目录

MCU OTA教程（3.1）

GAgent OTA教程

MCU OTA教程（2.0）

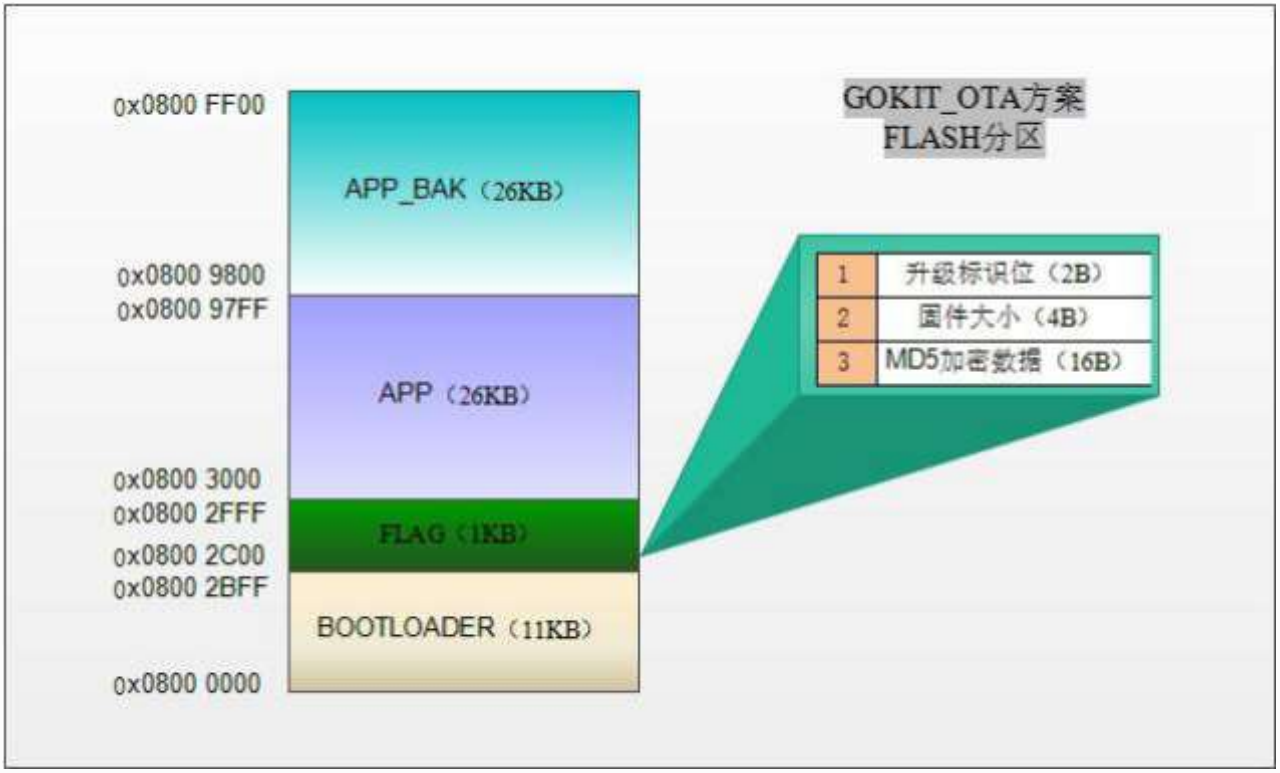
MCU OTA教程（3.0）

源文档及源参考代码下载

1.概述

MCU OTA可以对MCU程序进行无线远程升级。本文以STM32F103C8T6实现OTA作为例子。原本MCU程序软件版本号是01，想升级到02，但是设备已经量产了不可能再去一个一个设备重新烧录新的程序，这时候就需要用到MCU OTA。

STM32F103C8T6 芯片（GOKIT2 代）Flash 空间划分出 4 个区域：Bootloader、FLAG、APP 分区、APPBAK 分区。



Bootloader:存储 Bootloader 固件，MCU 上电后首先运行该固件。

FLAG:存储有关升级的相关标志位，Bootloader 和 APP 分区都需要操作该区域。

APP 分区:存储用户程序固件。

APPBAK 分区:临时存储云端下发的新固件，升级固件的一个过渡存储区。

源代码中有BootLoader和APP 分区两部分。BootLoader稍作编译设置，不需要改动代码，就可以编译烧写到MCU中；APP 分区的OTA功能相关代码复制到mcu方案自动生成代码中，再编译烧写到MCU中，mcu程序就具有OTA功能。

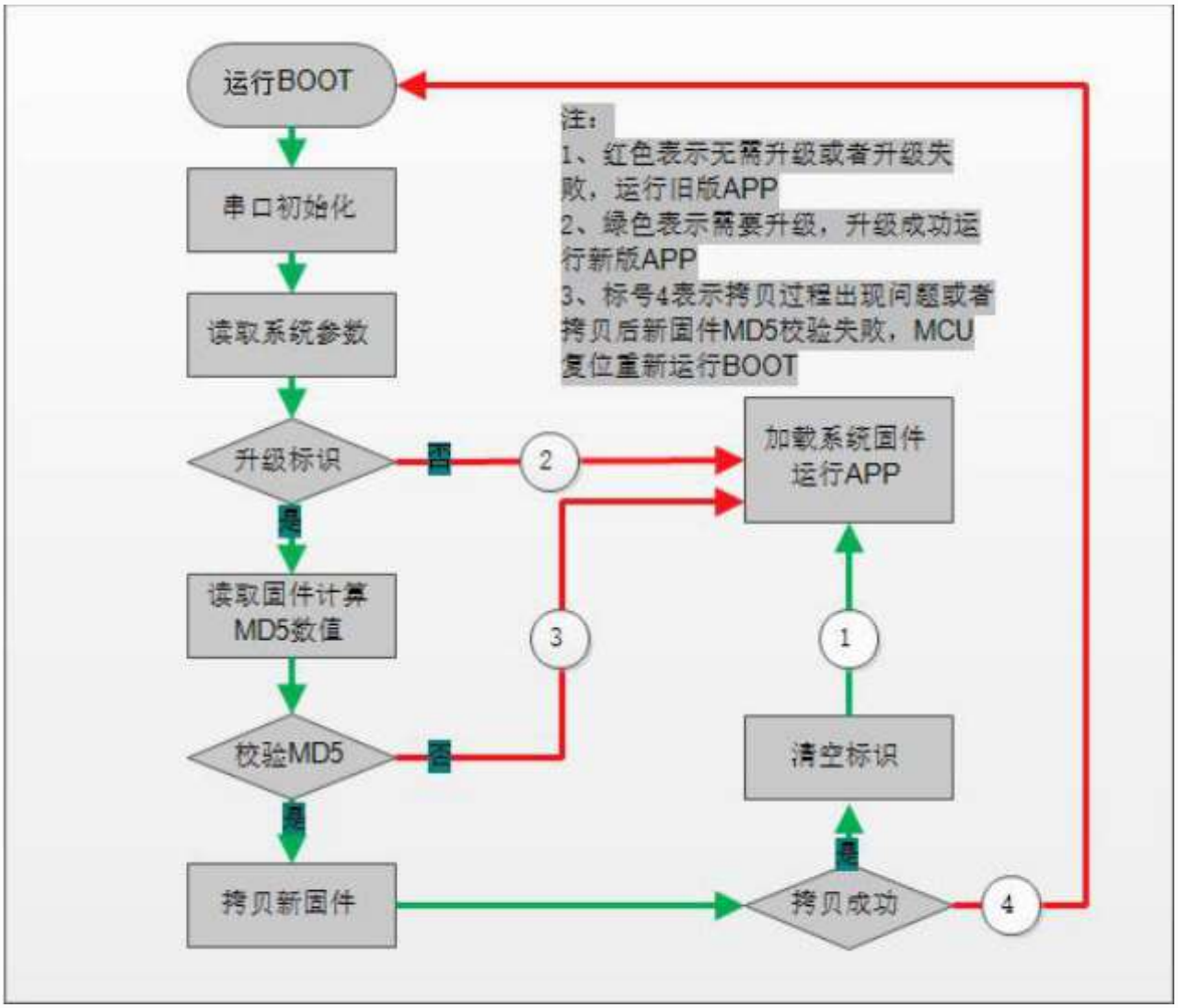
下文分别是BootLoader和APP 分区的详细移植步骤。

2.Bootloader 分区部分

2.1.Bootloader 程序流程

Bootloader 的主要职能是在有升级任务的时候将 APPBAK 分区里面的固件拷贝到 APP 区域。当然，这期间需要做很多的工作，比如升级失败的容错等等。具体的流程可

示。需要注意的是，在校验 MD5 正确后开始搬运固件数据期间，MCU 出现故障（包括突然断电），MCU 应发生复位操作（FLAG 区域数据未破坏），复位后重新开始执行 Bootloader，从而避免 MCU 刷成板砖。

