

---

## N32H7xx系列芯片UART Xmodem IAP升级应用笔记

---

### 简介

本文档主要介绍 N32H7xx 系列芯片（以下简称 N32H7xx）通过串口来进行 IAP 升级，以及提供相应例程。

以下均为侧重点说明，主要是提供一个 APP 跳转示例，详细代码内容可查看附件例程。

## 目录

<b>1 目录</b>	<b>I</b>
<b>1 IAP 说明</b>	<b>1</b>
1.1 IAP 简介	1
1.2 IAP 原理	1
1.2.1 无 IAP 机制下程序运行流程	1
1.2.2 加入 IAP 机制后程序运行流程	2
<b>2 IAP 实现</b>	<b>4</b>
2.1 XMODEM 协议介绍	4
2.2 IAP 工程关键代码说明	5
2.3 IAP 升级流程图说明	7
2.4 注意事项	9
2.4.1 Bootloader	12
2.4.2 App	16
<b>3 IAP 实现乒乓升级</b>	<b>19</b>
3.1 IAP 乒乓升级流程介绍	19
3.2 TERATERM 工具使用	22
<b>4 历史版本</b>	<b>27</b>
<b>5 声明</b>	<b>28</b>

# 1 IAP 说明

## 1.1 IAP 简介

IAP 是 In Application Programming 的缩写，即在应用编程。用户程序在运行过程中对 User Flash 部分区域进行烧写，为了在产品发布后可以方便地通过预留的通讯接口对固件程序进行更新升级。一句话概括就是，IAP 是一个专门用来升级应用程序的程序。

通常在实现 IAP 功能时，需要在编写固件程序时设计两部分代码：第一个部分不执行产品正常的功能操作，只是通过通讯接口去接收数据，执行对第二部分代码的更新操作（这部分代码后面简称为 Bootloader）；第二部分代码才是真正的产品功能代码（这部分代码后面简称为 App）。

这两部分代码都同时存放在 User Flash 的不同区域中（一般从最低地址开始存放 Bootloader，后面紧随 App），当芯片上电之后会首先执行 Bootloader，Bootloader 作如下操作：

- 1、检查是否需要对 App 进行更新
- 2、如果不需要更新则跳转到步骤 4
- 3、执行更新操作
- 4、跳转到 App 执行

Bootloader 必须通过其它方式如 JTAG/SWD 或 ISP 烧录，第二部分代码可以使用 Bootloader IAP 功能烧录，也可以和 Bootloader 一起烧录。后面 Bootloader 统称为 IAP 程序。

## 1.2 IAP 原理

### 1.2.1 无 IAP 机制下程序运行流程

在加入 IAP 升级机制之前，先了解程序正常的运行流程如下图：



以上即是正常的程序运行流程。

当加入 IAP 程序之后，程序运行流程如下图：

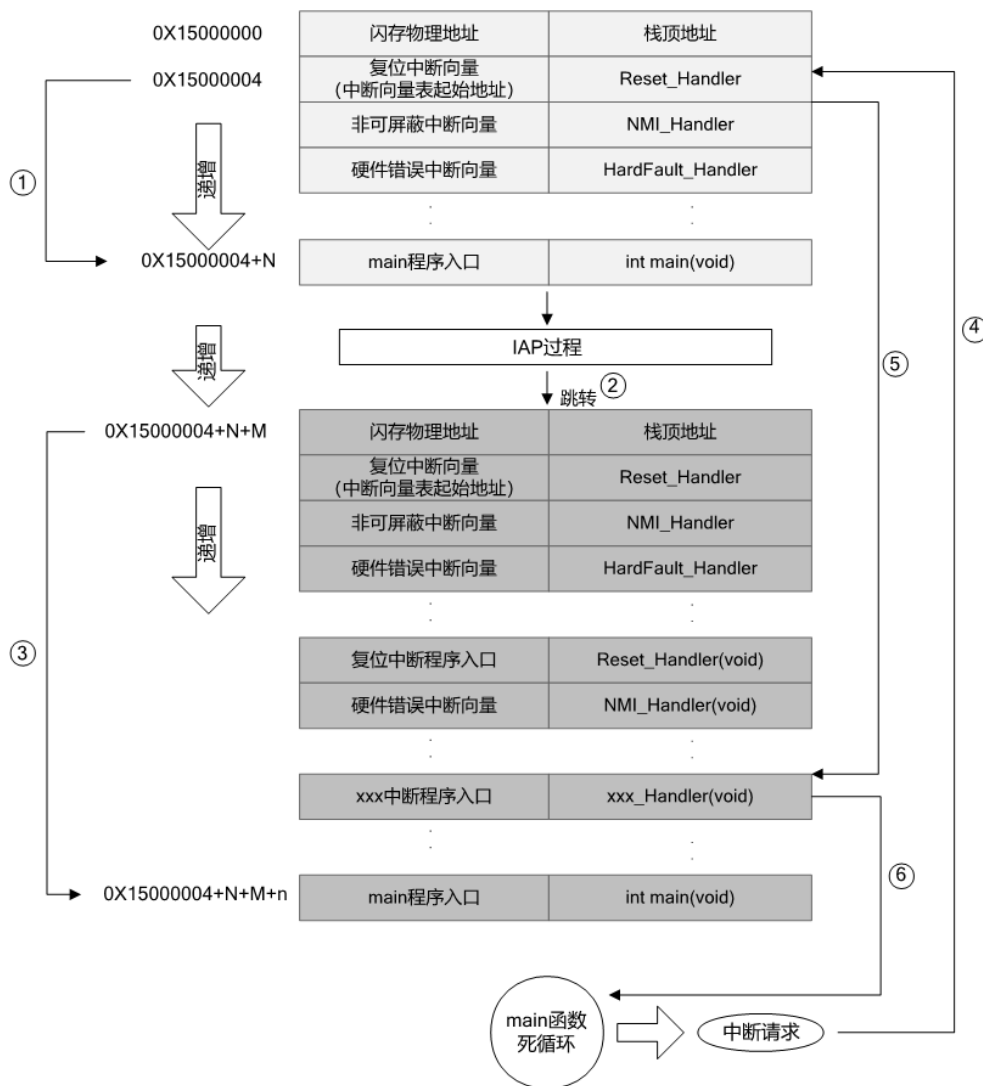


图 2

N32H7xx 复位后，还是从 0x1500\_0004 取出复位中断向量的地址并跳转到复位中断服务程序，在运行完复位中断服务程序之后跳转到 IAP 的 main 函数（如上图流程①），这部分与无 IAP 机制的程序运行一致。

在执行完 IAP 以后（即将新的 APP 代码写入 User Flash，即灰底部分），新 APP 的复位中断起始地址为：0x1500\_0004+N+M。跳转至新 APP 的复位向量表（如②），取出复位中断向量的地址并跳转执行新 APP 的复位中断服务程序，随后再跳转至新 APP 的 main 函数（如③）。同样，main 函数是一个死循环，此时可以看到 Flash 中在不同位置上有两个“中断向量表”。

但在新 APP 的 main 函数执行过程中，如果 CPU 得到一个中断请求，PC 指针还是强制跳转到 0x1500\_0004 这个地址的中断向量表处（如④），而不是新 APP 的中断向量表处。程序会再根据我们设置的中

断向量表偏移量（这个新的中断偏移量会在新 APP 中设置，APP 程序中会有体现），跳转到对应中断源新的中断服务程序中（如⑤）。执行完中断服务程序后，程序返回 main 函数继续运行（如⑥）。

以上就是加 IAP 机制之后程序的运行流程。

## 2 IAP 实现

了解原理之后，正式来实现 IAP 程序和 APP 程序。由上面过程分析，可以得到 IAP 程序需要满足两个条件：

- 1) 新 APP 必须在 IAP 程序之后的某个偏移量为 x 的地址开始；
- 2) 必须将新 APP 的中断向量表相应地移动，移动偏移量为 x。

（注：这里的 x 根据 IAP 程序的大小来自定义的，后面会详细说明）

由前述章节可知，IAP 升有两部分代码组成：

- 1) 用于升级的 IAP 部分代码；
- 2) 用户应用程序代码；

升级的 IAP 代码本例程采用了开源的 Xmodem 协议。

### 2.1 Xmodem 协议介绍

为了保证 IAP 升级过程中数据传输的可靠性，这里数据传输采用了开源的 Xmodem 协议。Xmodem 是一种在串口通信中广泛使用的异步文件传输协议，以 128 字节块的形式传输数据，并且每个块都使用一个校验过程来进行错误检测。在校验过程中如果接收方关于一个块的检验和与它在发送方的检验相同时，接收方就向发送方发送一个确认字节 **ACK**。如果有错则发送一个字节 **NAK** 要求重发，以保证传输过程中的正确性。另外除了 Xmodem 外还有 Ymodem 和 Zmodem，此文档只采用 Xmodem，读者可自行查阅 Ymodem 和 Zmodem。Xmodem 协议的传输数据单位为数据包，包含一个标题开始字符 **SOH(Start Of Header)** 或者 **STX**，一个单字节包序号，一个单字节包包序号的补码，128 个字节数据和一个双字节的 **CRC16** 校验，如下表所示

表 1 Xmodem 数据包格式

Byte0	Byte1	Byte2	Byte3~ Byte130	Byte131~Byte132
数据头 SOH	包序号	包序号补码	128 字节数据	CRC16 位校验

对于标准 Xmodem 协议来说，如果传送的数据不是 128 的整数倍，那么最后一个数据包的有效内容肯定小于 128 字节，不足的部分需要用 **CTRL-Z(0x1A)** 来填充。

Xmodem 协议的传输由接收方启动，接收方发送方发送 **C** 或者 **NAK**（这里的 **NAK** 是用来启动传输的，下面我们用到的 **NAK** 是用来对数据产生重传机制）。其中接收方发送 **NAK** 信号表示接收方打算用累加和校验；发送字符 **C** 则表示接收方打算使用 **CRC** 校验。传输过程如下图所示。

