KF32Bootloader 应用笔记

简介

本文介绍 KF32 系列的一种通用的 Bootloader 实现方法，又称 IAP（in-application programming），通俗的说法，Bootloader 的过程类似于 Windows 中软件自动升级功能。Bootloader 通过任意通信口都可以远程升级产品固件（程序），解决了 MCU 烧写程序需要拆解设备或者需要专业人员、专用工具、以及现场操作的烦恼。使用 Boot loader 刷新产品程序，只要用户愿意，轻轻按下按钮再等待一段时间就可以享受新的固件程序带来的便利。

概述

本文介绍的内容包括：使用 ChipON IDE KF32（以下简称 IDE）—— V1.0.10.2 以上版本，编译客户应用程序（以下简称 APP），获取 APP 对应的 HEX 文件（以下简称 APP.HEX）；使用 ChipON PRO KF32（以下简称 PRO）—— V1.0.10.2 以上版本，将 APP.HEX 转换为对应的 BIN 文件（以下简称 APP.BIN）；SecureCRT 作为 Bootloader 的上位机或者服务器，使用 Ymodem 协议传输 APP.BIN 文件到目标板；目标板里的 Bootloader 程序解析数据包，并将其内容组合起来，按照顺序写入到目标 Flash 空间内；Bootloader 程序引导目标 Flash 区的 APP 程序启动成功后，自动退出运行；APP 程序正常运行，Boot loader 升级固件成功。

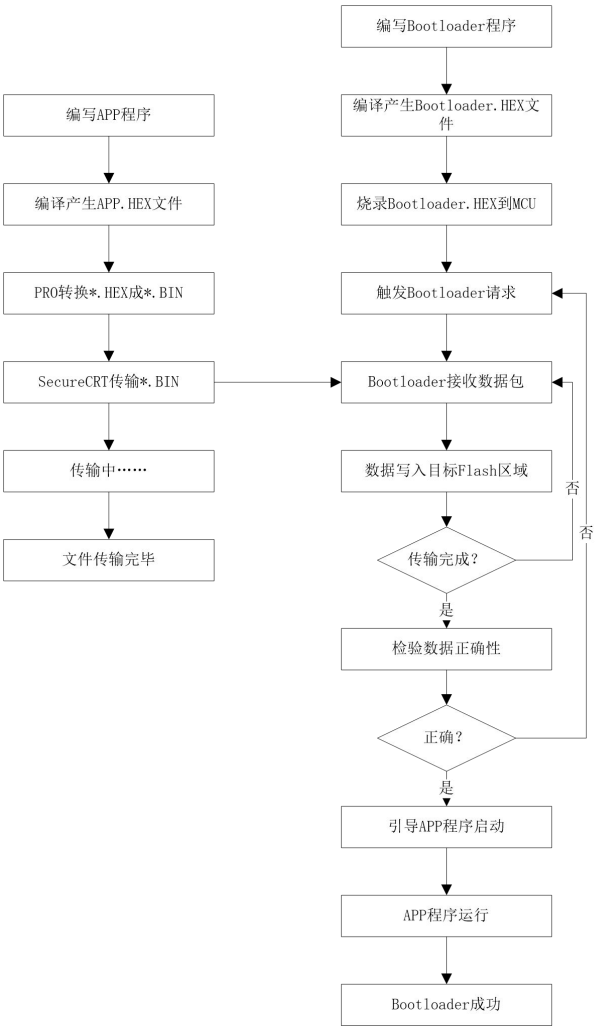

图 1. Bootloader 实现过程

一、背景知识

在正式开始编写 Bootloader 程序之前我们还需要了解一些相关的背景知识。如果已经掌握这些知识点的读者可以直接跳过这部分。如果读者觉得某些知识点讲解不够透彻或者有错误可以查阅网上相关信息。如果读者觉得某些知识点一时难以理解，可以先跳过该部分通读全篇后，结合实际操作将会更好理解一些概念。

1、Bin 文件、Hex 文件的区别

Bin 文件和 Hex 文件是 MCU 常用的两种烧写文件格式。Bin 文件是一种纯二进制文件，没有任何格式。它是一种 Flash 映像，但是它不包含任何地址信息，烧写时必须人为指定地址信息。Hex 文件是记录文本行的 ASCII 文本文件，每一行是一个 Hex 记录，由十六进制数组成的机器码或者数据常量。Hex 文件包含地址信息，所以烧写 Hex 文件时并不需要用户指定地址信息。

