

文档编号: AN082

上海东软载波微电子有限公司

应用笔记

ES8P 系列 MCU Bootloader

修订历史

版本	修改日期	更改概要
V1.0	2017-4-27	初版发布
V1.1	2017-11-28	增加 bootloader 移植说明，修改程序启动方式描述

地 址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层

邮 编：200235

E-mail: support@essemi.com

电 话：+86-21-60910333

传 真：+86-21-60914991

网 址：http://www.essemi.com/

版权所有©

上海东软载波微电子有限公司

本资料内容为上海东软载波微电子有限公司在现有数据资料基础上慎重且力求准确无误编制而成，本资料中所记载的实例以正确的使用方法和标准操作为前提，使用方在应用该等实例时请充分考虑外部诸条件，上海东软载波微电子有限公司不担保或确认该等实例在使用方的适用性、适当性或完整性，上海东软载波微电子有限公司亦不对使用方因使用本资料所有内容而可能或已经带来的风险或后果承担任何法律责任。基于使本资料的内容更加完善等原因，上海东软载波微电子有限公司保留未经预告的修改权。使用方如需获得最新的产品信息，请随时用上述联系方式与上海东软载波微电子有限公司联系。

目 录

内容目录

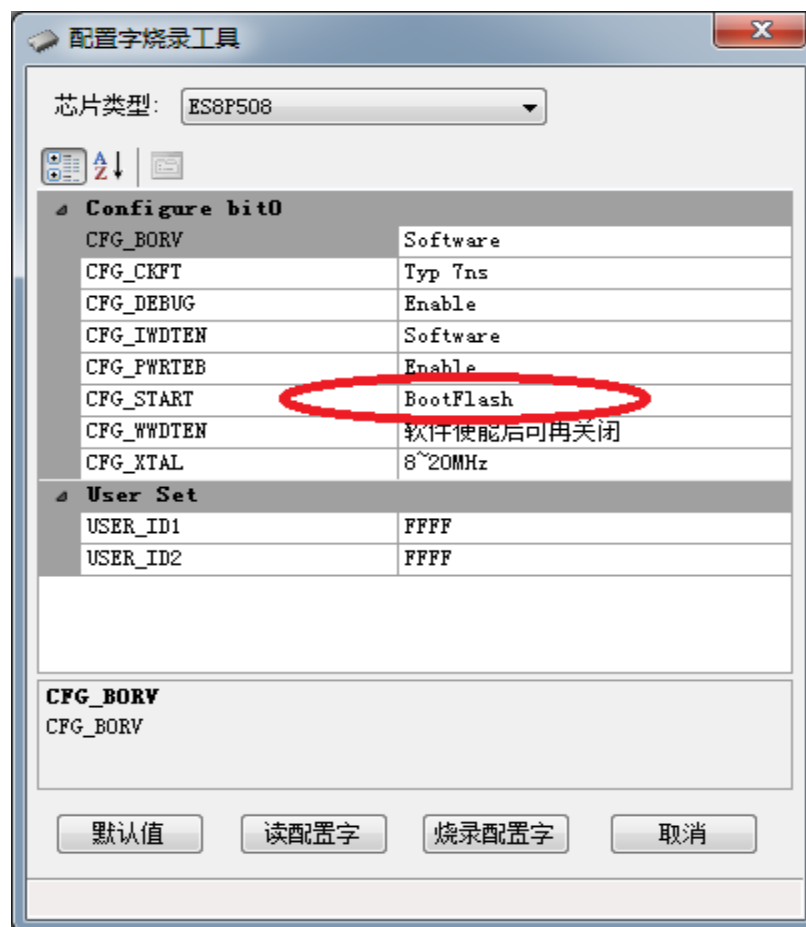
第 1 章	原理概述	4
1.1	工程建立与空间分配	4
1.2	程序引导过程	5
1.3	通信交互	5
第 2 章	程序设计说明	6
2.1	Bootloader 程序	6
2.1.1	指定程序存储区域	6
2.1.2	引导程序到 App	7
2.1.3	Bootloader 移植	7
2.2	App 程序	8
2.2.1	指定程序存储区域	8
2.2.2	进入 App 程序	8
2.2.3	调试 App 程序	8
2.2.4	编译生成 App.bin 文件	10
2.3	IAP 操作	11
2.4	RAM 区域分配	11
第 3 章	Bootloader 演示	12
3.1	概述	12
3.2	演示流程	12
3.2.1	流程图	12
3.2.2	更新步骤	13
附录 1	Ymodem 通信协议	16

第1章 原理概述

Bootloader 是实现芯片程序远程更新的方法。ES8P 系列 MCU 的 Bootloader 实质是进行 IAP (In Application Programming) 和中断向量重映射。本文提供一个 demo 方案, 供用户参考如何实现远程程序更新。

1.1 工程建立与空间分配

Bootloader 程序和 App 程序两个工程需要分开建立和设计, 先把 Bootloader 程序生成的 hex 烧录到 MCU 中, 然后 App 程序生成的 hex 借助上位机和 Bootloader 程序烧录到 MCU 中运行。对于内部没有做 APPFlash 和 BootFlash (以下统一将 App 所在的 FLASH 称为 APPFlash, 将 Bootloader 所在的 FLASH 称为 BootFlash) 地址空间划分的芯片来说, Bootloader 必须放在 FLASH 的起始位置, App 程序放在剩余的 FLASH 空间里, APPFlash 和 BootFlash 空间需要人为地划分, 具体操作请参考第 2 章; 对于内部已经做出 APPFlash 和 BootFlash 地址空间划分的芯片来说, 程序的起始位置是可以通过配置字的方式来选择。例如 ES8P508, 内部已经做了 APPFlash 和 BootFlash 空间的划分, 用户仅需要在下载代码前烧录好配置字, 选择 BootFlash 启动或者 APPFlash 启动。



1.2 程序引导过程

芯片复位后的中断向量表默认是在 **BootFlash** 起始的一段字节空间里，这时运行的是 **Bootloader** 程序。当 **Bootloader** 程序要将程序引导到 **App** 区执行时，首先要将中断向量表重映射到 **App** 程序的中断向量表所在的位置，然后从 **App** 程序的中断向量表里得到 **App** 程序的栈顶，最后再操作 **PC** 指向 **App** 程序中断向量表指定的程序入口地址。