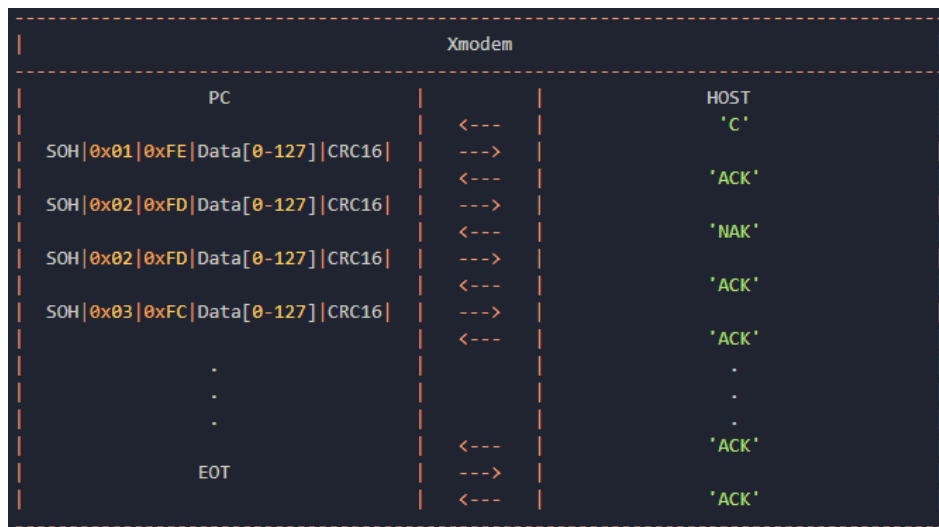


图 3

如果发送方正常传输全部数据，需要结束传输，正常结束需要发送方发送 **EOT** 通知接收方。接收方回以 **ACK** 进行确认。如果接收方发送 **CAN** 给发送方也可以强制停止传输，发送方受到 **CAN** 后不需要发送 **EOT** 确认，此时传输已经结束。

## 2.2 IAP 工程关键代码说明

### ► 芯片容量宏定义

由于 N32H7xx 系列 Flash 有 2M、4M 两个容量，例程中用宏定义将这两个容量进行了区分，在 flash.h 文件中做了如下宏定义：

```
#define N32H7xx_2M_SERIES
// #define N32H7xx_4M_SERIES

#if defined N32H7xx_2M_SERIES
#define FLASH_BANKA_START_ADDR ((uint32_t)0x15004000u) //bankA 880K
#define FLASH_BANKA_END_ADDR ((uint32_t)0x150DFFFu)
#define FLASH_BANKB_START_ADDR ((uint32_t)0x150E0000u) //bankB 1024K
#define FLASH_BANKB_END_ADDR ((uint32_t)0x151DFFFu)
#elif defined N32H7xx_4M_SERIES
#define FLASH_BANKA_START_ADDR ((uint32_t)0x15004000u) //bankA 1904K
#define FLASH_BANKA_END_ADDR ((uint32_t)0x151DFFFu) //
```

```
#define FLASH_BANKB_START_ADDR ((uint32_t)0x151E0000u) //bankB 2048K
#define FLASH_BANKB_END_ADDR ((uint32_t)0x152E0000u)
#endif
```

用户在开发自己的 IAP 程序时，只需根据自己项目所选 MCU 型号对应的 flash 大小来决定使用哪个宏定义。本例程使用的芯片型号为 N32H785IIB7，基于 N32H785IXB7\_STB V1.5 开发板。Flash 是 2M，因此只需开启 #define N32H7xx\_2M\_SERIES，其他容量宏定义注释掉即可。

## ➤ 用户 APP 区起始地址

例程将 MCU 2MB flash 分成 3 个部分：IAP 区、bank A、bank B，IAP 分配了 16K 用于存放升级代码，其余 1904K 用于用户存放用户应用程序，为了满足部分客户乒乓升级的需求，例程中还设计了 BankA 与 BankB，乒乓升级请参考后续章节，这里只讲述非乒乓升级的情况。

在 IAP.c 源文件中定义了两个 32bit 长度全局变量 FLASH\_APP\_START\_ADDRESS，FLASH\_APP\_END\_ADDRESS：

```
uint32_t FLASH_APP_START_ADDRESS = 0x15004000; //user app start address
uint32_t FLASH_APP_END_ADDRESS = 0x151DFFFF; //user app end address
```

这两个全局变量用于存放用户 APP 的起始和结束地址，当升级程序时以这两个变量中的地址为准，用户可以根据自己使用情况做适当调整，flash 分配框图如图 4 所示：

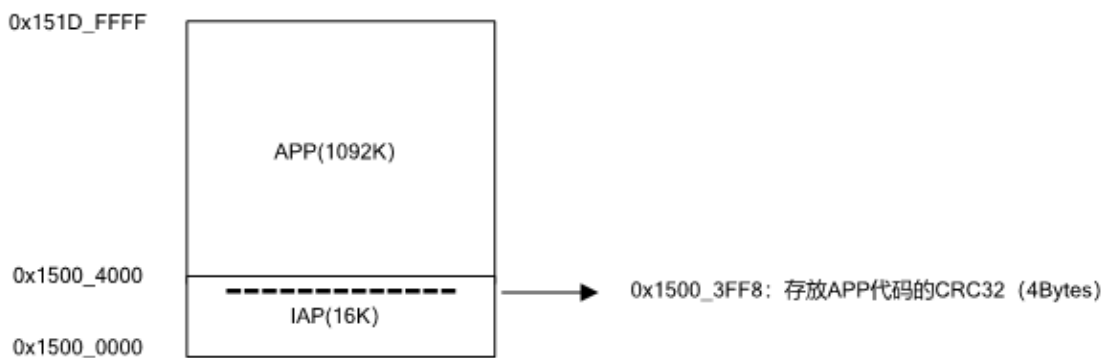


图 4

图 4 中标明在 0x1500\_3FF8 处存放 APP 代码的 CRC32，这是在 IAP 的 boot 区的倒数第二个 word 预留出来存放用户代码的 CRC32，当代码升级完成后，例程会根据 app 起始和结束地址来计算出 CRC32，然后写到 0x1500\_3FF8 地址处，MCU 每次上电或者复位后会去计算用户 APP CRC32，然后与升级时写入的 CRC32 做比较，如果相等则执行用户代码，否则打印出错误提示，代码如下所示：

```
if((check_app_crc(FLASH_APP_START_ADDRESS, FLASH_APP_END_ADDRESS)) == SUCCESS)
{
    flash_jump_to_app(FLASH_APP_START_ADDRESS); //Jump to app
```



```
}  
else  
{  
    printf("Error, APP CRC check err, app start_addr: 0x%08x, app end_addr: 0x%08x\r\n", FLASH_APP_START_ADDRESS,  
    FLASH_APP_END_ADDRESS);  
}
```

### ➤ 乒乓升级宏定义

例程中设置了一个支持乒乓升级的宏定义 `#define BANK_AB_UPDATA`，用户如果需要乒乓升级，只需开启这个宏定义，如果不需要只需注释掉此宏定义即可。

### ➤ BOOT 区最后一页掉电保护

为了防止在升级过程中编程 BOOT 区最后一页时，出现掉电导致 BOOT 区最后一页编程失败的情况，在编程 BOOT 区最后一页前，例程中会将最后一页的数据读出来写到倒数第二页，然后再修改相关设置数据，写到最后一页。如果升级失败，用户可以读取 BOOT 区倒数第二页获取升级前的相关信息。

## 2.3 IAP 升级流程图说明

IAP 升级流程图如图 5 所示：

- ① 初始化 uart、按键 GPIO 等，主要是初始化升级用到的外设，此例程中 uart 使用的是 DMA 方式接收，使用 uart idle 中断来判断一包数据是否接收完成。Xmodem 数据缓存数组 `received_packet_data[]` 在初始化 DMA 时，配置为 DMA 搬移的目的地址，即升级程序时上位机使用 Xmodem 协议分包发送给 MCU，MCU 采用 DMA + UART IDLE 中断方式来接收，然后 MCU 使用 uart 查询方式发送应答给上位机，程序中配置了 DMA 发送但是没有使用；
- ② 判断程序升级按键按下是否  $\geq 2s$ 。IAP 程序中设计了一个按键检测，让用户决定是否需要升级，当 MCU 上电或复位后重新运行程序时，会检测 PA0 引脚对应的按键是否按下，这个检测的时间是 5s，在这 5s 内按键持续按下超过 2s，则设置 `update_flag`，否则不设置 `update_flag`。程序会对 `update_flag` 进行判断如果大于等于 2s，则继续往下运行升级流程，如果不需要升级则跳到步骤⑪直接运行 APP 程序；
- ③ 如果需要升级，则调用 `xmodem_download()` 函数运行 Xmodem 协议；
- ④ Xmodem 协议 MCU 作为接收方向 PC 机发送字符 C，表明接收方打算使用 CRC 校验，一边发送字符 C，一边等待上位机发送 Xmodem 包；
- ⑤ 用户根据上位机工具接收到的字符 C，判断 MCU 已经处于接收状态。此时可以使用上位机工具发送 xmodem 协议包。本例程使用的上位机工具是 `teraterm`，与文档一起附带。后续章节对 `teraterm` 工具操作有介绍，详情请参考后续章节。