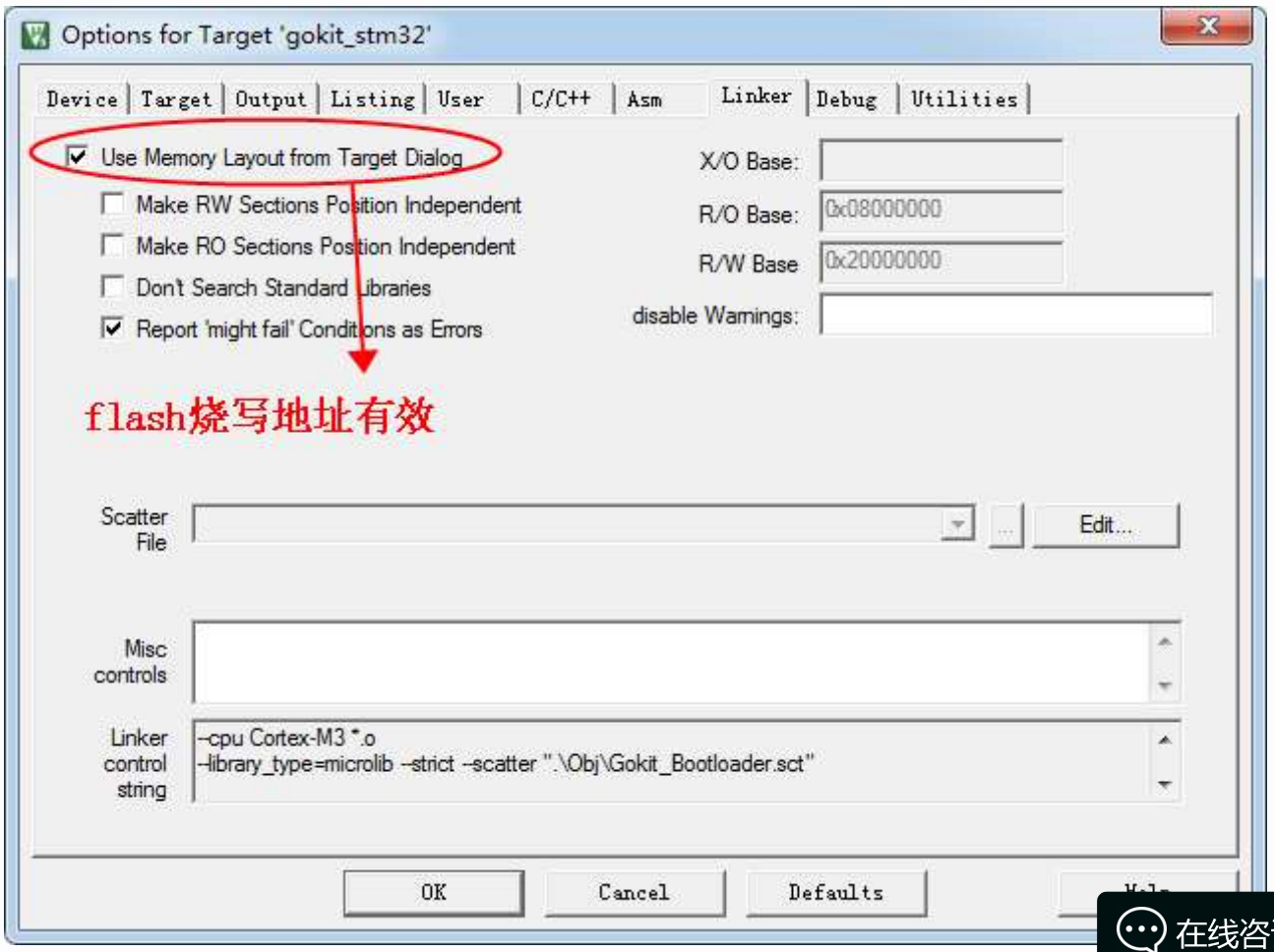


2.2.Bootloader 编译设置

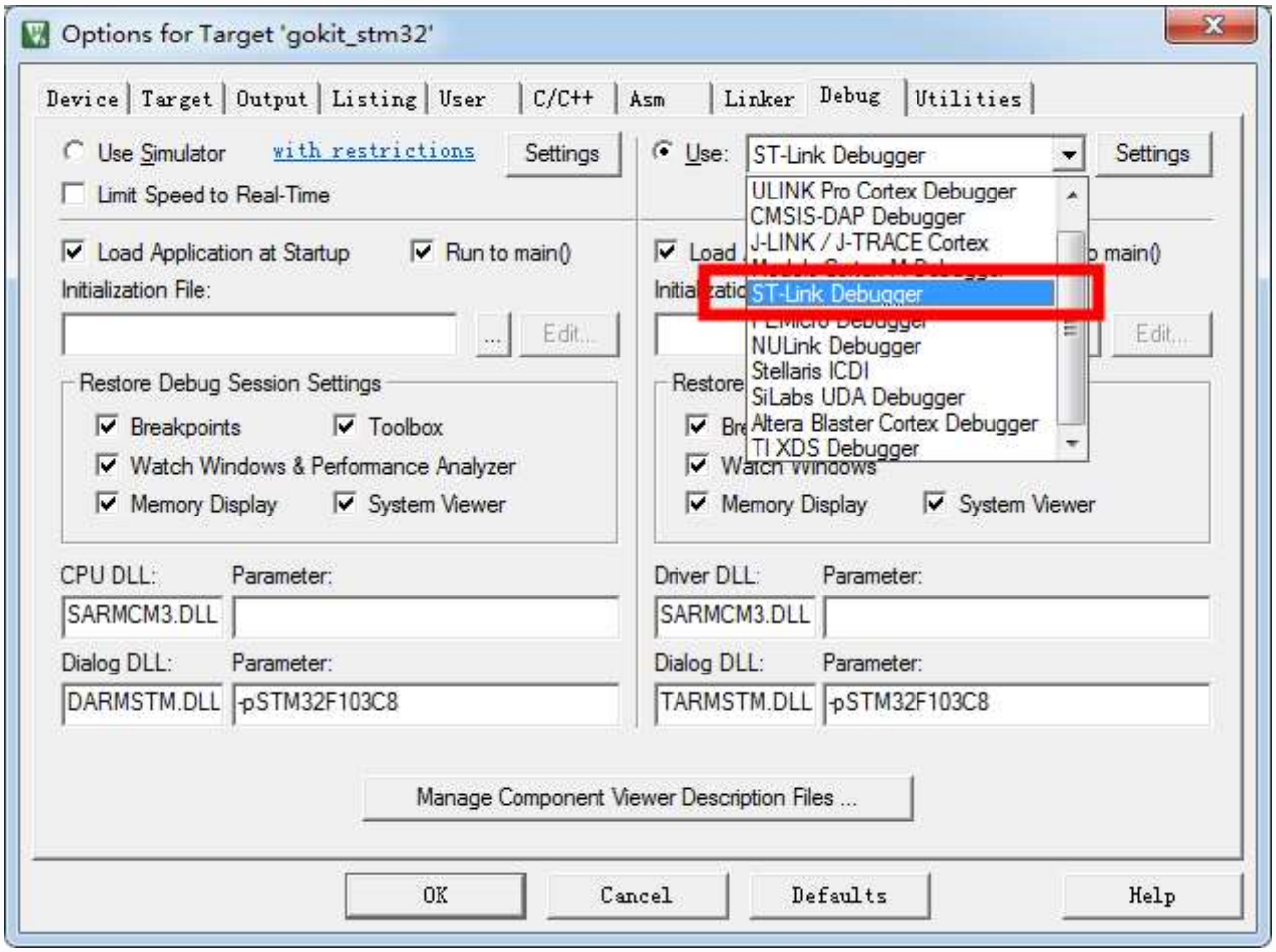
按照 Bootloader 流程编写好代码，需要我们对 KEIL 工程做相应配置，需要注意的是编译的 Bootloader 固件大小不超过最大可允许的 11KB。Keil 编译器需要设置如下：

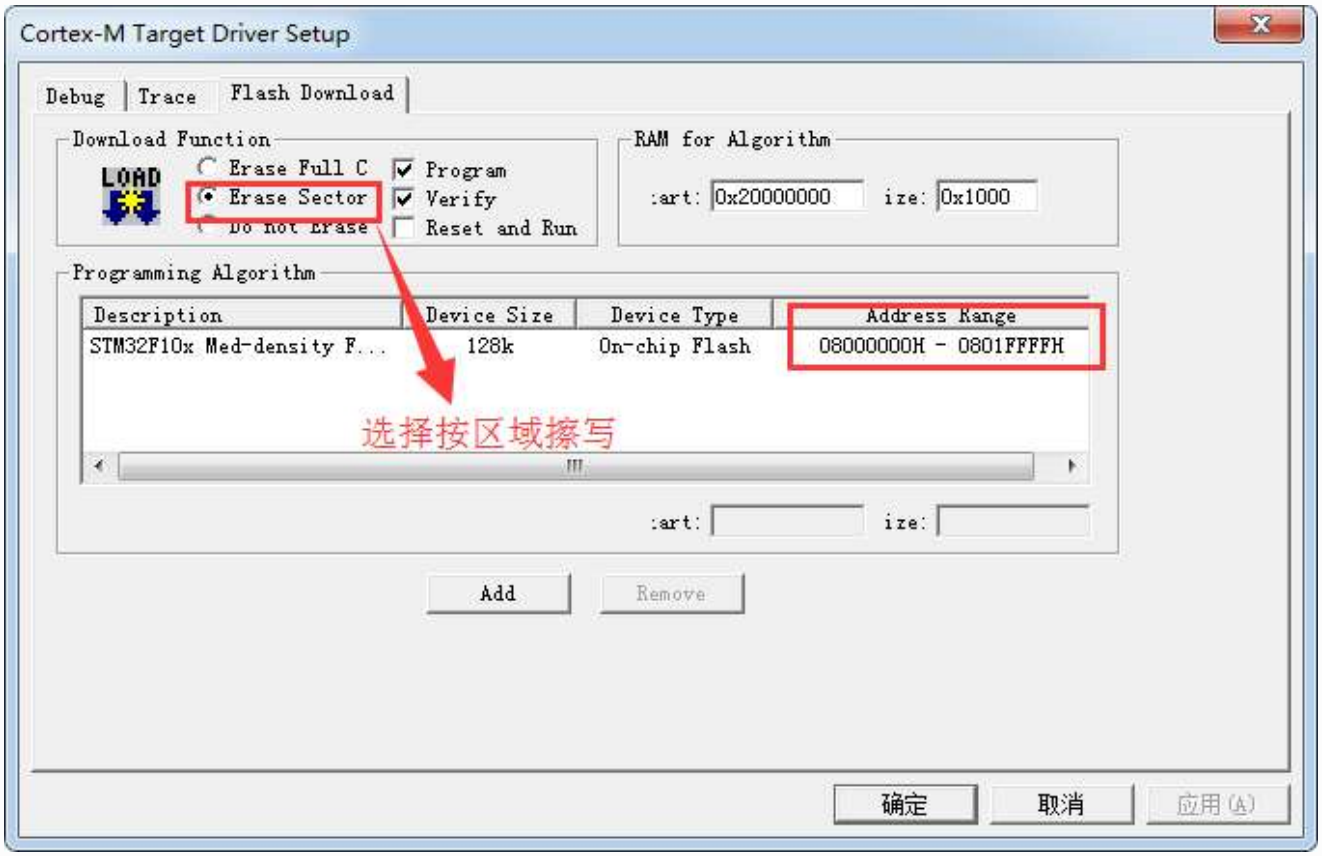
2.2.1按照 FLASH 分区方案，设置 FLASH 固件下载地址，如下图所示：

