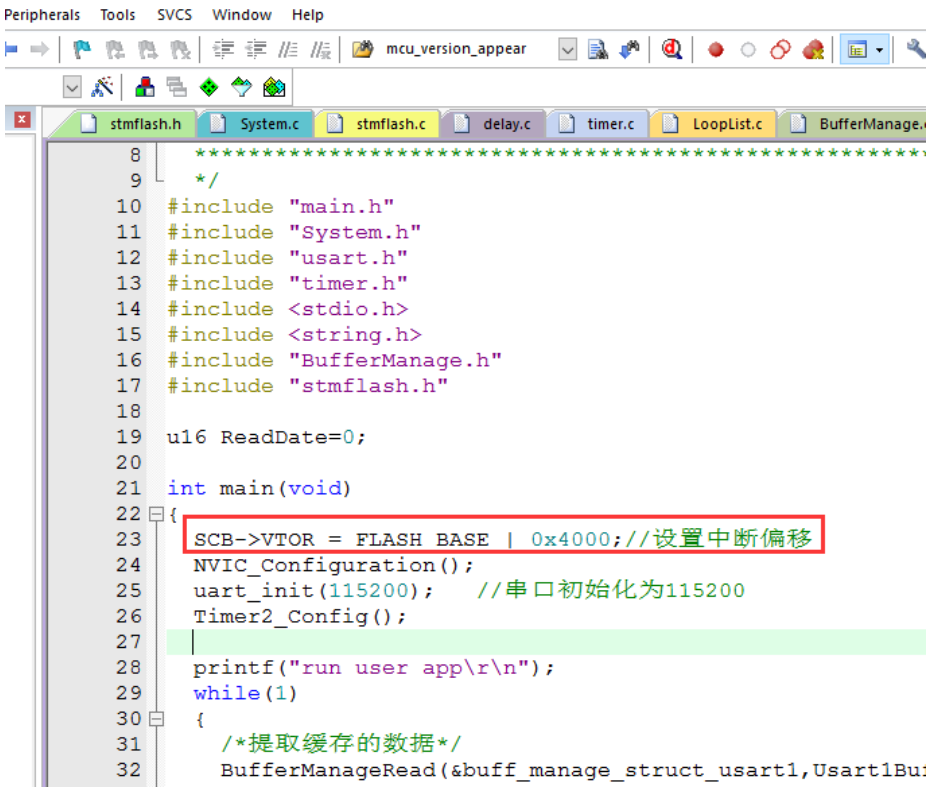
为什么需要设置中断偏移?

记住一句话:BootLoader里面配置的程序,即使执行了用户程序同样有效!

记住另一句话:所有的中断函数都有固定的地址入口!

假设BootLoader使用了某个中断,用户程序也使用了某个中断,如果不设置这个偏移,

那么用户程序和BootLoader就使用了同一个中断函数!



2.3 设置该程序文件运行的地址,及其大小

0x8004000 0xB800

解释:

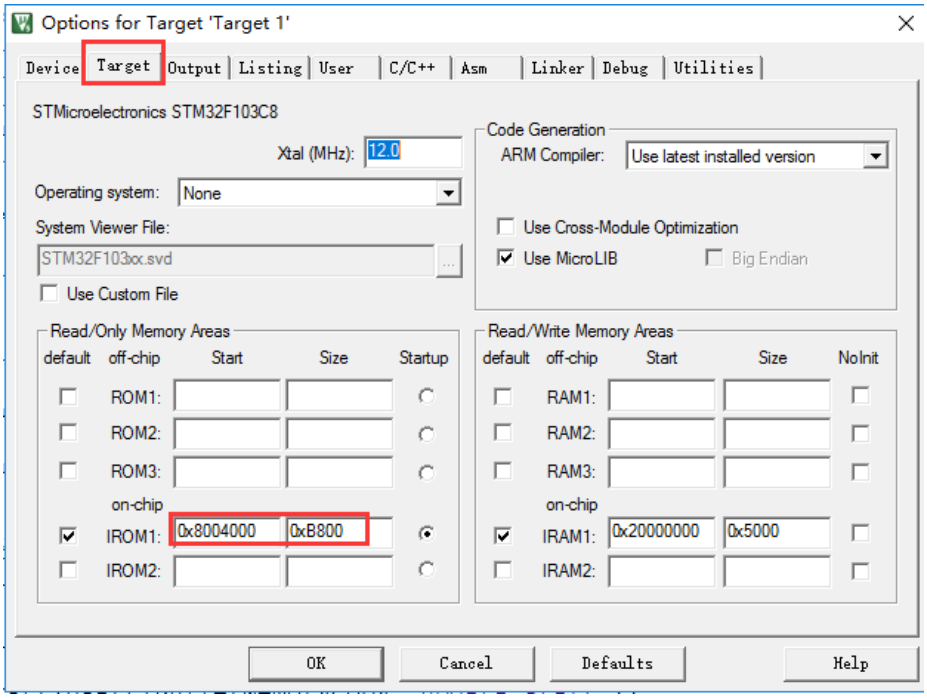
想让这个程序运行在某个地址上,必须在软件上设置一下,然后生成的文件才可以运行在这个地址上.

0x8004000 :这节的程序就希望程序运行在这个地址上

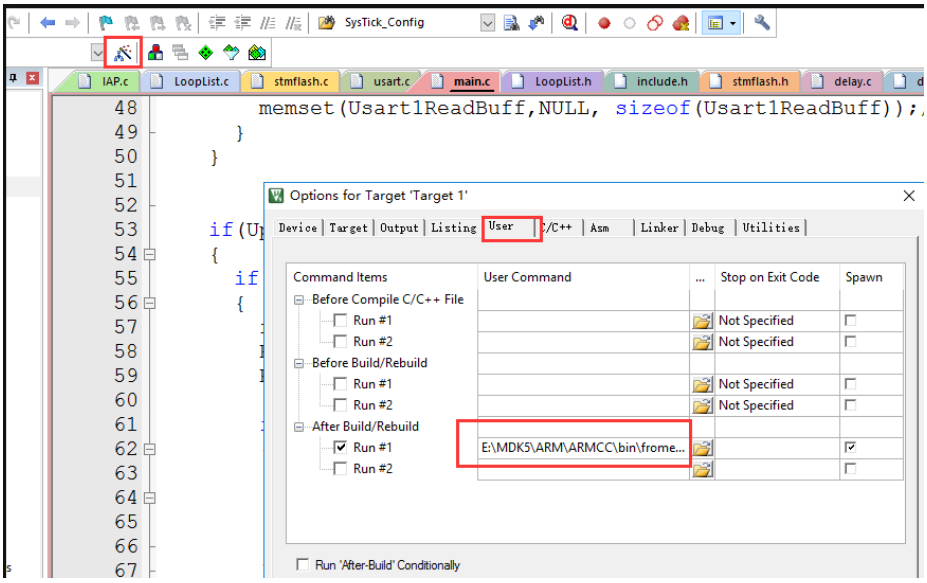
0xB800 : 我使用的是64KB的flash,从0x08000000 到 0x08004000 (16KB) 用于存储BootLoader程序

从 0x0800f800 到 0x0801000 (最后2KB) 用于存储其它数据

用户程序剩余 = 64KB - 16KB - 2KB = 46KB = 46*1024字节 = 47104字节 = 0xB800(16进制表示)



2.4让软件生成bin文件



我写的是

E:\MDK5\ARM\ARMCC\bin\fromelf.exe --bin -o .\Progect\user.bin .\output\Progect.axf

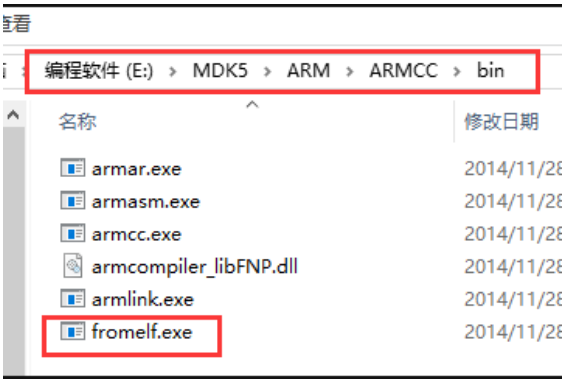
E:\MDK5\ARM\ARMCC\bin\fromelf.exe --bin -o 这是命令,意思是让软件生成bin文件

前面这个路径和自己安装软件的路径有关,下面是我的路径

根据自己软件安装的路径修改,否则编译报错!

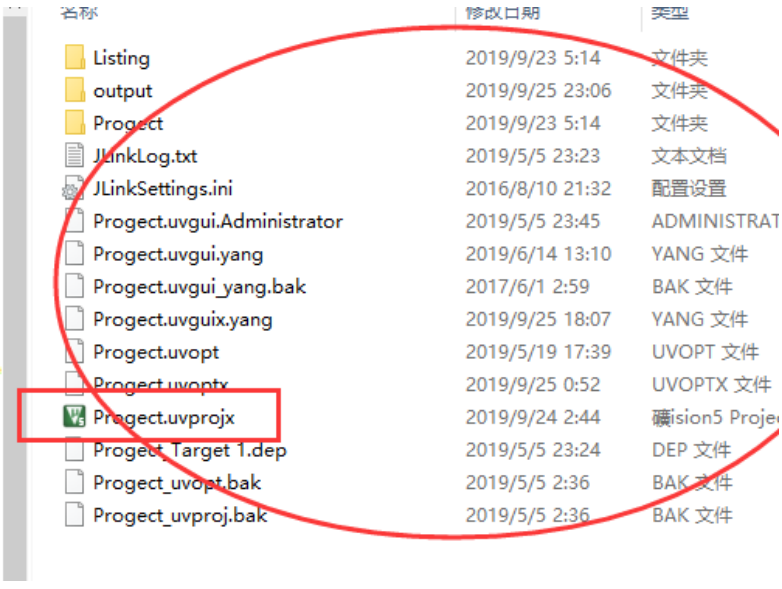
根据自己软件安装的路径修改,否则编译报错!

根据自己软件安装的路径修改,否则编译报错!