下面是同一个程序的 Bin 文件和 Hex 文件前 4 行的内容，我们可以通过对比来直观的感受二者的区别。

```
00000000 00 00 02 10 18 89 00 00 90 8A 00 00 94 8A 00 00
00000010 00 00 00 00 98 8A 00 00 00 00 00 00 2C D7 FB EF
00000020 00 00 00 00 00 00 00 00 00 00 00 00 9C 8A 00 00
00000030 00 00 00 00 00 00 00 00 A0 8A 00 00 A4 8A 00 00
```

图 2.Bin 文件截图

```
:10800000000002101889000090 8A0000948A000085
:1080100000000000988A0000000002CD7FBEF51
:10802000000000000000000000000009C8A00002A
:1080300000000000000000A08A0000A48A0000E8
```

图 3.Hex 文件截图

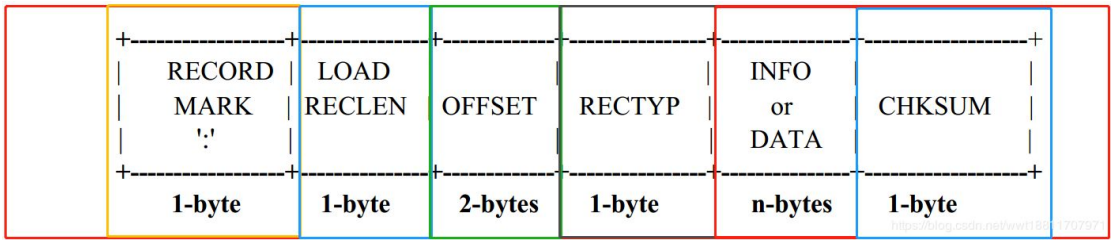

图 4.Hex 文件的基本格式

'00'	Data Record
'0l'	End of File Record
'02'	Extended Segment Address Record
'03'	Start Segment Address Record
'04'	Extended Linear Address Record
'05'	Start Linear Address Record

图 5.记录的几种类型

Hex 记录格式，冒号‘:’表示记录的开始。第一个字节是记录的数据长度，接着的 2 个字节是数据的地址，接下来的一个字节是记录类型，接着是 N 个数据字节，最后一个字节是数据字节的 Checksum 计算值。

以第一行为例，记录长度：0x10，记录地址：0x8000，数据类型：0x00，数据内容：0x00 0x00 0x02 0x10 0x18 0x89 0x00 0x00 0x90 0x8A 0x00 0x00 0x94 0x8A 0x00 0x00， Checksum：

0x85。对比发现，hex 和 Bin 文件的数据内容是一样的，但是 hex 文件多了地址信息和 Checksum 字节。但是，正是由于 Bin 文件的纯粹，所以用 bin 文件做 Bootloader 跟适合一些。如果用 Hex 文件做 Bootloader，则不可避免的需要解析 Hex 文件，这样只会徒增烦恼。

2、Ymodem 文件传输协议

Ymodem 协议是一种常见的通信协议。在本实验中，上位机向目标板下载 APP.bin 文件使用的正是 Ymodem 协议。

Ymodem 协议的数据帧内容通常包括 4 个部分，“帧标识符”+“帧序号_帧序号反码”+“128Byte 数据/1K byte 数据”+“2 字节 CRC16 校验码”。例如首个数据包内容：

SOH 00 FF   filename NUL   CRCH CRCL

其中 SOH 是 0x01，表示这个数据包包含 128Byte 数据。00 表示数据帧序号以后的数据包依次增长，FF 是 00 的反码。首个数据包一般会包含文件名称及其后缀，余下的数据用 0 补齐 128Byte。最后 CRCH\CRCL 分别代表 16 位 CRC 校验码的高 8 位和低 8 位。一般网上描述的 ymodem 协议，首个数据包包括 filename +filezise+NUL，实际上 SecureCRT 使用的 yemodem 协议版本可能不一样，SecureCRT 首个数据包只包含 filename 。另外一个区别之处在于，SecureCRT 使用的 Ymodem 协议数据包长度全部是 128Byte 类型的。例如，一般资料描述的 Ymodem 的文件内容帧的结构是：STX 01 FE data[1024] CRCH CRCL。实际上，SecureCRT 的文件内容帧是：SOH 01 FE data[128] CRC CRC。