2.2.2Flash 烧写地址设置生效

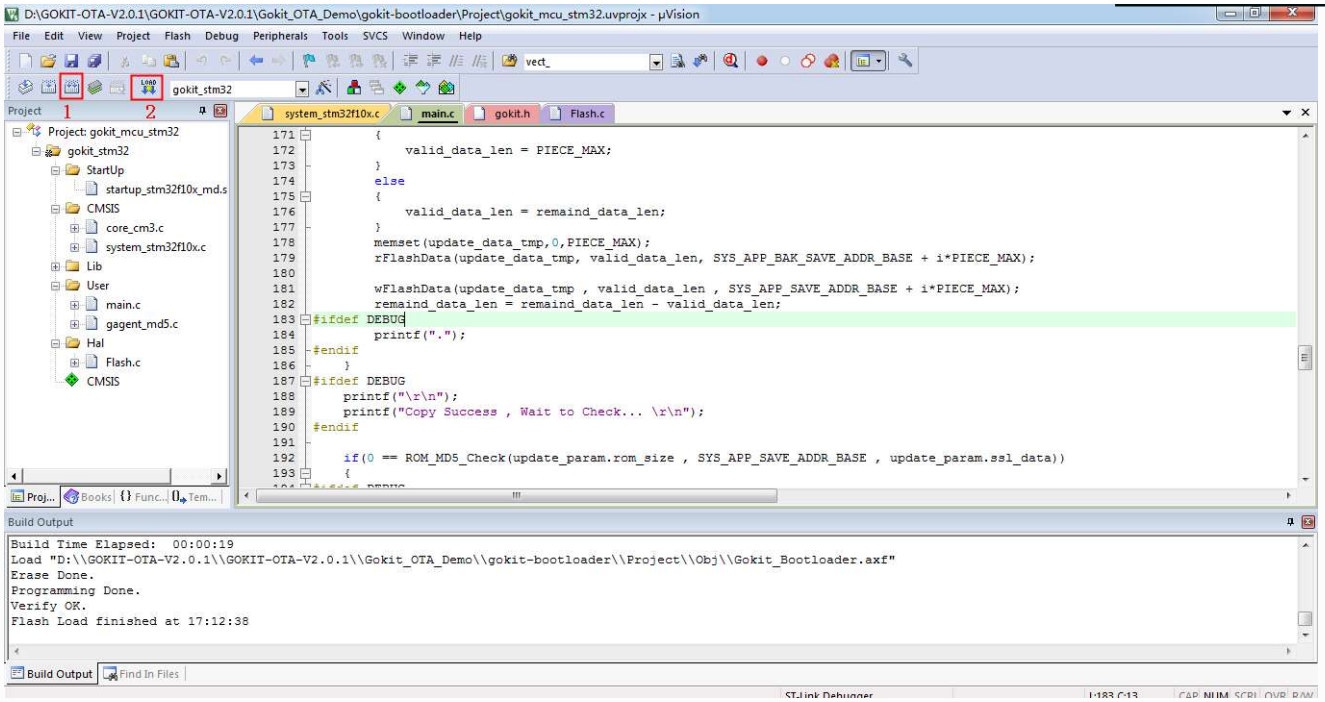


2.2.3设置ST-LINK 按块擦除 FLASH 区间和烧写程序





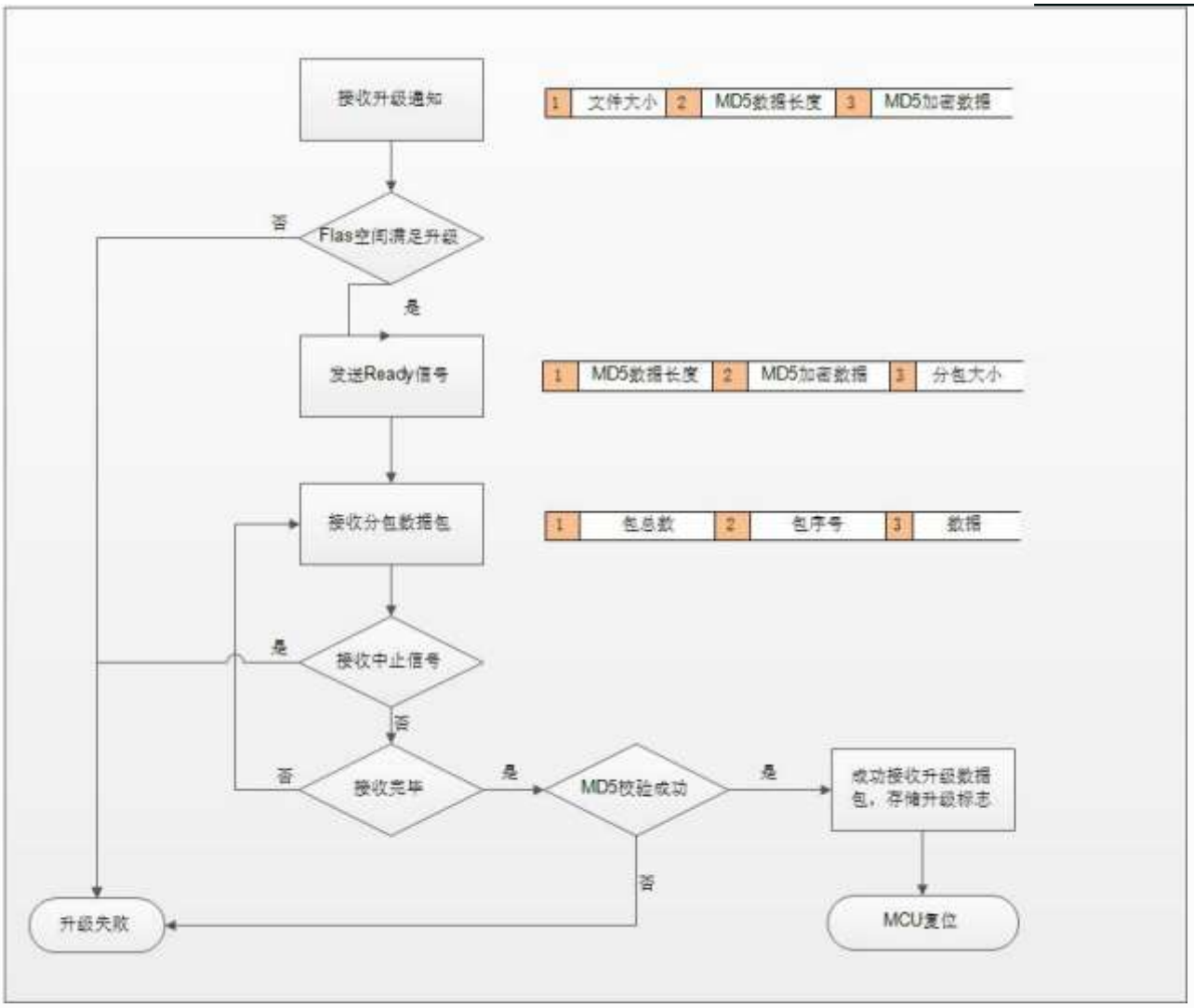
在线咨询



3.APP 分区部分

3.1.固件接收流程

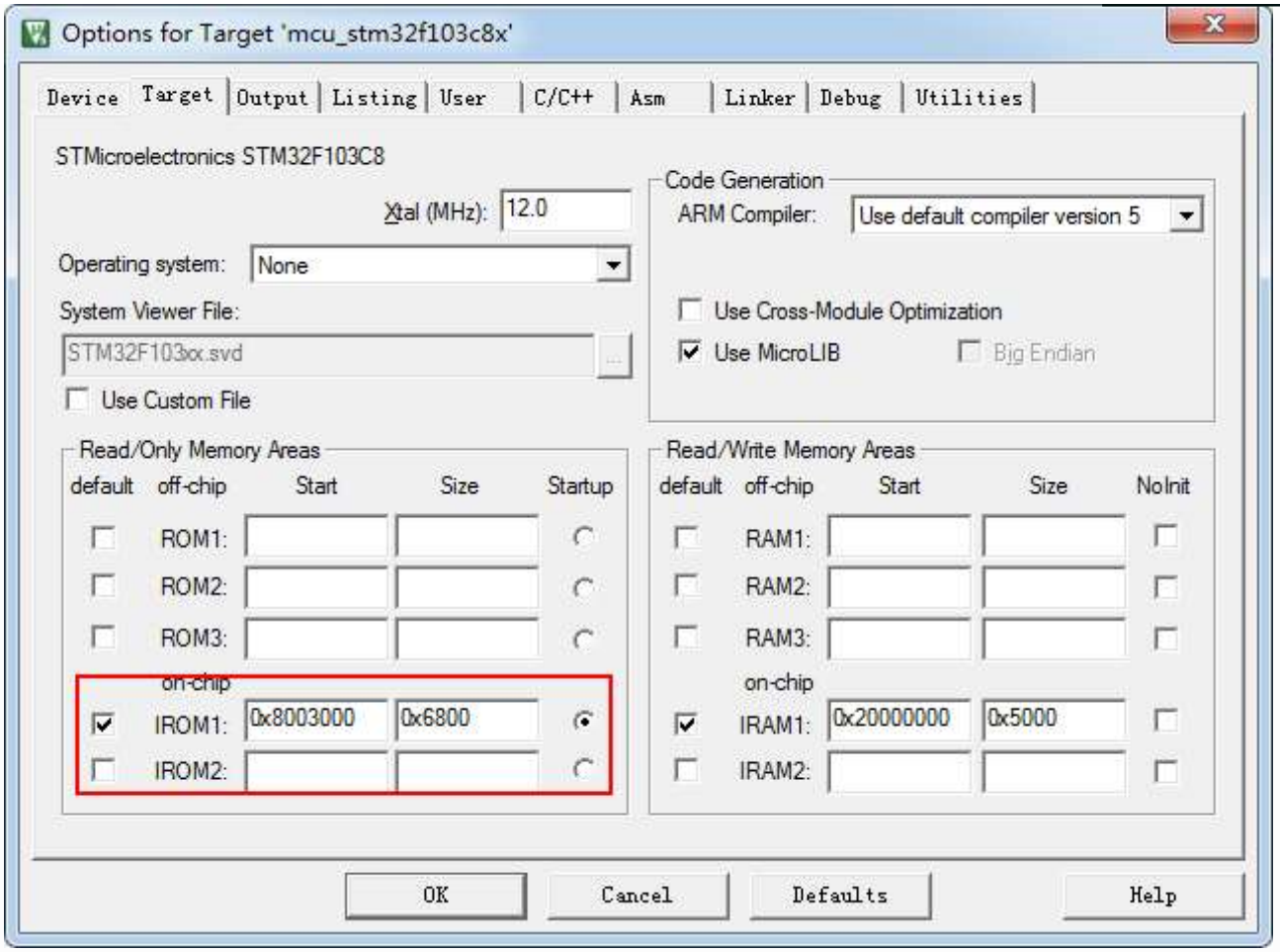
做好 BOOTLOADER 工作后，我们开始写 APP 分区的代码。APP 分区固件的编写要注意硬件版本号和软件版本号，软件版号作为升级迭代很重要的标志。APP 分区代码我们只需要在 GOKIT 微信宠物屋代码基础上增加大数据接受即接受云端新固件功能即可。需要注意的是，中断向量地址偏移的定义，这个地方需要我们尤其注意，我在开发过程中在这个地方排查了好长时间。STM32 标准库默认中断向量地址偏移为 0x0,但是我们 APP 分区实际的偏移是 0x3000。如果不修改，APP 分区也可以正常加载运行，但是不会相应中断。所以，我们需要根据实际 APP 分区下载的起始地址，对中断向量地址偏移做定义。按照协议规定，我们去实现大数据整个流程，具体如下：