.\Progect\user.bin 就是生成的bin文件名字是 user.bin ,路径是工程目录的 Progect文件夹里面(如果没有则会自动建个文件夹)

所谓工程目录



.\output\Progect.axf 当前工程目录的output文件夹里面的Progect.axf文件

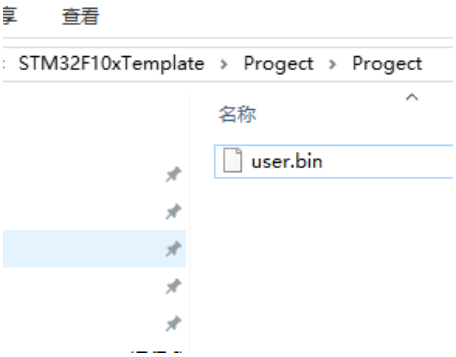
工程都会生成xxxx.axf文件,我的这个文件就生成在上面的目录里面

整理下就是

E:\MDK5\ARM\ARMCC\bin\fromelf.exe --bin -o .\Progect\user.bin .\output\Progect.axf

用fromelf.exe里面的--bin -o指令,把Progect.axf文件里面的bin数据提取出来以后生成一个新的文件 user.bin

点击编译便生成了 user.bin文件



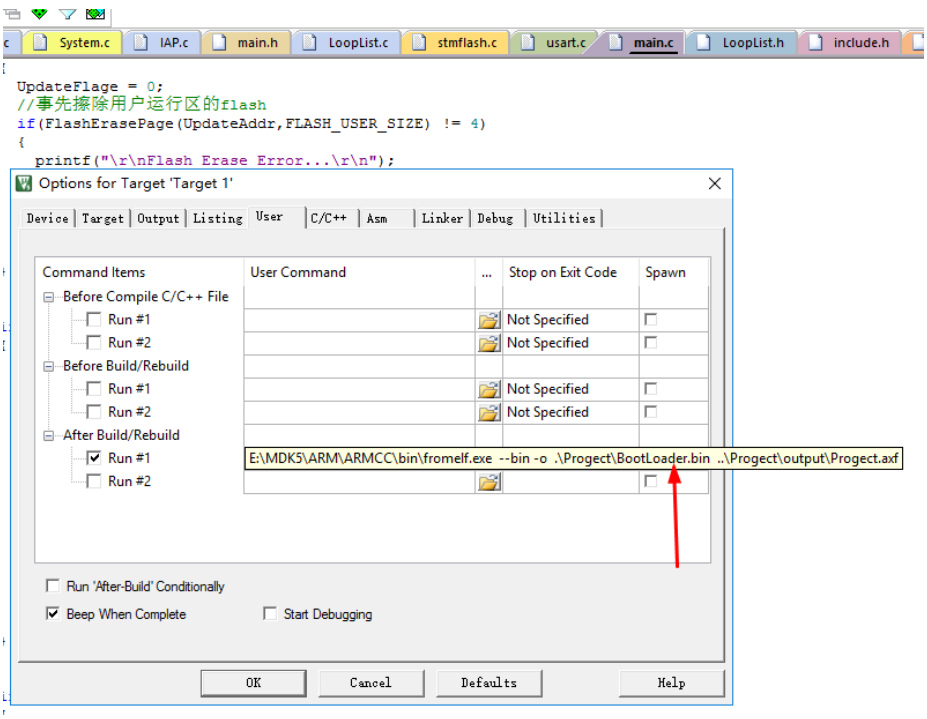
也有简洁的方式: $K\ARM\ARMCC\bin\fromelf.exe --bin --output=Bin\user.bin !L

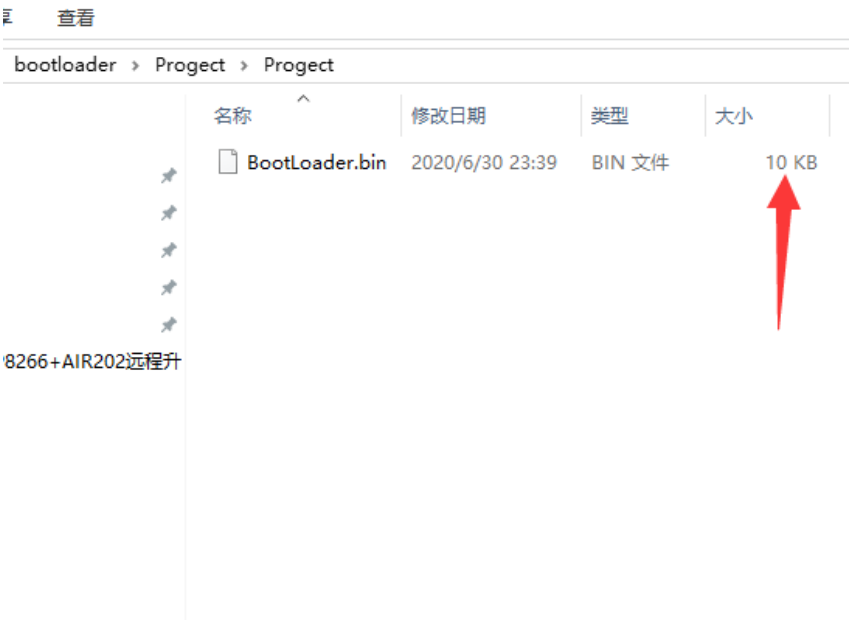
上面的意思是在工程目录的Bin文件夹里面生成user.bin文件

如何分配flash

1.首先需要明确,BootLoader程序是烧写到单片机里面永远不变的!

写完BootLoader程序以后,生成bin文件,看一下bin文件大小





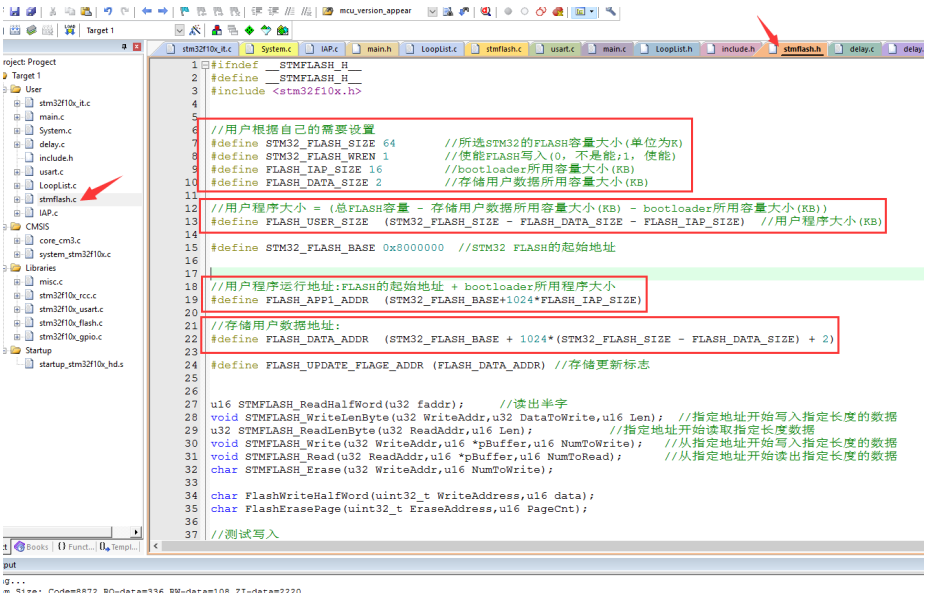
BootLoader是10KB,那么可以规定12KB(高容量单片机是2KB作为一页,用偶数可兼容全系列)

2.接下来是确定下存储其它用户数据所用flash大小

这个需要根据自己的项目自行规定大小,一般都是把数据放到最后的地址存储

3.以上的确定下来之后,剩余中间部分就是作为用户程序的了

为了便于修改,我做了以下程序



```
#ifndef __STMFLASH_H__
#define __STMFLASH_H__
#include <stm32f10x.h>

//用户根据自己的需要设置
#define STM32_FLASH_SIZE 64         //所选STM32的FLASH容量大小(单位为K)
#define STM32_FLASH_WREN 1          //使能FLASH写入(0，不是能;1，使能)
#define FLASH_IAP_SIZE 16           //bootloader所用容量大小(KB)
#define FLASH_DATA_SIZE 2           //存储用户数据所用容量大小(KB)

//用户程序大小 = (总FLASH容量 - 存储用户数据所用容量大小(KB) - bootloader所用容量大小(KB))
#define FLASH_USER_SIZE  (STM32_FLASH_SIZE - FLASH_DATA_SIZE - FLASH_IAP_SIZE)  //用户程序大小(KB)

#define STM32_FLASH_BASE 0x8000000  //STM32 FLASH的起始地址

//用户程序运行地址:FLASH的起始地址 + bootloader所用程序大小
#define FLASH_APP1_ADDR  (STM32_FLASH_BASE+1024*FLASH_IAP_SIZE)

//存储用户数据地址:
#define FLASH_DATA_ADDR  (STM32_FLASH_BASE + 1024*(STM32_FLASH_SIZE - FLASH_DATA_SIZE) + 2)

#define FLASH_UPDATE_FLAGE_ADDR (FLASH_DATA_ADDR) //存储更新标志


u16 STMFLASH_ReadHalfWord(u32 faddr);     //读出半字
void STMFLASH_WriteLenByte(u32 WriteAddr,u32 DataToWrite,u16 Len);  //指定地址开始写入指定长度的数据
u32 STMFLASH_ReadLenByte(u32 ReadAddr,u16 Len);           //指定地址开始读取指定长度数据
void STMFLASH_Write(u32 WriteAddr,u16 *pBuffer,u16 NumToWrite);   //从指定地址开始写入指定长度的数据
void STMFLASH_Read(u32 ReadAddr,u16 *pBuffer,u16 NumToRead);      //从指定地址开始读出指定长度的数据
char STMFLASH_Erase(u32 WriteAddr,u16 NumToWrite);

char FlashWriteHalfWord(uint32_t WriteAddress,u16 data);
char FlashErasePage(uint32_t EraseAddress,u16 PageCnt);

//测试写入
```