1.3 通信交互

Bootloader demo 程序现仅支持 **UART** 与上位机进行通信，实现 **App** 程序更新。**App** 程序的 **bin** 文件通过上位机解析，然后把解析后的数据通过 **UART** 发送到 **Bootloader**，由 **Bootloader** 进行 **IAP**，从而达到更新 **App** 程序的目的。

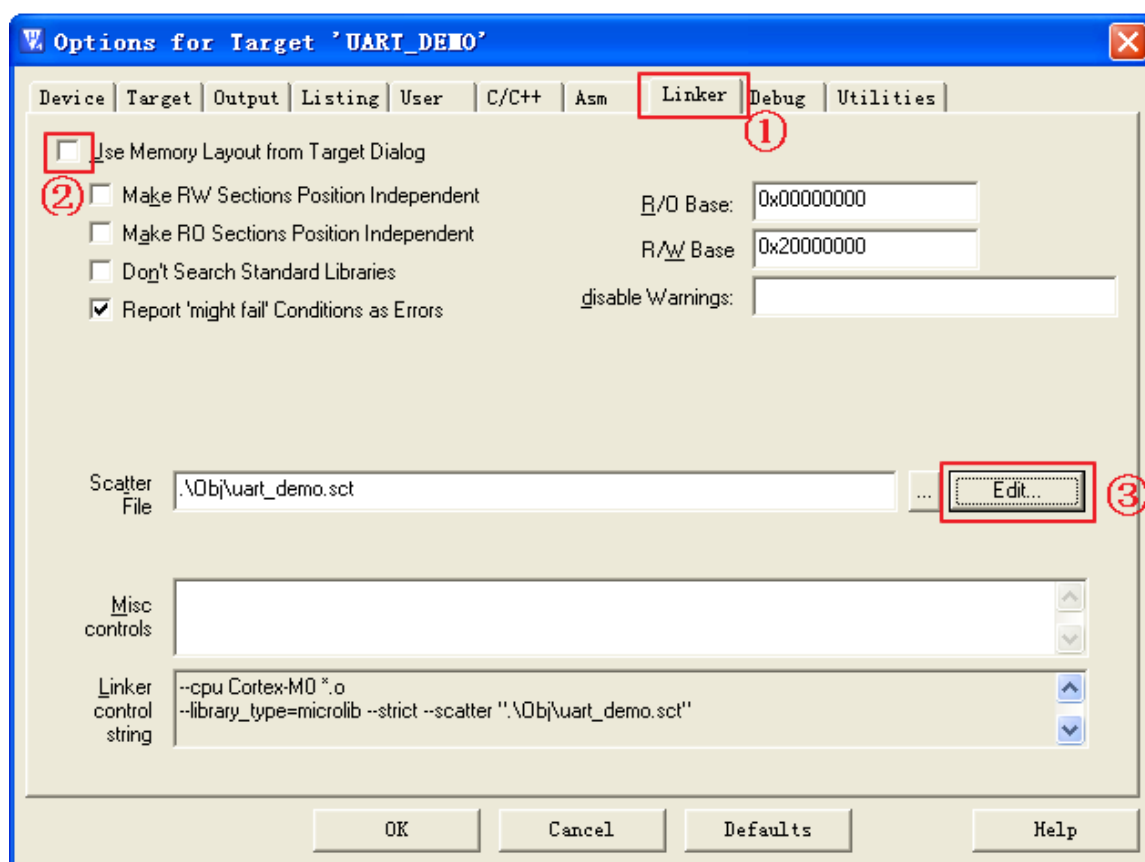
第2章 程序设计说明

2.1 Bootloader程序

工程师在设计 Bootloader 程序时，需要单独建立工程，并使程序分配在 FLASH 的起始位置。指定程序存储区域采用分散加载的方式来实现。

2.1.1 指定程序存储区域

分散加载文件即工程里的.sct 文件。可以为每一个代码或数据区在装载和执行时指定不同的存储区域地址，通过使用分散加载机制，可以使用文本文件中的描述为链接器指定映像的内存映射，如下所示进入并编辑.sct 文件：



```

*****
*** Scatter-Loading Description File generated by uVision ***
*****

LR_IROM1 0x00000000 0x00002000 { ;load region size_region
    ER_IROM1 0x00000000 0x00002000 { ;load address = execution address
        *.o (RESET, +FRIST)      ; 所有的.o 文件保存在这个区域
        *(InRoot$$Sections)      ; 所有的连接文件保存在这个区域
        .ANY (+RO)               ; 除了指定的 RO 段，其他所有 RO 段在这个区域执行
    }
}
    
```

```

RW_IRAM1 0x20000000 0x00003000 { ; RW data
    .ANY (+RW +ZI)          ; 除了指定的 RW 段, ZI 段, 其他所有 RW 段和 ZI 段在这个区域执行
    lib_flashiap.o(+RO)      ; lib_flashiap.o 文件的 RO 段在这区域执行
}

```

在以上.sct 文件里,“LR_IROM1”和“ER_IROM1”指定 FLASH 空间,分别为“存储空间”和“执行空间”,这两段空间通常相同,用来存储程序中的 RO 端。“RW_IRAM1”指定程序 RAM 空间,除了用来存储程序空间中的 RW 段和 ZI 段以外,还可以指定特定的 RO 段存储在 RAM 空间,如果 IAP 程序需要在 RAM 空间执行,那么需要在这里添加,如上面例子里对“lib_flashiap.o”文件的操作。

2.1.2 引导程序到App

在 Bootloader 程序里,除了需要对 App 程序进行正确的 IAP 以外,还需要将程序正确地引导到 App 区域执行,在提供的 Bootloader Demo 程序中的“JumpToApp”函数实现了这个功能。主要步骤如下:

1. __set_PRIMASK(1); //关中断
2. 关闭所有在 Bootloader 程序中被打开的中断和异常;
3. 把在 Bootloader 程序中使用过的外设关闭;
4. 进行中断向量重映射,把中断向量的位置映射到 App 程序的起始位置;
5. __set_PRIMASK(0); //开中断
6. 把 SP 寄存器配置为目标程序的栈顶地址,App 程序栈顶地址存储在 App 程序中断向量表的第一个 word;
7. 程序跳转到 App 的复位向量所指向的位置开始执行 App 程序,复位向量所指向的地址存储在 App 程序中断向量表的第二个 word。