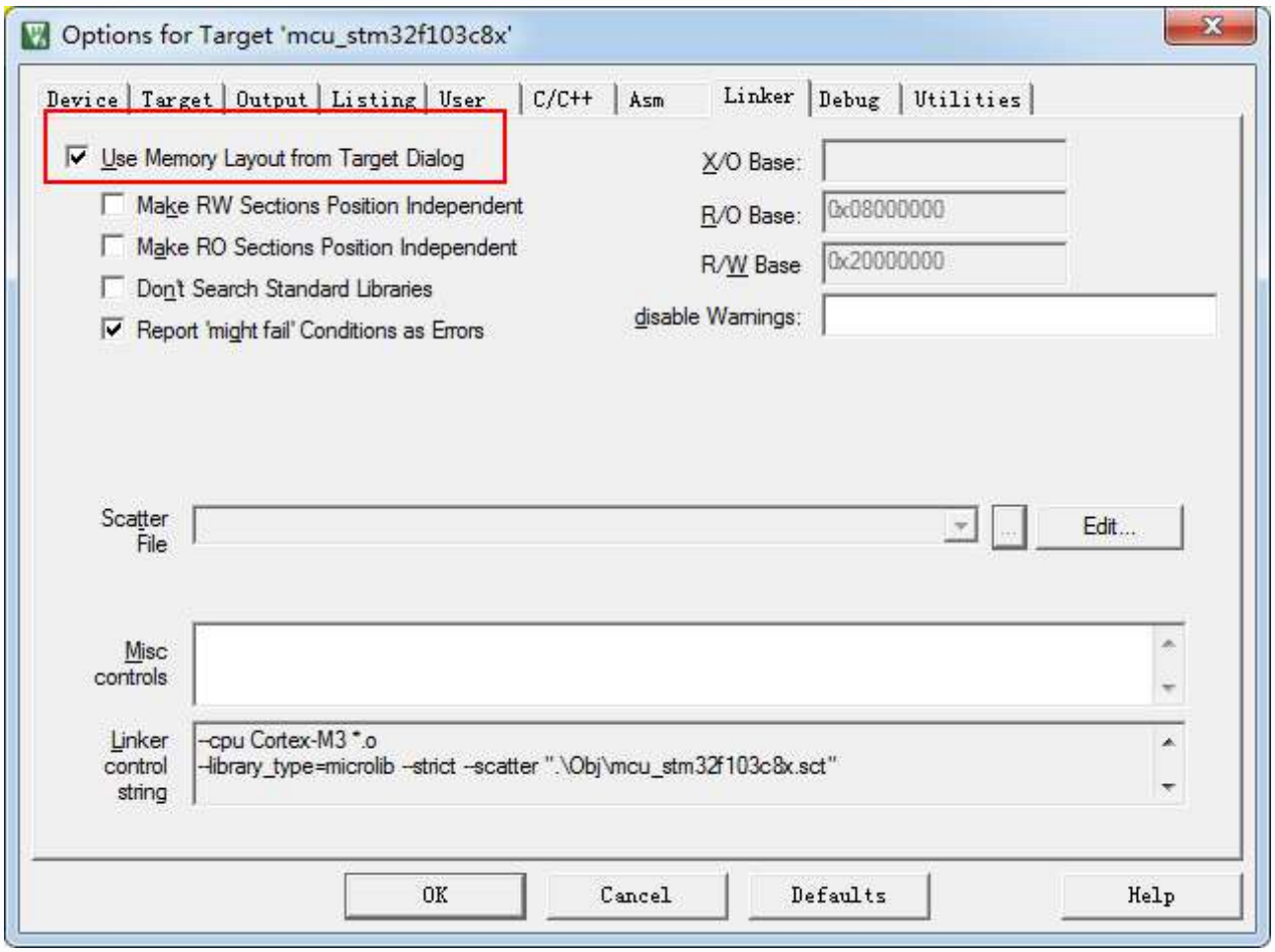


3.2.App 分区编译器设置

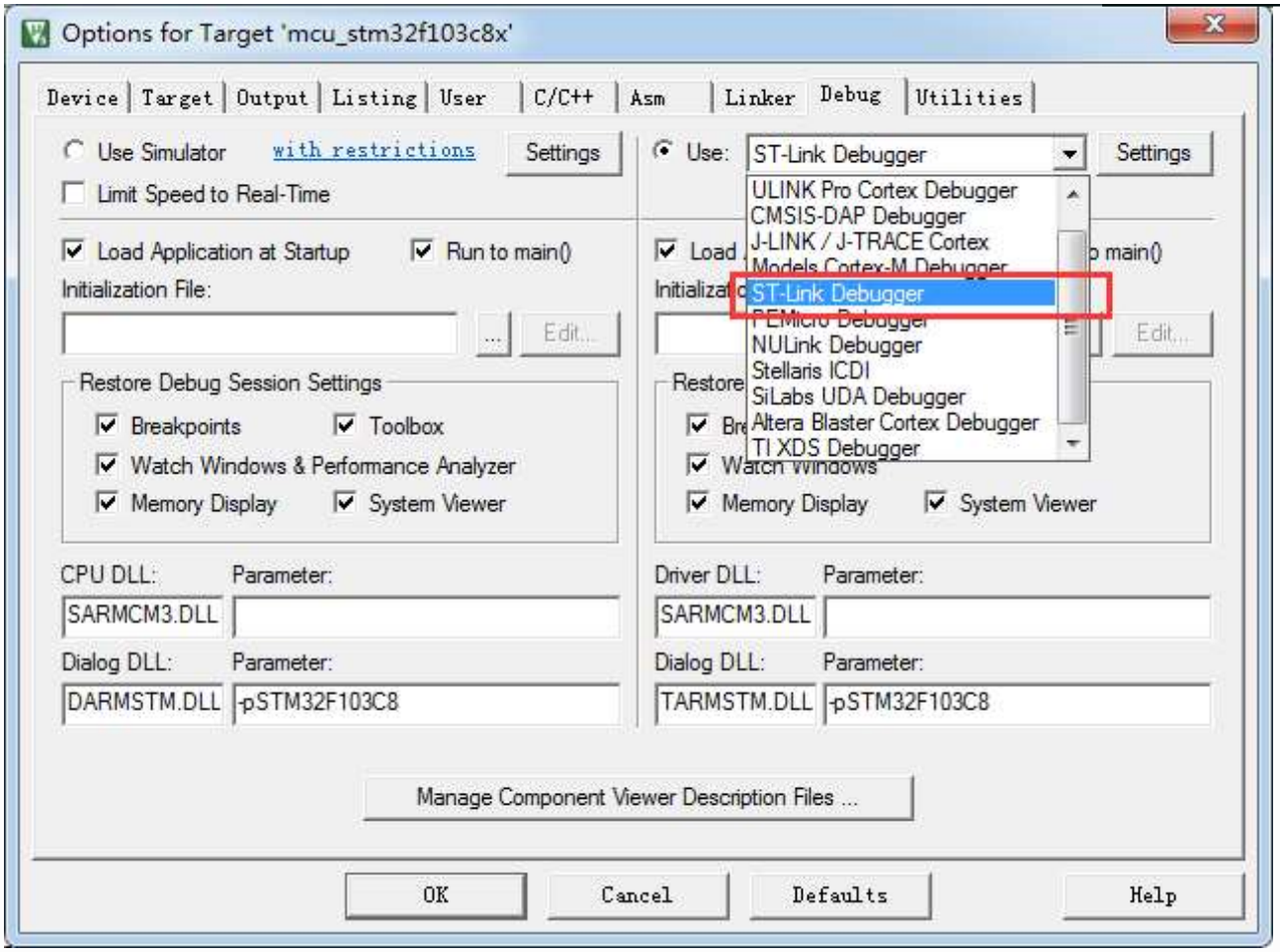
同样，因为硬件 FLASH 空间限定，我们需要对 APP 分区的固件大小做严格的限制。本方案，针对 GOKIT 我们可允许的最大固件为 26KB。需要升级的新固件同样最大可支持 26KB。