在BootLoader程序开始运行,会打印下当前的配置

stm32f10x_it.c | System.c | IAP.c | main.h | LoopList.c | stmflash.c | usart.c | main.c | Loop

```
10  #include "stmflash.h"
11  #include "IAP.h"
12
13
14  u16  ReadDate=0;
15  char ReadDat[2]={0};
16  char UpdataStartFlage = 0;//是不是开始更新
17  char UpdataReadEndFlage=0;//是不是接收完用户程序
18  char FlashWriteErrFlage=0;
19
20  int i;
21  u32 UpdateAddrCnt=0;//更新累加使用
22  u32 UpdateAddr=0;//程序写入的首地址
23
24
25  u16 UpdateFlage =0;//获取存储的更新标志
26
27  int main(void)
28  {
29    rbCreate(&pRb,UpdateUsart1Buff,UpdateUsart1ReadLen);//创建接收环形队列
30    NVIC_Configuration();
31    uart_init(115200);    //串口初始化为115200
32    DelayInit();
33
34    printf("\r\nUserROMStart:%x\r\n",FLASH_APP1_ADDR);//用户程序运行的地址
35    printf("userROMSize:%X\r\n",(FLASH_USER_SIZE)*1024);//设置的用户程序占用flash大小
36    printf("\r\nUserDATAStart:%x\r\n",FLASH_DATA_ADDR);//保存用户数据的地址
37
38    UpdateAddr = FLASH_APP1_ADDR;//用户程序运行的地址
39    UpdateAddrCnt = FLASH_APP1_ADDR;//用户程序运行的地址
40
41
42    STMFLASH_Read(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//获取更新标志
43    if(ReadDate == 1)
44    {
45      ReadDate = 0;
46      STMFLASH_Write(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//清除记录的更新标志
```

XCOM V2.0

```
run user app
run user app
run user app
run user app
run user app
run user app
run user app
run user app

UserROMStart:8004000
userROMSize:B800

UserDATAStart:800f802

wait updata...

UserROMStart:8004000
userROMSize:B800

UserDATAStart:800f802
run user app
```

程序具体怎么写入的flash并运行

1.一开始得到了程序写入的地址

stm32f10x_it.c | System.c | IAP.c | main.h | LoopList.c | stmflash.c | usart.c | main.c

```
19
20  int i;
21  u32 UpdateAddrCnt=0;//更新累加使用
22  u32 UpdateAddr=0;//程序写入的首地址
23
24
25  u16 UpdateFlage =0;//获取存储的更新标志
26
27  int main(void)
28  {
29    rbCreate(&pRb,UpdateUsart1Buff,UpdateUsart1ReadLen);//创建接收环形队列
30    NVIC_Configuration();
31    uart_init(115200);   //串口初始化为115200
32    DelayInit();
33
34    printf("\r\nUserROMStart:%x\r\n",FLASH_APP1_ADDR);//用户程序运行的地址
35    printf("userROMSize:%X\r\n",(FLASH_USER_SIZE)*1024);//设置的用户程序占用flash大小
36    printf("\r\nUserDATAStart:%x\r\n",FLASH_DATA_ADDR);//保存用户数据的地址
37
38    UpdateAddr = FLASH_APP1_ADDR;//用户程序运行的地址
39    UpdateAddrCnt = FLASH_APP1_ADDR;//用户程序运行的地址
40
41
42    STMFLASH_Read(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//获取更新标志
43    if(ReadDate == 1)
44    {
45      ReadDate = 0;
46      STMFLASH_Write(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//清除记录的更新标志
47      UpdateFlage = 1;
48    }
49
50
51    while(1)
52    {
53      if(!UpdateFlage)//没有更新标志,尝试加载运行用户程序
54      {
55        if(!UpdataStartFlage)//没有在执行更新
```

2.接收到 updata start ;擦除用户程序区; UpdataStartFlage = 1;

```
70          Reset_MCU();//重启
71        }
72
73        UpdataStartFlage = 1;//置位升级标志
74        printf("\r\nwait updata...\r\n");
75      }
76
77
78      if(Usart1ReadFlage)
79      {
80        Usart1ReadFlage=0;
81        if(strstr(Usart1ReadBuff,"updata start"))//接收到更新指令
82        {
83          //事先擦除用户运行区的flash
84          if(FlashErasePage(UpdateAddr,FLASH_USER_SIZE) != 4)
85          {
86            printf("\r\nFlash Erase Error...\r\n");
87            Reset_MCU();//重启
88          }
89
90          UpdataStartFlage = 1;//置位开始升级标志
91          printf("\r\nwait updata...\r\n");
92
93          memset(Usart1ReadBuff,NULL, sizeof(Usart1ReadBuff));//清零
94        }
95      }
96
97
98      if(UpdataStartFlage == 1)
99      {
100       if(rbCanRead(&pRb)>1)//接收到更新程序就开始写入
101       {
```

3.串口助手发送程序数据时,把程序数据写入了环形队列

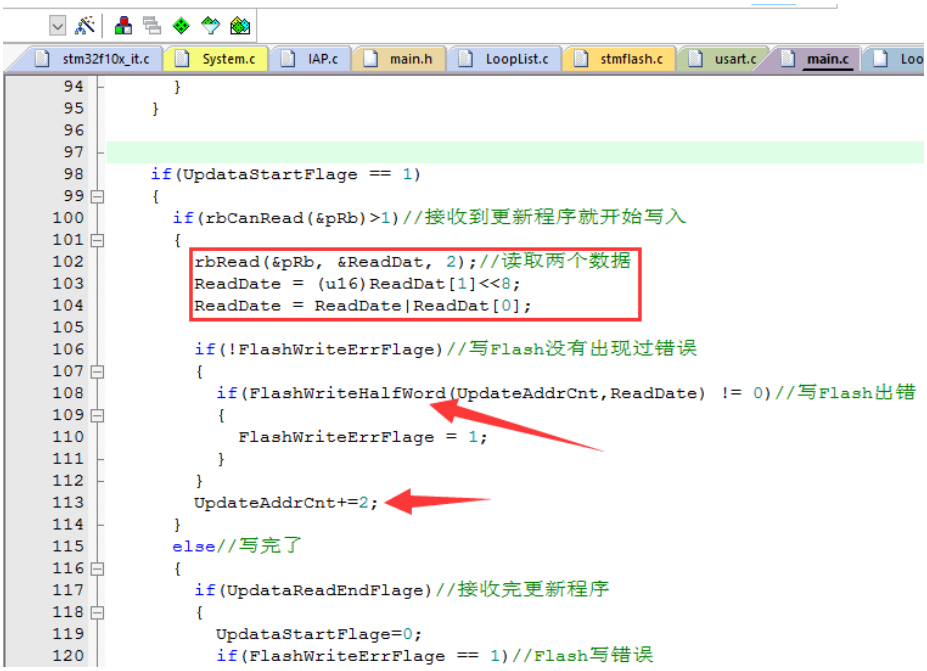
关于环形队列系列文章: https://mnifdv.cn/forum.php?mod=forumdisplay&fid=53

```
51
52
53  void USART1_IRQHandler(void)                    //串口1中断服务程序
54  {
55    u8 Res;
56    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
57    {
58      Res =USART_ReceiveData(USART1); //读取接收到的数据
59
60      if(Usart1ReadCnt < Usart1ReadLen-1)
61      {
62        Usart1ReadBuff[Usart1ReadCnt] = Res;
63      }
64      else
65      {
66        Usart1ReadCnt=0;
67      }
68      Usart1ReadCnt ++; //数据个数
69      Usart1IdleCnt = 0;
70
71      if(UpdataStartFlage)//如果置位升级标志
72      {
73        if(PutData(&pRb,&Res,1) == -1)//写入环形队列
74        {
75          UpdateOverflow = 1;//环形队列溢出
76        }
77      }
78    }
79  }
80
```