2.1.3 Bootloader移植

demo 中 FLASH 相关数值都以宏定义的方式定义在“flash_if.h”文件里,程序中涉及到 FLASH 数值的时候都会用宏定义替换,在不同平台移植前需要先定义好 FLASH 的相关参数。相关定义如下:

APPFLASH_ADDRESS	定义 APP 程序的起始位置
APPFLASH_PAGE_QUANTITY	定义 APP 程序占 FLASH 的页数
APPFLASH_SIZE	定义 APP 程序区的 FLASH 大小
BOOTFLASH_ADDRESS	定义 Bootloader 程序的起始位置
BOOTFLASH_PAGE_QUANTITY	定义 Bootloader 程序占的 FLASH 的页数
FLASH_PAGE_SIZE	定义 Bootloader 程序区的 FLASH 大小
SRAM_START_ADDRESS	定义 SRAM 的起始位置

例如 ES8P508FJ, APP FLASH 的起始位置为 0x00000000,而 Bootloader FLASH 的起始地址为 0x00010000,那么 APPFLASH_ADDRESS 的值为 0x00000000, APPFLASH_PAGE_QUANTITY

的值为 64，APPFLASH_SIZE 为 0x00010000。详细数值请查阅相关芯片数据手册。

在 ES8P 系列 MCU 的 Bootloader demo 里对于 FLASH 的擦写提供了两种实现方式。第一种是调用"lib_flashiap.c"中提供的库函数，操作 IAP 寄存器的方式实现；第二种则是利用芯片内部固化的 IAP 模块实现 FLASH 的擦写。如果芯片内部已经固化了 IAP 模块，则可以调用固化的函数，这种方式可以节省芯片的资源。

2.2 App程序

App 程序是用户需要更新的程序，是 Bootloader 更新的部分，其需要单独建立工程，将用户程序分配到剩余的地址空间。通过编写.sct 文件这种方式来指定程序存储区域。

2.2.1 指定程序存储区域

如 Bootloader 程序操作。

分散加载文件（SCT 文件）如下：

```
*****  
;   
*** Scatter-Loading Description File generated by uVision ***   
*****  
;   
LR_IROM1 0x00002000 0x00020000 { ;load region size_region  
    ER_IROM1 0x00002000 0x00020000 { ;load address = execution address  
        *.o (RESET, +FRIST)      ; 所有的.o 文件保存在这个区域  
        *(InRoot$$Sections)      ; 所有的连接文件保存在这个区域  
        .ANY (+RO)               ; 除了指定的 RO 段，其他所有 RO 段在这个区域执行  
    }  
    RW_IRAM1 0x20000000 0x00003000 { ; RW data  
        .ANY (+RW +ZI)           ; 除了指定的 RW 段，ZI 段，其他所有 RW 段和 ZI 段在这个区域执行  
        lib_flashiap.o(+RO)      ; lib_flashiap.o 文件的 RO 段在这个区域执行  
    }  
}
```