帧标识符和应答字符的内容如下：

```
#define YMODEM_SOH (0x01)
#define YMODEM_STX (0x02)
#define YMODEM_EOT (0x04)
#define YMODEM_ACK (0x06)
#define YMODEM_NAK (0x15)
#define YMODEM_C (0x43)
```

SecureCRT 使用的 Ymodem 协议的通信时序如下：

```
发送端                                                    接收端
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<     C

SOH 00 FF Filename NUL……CRC CRC >>> //128 数据包长度

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<     ACK

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<     C

SOH 01 FE data[128] CRC CRC>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<    ACK

SOH 02 FD data[128] CRC CRC>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<    ACK

SOH 03 FC data[128] CRC CRC>>>>>>>>>>>>>>>>>>>>>>
```

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   ACK

SOH 04 FB data[128] CRC CRC>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   ACK

SOH 05 FA data[100]   1A[28] CRC CRC>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   ACK

EOT >>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   NAK

EOT>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   ACK

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   C

SOH 00 FF NUL[128] CRCCRC >>>>>>>>>>>>>>>>>>>>>>>>>

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<   ACK

Bootloader 接收代码是按照这个时序接收和解析 Yemodem 时序的。

3、KF32 的存储空间映射

KF32 的存储空间的设计是按照线性地址排列的，这样做的好处是 RAM、ROM、Flash、寄存器的寻址更为方便和直观。例如，KF32F350MQV 的 Flash 空间地址是从 0x0000 0000——0x0008 0000，RAM 的地址范围是 0x1000 0000——0x1002 0000。具体情况如下图示。

图 6.KF32F350MQV 部分存储空间映射

4、KF32 的代码存储分布

KF32 的 Bootloader 放在从 0x0000 0000 开始的 Flash 的低地址空间内，MCU 上电后会自动从 Bootloader 启动。APP 程序放在 Flash 后面的一个区域内，本例程从 0x0000 8000 地址开始。实际上，这个起始地址也可以改成其他以 1K Byte 为单位的起始地址，只要不和 Bootloader 区重合，剩余的 Flash 空间也足够存放 APP 程序即可。具体 Flash 存储映射入下图所示。每个程序存储映射，以向量表（Vector table）开始，后面连续存放 Code segment（也称 text segment）、Data segment、Bss segment。向量表将在下面详细介绍。Code segment 存放的是.c 和.h 编译产生的二进制机器码。Data segment 存放的是已经初始化的全局变量，包括常量和已初始化值的变量。Bss segment 段保存未初始化值的全局变量。