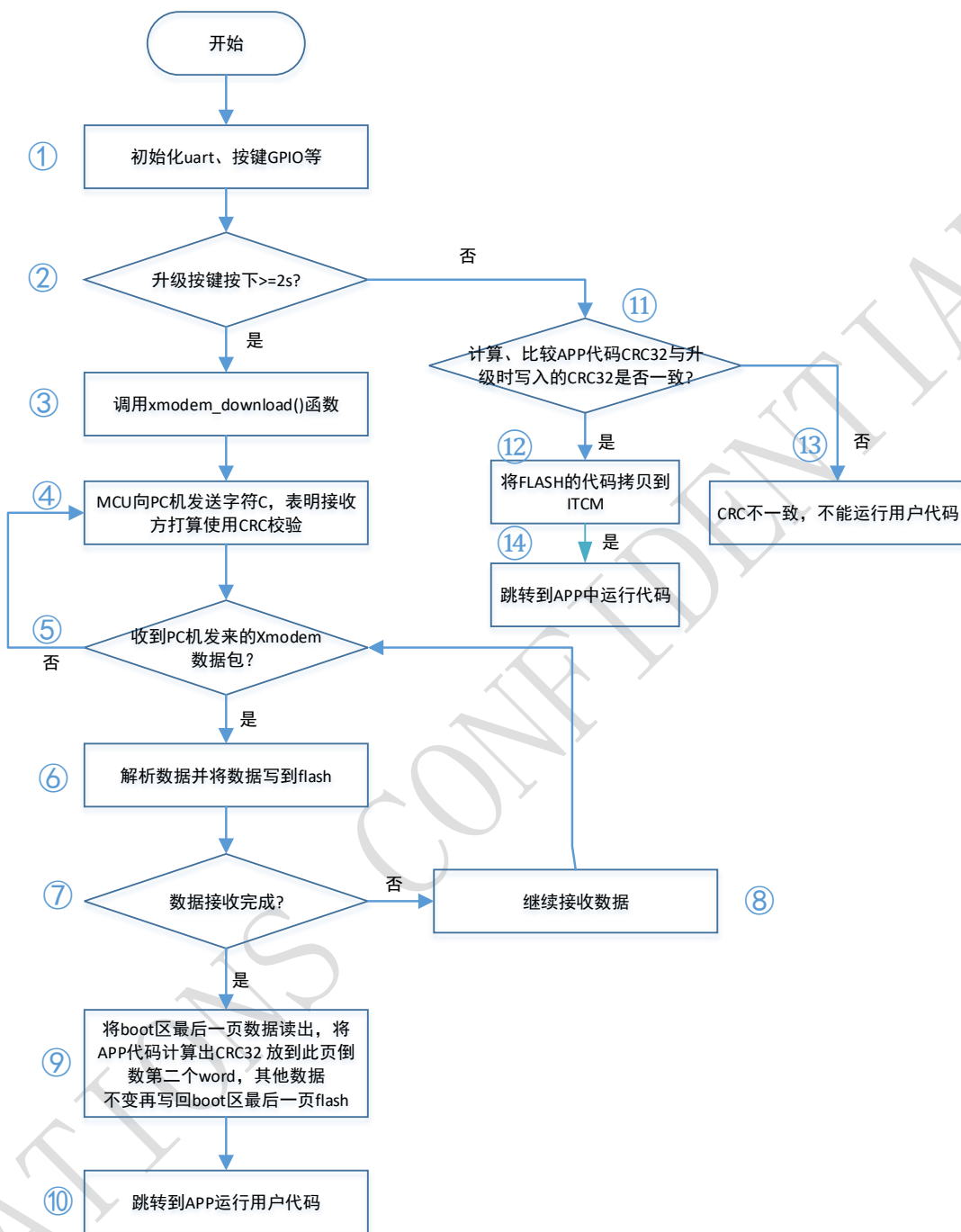


图 5

- ⑥ MCU采用 DMA + UART IDLE 中断方式接收数据，如果收到数据会产生 UART IDLE 中断。接收到的数据依据 Xmodem 协议进行解析并校验。本例程中 xmodem.c 源文件移植 github 网站上的开源代码(网址: <https://github.com/ferenc-nemeth/stm32-bootloader/tree/master>)，在数据处理环节做了部分改动。每确认接收一包数据，会调用 xmodem\_handle\_packet()函数实时写入到 flash 中。如果是第一包数据会调用

flash\_erase()去擦除整个 APP 区，擦除的起始和结束地址就是前面介绍的全局变量

FLASH\_APP\_START\_ADDRESS 和 FLASH\_APP\_END\_ADDRESS。

- ⑦ 判断数据是否接收完成。如果是最后一包数据,则最后一包数据的帧头为 X\_EOT(End Of Transmission);
- ⑧ 如果数据没有接收完,则跳到步骤⑤继续接收数据;
- ⑨ 如果数据接收完成,则将 BOOT 区中最后一页 flash 中的数据读出。BOOT 区最后一页的倒数第二个 word,地址 0x15003FF8-0x15003FFB 这四个字节用于存放用户代码的 APP CRC32。每此升级 MCU 都会根据 APP 区的起始和结束地址计算出 CRC32,记录到 BOOT 区最后一页的倒数第二个 word。最后一页中其他地址数据不变;
- ⑩ 跳转到 APP 区起始地址区,运行用户程序。在执行用户程序之前需要调用 UART, DMA 模块复位代码,将 IAP 用到的外设资源进行复位。
- ⑪ 如果用户在 MCU 在上电或者复位后没有按下升级按键,则先计算出用户 APP 代码的 CRC32,然后与步骤⑨中升级时保存的 CRC32 进行比较,判断两者是否相等;
- ⑫ 如果计算的 CRC32 与保存的 CRC32 相等,则运行用户代码;
- ⑬ 如果计算的 CRC32 与保存的 CRC32 不相等,则串口打印错误提示,并打印出 APP 区起始和结束地址;

## 2.4 注意事项

因为该系列芯片有高性能的 TCM (Tightly-Coupled Memory, 紧耦合内存), 并且有最大 1M 的 ITCM 可以使用, 为了发挥芯片的最佳性能, 此处介绍的升级方法是基于 APP 程序在 ITCM 中运行的情况下的。

ps: 如果在 SRAM 或者 FLASH 运行 App 程序, 基于上面介绍的基础, 结合芯片的数据手册以及用户手册, 找到 FLASH 和 SRAM 内存映射相关章节, 然后修改程序的运行地址以及重定向中断向量表即可, 就不加赘述了, 这里仅陈述程序运行在 ITCM 的情况下的 IAP 升级方法。

基于 App 程序在 ITCM 中运行的 IAP 升级流程: 芯片启动后, 在 Sram 运行 Bootloader 代码, 然后通过串口通讯, 从上位机软件下载 App 程序到 Flash 中。下载完成后, Bootloader 会将 App 程序从 Flash 的对应地址复制搬运到 ITCM 区域。复制完成后, 关闭已经打开的外设模块及相关中断, 再跳转到 ITCM 中运行代码。

注意: 这里 ITCM 只有 1M 大小, 所以固件必须在 1M 以下才可以使用这个升级方法。所以, 如果采用非乒乓升级的情况, App 固件也是不能超过 1M 的, 所以可升级的区域默认选择 BANK B 区。BANK 分区的内容, 后续章节“3 IAP 实现乒乓升级”会介绍到。如果固件实在太太大, 超过 1M 的话, 就不能使用这种方法, 而是要在 Flash 下载和执行代码, 也是可以正常运行的, 舍弃最佳性能去保证程序的正常运行。

程序要在 ITCM 段中运行首先要知道各区域的内存映射情况。当前使用芯片是 N32H785IIB7，可到其对应的数据手册《CN\_DS\_N32H785\_Series\_Datasheet.pdf》的章节“2.2.1 嵌入式 SRAM”、用户手册《EN\_UM\_N32H7xx\_Series\_User\_Manual\_V1.0.0.pdf》的章节“2.3.5 SIP FLASH”进行查阅。

### 2.2.1 嵌入式SRAM

#### TCM RAMs(只对 Cortex-M7)

TCM RAM 由 1MB SRAM 组成，可配置为 ITCM-RAM， DTCM-RAM 和 AXI-SRAM2 /3，软件可灵活配置 DTCM 和 ITCM 大小，剩余 SRAM 自动配置为 AXI-SRAM2 /3，粒度为 128KB。ITCM RAM 映射到地址 0x0000 0000，它只能被 Cortex®- M7 CPU 和 MDMA 访问(即使 CPU 处于休眠模式)，DTCM RAM 映射在地址 0x2000 0000，也只能通过 Cortex®- M7 CPU 内核和 MDMA 访问。

#### AXI SRAM

高达 1152KB 的 AXI RAM，它可以通过字节(8 位)，半字(16 位)，全字(32 位)或双字(64 位)访问。AXI SRAM 分为以下几种：

- 128kb 的 AXI SRAM 1 映射到地址 0x2400 0000
- 最大 512KB 的 AXI RAM 2(与 ITCM 和 DTCM 共享)，映射地址为 0x2402 0000
- 最大 512KB 的 AXI RAM 3(与 ITCM 和 DTCM 共享)，映射地址为 0x240A 0000

#### AHB SRAM

高达 352KB 的 AHB SRAM1-5,它们可以通过字节、半字(16 位)或字(32 位)访问。AXI SRAM 分为以下几种：

- 128 KB 的 AHB SRAM1 映射到地址 0x3000 0000。
- 128 KB 的 AHB SRAM2 被映射到地址 0x3002 0000。
- 32 KB 的 AHB SRAM3 被映射到地址 0x3004 0000。
- 32 KB 的 AHB SRAM4 被映射到地址 0x3004 8000。
- 32 KB 的 AHB SRAM5 被映射到地址 0x3005 0000。

AHB SRAM 5 被分成两个块，与 CANFD1-8 共享，用为 CANFD 数据帧缓冲 buffer。

图 6

Table 2-8 SIP Flash Main bank addressing

Flash Size (KB) <sup>(1)</sup>	Physical Address		Bus Address	
	Begin	End	Begin	End
2048	0x80020000	0x801FFFFF	0x15000000	0x151DFFFF
4096	0x80020000	0x803FFFFF	0x15000000	0x153DFFFF

(1) First 128KB always is fixed to be used as BOOTPATCH/OB\_FLASH.