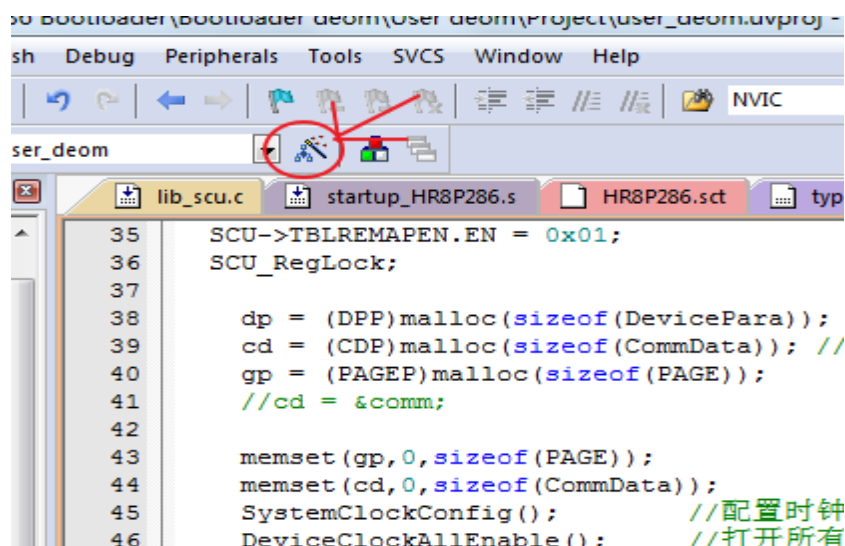
2.2.2 进入App程序

当 App 程序已烧录到 MCU 中，要想运行 App 程序，通过点击上位机（上位机操作见第 3 章介绍）进入 App 状态，就可进入 App 程序。

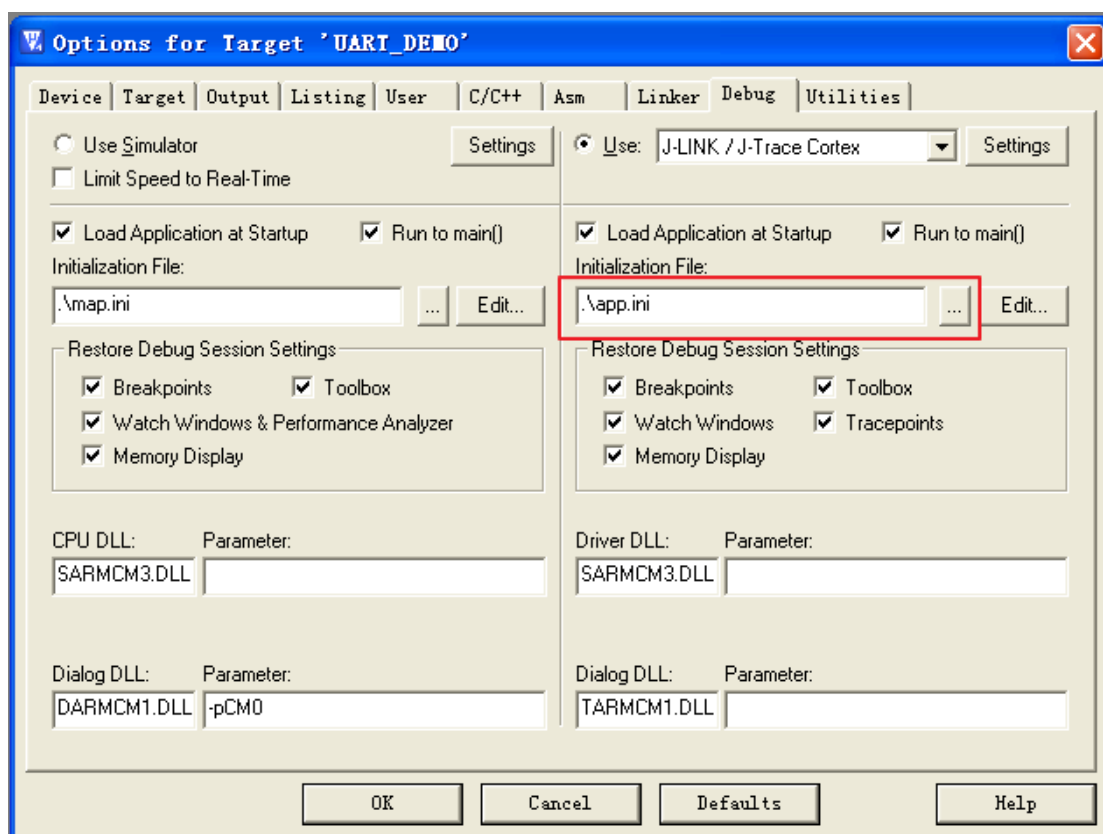
2.2.3 调试App程序

对 App 程序进行单独调试时，需要编写一个 xxx.ini 文件。此文件的目的是用 JLINK 调试时，在程序区没有 Boot 程序的情况下，直接引导和调试用户程序的运行。其设置如下：