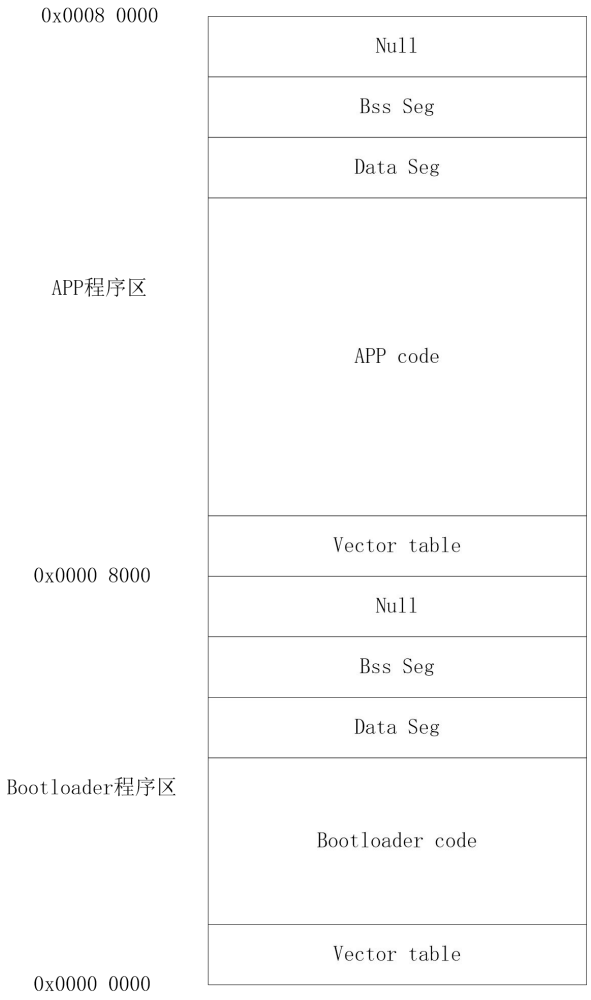

图 7.KF32 的中断向量表及其重映射

形象一点说，中断向量表以表格的形式，按照保存了所有中断服务函数的入口指针。中断向量表在程序启动时和执行中断服务函数的过程中是必不可少的，它相当于程序的“目录”。尤其特别的是 0x0000 0000——0x0000 0003 保存的是堆栈空间 MSP——栈顶指针的值，在 KF32 中是 Ram 空间的最高地址 0x1002 0000。另外，0x0000 0004——0x0000 0007 保存的是 startup()函数的指针，这个函数是 MCU 启动的入口，它负责初始化 RAM，跳转到 main()函数。

截取部分部分 Vector.c 文件内容，如下：

```
asm(
"  .global    _start"                                    "\n"
```

```
"  .text"                                          "\n"
"_start:"                                          "\n"
"  .long      __initial_sp"                    "\n" //栈顶指针
"  .long      startup"                         "\n" //启动函数
"  .long      _NMI_exception"                  "\n"
"  .long      _HardFault_exception"       "\n"
"  .long      _Soft4_exception"           "\n"
"  .long      _StackFault_exception"      "\n"
"  .long      _AriFault_exception"        "\n"
"  .long      0"                               "\n"
"  .long      _Soft8_exception"           "\n"
"  .long      _Soft9_exception"           "\n"
"  .long      _Soft10_exception"          "\n"
"  .long      _SVC_exception"             "\n"
……
```

当 Bootloader 引导 APP 程序启动的时候，需要使用 APP 程序的中断向量表。APP 启动后需要将 Bootloader 的向量表切换为 APP 程序自己的向量表，这个过程称为向量表重映射。常规程序中，中断产生后，在做好中断现场压栈工作后，硬件自动寻址对应的中断服务函数入口，并跳转到中断函数执行。硬件寻址中断服务函数的方法是：中断入口地址 = SYS_VECTOFF + 中断向量编号*4， SYS_VECTOFF 复位值是 0x0000 0000。中断向量重映射比较简单，只需要将 SYS_VECTOFF 赋值 APP 程序的偏移地址就可以了。

5、KF32 的上电启动过程

KF32 系列 MCU 上电后，硬件会读出 SYS_VECTOFF 地址的值作为 MSP（栈顶指针）。读取 0x0000 0004——0x0000 0007 地址内容，并以此作为 startup()指针，跳转执行 startup()。startup()包括三个内容：Data 段数据复制到对应的 RAM 空间中；Bss 段数据对应的 RAM 空间初始化为 0；跳转到 main()函数。startup()源码如下：

```
void startup()
{
    unsigned char length;
    unsigned int *s,*begin,*end;
    s = (unsigned int*)&__text_end__;
    begin = (unsigned int*)&__data_start__;
    end = (unsigned int*)&__bss_start__;
    while(begin < end)
        *begin++ = *s++; //初始化data段对应的ram
    begin = (unsigned int*)&__bss_start__;
    end = (unsigned int*)&__bss_end__;
    while(begin < end)
        *begin++ = 0; //初始化bss段对应的ram
    main();
}
```

在 main()执行的过程中，如果产生中断，MCU 硬件机制会自动寻址中断服务函数入口——中断入口地址 = SYS_VECTOFF + 中断向量编号*4，在获取中断服务函数地址后将跳转过去执行，中断服务函数执行完毕后，将再返回 main()。