图 7

由图 6、图 7 可知，FLASH 的起始地址是 0x1500\_0000，结束地址是 0x151D\_FFFF。

ITCM 的起始地址是 0x0000\_0000, 最大可分配 1MByte。

SRAM 的起始地址是 0x2400\_0000，最大可分配 1152Kbyte。

了解升级思路以及基本的内存映射之后，开始着手修改 Bootloader 和 App 程序。详见章节“2.4.1 Bootloader”和“2.4.1 App”。

NATIONS CONFIDENTIAL

## 2.4.1 Bootloader

将程序配置成在 ITCM 区域运行，首先，在工程设置的“C/C++”选项卡中添加宏定义 USING\_TCM。并将优化等级改(-O1)以上，这会影响到后续程序的执行速度，影响 App 程序的启动时间，非常重要。如图 9 显示。

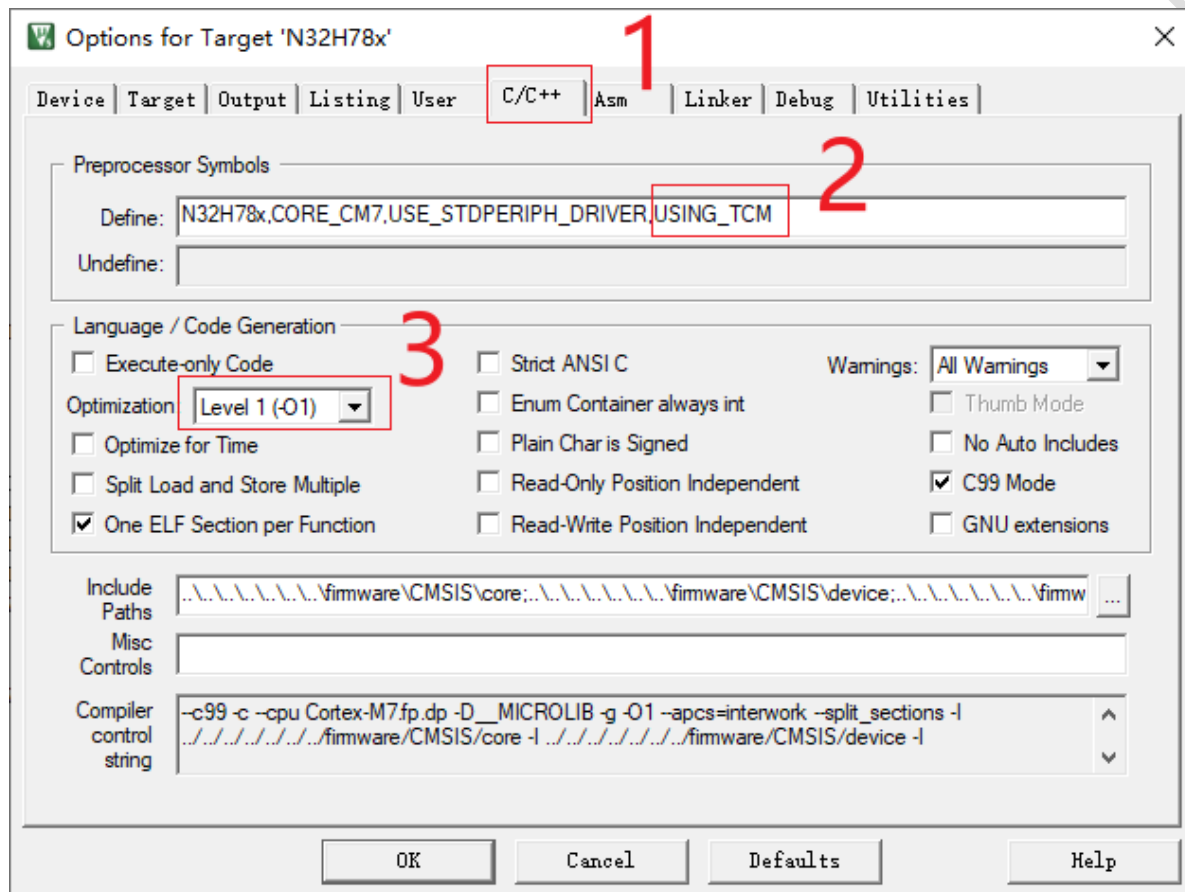
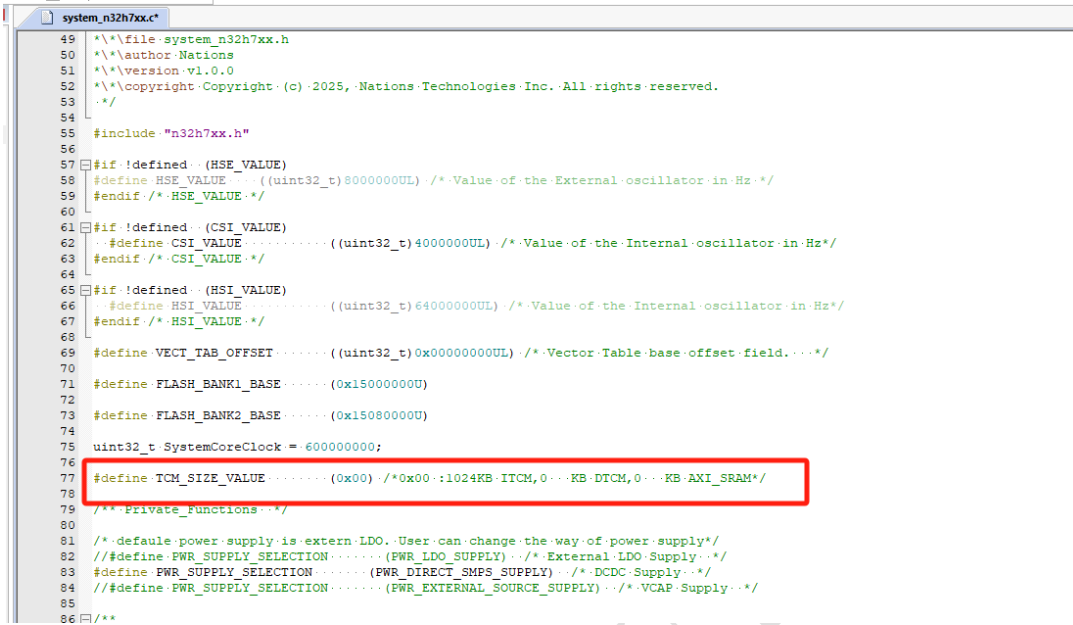


图 9

第二步，打开“system\_n32h7xx.c”文件，修改 TCM\_SIZE\_VALUE 的宏定义，将内存的 1M 可分配空间全部分配给 ITCM。如图 10。



```

49  /*\file:system_n32h7xx.h
50  /*\author:Nations
51  /*\version:v1.0.0
52  /*\copyright:Copyright (c) 2025, Nations Technologies Inc. All rights reserved.
53  */
54
55  #include "n32h7xx.h"
56
57  #if !defined (HSE_VALUE)
58  #define HSE_VALUE ((uint32_t)8000000UL) /* Value of the External oscillator in Hz */
59  #endif /* HSE_VALUE */
60
61  #if !defined (CSI_VALUE)
62  #define CSI_VALUE ((uint32_t)4000000UL) /* Value of the Internal oscillator in Hz */
63  #endif /* CSI_VALUE */
64
65  #if !defined (HSI_VALUE)
66  #define HSI_VALUE ((uint32_t)64000000UL) /* Value of the Internal oscillator in Hz */
67  #endif /* HSI_VALUE */
68
69  #define VECT_TAB_OFFSET ((uint32_t)0x00000000UL) /* Vector Table base offset field */
70
71  #define FLASH_BANK1_BASE (0x15000000U)
72
73  #define FLASH_BANK2_BASE (0x15080000U)
74
75  uint32_t SystemCoreClock = 600000000;
76
77  #define TCM_SIZE_VALUE ((0x00) /* 0x00:1024KB-ITCM,0...KB-DTCM,0...KB-AXI-SRAM */
78
79  /** Private Functions */
80
81  /* default power supply is extern LDO, User can change the way of power supply */
82  #define PWR_SUPPLY_SELECTION (PWR_LDO_SUPPLY) /* External LDO Supply */
83  #define PWR_SUPPLY_SELECTION (PWR_DIRECT_SMPS_SUPPLY) /* DCDC Supply */
84  #define PWR_SUPPLY_SELECTION (PWR_EXTERNAL_SOURCE_SUPPLY) /* VCAP Supply */
85
86  /**

```

图 10

第三步，修改 Bootloader 的分散加载文件，配置程序的运行、加载区域。如图 11 打开分散加载文件：

- (1) 点击魔法棒打开工程设置。
- (2) 点击“Linker”选项卡。
- (3) 取消勾选“Use Memory Layout from Target Dialog”选项。
- (4) 点击“...”选择分散加载文件。
- (5) 点击“edit”打开分散加载文件进行修改。