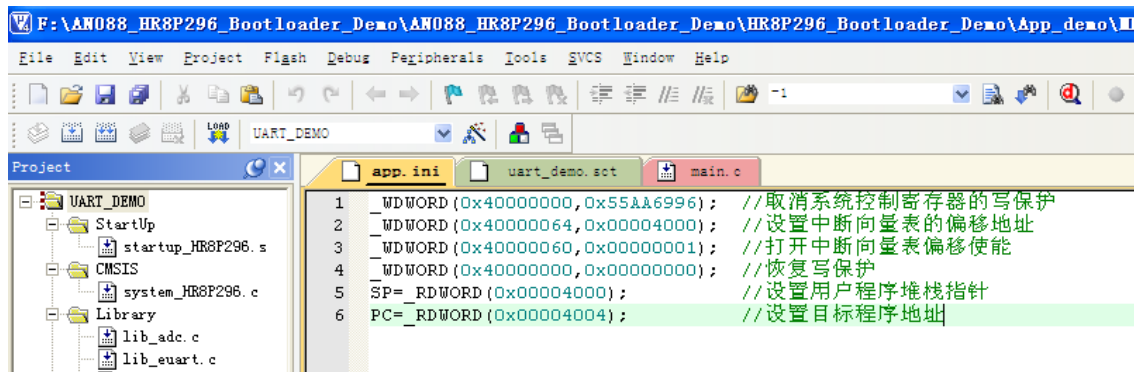
点击魔法棒，进入设置界面。



点击 Debug 设置，在 Initialization File 添加自己编写的 xxx.ini 文件。



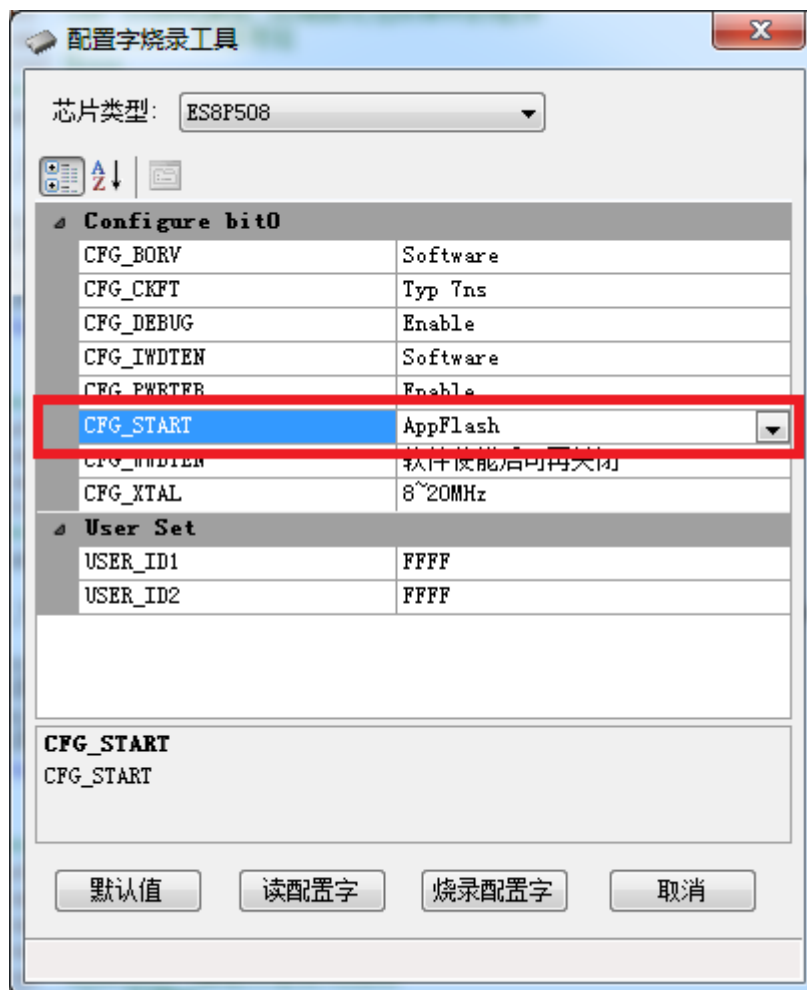
按照下图中对 ini 文件内容进行编辑即可。



添加正确的 ini 文件后，不需要下载 Boot 程序，用户程序可以在 FLASH 里任何 256 Bytes 对齐的起始位置启动并调试。

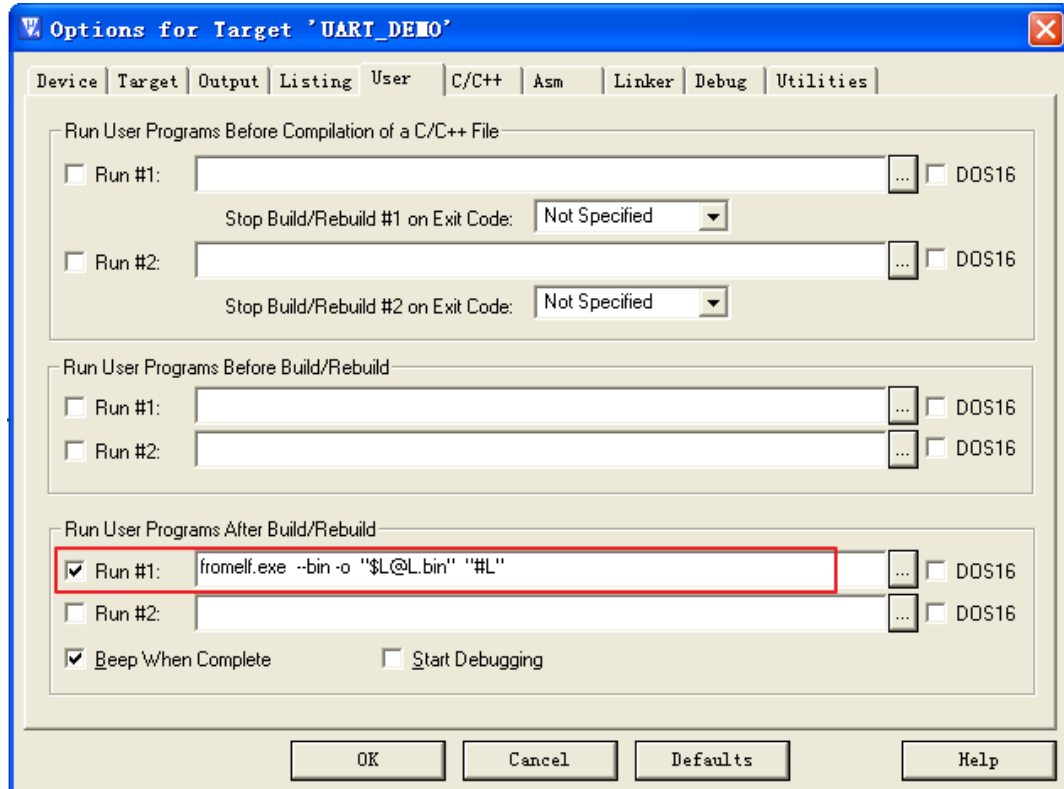
编译无误后，按 Debug 图标即可进入正常的程序调试。

对于内部可以通过配置字来选择启动位置的芯片来说，当需要调试 App 程序时，可以直接设置相关的配置字。例如 ES8P508，设置 CFG_START 位为 AppFlash，程序便会从 APPFlash 启动，中断向量也被映射到 APP FLASH 的起始处。



2.2.4 编译生成App.bin文件

本 Demo 采用超级终端软件下载 bin 文件，用户生成 bin 文件的方法如下：



2.3 IAP操作

对于具有 IAP ROMtable 的芯片，直接调用 IAP ROMtable 里的 IAP 程序即可。对于没有 IAP ROMtable 的芯片，IAP 程序必须在 RAM 中运行，按照 2.2.1 里介绍的方法配置分散加载文件，将 IAP 程序分配到 RAM 中运行：

```
RW_IRAM1 0x20000000 0x00003000 { ; RW data
    .ANY (+RW +ZI)      ; 除了指定的 RW 段，ZI 段，其他所有 RW 段和 ZI 段在这个区域执行
    lib_flashiap.o(+RO) ; lib_flashiap.o 文件的 RO 段在这区域执行
}
}
```

2.4 RAM区域分配

在此 demo 里，复位后 RAM 空间的数据不需要保存，所以 Bootloader 程序和 App 程序使用的 RAM 空间是完全相同的。如果 App 程序需要有一段 RAM 空间复位不丢失，那么 Bootloader 程序就不可占用这段 RAM 空间。

第3章 Bootloader 演示

3.1 概述

上一章节详细说明了 Bootloader 的两个工程，分别是 App 工程和 Bootloader 工程。两个工程分别编译后可生成 App.bin 和 Bootloader.hex，Bootloader.hex 可用烧录工具烧录到 MCU 运行。当芯片上电工作后先进入 Bootloader 程序，Bootloader 有如下 3 个功能：

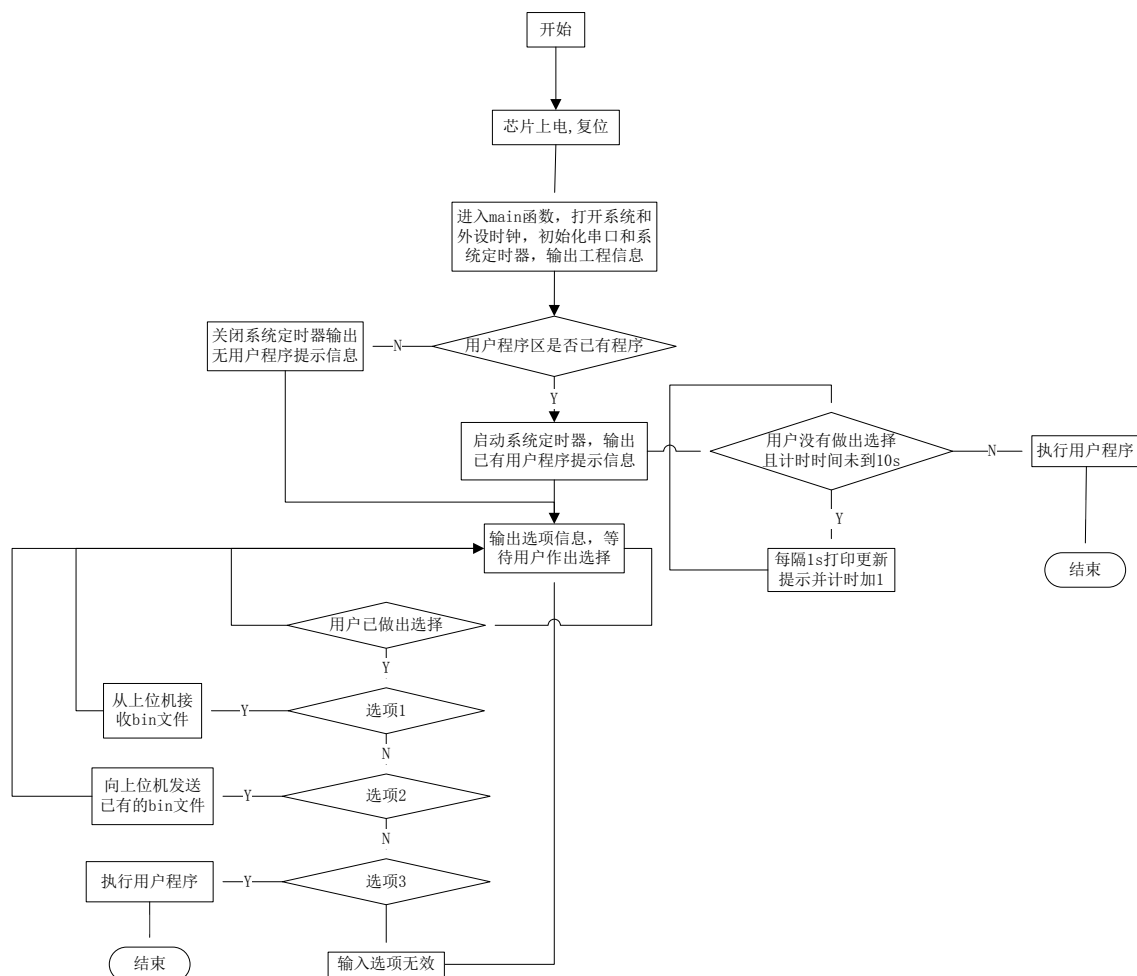
1. 下载 bin 文件到用户程序区；
2. 上传 bin 文件到上位机；
3. 执行用户程序。