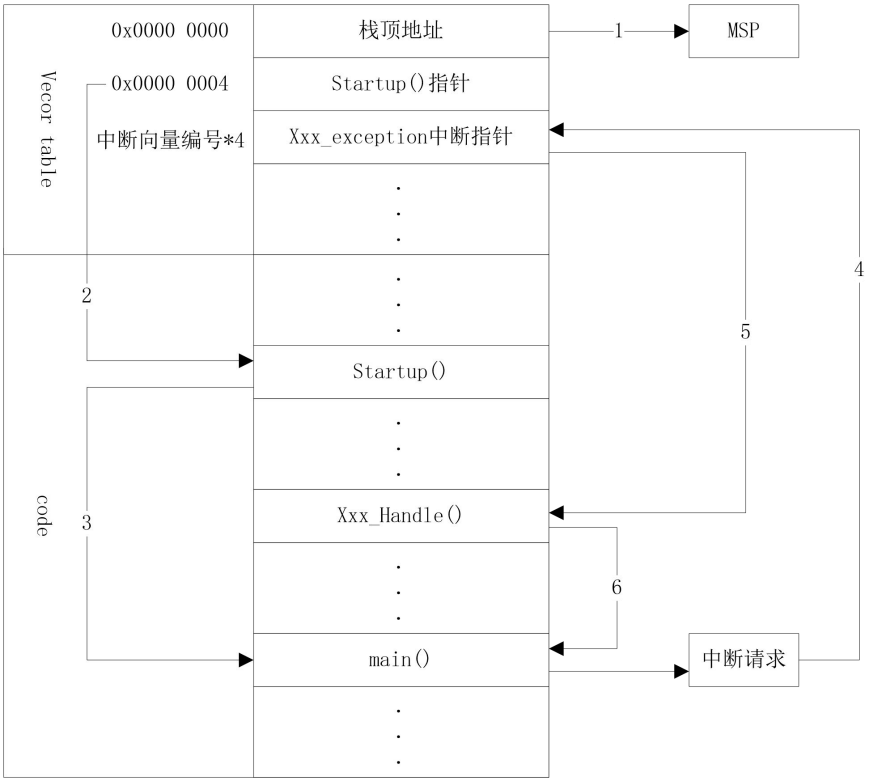

图 8.KF32 常规运行流程图

Bootloader 后程序运行的流程如下图所示。Bootloader 成功后 MCU 运行的流程稍微有些变化，主要表现为 2 个方面：从 Bootloader 的 main()跳转到 APP 程序的 startup()；向量表重映射到 APP 中，其实就是生效 APP 程序中的向量表。APP 运行起来后，RAM 被重新初始化，但是由于 MCU 没有复位，外设仍然保持着 Bootloader 初始化后的状态，这点在应用过程中需要注意。中断向量表重映射的操作其实就是把 SYS_VECTOFF 赋值为 APP 的 Flash 偏移量。这样操作完成后，每次发生中断硬件将自动以 SYS_VECTOFF 为基地址寻找中断向量，APP 的 Vector table 代替 Bootloader 的 Vector table 作用。当然，也可以在进入 APP 程序后，以同样的方法返回执行 Bootloader 程序。