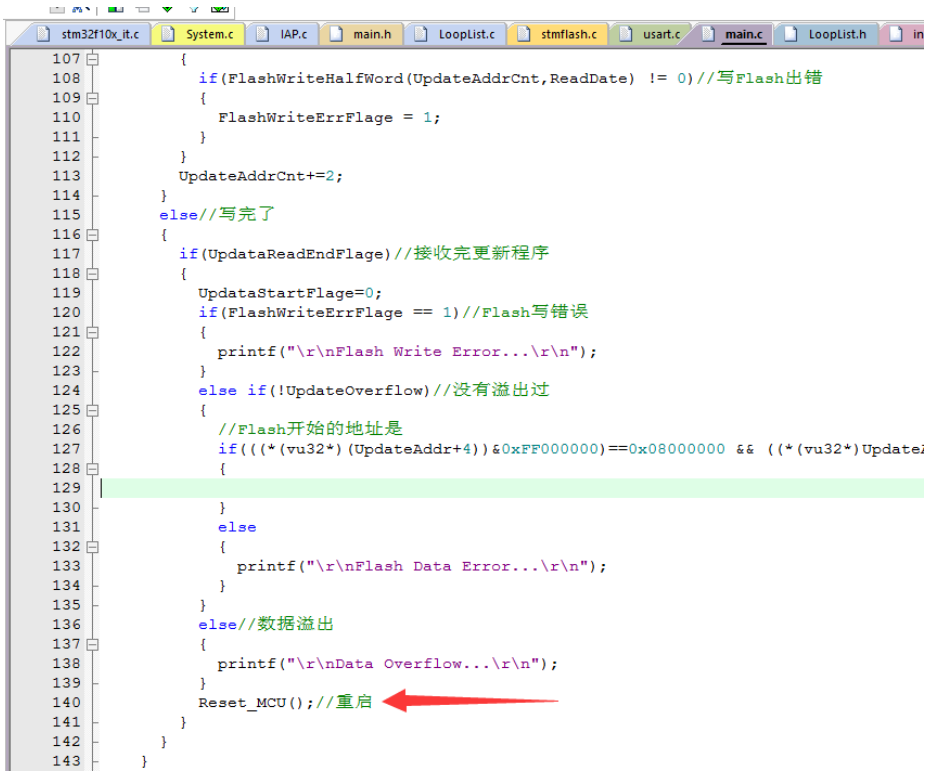
4.主函数取出数据拼接成16位数据以后写入flash

从0x08004000开始写入,地址每次累加2

注:STM32写入flash每次需要写16位数据

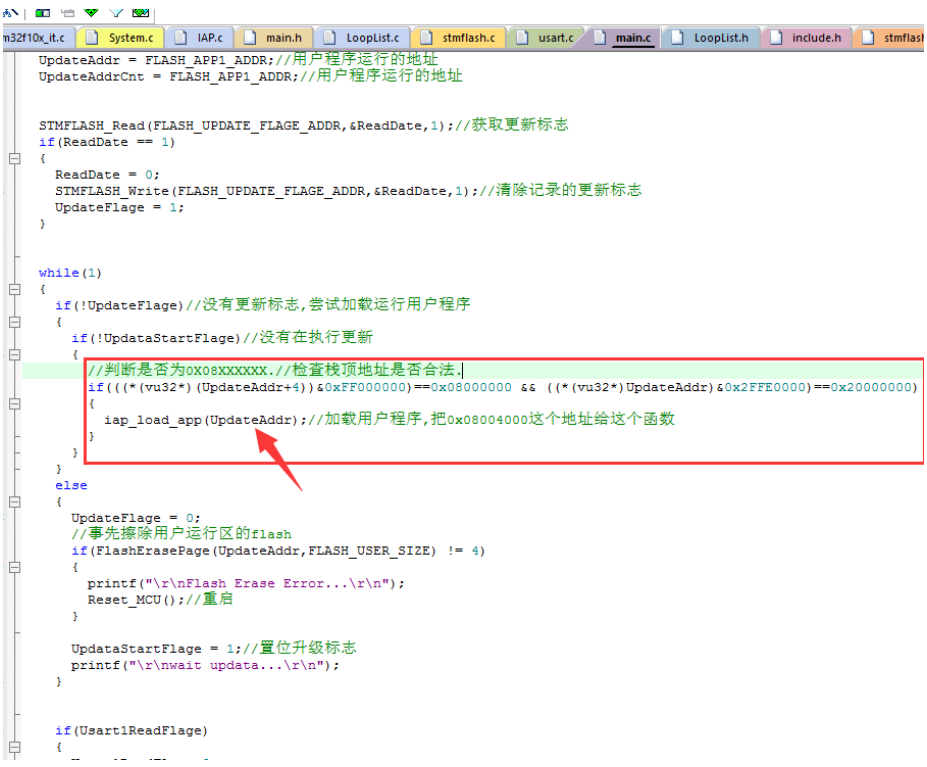


5.接收完数据,打印下相应的错误,重启.



```
        {
          if(FlashWriteHalfWord(UpdateAddrCnt,ReadDate) != 0)//写Flash出错
          {
            FlashWriteErrFlage = 1;
          }
        }
        UpdateAddrCnt+=2;
      }
      else//写完了
      {
        if(UpdataReadEndFlage)//接收完更新程序
        {
          UpdataStartFlage=0;
          if(FlashWriteErrFlage == 1)//Flash写错误
          {
            printf("\r\nFlash Write Error...\r\n");
          }
          else if(!UpdateOverflow)//没有溢出过
          {
            //Flash开始的地址是
            if(((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000 && ((*(vu32*)Updatei
            {

            }
            else
            {
              printf("\r\nFlash Data Error...\r\n");
            }
          }
          else//数据溢出
          {
            printf("\r\nData Overflow...\r\n");
          }
          Reset_MCU();//重启
        }
      }
    }
```

6.重启以后,判断了用户地址里面有用户程序,加载用户程序



```
UpdateAddr = FLASH_APP1_ADDR;//用户程序运行的地址
UpdateAddrCnt = FLASH_APP1_ADDR;//用户程序运行的地址


STMFLASH_Read(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//获取更新标志
if(ReadDate == 1)
{
  ReadDate = 0;
  STMFLASH_Write(FLASH_UPDATE_FLAGE_ADDR,&ReadDate,1);//清除记录的更新标志
  UpdateFlage = 1;
}


while(1)
{
  if(!UpdateFlage)//没有更新标志,尝试加载运行用户程序
  {
    if(!UpdataStartFlage)//没有在执行更新
    {
      //判断是否为0x08xxxxxx.//检查栈顶地址是否合法.
      if(((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000 && ((*(vu32*)UpdateAddr)&0x2FFE0000)==0x20000000)
      {
        iap_load_app(UpdateAddr);//加载用户程序,把0x08004000这个地址给这个函数
      }
    }
  }
  else
  {
    UpdateFlage = 0;
    //事先擦除用户运行区的flash
    if(FlashErasePage(UpdateAddr,FLASH_USER_SIZE) != 4)
    {
      printf("\r\nFlash Erase Error...\r\n");
      Reset_MCU();//重启
    }

    UpdataStartFlage = 1;//置位升级标志
    printf("\r\nwait updata...\r\n");
  }


  if(Usart1ReadFlage)
  {
```

细节说明

1.环形队列大小5字节

就是说,只使用了5字节就接收处理了全部的程序文件!

main.c | stm32f10x_it.c | System.c | IAP.c | startup_stm32f10x_hd.s | usart.h

```
11  #include "IAP.h"
12
13
14  u16  ReadDate=0;
15  char ReadDat[2]={0};
16  char UpdataStartFlage = 0;//是不是开始更新
17  char UpdataReadEndFlage=0;//是不是接收完用户程序
18  char FlashWriteErrFlage=0;
19
20  int i;
21  u32 UpdateAddrCnt=0;//更新累加使用
22  u32 UpdateAddr=0;//程序写入的首地址
23
24
25  u16 UpdateFlage =0;//获取存储的更新标志
26
27  int main(void)
28  {
29    rbCreate(&pRb,UpdateUsart1Buff,UpdateUsart1ReadLen);//创建接收环形队列
30    NVIC_Configuration();
31    uart_init(115200);   //串口初始化为115200
32    DelayInit();
33
34    printf("\r\nUserROMStart:%x\r\n",FLASH_APP1_ADDR);//用户程序运行的地址
35    printf("userROMSize:%X\r\n",(FLASH_USER_SIZE)*1024);//设置的用户程序占
36    printf("\r\nUserDATAStart:%x\r\n",FLASH_DATA_ADDR);//保存用户数据的地址
37
38    UpdateAddr = FLASH_APP1_ADDR;//用户程序运行的地址
39    UpdateAddrCnt = FLASH_APP1_ADDR;//用户程序运行的地址
```

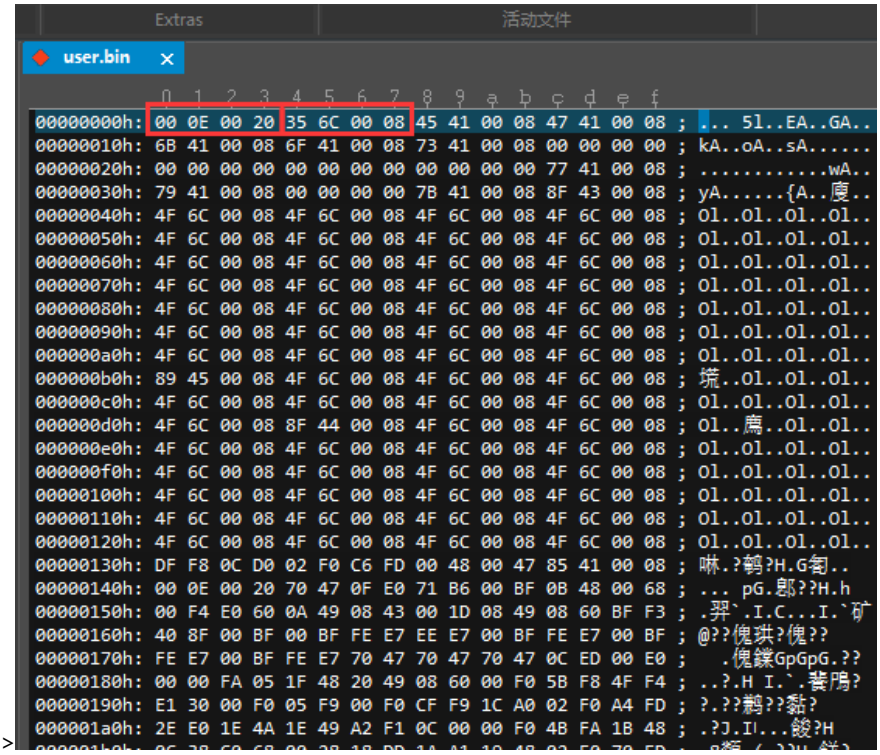
mcu_version_appear

main.c | stm32f10x_it.c | System.c | IAP.c | startup_stm32f10x_hd.s | usart.h

```
1  #ifndef __USART_H
2  #define __USART_H
3  #include <stm32f10x.h>
4  #ifndef USART_C_//如果没有定义
5  #define USART_C_ extern
6  #else
7  #define USART_C_
8  #endif
9
10
11 #define UpdateUsart1ReadLen       5    //定义最大接收字节数(缓冲程序)
12
13 #define Usart1ReadLen 150
14
15 USART_C_ u8  UpdateStartFlage;
16
17
18 USART_C_ char  Usart1ReadBuff[Usart1ReadLen];  //接收数据缓存
19 USART_C_ u32 Usart1ReadCnt;//串口1接收到的数据个数
20 USART_C_ u32 Usart1ReadCntCopy;//串口1接收到的数据个数
21 USART_C_ u32 Usart1IdleCnt;//空闲检测用
22 USART_C_ u8  Usart1ReadFlage;//串口1接收到一条完整数据
23
24 USART_C_ u8  UpdateUsart1Buff[UpdateUsart1ReadLen];  //接收数据缓存
25 USART_C_ u8  UpdateOverflow;//是不是溢出
26
```

2.关于 if(((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000 && ((*(vu32*)UpdateAddr)&0x2FFE0000)==0x20000000)这句话是判断程序文件

实际上就是判断的程序文件的这两个位置



前面的四位flash地址记录的值 00 0E 00 20 是记录的整个程序占用RAM空间的最高地址(STM32默认的)

STM32的RAM是从 20000000 开始

注意:STM32是小端模式,低位存在低位,高位存在高位

0x20000E00　　20 00 0E 00　就是说总共使用了 0E 00 (3584字节的RAM)