3.2.1.设置 FLASH 固件下载地址



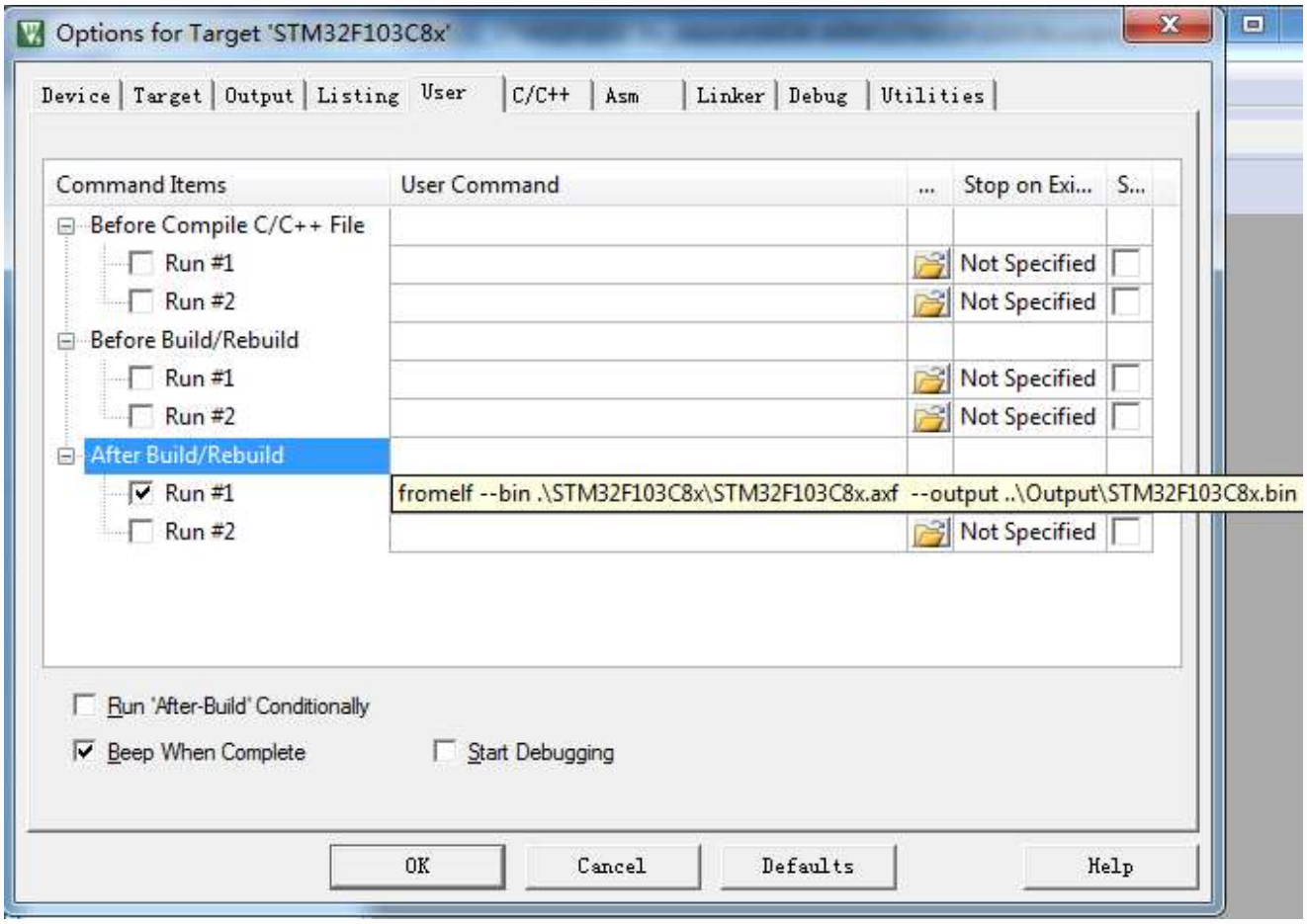


3.2.2.设置keil烧写方式为st-link

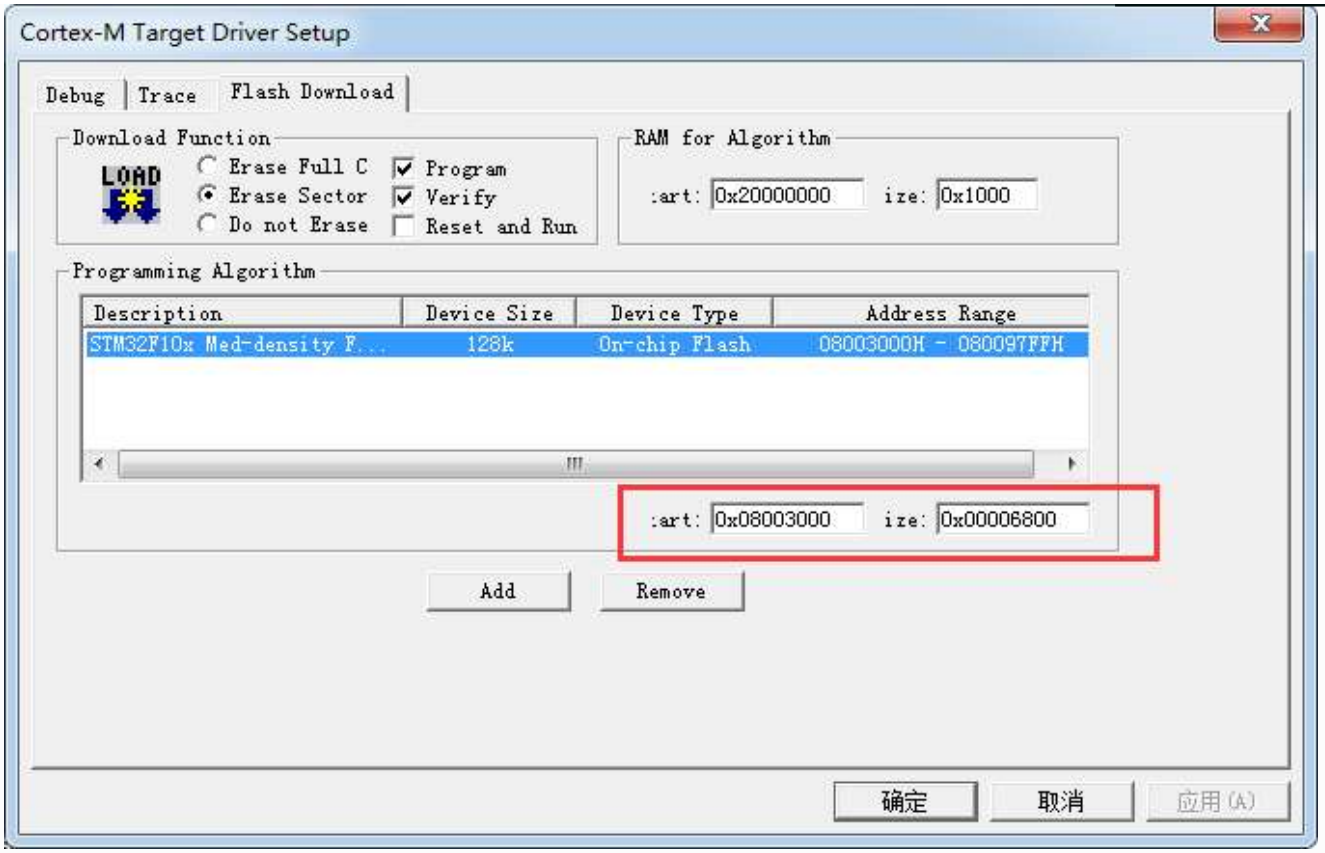


3.2.3.通过编译优化等级控制APP 分区固件大小

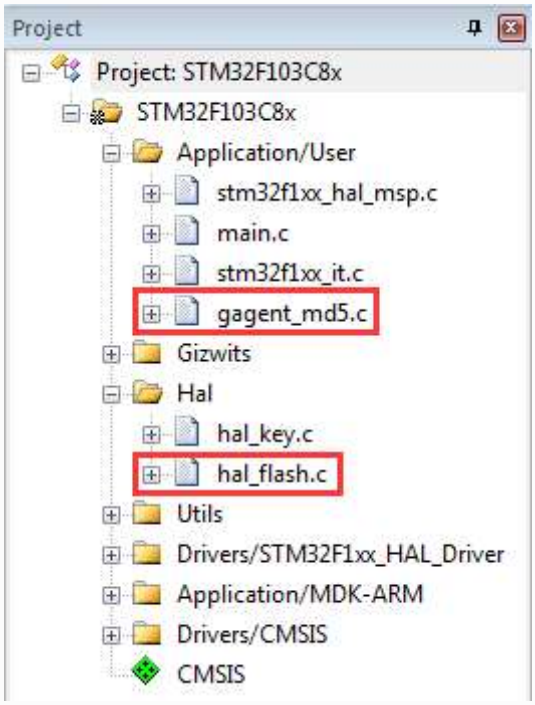
3.2.4.设置编译的时候生成.bin文件（OTA的时候需要选择把.bin文件上传到机智云）



3.2.5.设置按区域擦除



3.2.6.往工程目录添加hal_lash.c和gagent_md5.c，单击keil的build按钮展开工程目录，可以看到hal_flash.h和gagent_md5.h



3.3.App 分区OTA功能代码移植

3.3.1.中断向量偏移地址

```
system_stm32f10x.c | hal_key.c | hal_uart.c | hal_watchdog.c | main.c | gizwits_protocol.c | misc.h
115  #define SYSCLK_FREQ_72MHz  72000000
116  #endif
117
118  /*!< Uncomment the following line if you need to use external SRAM mounted
119       on STM3210E-EVAL board (STM32 High density and XL-density devices) or on
120       STM32100E-EVAL board (STM32 High-density value line devices) as data memory */
121  #if defined (STM32F10X_HD) || (defined STM32F10X_XL) || (defined STM32F10X_HD_VL)
122  /* #define DATA_IN_ExtSRAM */
123  #endif
124
125  /*!< Uncomment the following line if you need to relocate your vector Table in
126       Internal SRAM. */
127  /* #define VECT_TAB_SRAM */
128  #define VECT_TAB_OFFSET  0x3000 /*!< Vector Table base offset field.
129                                   This value must be a multiple of 0x200. */
130
131
132  /**
133    * @}
134    */
135
136  /** @addtogroup STM32F10x_System_Private_Macros
```

中断向量地址偏移

3.3.2.接下来是代码移植步骤，只需把图片内红框相应代码复制到开发者自动生成mcu代码，即可实现mcuOTA功能。

```
Project
  StartUp
    startup_stm32f10x_m
  CMSIS
  Lib
  User
    main.c
    stm32f10x_it.c
    gagent_md5.c
  Hal
    hal_key.c
    hal_uart.c
    hal_watchdog.c
    Flash.c
  Gizwits
    gizwits_product.c
    gizwits_protocol.c
  Utils
    common.c
Project | Books | {} Func... | {} Tem...
```

```
gizwits_product.c | system_stm32f10x.c | main.c* | gizwits_product.h | gizwits_protocol.h | gi
109     userInit();
110     gizwitsInit();
111
112     keyInit();
113     timerInit();
114     uartInit();        //communication serial port init
115
116     watchdogInit(2);    //watchdog reset time :2s
117
118     GIZWITS_LOG("MCU Init Success \n");
119
120     printf("mmmmmApp Init Success , SoftVersion = %s\r\n",SOFTWARE_VERSION);
121
122     Pro_D2W_Ask_Module_Reboot();
123
124     while(1)
125     {
126         watchdogFeed();
127
128         userHandle();
129
130         gizwitsHandle((dataPoint_t *)&currentDataPoint);
131     }
```

名字自拟

```
gizwits_protocol.h
25  #include <stdbool.h>
26  #include <stdio.h>
27  #include <stdlib.h>
28  #include <string.h>
29  #include "common.h"
30  #include "../Src/md5/gagent_md5.h"
31
```

gizwits_protocol.h

```
/*
#define MAX_PACKAGE_LEN     (sizeof(devStatus_t)+sizeof(attrFlags_t)+20)
#define RB_MAX_LEN           (MAX_PACKAGE_LEN*2)      ///< Maximum length of ring buffer
*/

#define MAX_PACKAGE_LEN     900                  ///< Data buffer maximum length
#define RB_MAX_LEN          900                  ///< Maximum length of ring buffer

```

gizwits_protocol.h*

```
#pragma pack()


/*****************************************************
* OTA
*****************************************************/
#define PIECE_MAX_LEN  128

#define FILE_MD5_MAX_LEN  32
#define SSL_MAX_LEN (FILE_MD5_MAX_LEN/2)

#define UPDATE_IS_HEX_FORMAT  0 // Piece Send Format 0,nohex; 1,hex

typedef enum
{
    HEX = 0,
    BIN,
} otaDataType;

__packed  typedef struct
{
    uint16_t        piecenum;
    uint16_t        piececount;
    uint8_t         piececontent[PIECE_MAX_LEN];
} updataPieceData_TypeDef;

typedef struct
{
    uint16_t rom_statue;
    uint32_t rom_size;
    uint8_t  ssl_data[SSL_MAX_LEN];
} updateParamSave_t;

typedef struct
{
    uint8_t otaBusyModeFlag;
    uint32_t updateFileSize;  //Rom Size
    MD5_CTX ctx;
    updateParamSave_t update_param;//Save Update Param
} mcuOTA_t;
int8_t Pro_W2D_UpdateDataHandle(uint8_t *inData , uint32_t dateLen , otaDataType formatType);
int8_t Pro_D2W_UpdateReady(uint8_t *md5Data , uint16_t md5Len);
int8_t Pro_W2D_UpdateCmdHandle(uint8_t *inData,uint32_t dataLen);
void Pro_D2W_UpdateSuspend(void);
void Pro_D2W_Ask_Module_Reboot(void);

/**@name Gizwits user API interface
```

gizwits_protocol.c

```c
#include "ringBuffer.h"
#include "gizwits_product.h"
#include "dataPointTools.h"
#include "hal_flash.h"
/** Protocol global variables **/
gizwitsProtocol_t gizwitsProtocol;
mcuOTA_t  romUpdate;

/**@name The serial port receives the ring buffer implementation
* @{
*/
rb_t pRb;                                                   ///< Ring
static uint8_t rbBuf[RB_MAX_LEN];                           ///< Ring
```

gizwits_protocol.c

```c
/**
* @brief Protocol handling function

*

* @param [in] currentData :The protocol data pointer
* @return none
*/
int32_t gizwitsHandle(dataPoint_t *currentData)
{
    int8_t ret = 0;
#ifdef PROTOCOL_DEBUG
    uint16_t i = 0;
#endif
    uint8_t ackData[RB_MAX_LEN];
    uint16_t protocolLen = 0;
    uint32_t ackLen = 0;
    protocolHead_t *recvHead = NULL;
    char *didPtr = NULL;
    uint16_t offset = 0;
    int8_t updatePieceResult = 0;

    if(NULL == currentData)
```