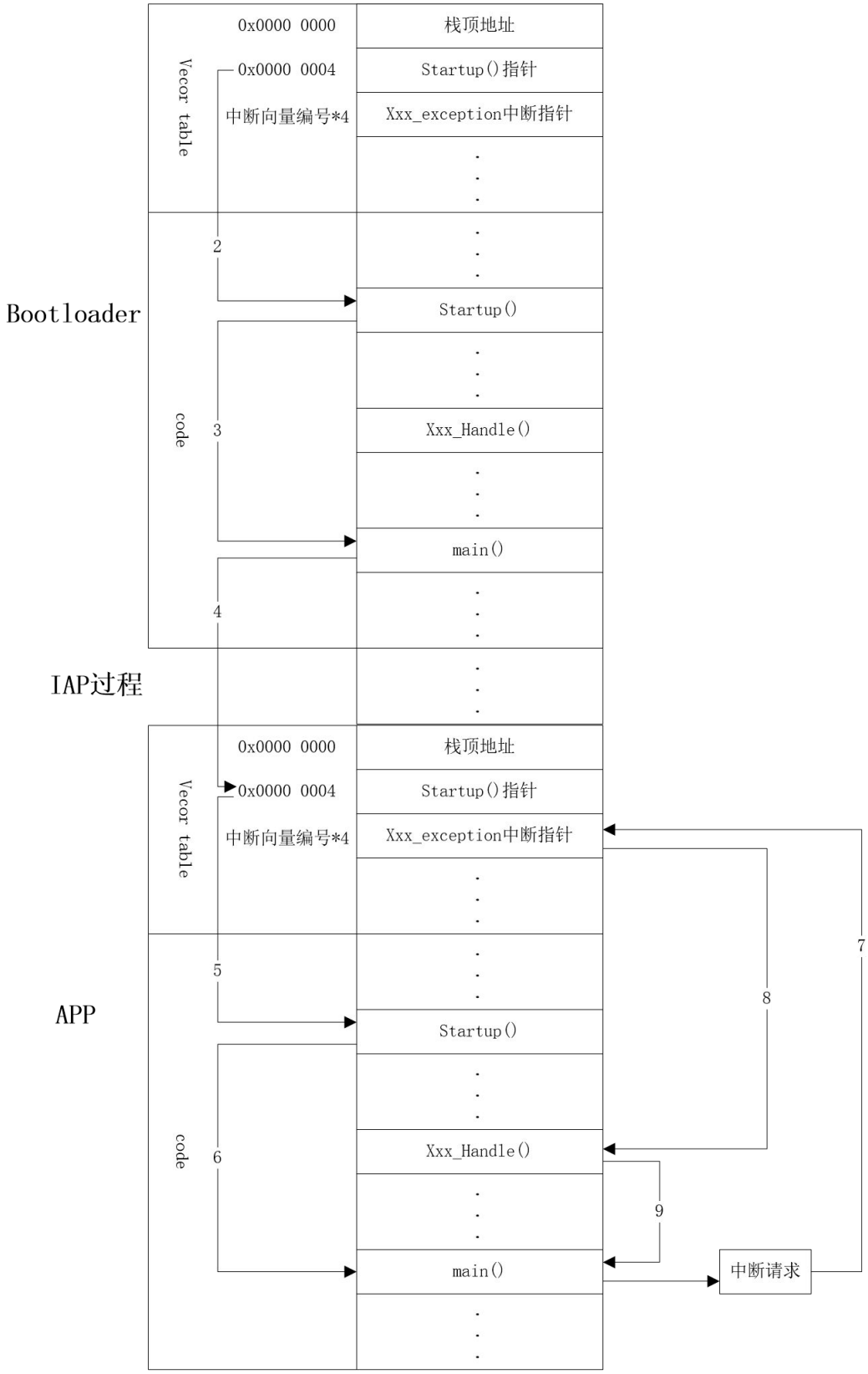

图 9.Bootloader 成功后的程序运行流程

6、KF32 的 Flash 自写以及信息区

以 KF32F350MQV 为例，Flash 包括 512K Byte 的程序区和 8K 的信息区，其中信息区只有 3k Byte 可供用户使用。信息区只能用于保存数据，不可用于保存程序。Flash 一页的大小是

1K Byte，Flash 写缓存是 64Bit 大小。Flash 自写需要经过 3 个步骤：解锁、擦除、编程。复位以后，Flash 程序区处于锁定状态，不能直接擦、写。这一点对 Flash 内容保护非常重要。擦除操作分为 2 种：片擦除就是直接把整个程序区都擦除，程序运行在 ROM 或 RAM 中才可以执行片擦除命令；叶擦除把 FLASH_ISPADDR[19：10]地址指向的 PAGE 擦掉，执行页擦除功能时，硬件将自动暂停直到擦除结束。编程有 2 种基本操作：单字编程和半页编程。每次编程之前，必须保证当前操作的区域必须已经擦除完毕。而且擦除后只能编程一次。