((*(vu32*)UpdateAddr)&0x2FFE0000)==0x20000000

上面的判断其实就是判断了下是不是 20 00

后面的 08 00 6C 35

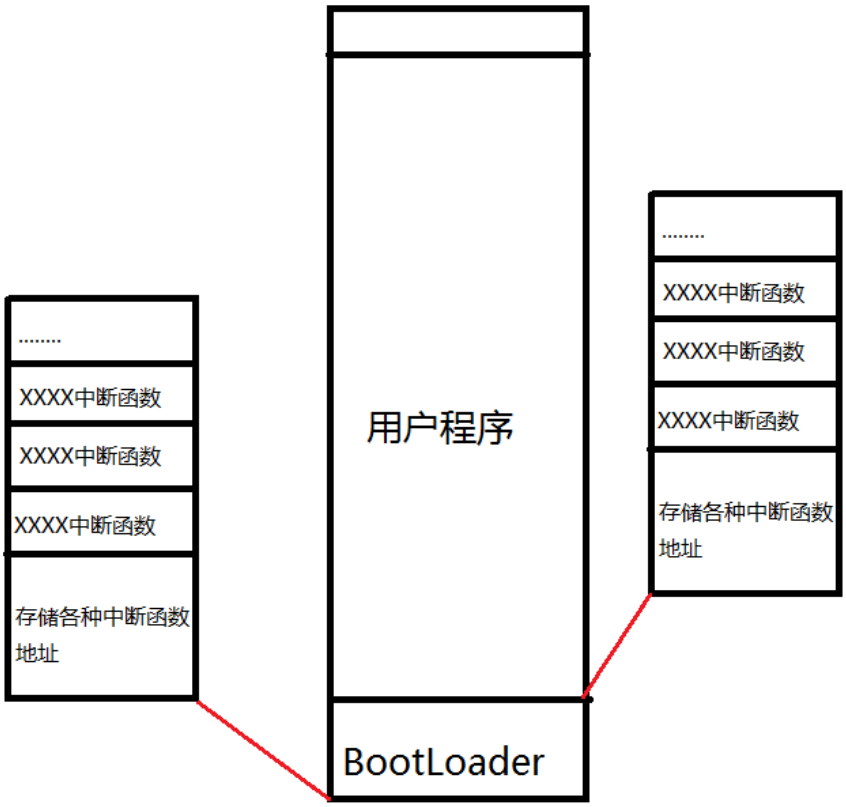
实际上这个存储的是复位中断入口的地址.(STM32默认的后面四位存储的是复位中断入口的地址)

当然再往后08 00 41 45 是不可屏蔽中断函数的地址

再往后 08 00 41 47 是硬件错误中断函数的地址

咱所有的程序都是存储在flash里面的,复位中断函数也不例外

其实是下面这个样子



BootLoader和用户程序的flash里面具体存储的都是按照上面的图示.

程序是存储在flash里面的,flash的地址肯定是从 0x08000000 - XXXXXX,所以才会有了

((*(vu32*)(UpdateAddr+4))&0xFF000000)==0x08000000

主要判断的是不是 最高位是不是08

3.一定要记住一件事情:

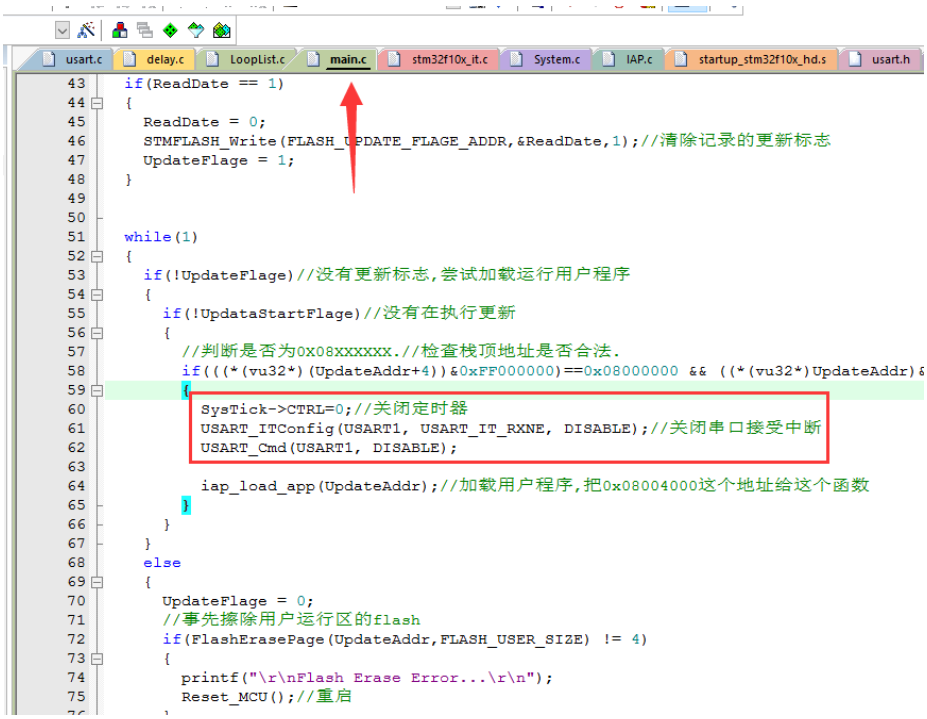
我上面说过一句话: BootLoader里面配置的程序,即使执行了用户程序同样有效!

仔细考虑这句话!

假设在BootLoader里面使用了中断定时器,用户程序里面没有使用,跳转到用户程序以后定时器还在运行!

但是由于所有的变量全部重新分配,导致凡是定时器中断里面的变量都没有了!从而导致死机!

所以在跳转用户程序的时候关闭了BootLoader里面使用的中断



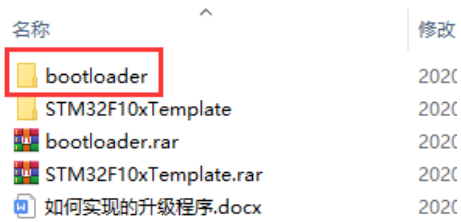
如何下载用户程序到单片机

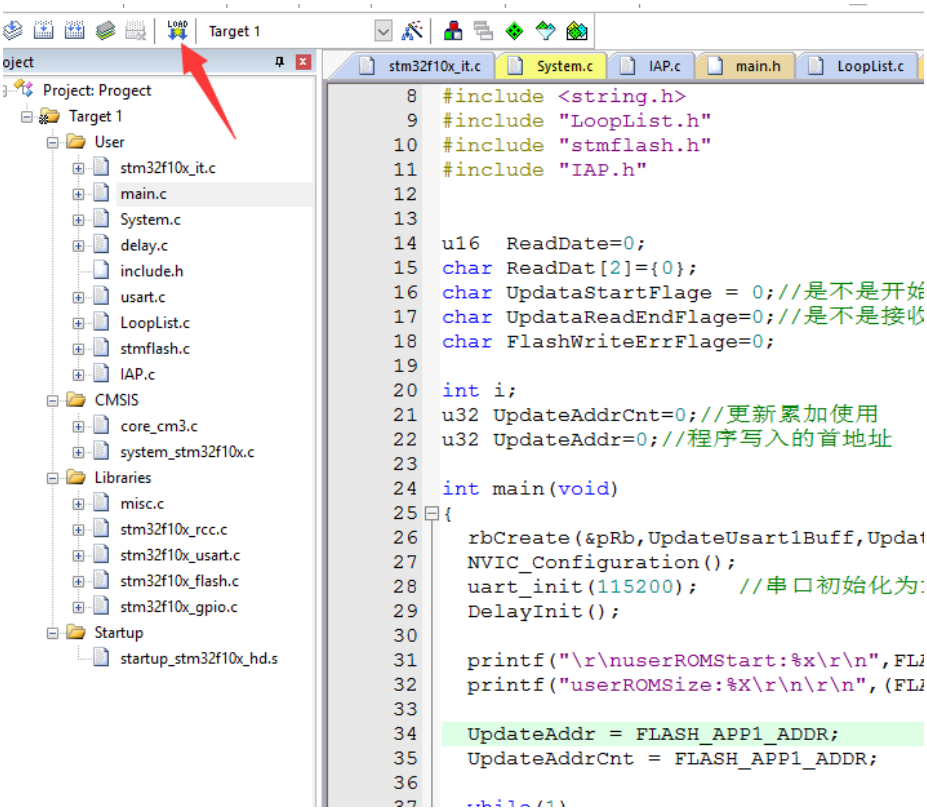
方式1(使用软件下载)