在线咨询

gizwits_protocol.c*

```
1252            case ACK_GET_NTP:
1253                gizProtocolWaitAckCheck(recvHead);
1254                gizProtocolNTP(recvHead);
1255                GIZWITS_LOG("Ack GET_UTT success \n");
1256                break;
1257            case ACK_ASK_MODULE_INFO:
1258                gizProtocolWaitAckCheck(recvHead);
1259                gizProtocolModuleInfoHandle(recvHead);
1260                GIZWITS_LOG("Ack GET_Module success \n");
1261                break;
1262            case ACK_BIGDATA_READY:
1263                gizProtocolWaitAckCheck(recvHead);
1264                break;
1265            case ACK_D_STOP_BIGDATA_SEND:
1266                gizProtocolWaitAckCheck(recvHead);
1267                break;
1268            case CMD_ASK_BIGDATA:
1269                GIZWITS_LOG("CMD_ASK_BIGDATA \n");
1270                gizProtocolCommonAck(recvHead);
1271                if(0 == Pro_W2D_UpdateCmdHandle((uint8_t *)gizwitsProtocol.protocolBuf + sizeof(prot
1272                {
1273                    romUpdate.otaBusyModeFlag = 1;
1274                    GIZWITS_LOG("System In OTA Mode BusyFlag = %d \n", romUpdate.otaBusyModeFlag);
1275                }
1276                else
1277                {
1278                    GIZWITS_LOG("Update Ask Handle Failed \n");
1279                    Pro_D2W_UpdateSuspend();
1280                }
1281                break;
1282            case CMD_BIGDATA_SEND:
1283                updatePieceResult = Pro_W2D_UpdateDataHandle((uint8_t *)gizwitsProtocol.protocolBuf
1284                gizProtocolCommonAck(recvHead);
1285                if(0 != updatePieceResult)
1286                {
1287                    romUpdate.otaBusyModeFlag = 0;
1288                    GIZWITS_LOG("CMD_BIGDATA_SEND , Piece Handle Faild . System OTA Mode Over BusyFl
1289                    Pro_D2W_UpdateSuspend();
1290                    GIZWITS_LOG("System Go On \n");
1291                }
1292                break;
1293            case CMD_S_STOP_BIGDATA_SEND:
1294                gizProtocolCommonAck(recvHead);
1295                romUpdate.otaBusyModeFlag = 0;
1296                GIZWITS_LOG("System OTA Mode Over BusyFlag = %d \n",romUpdate.otaBusyModeFlag);
1297                break;
1298            case ACK_REBOOT_MODULE:
1299                gizProtocolWaitAckCheck(recvHead);
1300                GIZWITS_LOG("Wifi Module Will Restart \n");
1301                break;
1302            default:
1303                gizProtocolErrorCmd(recvHead,ERROR_CMD);
1304                GIZWITS_LOG("ERR: cmd code error!\n");
1305                break;
1306        }
```

gizwits_protocol.c*

```
1400 /**
1401 * @brief Pro_W2D_UpdateCmdHandle
1402
1403 * Handle OTA Ask , Transform MD5 Char2Hex
1404
1405 * @param[in]  :
1406 * @param[out] :
1407 * @return  0,Update Ask Handle Success , Send Ready Success
1408 *         -1,Input Param Illegal
1409 *         -2,Update Ask Handle Success , Send Ready Faild
1410 *
1411 */
1412 int8_t Pro_W2D_UpdateCmdHandle(uint8_t *inData,uint32_t dataLen)
1413 {
1414     uint8_t k = 0;
1415     int8_t ret = 0;
1416
1417     uint8_t  fileMD5value[FILE_MD5_MAX_LEN];
1418     uint16_t fileMD5len;      //MD5 Length
1419
1420     if(NULL == inData)
1421     {
1422         return -1;
1423     }
1424
1425     romUpdate.updateFileSize = ((uint32_t)(inData[0]<<24))|((uint32_t)(inData[1]<<
1426
```

```
1427     //judge flash size
1428     if(romUpdate.updateFileSize > APP_BAK_DATA_MAX_SIZE)
1429     {
1430         GIZWITS_LOG("UpdateFileSize Error ,Update Failed ,fileSize = %d \n",romUpdate.updateFileSi
1431         return -1;
1432     }
1433     else
1434     {
1435         GIZWITS_LOG("UpdateFileSize Legal ,size = %d \n",romUpdate.updateFileSize);
1436         romUpdate.update_param.rom_size = romUpdate.updateFileSize;
1437         flash_erase(APP_BAK_DATA_MAX_SIZE,SYS_APP_BAK_SAVE_ADDR_BASE);
1438     }
1439
1440     fileMD5len = inData[4]*256 + inData[5];
1441     GIZWITS_LOG("FileMD5len  = %d ", fileMD5len);
1442
1443     memcpy(fileMD5value,&inData[6],fileMD5len);
1444 #ifdef PROTOCOL_DEBUG
1445     GIZWITS_LOG("MD5: ");
1446     for(uint16_t i=0; i<32; i++)
1447     {
1448         GIZWITS_LOG("%02x ", fileMD5value[i]);
1449     }
1450     GIZWITS_LOG("\r\n");
1451 #endif
1452
1453     for(uint16_t j = 0; j<SSL_MAX_LEN; j++)
1454     {
1455         romUpdate.update_param.ssl_data[j] = char2hex(fileMD5value[k],fileMD5value[k+1]);
1456         k += 2;
1457     }
1458 #ifdef PROTOCOL_DEBUG
1459     GIZWITS_LOG("MD5_Hex: ");
1460     for(uint16_t i=0; i<SSL_MAX_LEN; i++)
1461     {
1462         GIZWITS_LOG("%02x ", romUpdate.update_param.ssl_data[i]);
1463     }
1464     GIZWITS_LOG("\r\n");
1465 #endif
1466
1467     GIZWITS_LOG("GAgent_MD5Init \n");
1468     GAgent_MD5Init(&romUpdate.ctx);
1469
1470     //send ready
1471     ret = Pro_D2W_UpdateReady(fileMD5value,fileMD5len);
1472     if(0 != ret)
1473     {
1474         GIZWITS_LOG("Pro_D2W_UpdateReady Error ,Error Code = %d \n",ret);
1475         return -2;
1476     }
1477
1478     return 0;
1479 }
1480
1481 /**
1482 * @brief Pro_D2W_UpdateReady
1483
1484 * MCU Send Update Ready
1485
1486 * @param[in]  md5Data: Input md5 char data
1487 * @param[in]  md5Len : Input md5 length
1488 * @param[out] :
1489 * @return  0,Update Ask Handle Success , Send Ready Success
1490 *         -1,Input Param Illegal
1491 *         -2,Uart Send Faild
1492 *
1493 */
1494 int8_t Pro_D2W_UpdateReady(uint8_t *md5Data , uint16_t md5Len)
1495 {
1496     int8_t ret = 0;
1497     uint8_t txBuf[100];
1498     uint8_t *pTxBuf = txBuf;
1499
1500     if(NULL == md5Data)
1501     {
1502         return -1;
1503     }
1504
1505     uint16_t dataLen = sizeof(protocolCommon_t) + 2 + md5Len + 2 - 4 ;
1506     *pTxBuf ++= 0xFF;
1507     *pTxBuf ++= 0xFF;
1508     *pTxBuf ++= (uint8_t)(dataLen>>8);
1509     *pTxBuf ++= (uint8_t)(dataLen);
1510     *pTxBuf ++= CMD_BIGDATA_READY;
1511     *pTxBuf ++= gizwitsProtocol.sn++;
1512
1513     *pTxBuf ++= 0x00;//flag
```