如果 10 秒钟没有进行功能选择，且用户程序区有程序代码，那么 Bootloader 将自动把程序引导到用户程序区运行。

3.2 演示流程

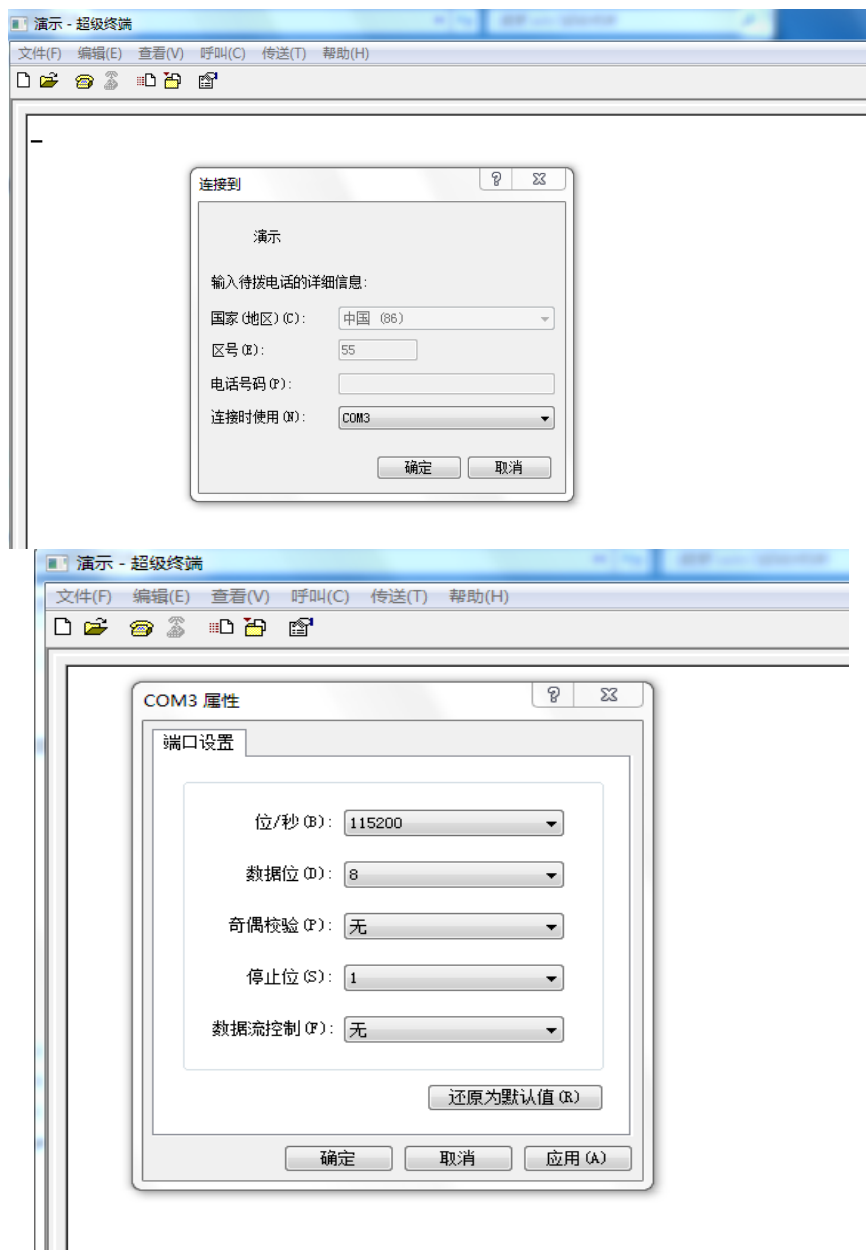
进行 Bootloader 功能试验前，需要使用烧录工具将 Bootloader 烧录到芯片 FLASH 起始区域中。