上面的例子是先下载BootLoader程序,然后把用户程序升级进去

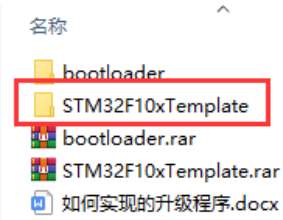
现在说一下在下载完BootLoader程序以后,如何把用户程序下载进去运行

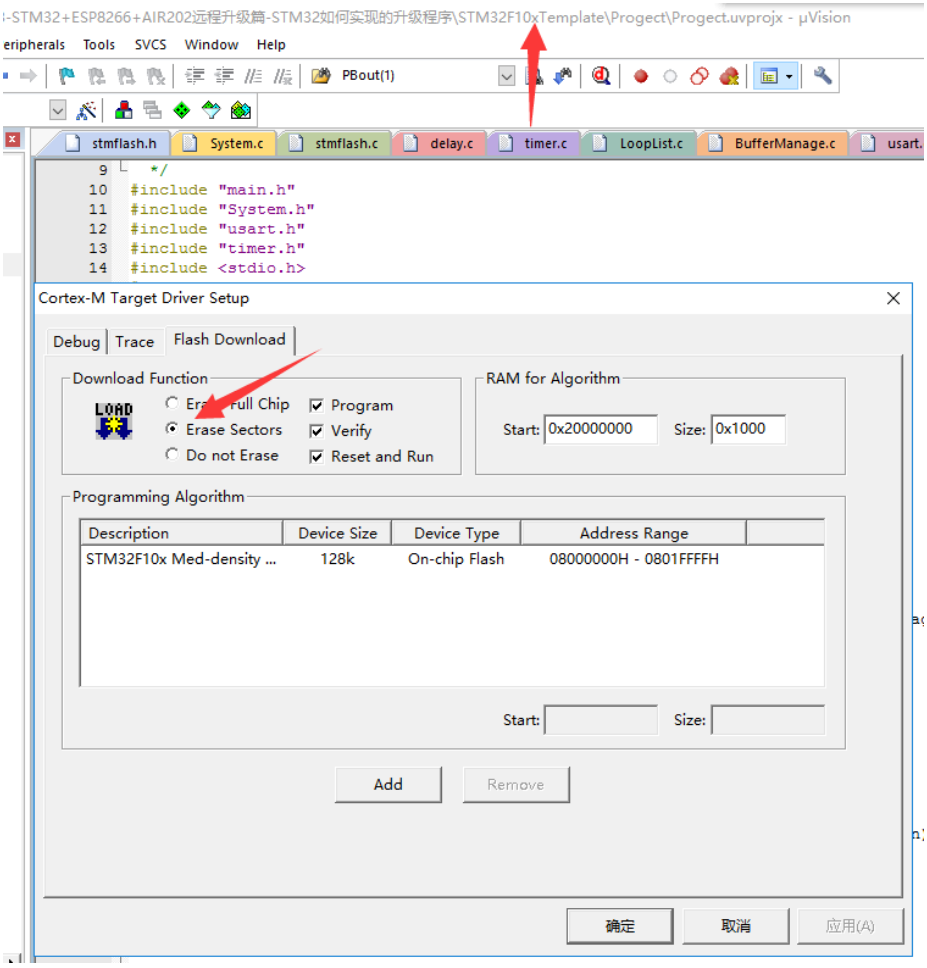
1.下载BootLoader程序到单片机



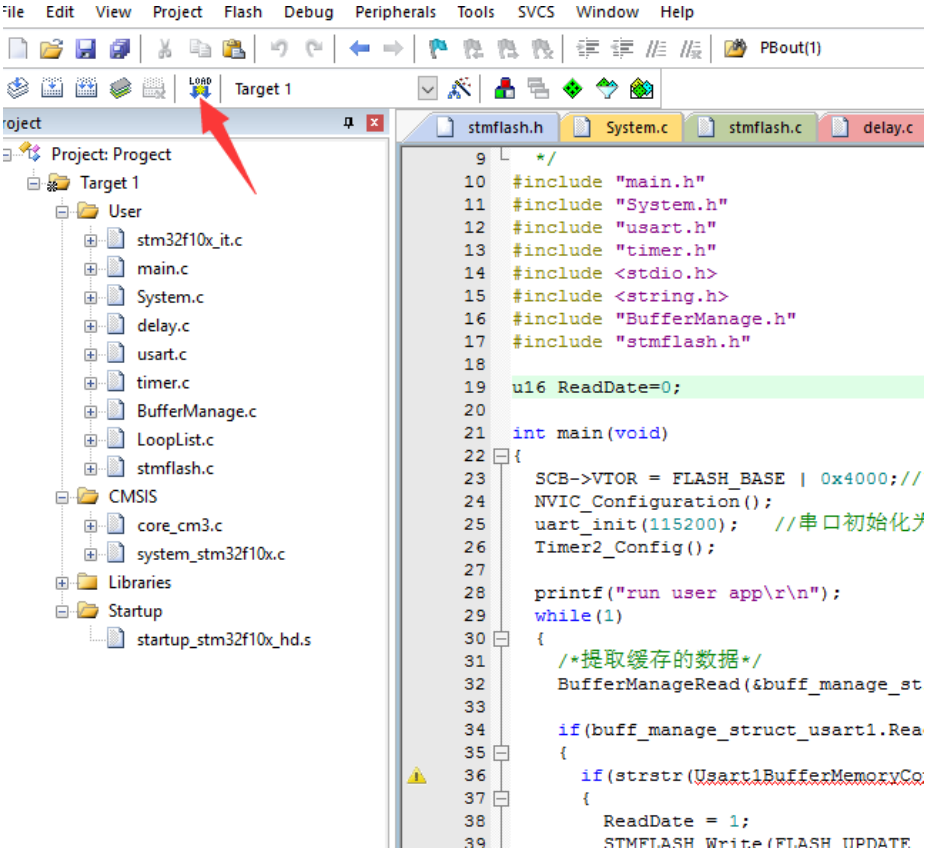


2.打开用户程序,调整用户程序的下载设置,只擦除使用的部分

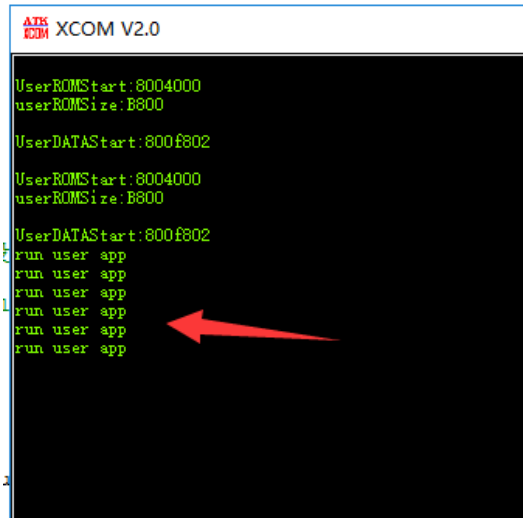



-STM32+ESP8266+AIR202远程升级篇-STM32如何实现的升级程序\STM32F10xTemplate\Progect\Progect.uvprojx - µVision
eripherals Tools SVCS Window Help
PBout(1)
stmflash.h System.c stmflash.c delay.c timer.c LoopList.c BufferManage.c usart.
*/
#include "main.h"
#include "System.h"
#include "usart.h"
#include "timer.h"
#include <stdio.h>
Cortex-M Target Driver Setup
Debug Trace Flash Download
Download Function
Full Chip
Erase Sectors
Do not Erase
Program
Verify
Reset and Run
RAM for Algorithm
Start: 0x20000000
Size: 0x1000
Programming Algorithm
Description
Device Size
Device Type
Address Range
STM32F10x Med-density ...
128k
On-chip Flash
08000000H - 0801FFFFH
Start:
Size:
Add
Remove
确定
取消
应用(A)

3.然后点击下载

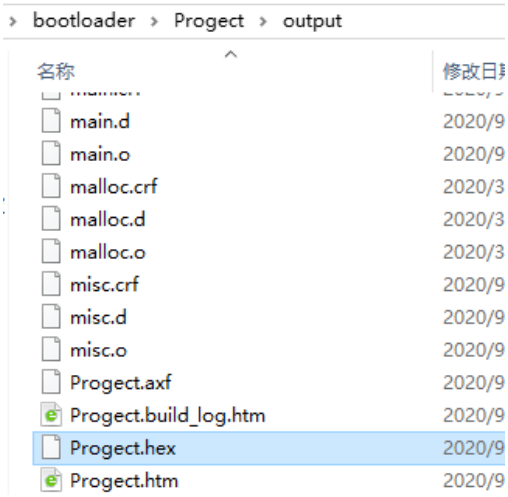


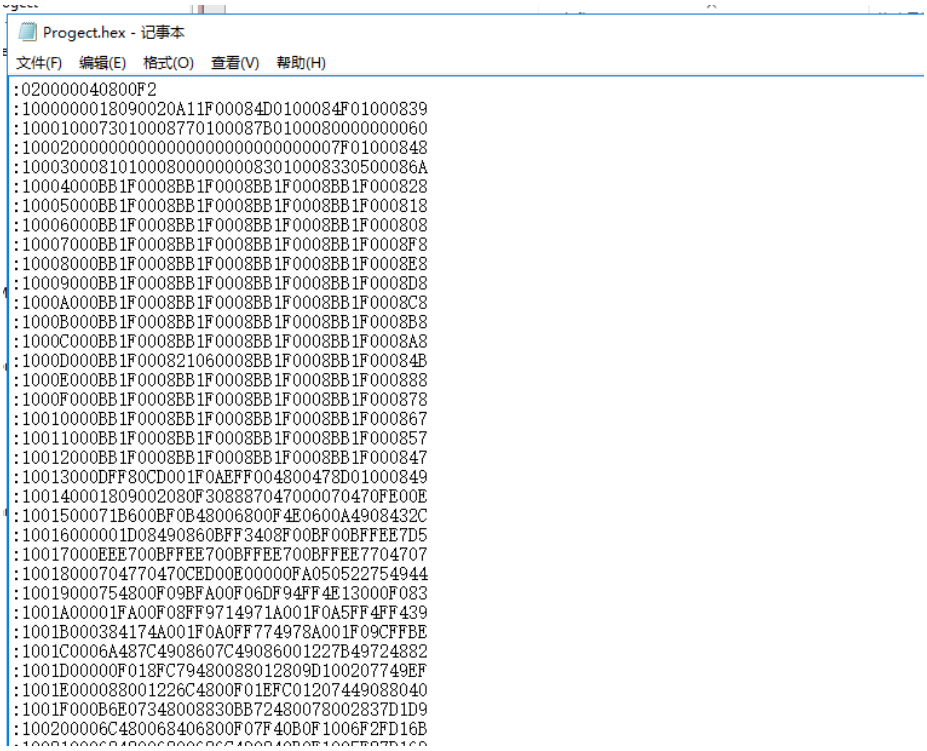
4.下载成功以后将会看到正在运行用户程序



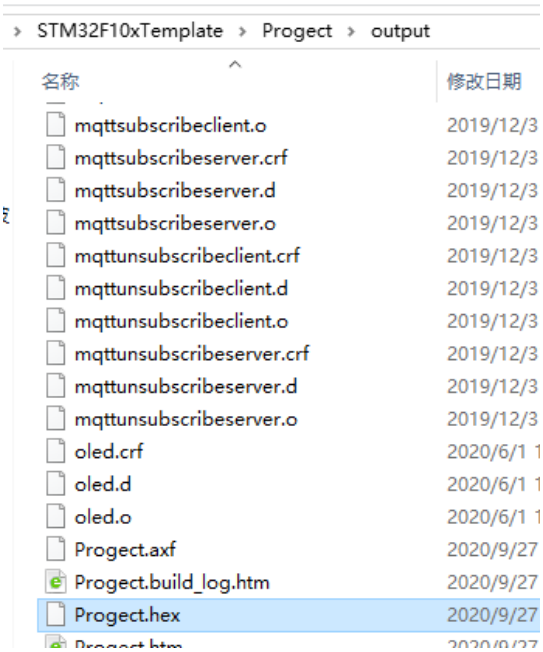
方式2(合并Hex文件)