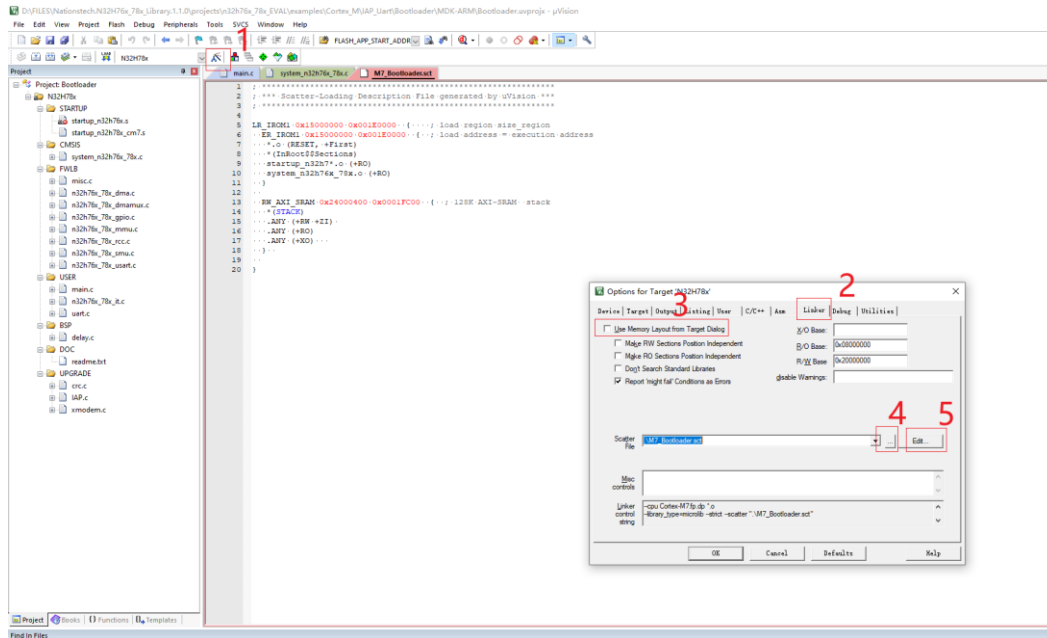
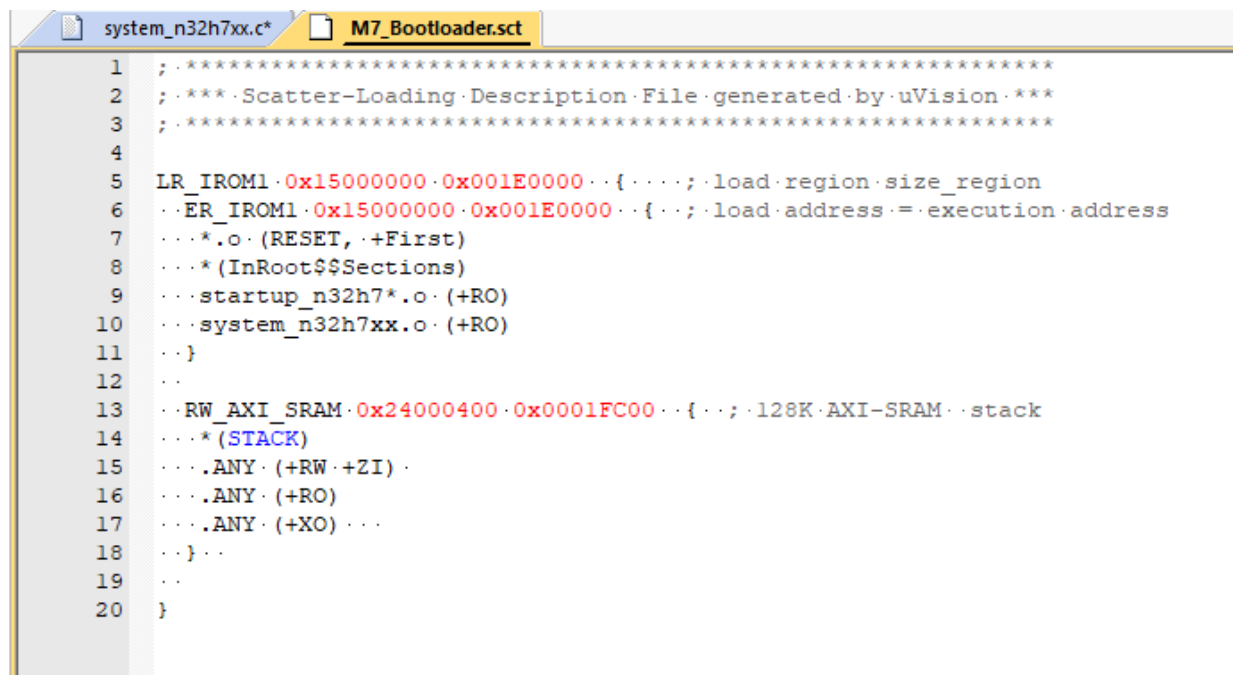


图 11



具体分散加载文件配置如图 12。



```

1  ; *****
2  ; *** Scatter-Loading-Description-File-generated-by-uVision ***
3  ; *****
4
5  LR_IROM1·0x15000000·0x001E0000·{····;·load·region·size·region
6  ··ER_IROM1·0x15000000·0x001E0000·{··;·load·address·=·execution·address
7  ···*.o·(RESET,·+First)
8  ···*(InRoot$$Sections)
9  ···startup_n32h7*.o·(+RO)
10 ···system_n32h7xx.o·(+RO)
11 ··}
12 ··
13 ··RW_AXI_SRAM·0x24000400·0x0001FC00·{··;·128K·AXI-SRAM·stack
14 ···*(STACK)
15 ···.ANY·(+RW·+ZI)·
16 ···.ANY·(+RO)
17 ···.ANY·(+XO)···
18 ··}··
19 ··
20 }

```

图 12

如图 12 所示的 Bootloader 的分散加载文件，Bootloader 从 Flash（0x1500\_0000）加载启动，然后将栈、读写数据、未初始化的数据、只读数据和可执行代码都分配到 Sram 中运行，提高 Bootloader 运行速度。

复位或者下载完成后，Bootloader 过程中会对 App 程序进行 CRC 校验，通过校验之后先将 Bootloader 中使用到的外设功能和中断全部关闭，再将需要执行的 App 程序复制到 ITCM 中。复制完成后，会跳转到 ITCM 中去执行 App 程序。图 13 红色方框中的语句就是执行将 App 程序从 Flash 复制转移到 ITCM 的操作

```

.../
void flash_jump_to_app(uint32_t address)
{
    ...uint32_t JumpAddress;

    .../*Judge whether the top of stack address is legal or not*/
    ...if((( (__IO uint32_t *) address) & 0x24000000) == 0x24000000)
    ...{
        ...uint32_t *pResetHandler = (uint32_t *) (address + 4);
        ...
        ...if(( *pResetHandler & FLASH_BANKA_START_ADDR) == ITCM_BASE_ADDR)
        ...{
            .../*Reset all peripherals*/
            ...RCC_EnableAHB5PeriphReset1(RCC_AHB5_PERIPHRS GPIOA | RCC_AHB5_PERIPHRS GPIOB); //reset GPIOA~GPIOB
            ...RCC_EnableAPB1PeriphReset3(RCC_APB1_PERIPHRS USART1); //reset usart1
            ...RCC_EnableAHB1PeriphReset1(RCC_AHB1_PERIPHRS DMAMUX1); //reset DMAMUX1
            ...RCC_EnableAHB1PeriphReset3(RCC_AHB1_PERIPHRS DMA1); //reset DMA1
            ...
            .../*Copy application data to itcm*/
            ...copy_app_to_itcm(FLASH_APP_END_ADDRESS - FLASH_APP_START_ADDRESS + 1);
            ...
            ...JumpAddress = ( (__IO uint32_t *) (ITCM_BASE_ADDR + 4));
            ...Jump_To_Application = (pFunction) JumpAddress;
            .../*Set Vector Table*/
            ...SCB->VTOR = ITCM_BASE_ADDR;
            .../*Initialize user application's Stack Pointer*/
            ...set_MSP(( __IO uint32_t *) address);
            .../*Jump to user application*/
            ...Jump_To_Application();
        }
    }
}

```

图 13

至此，Bootloader 的配置以及主要工作已经完成。

## 2.4.2 App

App 程序所需操作包含修改分散加载文件和重定向中断向量表两项。

首先，在工程设置的“C/C++”选项卡中添加宏定义 USING\_TCM。并将优化等级提高到-(O1)或以上。如图 14 显示。

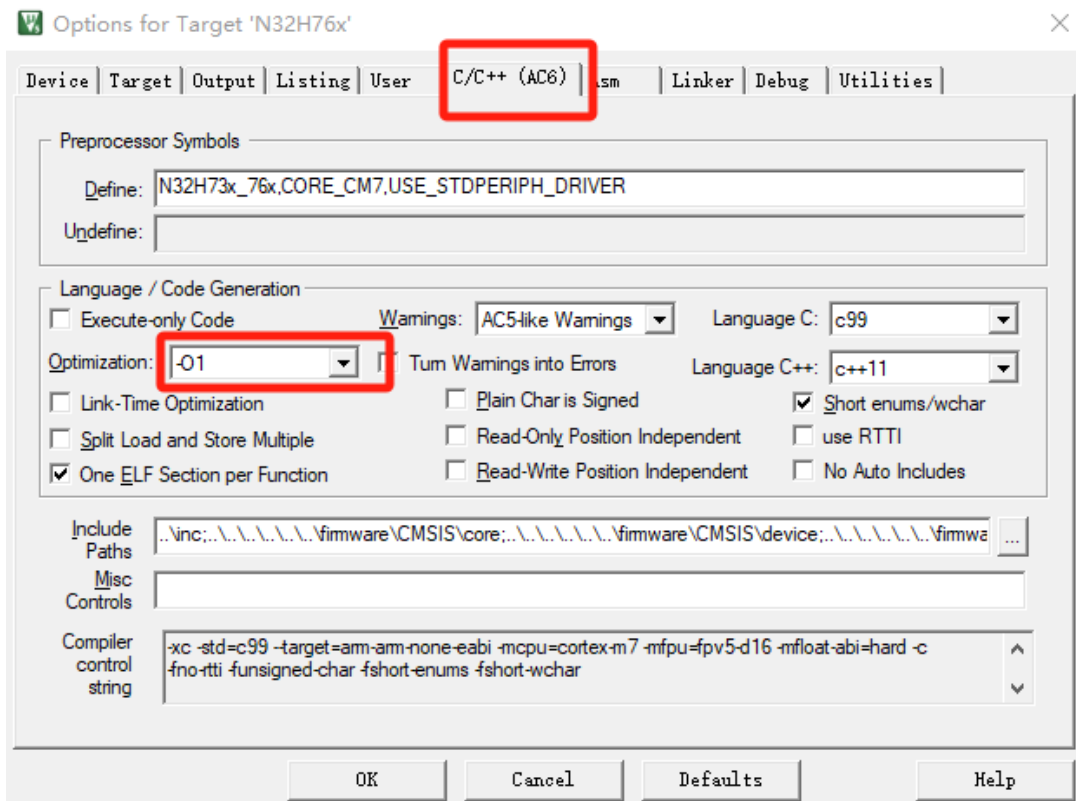


图 14

分散加载文件:

打开分散加载文件。参考图 15。

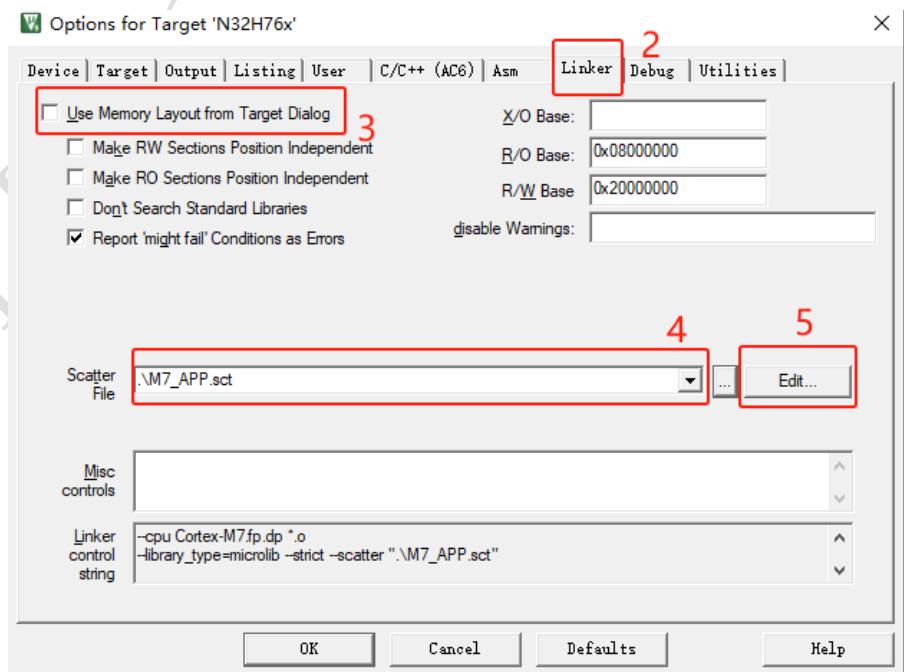


图 15

打开 App 工程的分散加载文件，需要如图 16 修改。

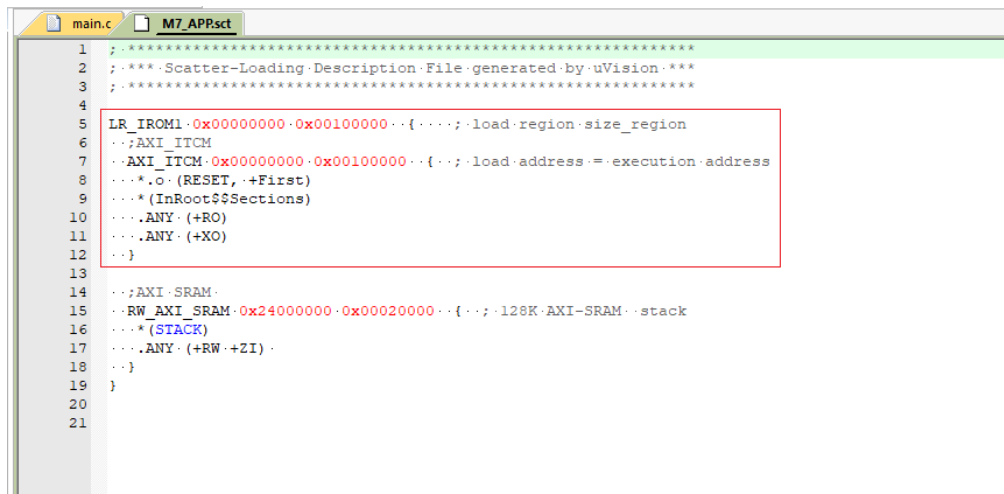


图 16

这里有两个地方需要注意：

- 第一，App 的可执行代码是放到 ITCM 中运行的，执行地址是从 0x0000\_0000 地址开始，空间大小是 0x0010\_0000（1MB）。
- 第二，APP 的加载地址也是从 0x0000\_0000 地址开始的，因为在 Bootloader 过程中，程序已经被搬运到 0x0000\_0000 开始的区域了，所以加载地址也是 0x0000\_0000 地址，而不是 IAP 升级时候实际下载的 Flash 地址。这样一来，使用 AB 分区的程序的重定向中断向量表的偏移量就可以统一，而不是需要根据实际储存的 Flash 地址做修改，保持了 AB 区程序代码的一致性。（重定向中断向量表的操作会在后续篇章说明）

### 重定向中断向量表:

App 的中断向量表重定向见图 16.

因为 Bootloader 过程中，App 代码已经从 Flash 复制到 ITCM 中了，所以中断向量表需要指向 ITCM 的起始地址。红方方框中的代，就是执行重定向中断向量表的操作。

```

main.c
49  /*\file main.c
50  /*\author Nations
51  /*\version v1.1.0
52  /*\copyright Copyright (c) 2025, Nations Technologies Inc. All rights reserved.
53  **/
54
55  #include "main.h"
56  #include "delay.h"
57  #include "uart.h"
58
59
60  #define ITCM_BASE_ADDR 0x00000000
61  #define VECT_APP_ITCM_OFFSET 0x0
62
63  void GPIO_Configuration(void);
64  void RCC_Configuration(void);
65
66  __IO uint8_t RxCounter = 0x00;
67  uint8_t RxBuffer[256];
68
69  /**
70  /*\name main.
71  /*\fun Main program.
72  /*\param none
73  /*\return none
74  */
75  int main(void)
76  {
77      SCB->VTOR = ITCM_BASE_ADDR | VECT_APP_ITCM_OFFSET; /* Vector Table Relocation in Internal FLASH */
78      /* Initialize System Clock */
79      RCC_SetSysClkToMode0();
80      /* RCC configuration */
81      RCC_Configuration();
82      /* Uart configuration */
83      uart_init();
84
85      printf("\r\n running APP A..., %s, %s\r\n", __DATE__, __TIME__);
86      printf("\r\n please send characters, and device will send back, check the received data\r\n");
87
88

```

图 17

### 3 IAP 实现乒乓升级

乒乓升级的原理是人为的将 MCU 的 flash 分成 A、B 两个部分(A、B 大小由用户自己定义)，第一次用 IAP 下载代码时将代码下载到 A 区域，下载完成后程序在 A 区运行，B 区保留。第二次升级代码时，将代码下载到 B 区域，代码升级完成后直接在 B 区域运行，A 区域保留。再下次升级时 A 区域存放代码，B 区域保留，如此往复称之为乒乓升级。

#### 3.1 IAP 乒乓升级流程介绍

乒乓升级相对于普通 IAP 升级多了 A、B 分区，以及记录 Bank A 和 Bank B 的升级次数。IAP BOOT 区最后四个字节用于存放 Bank A 及 Bank B 的升级次数，每次升级完会将升级次数读出来加 1 后再回写到此地址处。其他部分与普通 IAP 升级流程一致。

本例程中将 APP 用户区 flash 大小分成两个大小相等的 A,B 区，不同型号的 MCU flash 大小不同，具体请参考前面章节关键代码的容量宏定义说明。此处以本工程所用 N32H785IIB7 型号进行说明，IAP BOOT 分配 16Kbytes，剩下 1092K。BankA 分配 880Kbytes，BankB 分配 1Mbytes。用户可以根据自身项目需要来分配合适的大小。本例程 flash 分配如图 8 所示：

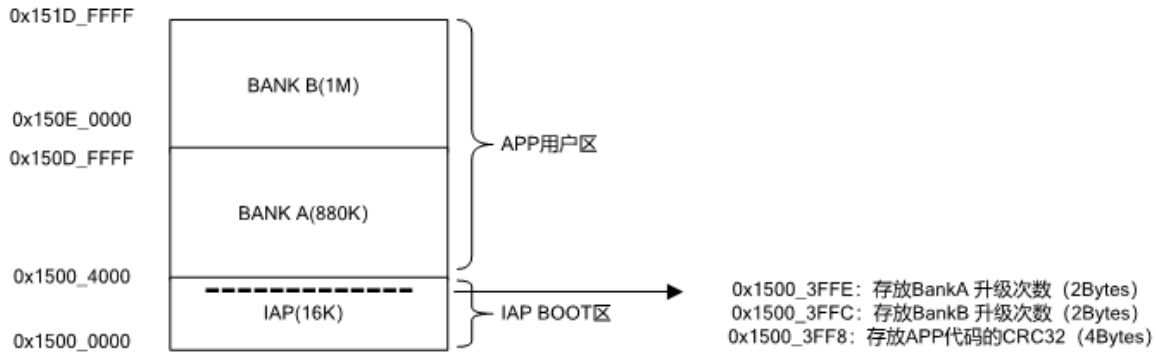


图 18

乒乓升级的流程图如下：

- ① 初始化 uart、按键 GPIO 等，主要是初始化升级用到的外设，此例程中 uart 使用的是 DMA 方式接收，使用 uart idle 中断来判断一包数据是否接收完成。Xmodem 数据缓存数组 received\_packet\_data[] 在初始化 DMA 时，配置为 DMA 搬移的目的地址，即升级程序时上位机使用 Xmodem 协议分包发送给 MCU，MCU 采用 DMA + UART IDLE 中断方式来接收，然后 MCU 使用 uart 查询方式发送应答给上位机，程序中配置了 DMA 发送但是没有使用；
- ② 判断程序升级按键按下是否  $\geq 2s$ 。IAP 程序中设计了一个按键检测，让用户决定是否需要升级，当 MCU 上电或复位后重新运行程序时，会检测 PA0 引脚对应的按键是否按下，这个检测的时间是 5s，在这 5s 内按键持续按下超过 2s，则设置 update\_flag，否则不设置 update\_flag。程序会对 update\_flag 进行判断如果大于等于 2s，则继续往下运行升级流程，如果不需要升级则跳到步骤⑫直接运行 APP 程序；
- ③ 调用 check\_bank() 函数，根据 IAP BOOT 区最后四个字节 Bank A 及 Bank B 的升级次数判断，即将升级的 APP 程序下载到哪个 Bank。如上一次使用 BankA，则下一次使用 BankB。程序中会根据不同的 Bank，给 FLASH\_APP\_START\_ADDRESS 和 FLASH\_APP\_END\_ADDRESS 赋不同的值。
- ④ 如果需要升级，则调用 xmodem\_download() 函数运行 Xmodem 协议；
- ⑤ 用户根据上位机工具接收到的字符 C，判断 MCU 已经处于接收状态。此时可以使用上位机工具发送 xmodem 协议包。
- ⑤ MCU 向 PC 机发送字符 C，表明 MCU 作为 Xmodem 协议数据接收方打算使用 CRC 校验；