3.2.1 流程图

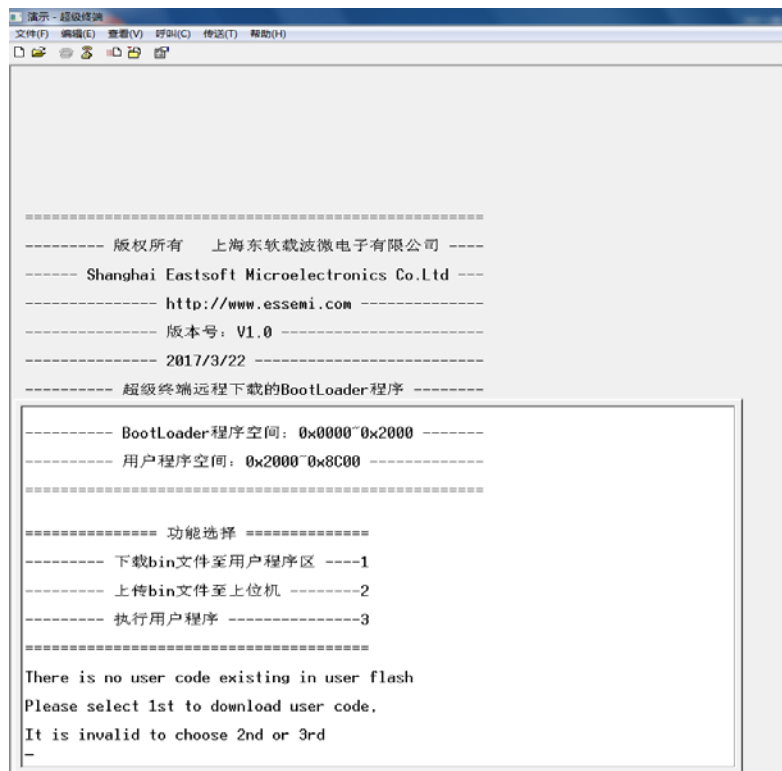


3.2.2 更新步骤

1. 打开超级终端软件，进行串口通讯配置。

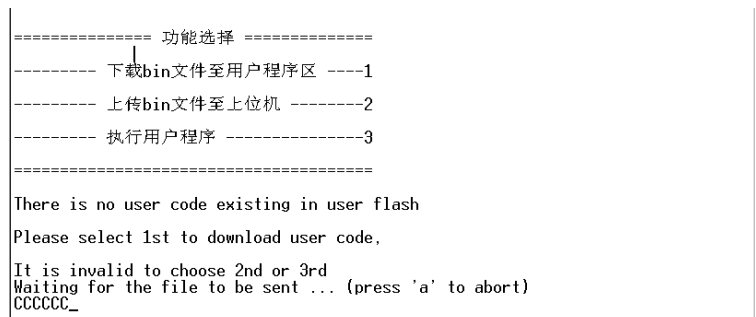


2. 复位 MCU，超级终端打印出如下界面：

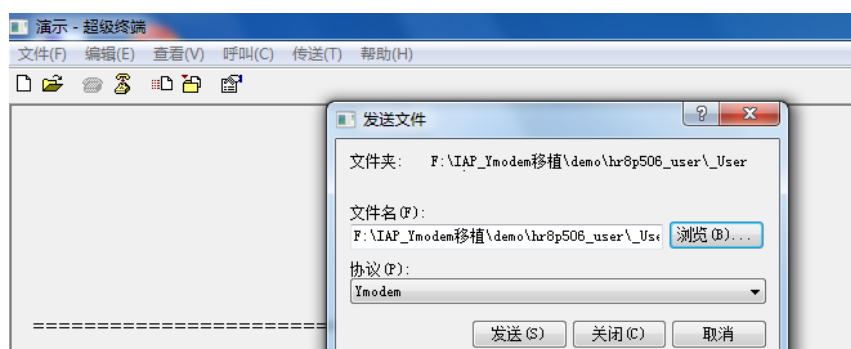


3. 下载.bin 文件至用户程序区。

按一下键盘上的数字“1”，出现如下界面：



4. 点击“传送”菜单，选择以 Ymodem 方式发送文件。



5. 发送完毕后，重新进入界面选择。

```

===== 功能选择 =====
----- 下载bin文件至用户程序区 ----1
----- 上传bin文件至上位机 -----2
----- 执行用户程序 -----3
=====

There is no user code existing in user flash
Please select 1st to download user code,
It is invalid to choose 2nd or 3rd
Waiting for the file to be sent ... (press 'a' to abort)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
文件传输成功，且成功写入用户Flash程序区！

Size: 5212 Bytes
===== 功能选择 =====
----- 下载bin文件至用户程序区 ----1
----- 上传bin文件至上位机 -----2
----- 执行用户程序 -----3
=====

```

6. 执行用户程序。

按一下键盘上的数字“3”，即可执行用户程序。

```

===== 功能选择 =====
----- 下载bin文件至用户程序区 ----1
----- 上传bin文件至上位机 -----2

----- 执行用户程序 -----3
=====

start execute
ShangHai Eastsoft
ShangHai Eastsoft
ShangHai Eastsoft
ShangHai Eastsoft
ShangHai Eastsoft
ShangHai Eastsoft

```

附录1 Ymodem 通信协议

Ymodem 是由 Xmodem 发展而来,使用更大数据块以追求更高效率的一种纠错协议(Xmodem 传输单元为 128Bytes, Ymodem 为 128/1024Bytes),广泛应用于串口通信,调制解调器等领域。

1.1 用作协议命令的控制字符

- 1) SOH(0x01), 表示一帧数据大小为 128Kbytes。
- 2) STX(0x02), 表示一帧数据大小为 1Kbytes。
- 3) EOT(0x04), 表示文件数据传输结束, 用来指示连接断开。
- 4) ACK(0x06), 确认应答信号。
- 5) NAK(0x15), 不确认应答信号。
- 6) CA(0x18), 取消或中断数据传输, 接收方和发送方都可以发送。
- 7) 'C'(0x43), 用来开启或断开传输连接的标志, 只有数据接收方可以发送。

上述特殊字符中前面 6 个都是 ASCII 码表里的不可显示字符, 在通信协议里用作控制。

1.2 数据帧格式

SOH/STX	帧编号	帧编号反码	数据	数据	CRCH	CRCL
---------	-----	-------	----	----	-------	------	------

说明:

- 1) 在 Demo 工程中, Ymodem 协议里的数据帧, 所传输的数据既有程序数据(也即 bin 文件), 也有由 bin 文件的文件名和文件大小组装成的数据帧(作为协议的首帧进行发送)。
- 2) 除了省略号单元格外, 其他每个单元格都代表一个字节数据。
- 3) 每一个数据帧都必须以 SOH/STX 作为帧首字节, SOH/STX 表示的是文件实际数据大小, 调用底层通信接口传输时, 每帧数据都要附加三个首字节(每一数据帧都必须以 SOH/STX 作为帧首字节)和两个尾部校验字节, 即实际帧数据大小为 133 或 1029Bytes。
- 4) 帧编号从 0 至 255, 循环使用。
- 5) 对前面的帧编号字节的每一位取反, 即得到帧编号反码字节数据, 帧编号和帧编号反码的作用提供连续数据帧的正确传输, 防止出现漏帧, 重复传输, 帧错位等错误。
- 6) 最后两字节为 16 位的 CRC 校验, 保障一帧数据不出现错误比特, 只对数据区进行校验, 最前面的三个帧首字节不参与校验。