红框内添加memset(txBuf,0,100);

```
    txBuf[7] |= UPDATE_IS_HEX_FORMAT<<0;//TERRY WARNING
    pTxBuf += 1;

    *pTxBuf ++= (uint8_t)(md5Len>>8);//len
    *pTxBuf ++= (uint8_t)(md5Len);

    memcpy(&txBuf[8 + 2],md5Data,md5Len);
    pTxBuf += md5Len;

    *pTxBuf ++= (uint8_t)(PIECE_MAX_LEN>>8);//len
    *pTxBuf ++= (uint8_t)(PIECE_MAX_LEN);
    *pTxBuf ++= gizProtocolSum(txBuf , (dataLen+4));

    ret = uartWrite(txBuf , (dataLen+4));
    if(ret < 0)
    {
        GIZWITS_LOG("ERROR: uart write error %d \n", ret);
        return -2;
    }
    GIZWITS_LOG("MCU Ready To Update ROM \n");
    return 0;
}

/**
* @brief Pro_W2D_UpdateDataHandle

* update Piece Handle ,  Judge Last Piece

* @param[in] indata   : Piece Data
* @param[in] dataLen    : Piece Length
* @param[in] formatType : Piece Data Format
* @param[out]
* @return  0,Handle Success
*         -1,Input Param Illegal
*         -2,Last Piece , MD5 Check Faild
*
*/
int8_t Pro_W2D_UpdateDataHandle(uint8_t *inData , uint32_t dataLen , otaDataType formatType)
{
    uint16_t piecenum = 0;
    uint16_t piececount = 0;
    uint32_t tempWFlashAddr = 0;
    updataPieceData_TypeDef pieceData;
    uint8_t md5_calc[SSL_MAX_LEN];//MD5 Calculate Fact

    if(NULL == inData)
    {
        return -1;
    }

    memcpy((uint8_t *)&pieceData, inData, dataLen);

    piecenum = exchangeBytes(pieceData.piecenum);
    piececount = exchangeBytes(pieceData.piececount);

    GIZWITS_LOG("******piecenum = %d , piececount = %d, pieceSize = %d******** \r\n",piecenum,piec


    tempWFlashAddr = SYS_APP_BAK_SAVE_ADDR_BASE + (piecenum-1) * PIECE_MAX_LEN;
    wFlashData((uint8_t *)pieceData.piececontent , dataLen - 4, tempWFlashAddr);

    GAgent_MD5Update(&romUpdate.ctx, (uint8_t *)pieceData.piececontent, dataLen - 4);

    /*updata package data ,ack*/
    if(piecenum == piececount)
    {
        memset(md5_calc,0,SSL_MAX_LEN);
        GAgent_MD5Final(&romUpdate.ctx, md5_calc);
        GIZWITS_LOG("MD5 Calculate Success , Will Check The MD5 ..\n ");

        if(0 != memcmp(romUpdate.update_param.ssl_data, md5_calc, SSL_MAX_LEN))
        {
            GIZWITS_LOG("Md5_Cacl Check Faild ,MCU OTA Faild\r\n ");
#ifdef PROTOCOL_DEBUG
            GIZWITS_LOG("Calc MD5: ");
            for(uint16_t i=0; i<SSL_MAX_LEN; i++)
            {
                GIZWITS_LOG("%02x ", md5_calc[i]);
            }
            GIZWITS_LOG("\r\n");
#endif
#ifdef PROTOCOL_DEBUG
            GIZWITS_LOG("SSL MD5: ");
            for(uint16_t i=0; i<SSL_MAX_LEN; i++)
            {
                GIZWITS_LOG("%02x ", romUpdate.update_param.ssl_data[i]);
```

```
1600                }
1601                GIZWITS_LOG("\r\n");
1602 #endif
1603                memset((uint8_t *)&romUpdate.update_param,0,sizeof(updateParamSave_t));
1604
1605                return -2;
1606            }
1607            else
1608            {
1609                GIZWITS_LOG("MD5 Check Success ,Storage  ROM Success , Write Update Flag\n ");
1610                flash_erase(sizeof(updateParamSave_t) , UPDATE_PARAM_SAVE_ADDR_BASE);
1611
1612                romUpdate.update_param.rom_statue = 0xEEEE;
1613                wFlashData((uint8_t *)&romUpdate.update_param, sizeof(updateParamSave_t), UPDATE_PARAM
1614
1615                GIZWITS_LOG("System Will Restart... \n");
1616                /****************************MCU RESTART****************************/
1617                mcuRestart();
1618                /*******************************************************************************/
1619                //last package , updata ok
1620                //MD5 checkout :Failed clear updata,Success , write flash ,begin updata
1621            }
1622        }
1623        return 0;
1624    }
1625
1626    /**
1627    * @brief Pro_D2W_UpdateSuspend
1628
1629    * Data Receiver
1630
1631    * @param[in]     : Void
1632    * @param[out]    :
1633    * @return        : Void
1634    *
1635    */
1636    void Pro_D2W_UpdateSuspend()
1637    {
1638        int32_t ret = 0;
1639        protocolCommon_t protocolCommon;
1640        memset(&protocolCommon, 0, sizeof(protocolCommon_t));
1641        gizProtocolHeadInit((protocolHead_t *)&protocolCommon);
1642        protocolCommon.head.len = exchangeBytes(sizeof(protocolCommon_t)-4);
1643        protocolCommon.head.cmd = CMD_D_STOP_BIGDATA_SEND;
1644        protocolCommon.head.sn = gizwitsProtocol.sn++;
1645        protocolCommon.sum = gizProtocolSum((uint8_t *)&protocolCommon, sizeof(protocolCommon_t));
1646
1647        ret = uartWrite((uint8_t *)&protocolCommon,sizeof(protocolCommon_t));
1648        if(ret < 0)
1649        {
1650            GIZWITS_LOG("ERROR: uart write error %d \n", ret);
1651        }
1652
1653        gizProtocolWaitAck((uint8_t *)&protocolCommon,sizeof(protocolCommon_t));
1654
1655    }
1656
1657    /**
1658    * @brief Pro_D2W_Ask_Module_Reboot
1659
1660    * Ask Module Reboot
1661
1662    * @param[in]     : Void
1663    * @param[out]    :
1664    * @return        : Void
1665    *
1666    */
1667    void Pro_D2W_Ask_Module_Reboot()
1668    {
1669        int32_t ret = 0;
1670        protocolCommon_t protocolCommon;
1671        memset(&protocolCommon, 0, sizeof(protocolCommon_t));
1672        gizProtocolHeadInit((protocolHead_t *)&protocolCommon);
1673        protocolCommon.head.len = exchangeBytes(sizeof(protocolCommon_t)-4);
1674        protocolCommon.head.cmd = CMD_REBOOT_MODULE;
1675        protocolCommon.head.sn = gizwitsProtocol.sn++;
1676        protocolCommon.sum = gizProtocolSum((uint8_t *)&protocolCommon, sizeof(protocolCommon_t));
1677
1678        ret = uartWrite((uint8_t *)&protocolCommon,sizeof(protocolCommon_t));
1679        if(ret < 0)
1680        {
1681            GIZWITS_LOG("ERROR: uart write error %d \n", ret);
1682        }
1683
1684        gizProtocolWaitAck((uint8_t *)&protocolCommon,sizeof(protocolCommon_t));
1685
1686    }
```

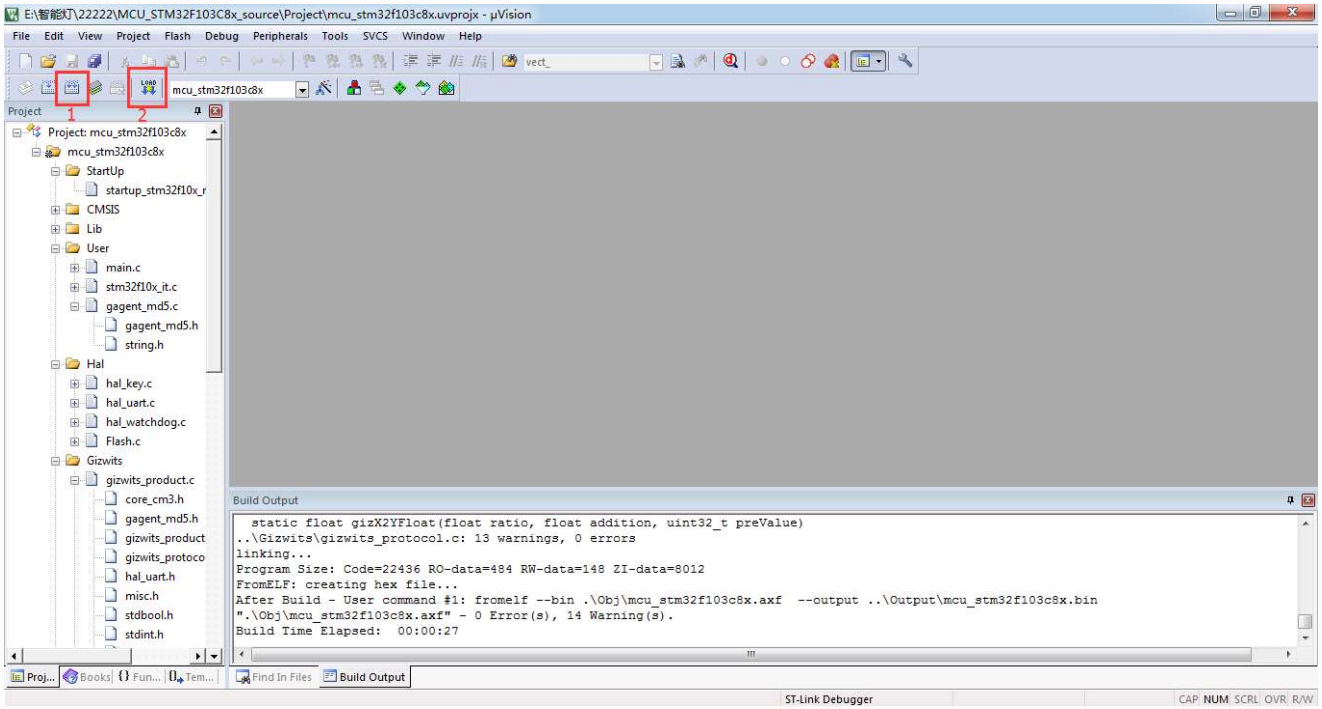
```
1687
1688 /**@} */
1689
```

gizwits_product.h*

```
33
34 /**@} */
35
36 #define typedef_t __packed typedef
37
38 /**
39 * MCU软件版本号
40 */
41 #define SOFTWARE_VERSION "03010101"
42 /**
43 * MCU硬件版本号
44 */
45 #define HARDWARE_VERSION "03010100"
46
47 #ifndef SOFTWARE_VERSION
48 #error "no define SOFTWARE_VERSION"
49 #endif
50
51 #ifndef HARDWARE_VERSION
52 #error "no define HARDWARE_VERSION"
53 #endif
54
```