- ⑥ 用户根据上位机工具接收到的字符 C，判断 MCU 已经处于接收状态。此时可以使用上位机工具发送 xmodem 协议包。

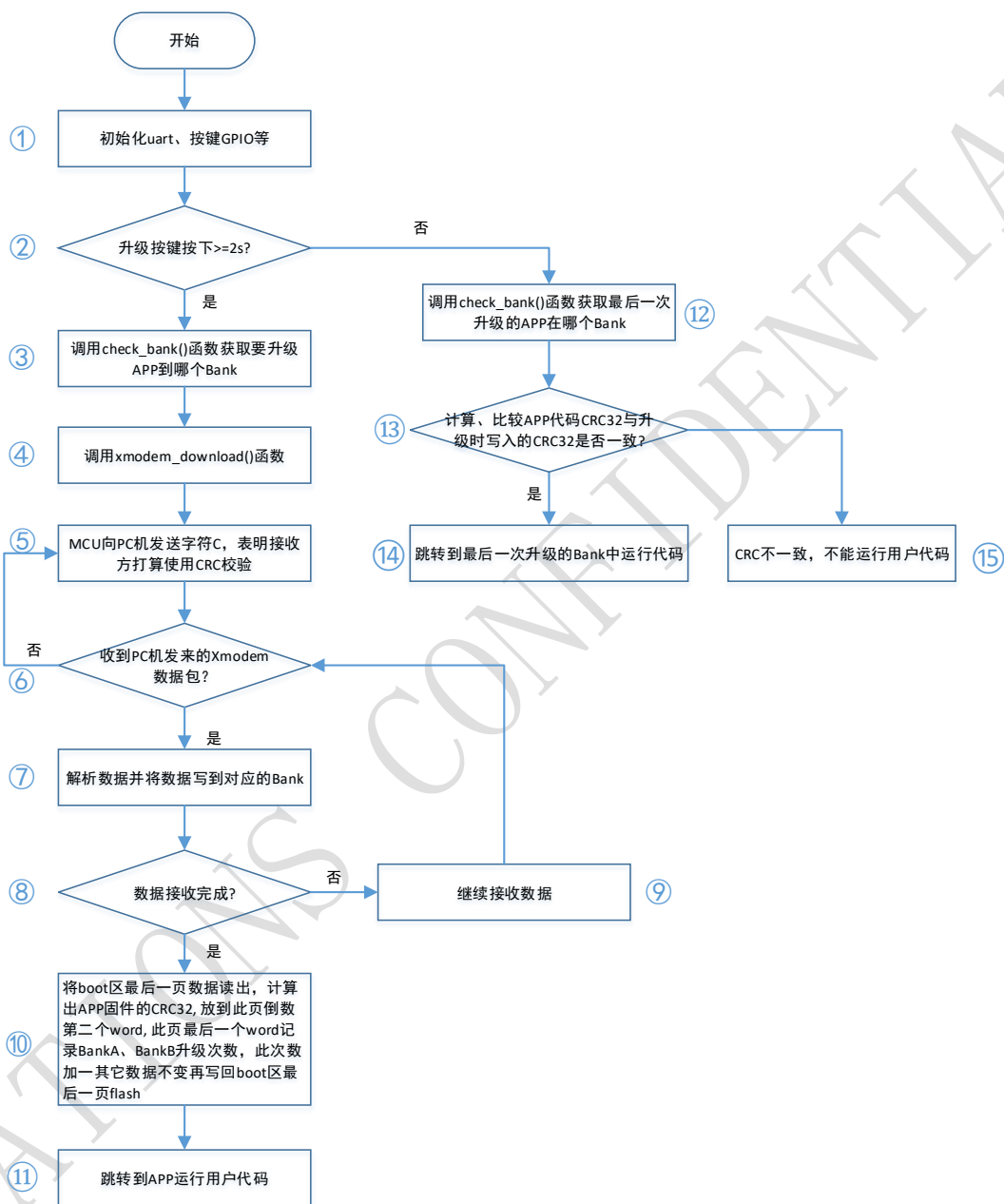


图 19 IAP 乒乓升级流程图

- ⑦ MCU 采用 DMA + UART IDLE 中断方式接收数据，如果收到数据会产生 UART IDLE 中断。接收到的数据依据 Xmodem 协议进行解析并校验。本例程中 xmodem.c 源文件移植 github 网站上的开源代码(网址：<https://github.com/ferenc-nemeth/stm32-bootloader/tree/master>)，在数据处理环节做了部分改动。每确认接收一包数据，会调用 xmodem\_handle\_packet()函数实时写入到 flash 中。如果是第一包数据会调用



flash\_erase()去擦除整个 APP 区，擦除的起始和结束地址就是前面介绍的全局变量

FLASH\_APP\_START\_ADDRESS 和 FLASH\_APP\_END\_ADDRESS。

- ⑧ 判断数据是否接受完成。如果是最后一包数据,则最后一包数据的帧头为 X\_EOT(End Of Transmission);
- ⑨ 如果数据没有接收完,则跳到步骤⑥继续接收数据;
- ⑩ 如果数据接收完成,则将 BOOT 地址空间中最后一页 flash 中的数据读出。计算出 APP 固件的 CRC32,放到此页倒数第二个 word。最后四个字节用于存放 BankA, BankB 区的升级次数。BankA 或者 BankB 区每升级一次响应的次数加 1。将记录 Bank 的次数从 flash 读出后,进行加 1 操作,再和 CRC32 一起回写到 flash 中,最后一页 flash 中其他地址数据不变;
- ⑪ 跳转到刚升级的对应的 Bank,运行用户 APP 程序;
- ⑫ 如果步骤②中用户没有按下按键 $\geq 2s$ ,则调用 check\_bank()函数,查询即将要跳转的 APP 应用程序在哪个 Bank; 获取到 APP 区的 start,end 地址;
- ⑬ 根据步骤⑫获取的 APP 区 start,end 地址,计算 APP 区代码 CRC32,与步骤⑩中保存的 CRC32 进行比较;
- ⑭ 如果上一步比较的 CRC32 结果相等,则运行用户 APP 程序;
- ⑮ 如果比较的 CRC32 结果不相等,则串口打印错误提示,并打印出 APP 区起始和结束地址。

## 3.2 teraterm 工具使用

打开文件包中附带的 ttermpro.exe 工具,选择串口,将串口波特率配成 115200。如下图所示:

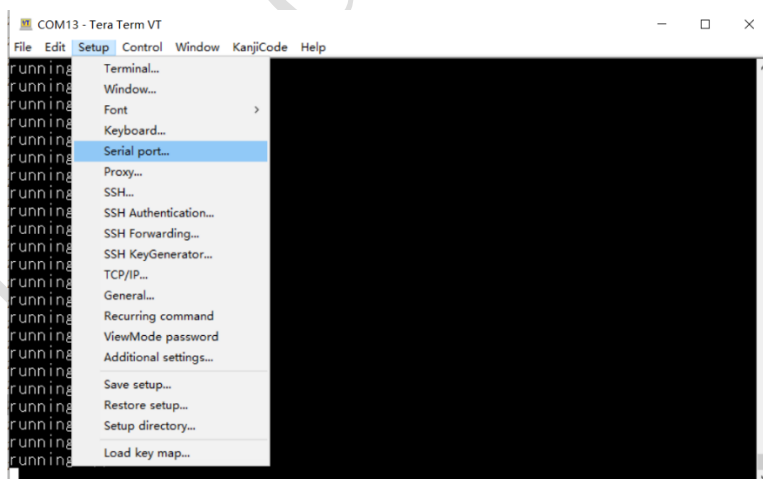


图 20



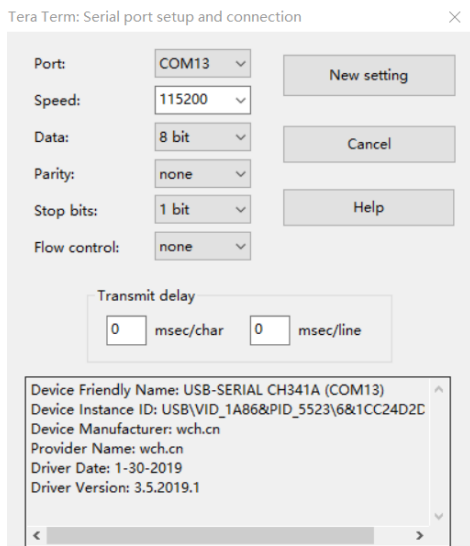


图 21

将 IAP 升级程序下载到 MCU 中，按下开发板上升级按键(WAKEUP)按键，复位 MCU，可以看到 teraterm 界面上有 5s 倒计时打印，5s 后会持续有 c 字符打印，等待上位机发送数据，这是 MCU 通知上位，机数据包发送使用 CRC 校验。