1.3 控制字符在 Ymodem 协议中的使用说明

- 1) 除 SOH/STX 作为数据帧的第一个字节外, 其余特殊字符都是单个字节做控制信号发送出去, 无校验, 无编号, 无确认。
- 2) ACK/NAK 均为数据接收方收到一帧数据后发给发送方。
- 3) 当收到的一帧数据的数据格式, 帧编号, 数据区的 CRC 校验都合法时, 接收方发送 ACK

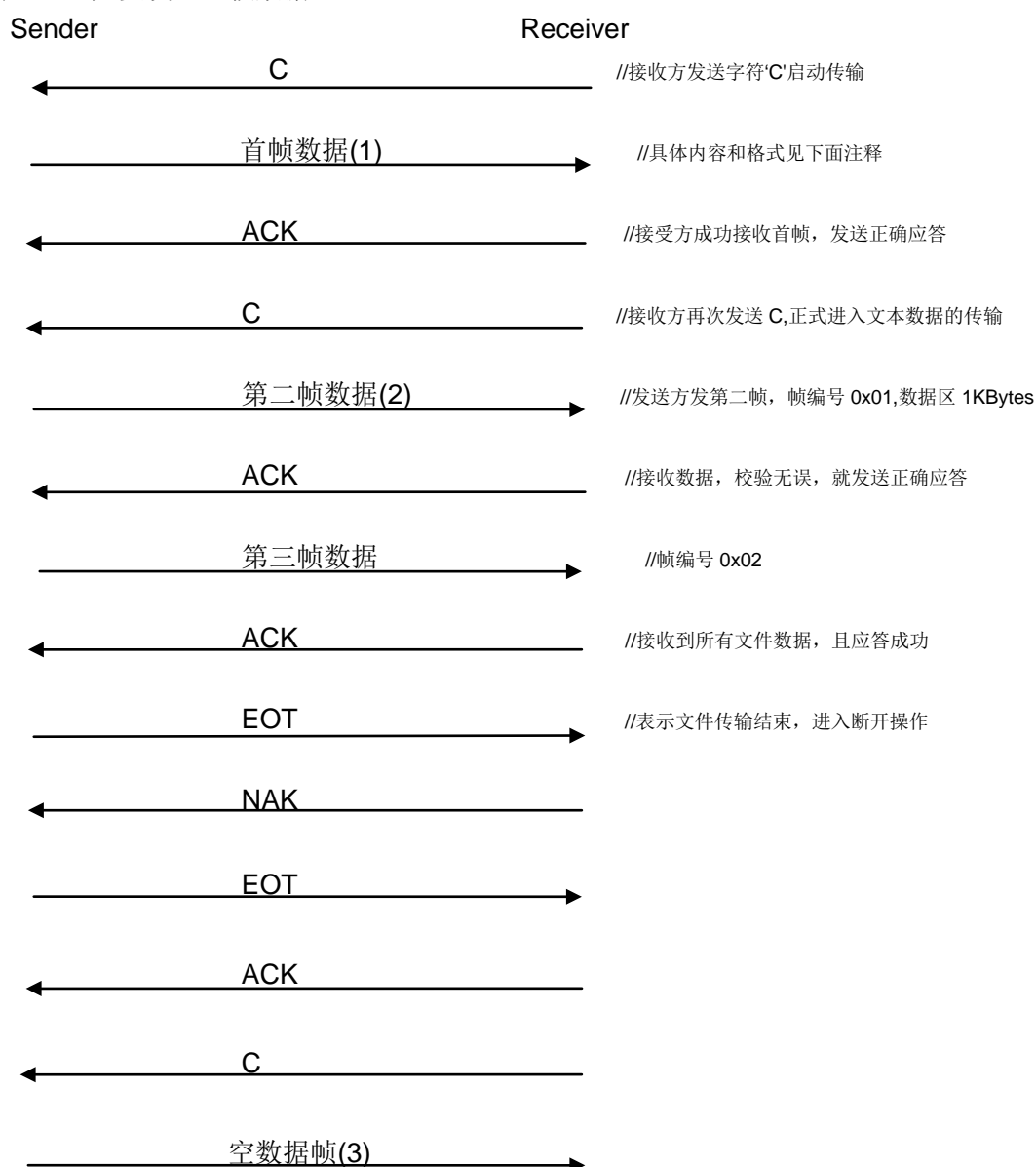
给发送方，否则发送 NAK。

- 4) 发送方发送完一帧只有在规定时间内收到 ACK 信号才会发下一帧，若一直收不到 ACK 或收到 NAK，则重发，达到最大重发次数还是收不到 ACK 则传输异常中止，Windows 自带的超级终端 Ymodem 的重发次数最大值为 10 次。
- 5) 使用 CA 取消传输时，理论上需要连发两次 CA，实际上可能多发，为确保取消传输的有效执行，windows XP 的超级终端会连发 5 次 CA。

1.4 完整无误的传输过程演示

现在假设要发送文件名为“123.txt”，大小为 1234Bytes 的文件。

按照 Ymodem 协议里的流程，文件名和文件大小也是文件数据，封装成一帧数据作为首帧进行发送，也要占用一个帧编号 0x00。一般首帧数据区大小 128Bytes。后面的数据帧都是 1024Bytes，很明显，一共要发送三帧数据。



注意事项:

- 1) 首个数据帧具体内容和格式如下所示，全部用十六进制表示。

SOH	00	FF	31 32 33 2e 74 78 74	00	31 32 33 34	00	00.....00	CRCH	CRCL
-----	----	----	----------------------	----	-------------	----	-----------	------	------

- 2) 第二帧开始，就是传输的实际数据，如果传输的是 Hex 文件，那么，数据是按照 ASCII 码形式传送的，如果是 bin 文件，就是传输的二进制数据。
- 3) 最后断开连接时，发送方需要发送一个空数据帧，也就是说数据区的所有数据都是 0，没有实际意义。

1.5 Ymodmem 协议特征总结

- 1) 点对点通信

上位机的超级终端，包括 SecureCRT 和串口调试助手等通信软件都只能和一个下位机单独传输，无法使用超级终端同时控制两个单片机。

- 2) 封装成帧，具有差错检测功能

既尽量保证了一个数据帧所有比特位不出错，也尽量保证所有数据帧在整个传输过程中的正确顺序。

- 3) 停止等待协议，面向可靠的连接传输，无缓冲机制

发送完一帧，发送方必须在规定时间内收到正确的应答信号，否则不会继续发送下一帧，超时未收到或者收到非正确应答，发送方将重发。