1.用记事本打开BootLoader程序的hex文件

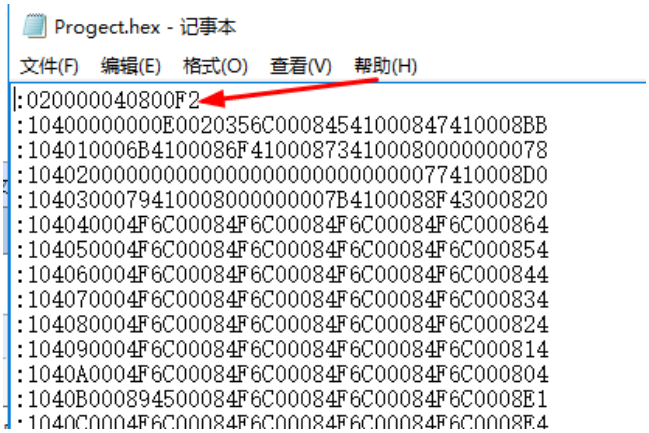


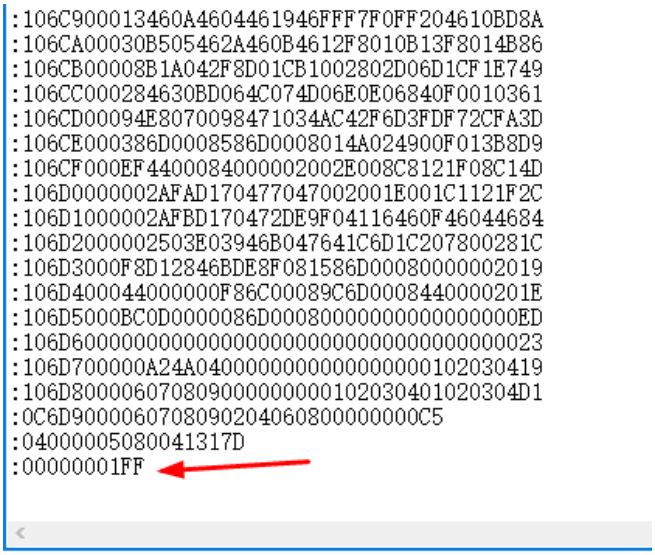


2.用记事本打开用户程序的hex文件

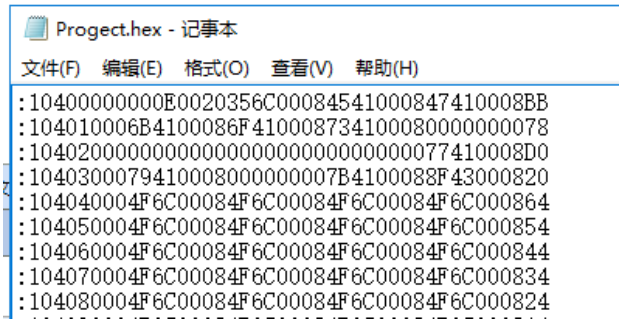


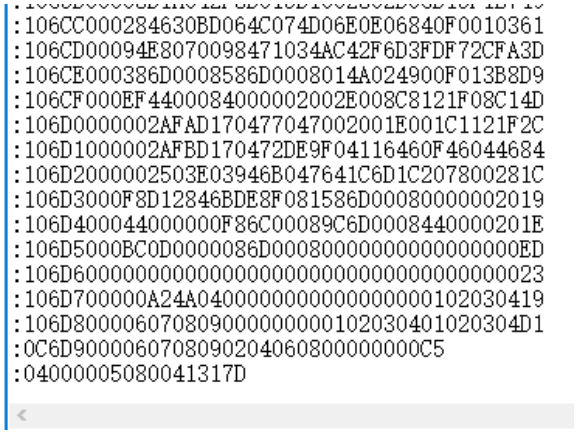
3.删除用户程序的hex数据的第一行和最后一行



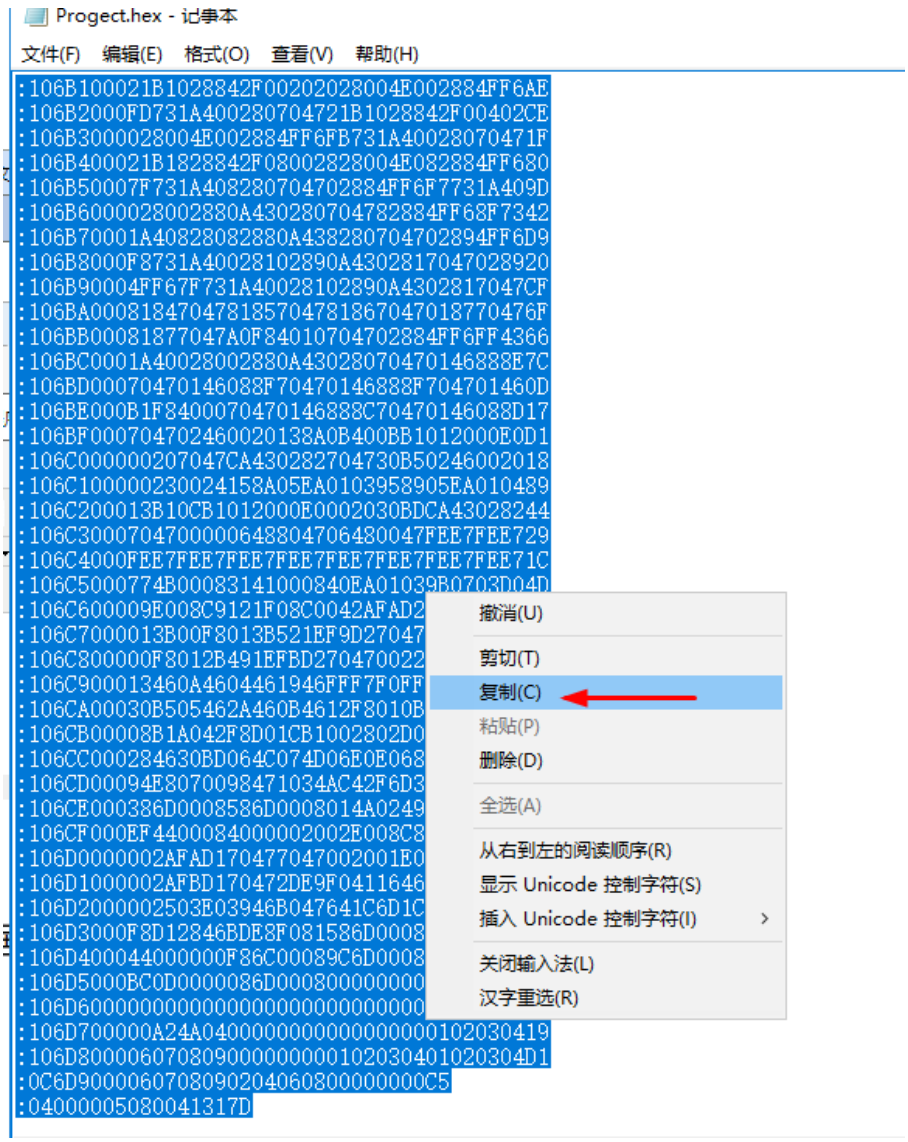


4.最终用户程序





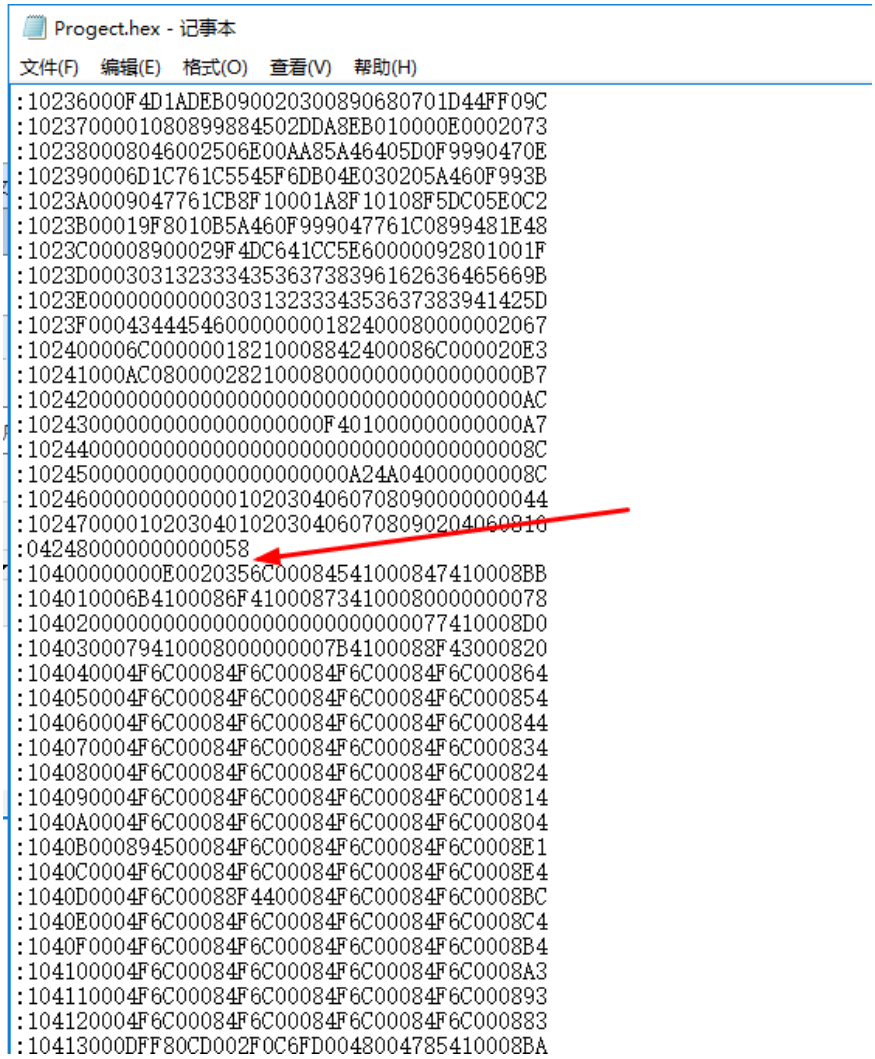
5.复制修改后的用户程序的hex数据(全部复制)