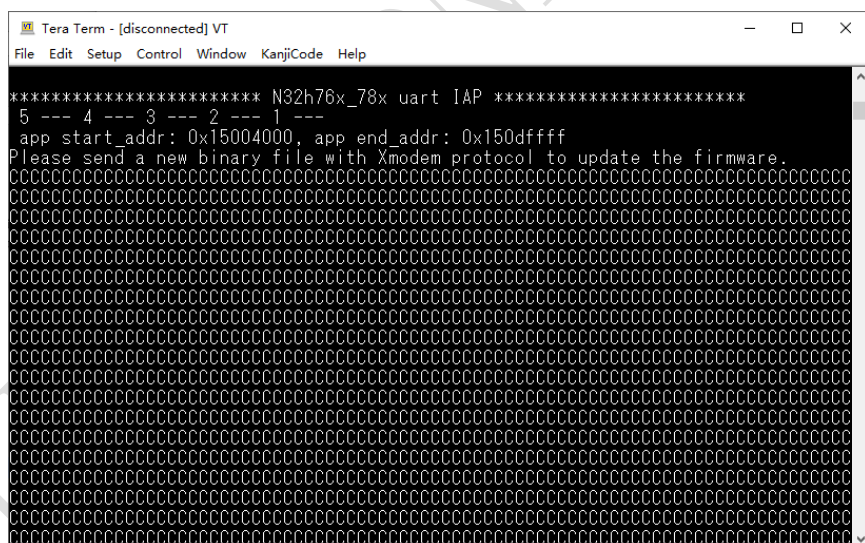


图 22

然后，选择 File -> Transfer -> XMODEM -> Send 选择要升级的 app 程序。

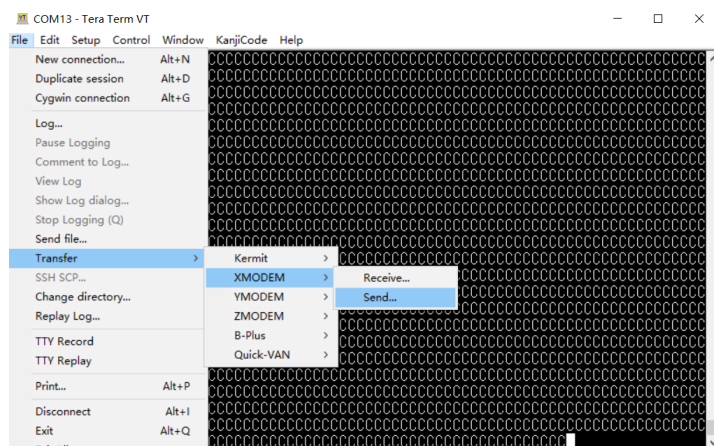


图 23

这里选择例程中的 APP\_A.bin 或者 APP\_B.bin，下载过程中会显示下载的进度。

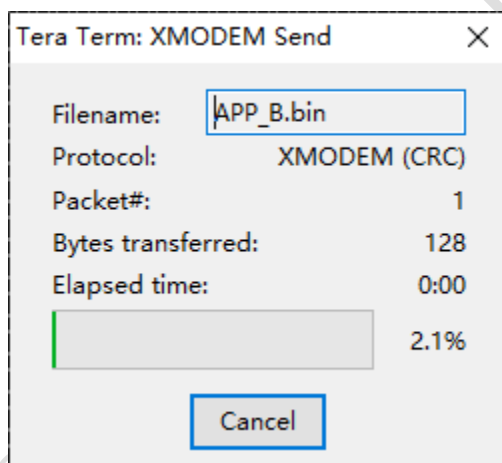


图 24

下载完成后，程序会自动跳转到 app 用户程序，并开始执行用户程序。

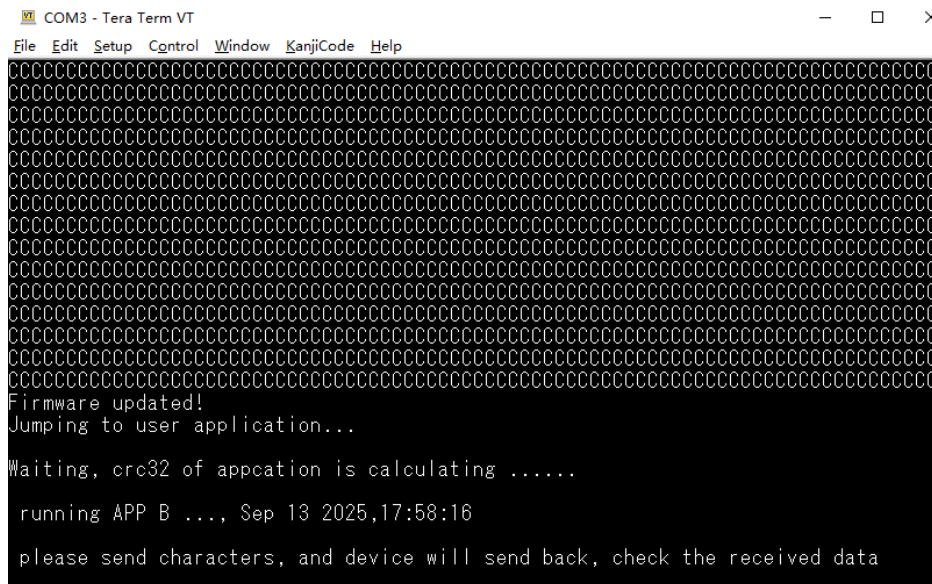


图 25

按下复位按键，复位 MCU，重启后后程序会自动跳转到 Bank 执行用户程序。

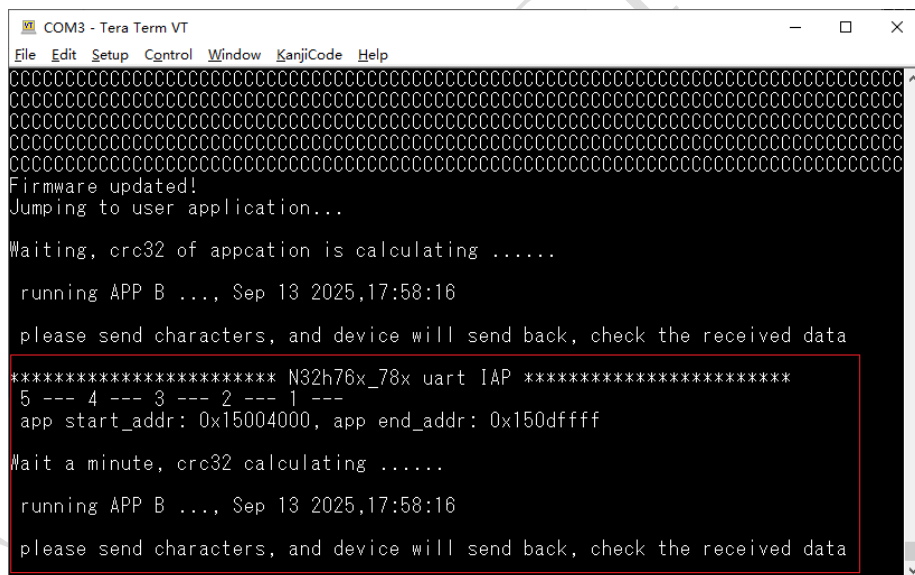


图 26

BankB 中应用程序实现的功能为 uart 串口将 RX 接口接收到的数据再通过 TX 功能发送出去，测试结果如图 27 所示：

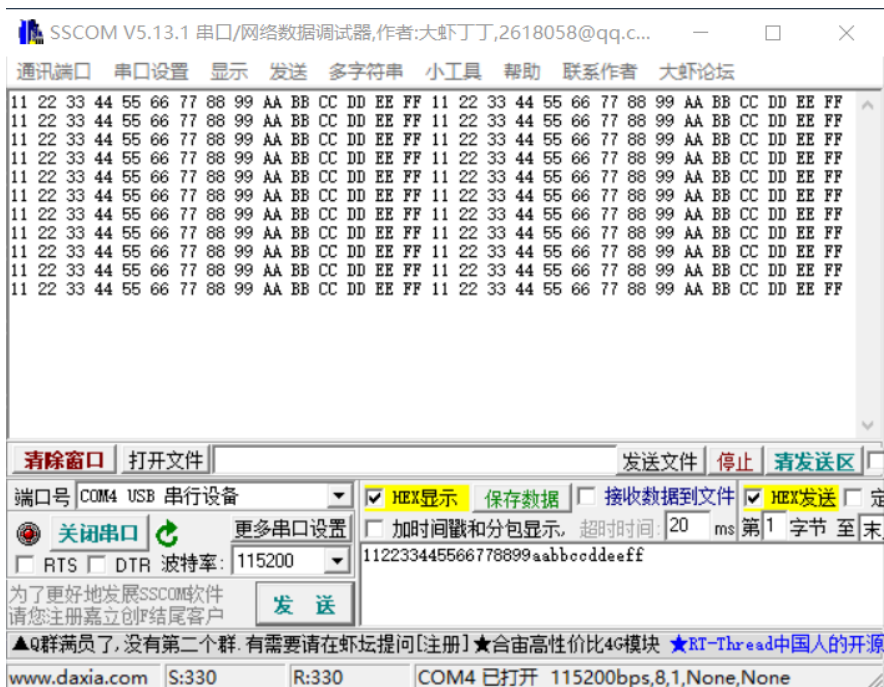


图 27

## 4 历史版本

版本	日期	备注
V1.0	2025.9.16	新建文档

## 5 声明

国民技术股份有限公司（下称“国民技术”）对此文档拥有专属产权。依据中华人民共和国的法律、条约以及世界其他法域相适用的管辖，此文档及其中描述的国民技术产品（下称“产品”）为公司所有。

国民技术在此并未授予专利权、著作权、商标权或其他任何知识产权许可。所提到或引用的第三方名称或品牌（如有）仅用作区别之目的。

国民技术保留随时变更、订正、增强、修改和改良此文档的权利，恕不另行通知。请使用人在下单购买前联系国民技术获取此文档的最新版本。

国民技术竭力提供准确可信的资讯，但即便如此，并不推定国民技术对此文档准确性和可靠性承担责任。

使用此文档信息以及生成产品时，使用者应当进行合理的设计、编程并测试其功能性和安全性，国民技术不对任何因使用此文档或本产品而产生的任何直接、间接、意外、特殊、惩罚性或衍生性损害结果承担责任。

国民技术对于产品在系统或设备中的应用效果没有任何故意或保证，如有任何应用在其发生操作不当或故障情况下，有可能致使人员伤亡、人身伤害或严重财产损失，则此类应用被视为“不安全使用”。

不安全使用包括但不限于：外科手术设备、原子能控制仪器、飞机或宇宙飞船仪器、所有类型的安全装置以及其他旨在支持或维持生命的应用。

所有不安全使用的风险应由使用人承担，同时使用人应使国民技术免于因为这类不安全使用而导致被诉、支付费用、发生损害或承担责任时的赔偿。

对于此文档和产品的任何明示、默示之保证，包括但不限于适销性、特定用途适用性和不侵权的保证责任，国民技术可在法律允许范围内进行免责。

未经明确许可，任何人不得以任何理由对此文档的全部或部分进行使用、复制、修改、抄录和传播。