改，OTA版本号必须比原版本大，例如"03010102"……

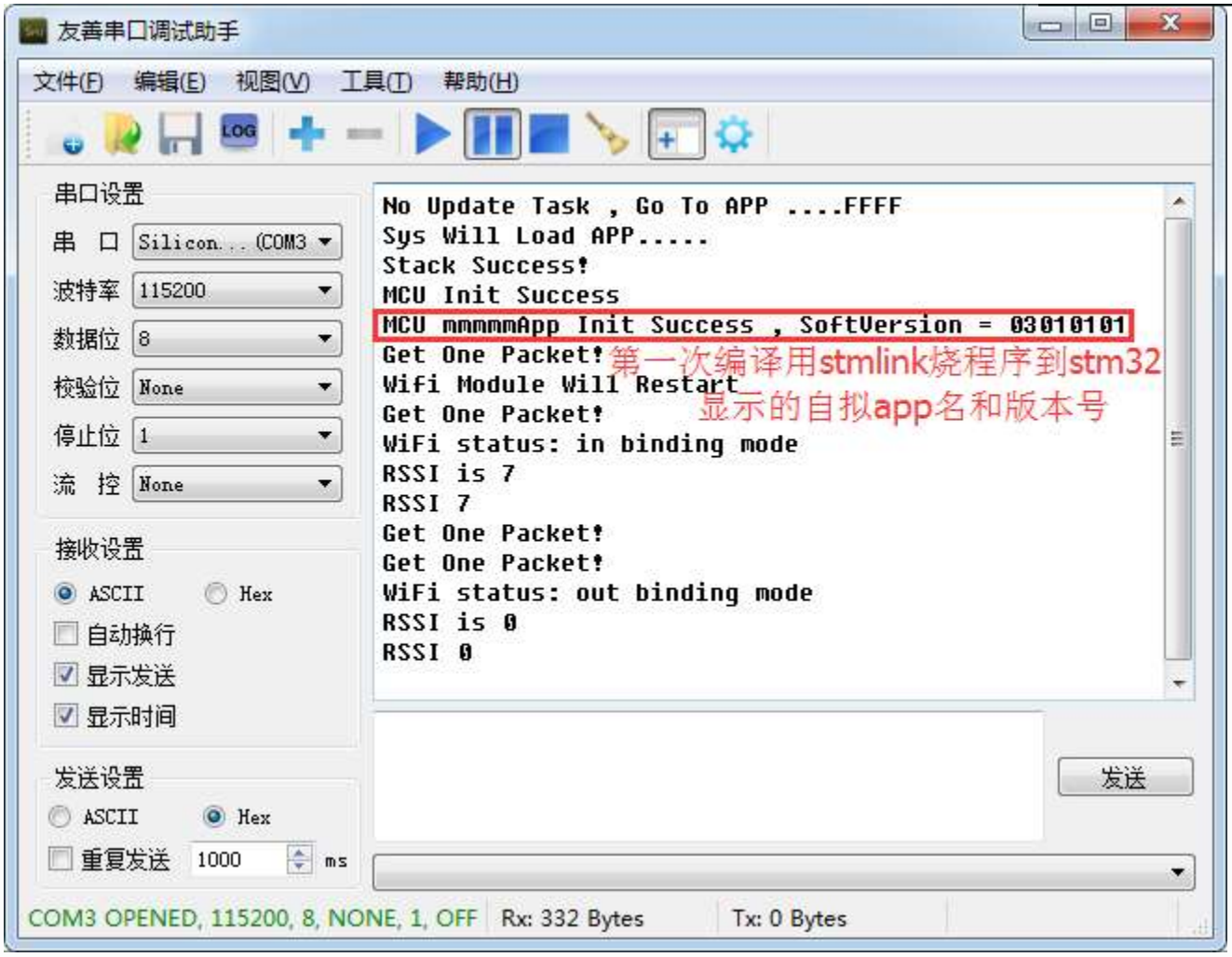
3.3.3.编译和烧程序



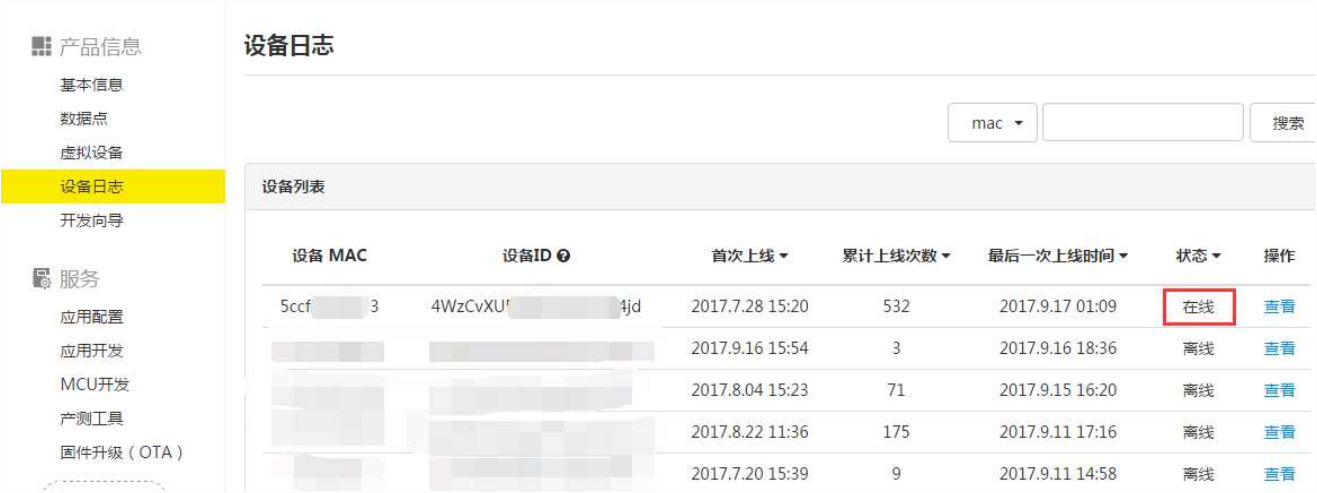
4.MCU OTA验证

4.1.第一次用stlink烧录mcu代码后，mcu日志如图

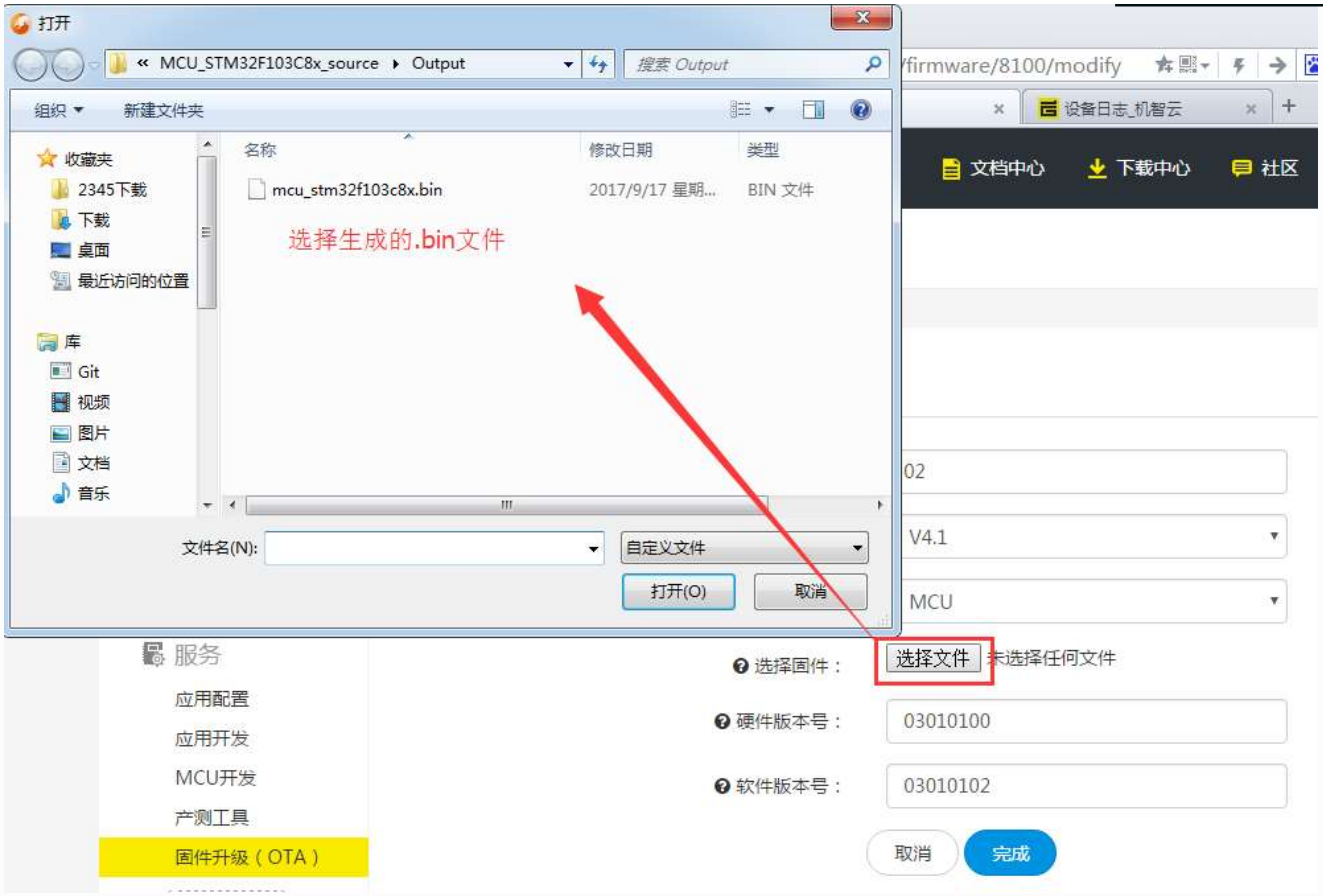
在线咨询



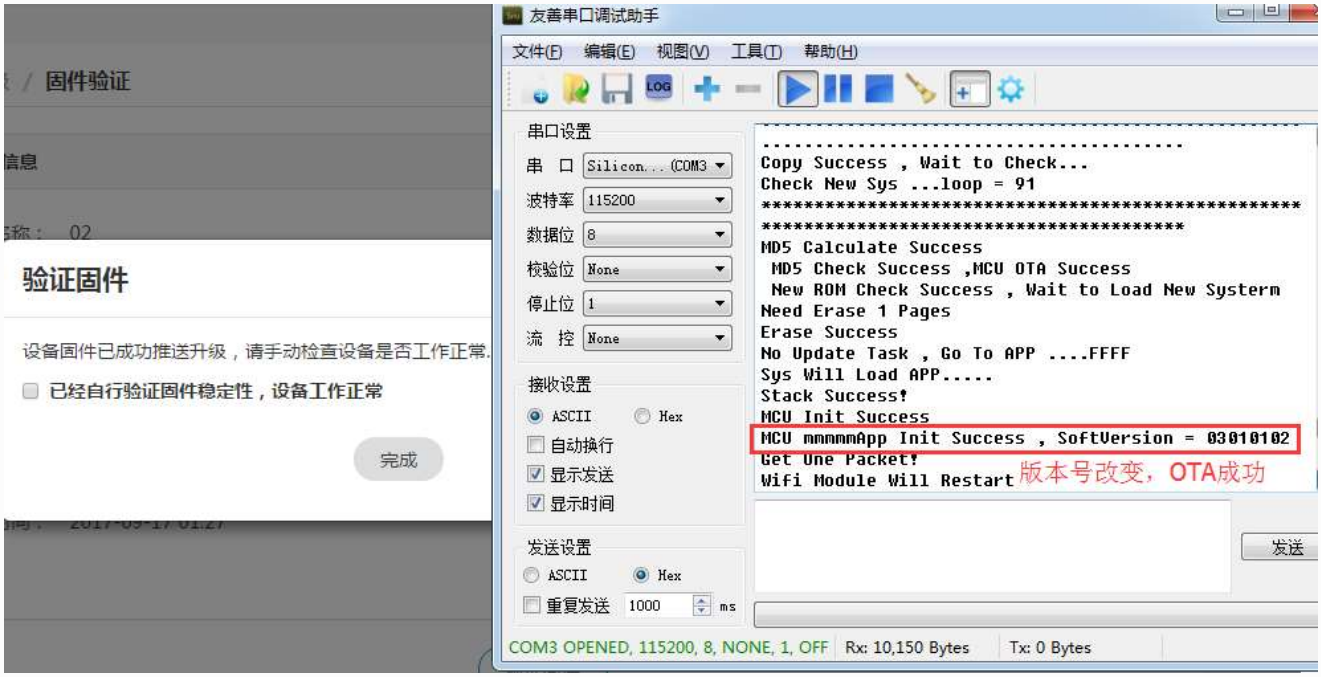
4.2.准备OTA，先让设备连上机智云。



4.3.改mcu代码里面的软件版本号，要比原来的高，选择编译出来的.bin文件。（注意：如果图中有手动/静默，请选择静默，没有则忽略注意）



4.4.OTA成功



本文档主要写移植OTA功能移植过程，想要了解MCU OTA详细过程（例如mcu启动流程检查有无OTA任务，OTA flash分区等等），请看源文档。

在线咨询