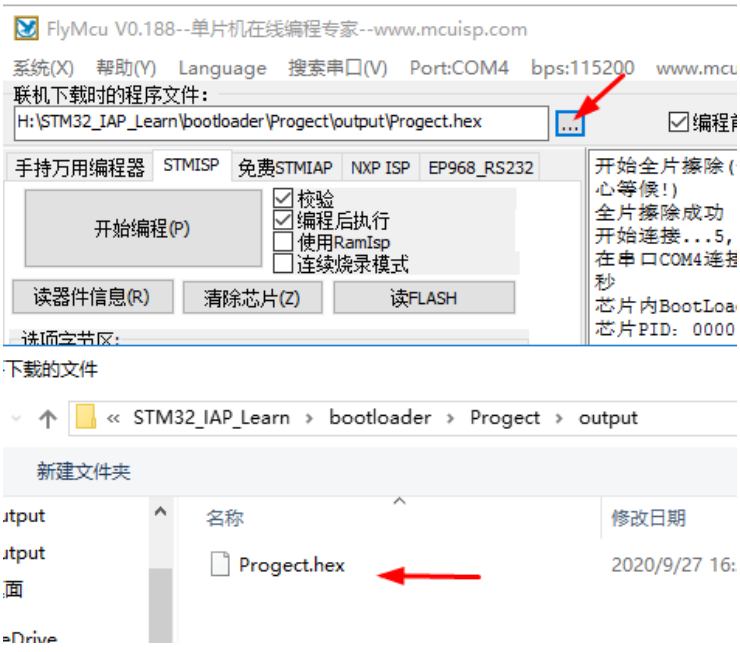
6.把复制的数据粘贴到BootLoader文件的下面的位置

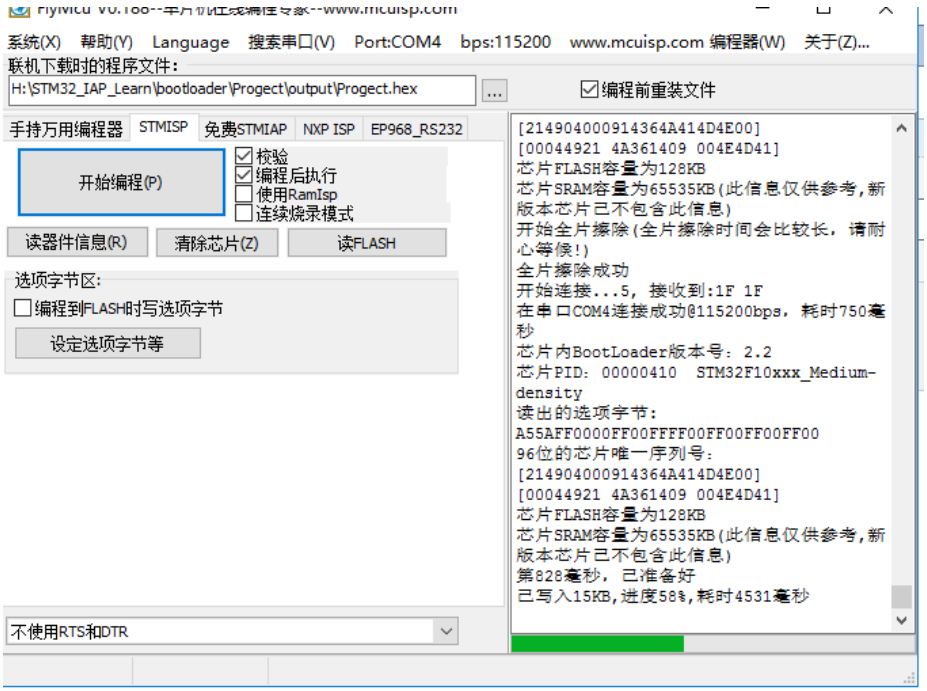

Progect.hex - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
:10222000040B8DF8000000208DF80100E946012024
:1022300003E057F8049B4FF0FF3069074FF00005AB
:1022400001D40AE06D1C45450DDA8542FADB19F828
:1022500005100029F6D106E06D1C8542FCDB19F85B
:1022600005100029F8D12E4404E019F8010B5A4654
:102270000F9990476D1EF8D2A5E00A200021CDE904
:10228000080lC5F3025002280DD001CFC117031E6B
:1022900071F100030DDA4FF0000CD0EB0C006CEB89
:1022A00001012D2308E0FF1D27F00707F7E80201D1
:1022B000EDE72B0504D52B238DF8283001233BE0D7
:1022C000EB0739D02023F7E70A2000E01020002197
:1022D000CDE9080107E01020002145F00405CDE913
:1022E00008014FF00808C5F3025002280FD001CFB3
:1022F00000214FF0000A2B071FD5702A0DD0DDE911
:1023000008C38CF0100C5CEA030C0CD015E0FF1D28
:1023100027F00707F7E80201EBE740228DF82820B5
:102320004FF0010A0BE050EA010306D030238DF88C
:1023300028308DF8292002239A46582A04D024A256
:102340000DF120090B9209E026A2F9E7DDE9082347
:10235000FFF76EFE0B9B9B5C09F8013D50EA010202
:10236000F4D1ADEB090020300890680701D44FF09C
:1023700001080899884502DDA8EB010000E0002073
:10238000804600250 6E00AA85A46405D0F9990470E
:102390006D1C761C5545F6DB04E030205A460F993B
:1023A0009047761CB8F10001A8F10108F5DC05E0C2
:1023B00019F8010B5A460F999047761C0899481E48
:1023C00008900029F4DC641CC5E60000092801001F
:1023D000303132333435363738396162636465669B
:1023E000000000000303132333435363738394142 5D
:1023F000434445460000000018240008000000 2067
:102400006C0000001821000884240008 6C000020E3
:10241000AC0800002821000800000000000000 B7
:10242000000000000000000000000000000000AC
:1024300000000000000000F401000000000000A7
:1024400000000000000000000000000000000008C
:1024500000000000000000000A24A040000000 8C
:1024600000000000001020304060708090000000044
:1024700001020304010203040607080902040608 16
:042480000000000 58
:04000005080001 3BD
:00000001FF

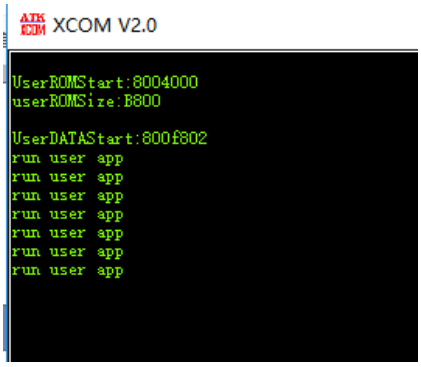
7.粘贴后的样子


Progect.hex - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
:10236000F4D1ADEB0900203008906807 01D44FF09C
:1023700001080899884502DDA8EB010000E0002073
:10238000804600250 6E00AA85A46405D0F9990470E
:102390006D1C761C5545F6DB04E030205A460F993B
:1023A0009047761CB8F10001A8F10108F5DC05E0C2
:1023B00019F8010B5A460F999047761C0899481E48
:1023C00008900029F4DC641CC5E60000092801001F
:1023D000303132333435363738396162636465669B
:1023E000000000003031323334353637383941425D
:1023F000434445460000000018240008000000 2067
:102400006C0000001821000884240008 6C000020E3
:10241000AC0800002821000800000000000000 00B7
:10242000000000000000000000000000000000 00AC
:1024300000000000000000000F401000000000000A7
:10244000000000000000000000000000000000008C
:1024500000000000000000000A24A040000000 08C
:10246000000000001020304060708090000000044
:10247000010203040102030406070809020406 0810
:042480000000000058
:10400000000E0020356C0008454100084741000 8BB
:104010006B4100086F4100087341000800000000 78
:10402000000000000000000000000077410008D0
:1040300079410008000000007B4100088F43000820
:104040004F6C00084F6C00084F6C00084F6C000864
:104050004F6C00084F6C00084F6C00084F6C000854
:104060004F6C00084F6C00084F6C00084F6C000844
:104070004F6C00084F6C00084F6C00084F6C000834
:104080004F6C00084F6C00084F6C00084F6C000824
:104090004F6C00084F6C00084F6C00084F6C000814
:1040A0004F6C00084F6C00084F6C00084F6C000804
:1040B000894500084F6C00084F6C00084F6C0008E1
:1040C0004F6C00084F6C00084F6C00084F6C0008E4
:1040D0004F6C00088F4400084F6C00084F6C0008BC
:1040E0004F6C00084F6C00084F6C00084F6C0008C4
:1040F0004F6C00084F6C00084F6C00084F6C0008B4
:104100004F6C00084F6C00084F6C00084F6C0008A3
:104110004F6C00084F6C00084F6C00084F6C000893
:104120004F6C00084F6C00084F6C00084F6C000883
:10413000DFF80CD002F0C6FD00480047854 10008BA

8.然后把组合后的hex文件下载到单片机里面即可







结语

这节主要目的是让用户彻底了解单片机是如何更新的程序

这节的程序只能作为学习参考不能作为项目应用!

后面的章节将为用户提供稳定可靠的升级方案!

分类: STM32+CH395Q(以太网)物联网开发


杨奉武
关注 - 1
粉丝 - 721

0 0

« 上一篇： 2.7-Air302(NB-IOT)-基础外设-DS18B20温度传感器实验
» 下一篇： 2-STM32+CH395Q(以太网)远程升级篇(自建物联网平台)-什么是http,怎么通过http下载文件数据,http分段下载

posted on 2021-08-26 13:52 杨奉武 阅读(213) 评论(0) 编辑 收藏 举报

单片机,物联网,上位机,…

扫一扫二维码，加入群聊。