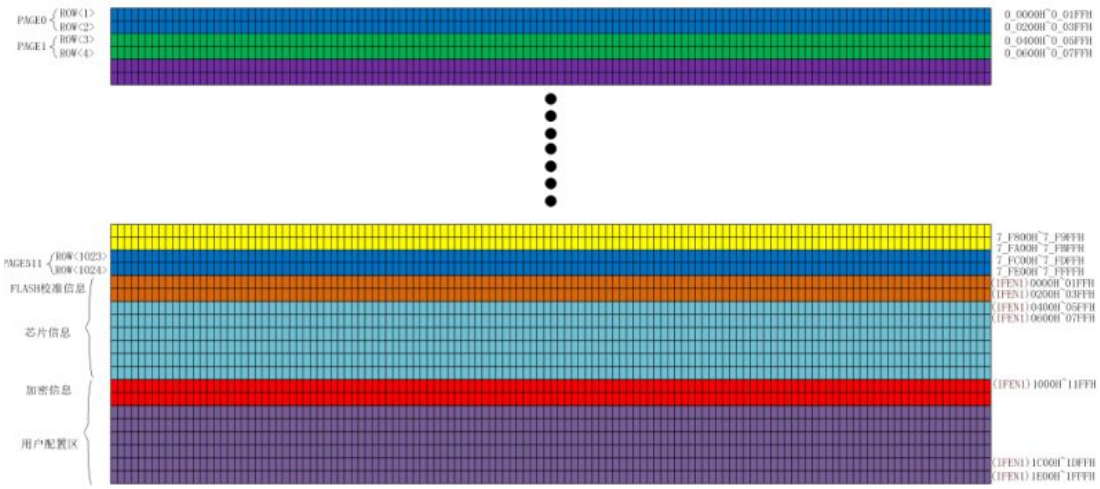

图 10.Flash 地址映射

二、APP 程序准备

APP 程序可以是任意程序，我们为了测试 Bootloader 的功能，在 APP 程序中加入了 LED 闪烁功能、中断、已经赋初值的全局变量。有一点需要注意，Bootloader 程序跳转到 APP 程序执行，由于 RAM 会重新初始化，所以 Bootloader 的变量信息将不复存在。但是，由于本质上只是程序流的跳转而非芯片复位，MCU 外设的配置仍然保持，APP 程序可以沿用 Bootloader 的外设配置或者重新配置外设。

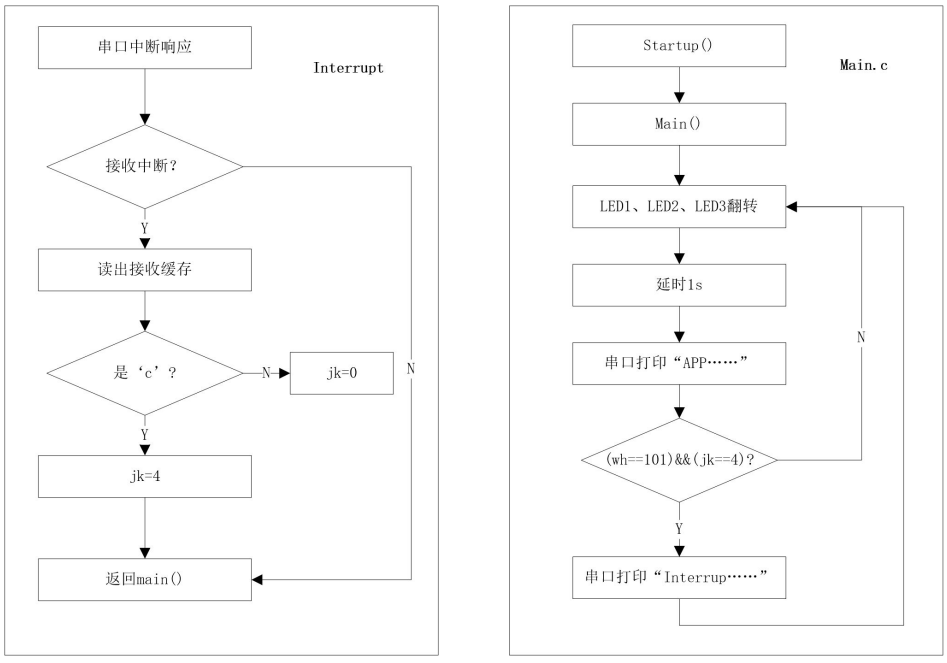

图 11.APP 程序流程图

APP 程序编译的时候需要注意一个问题：APP 需要按照目标地址编译，这是因为程序跳转指令是按照绝对地址进行指令寻址的。这样编译产生的目标文件，是不能像常规程序那样上电后自己就能运行起来，它必须被写入到指定的地址，在 Bootloader 程序的引导下才能顺利启动。默认情况下，编译工具链以 0x0000 0000 为 Flash 起始地址编译代码。本例程中，需要修改 Flash 起始地址为 0x0000 8000，这是我们指定的 APP 程序区的起始地址。KF32 中修改程序编译地址的方法：

1、 如果是默认路径安装的 IDE 软件，在目录 C:\Program Files (x86)\ChipON IDE\KungFu32\ChiponCC32\scripting\ccr1_issue 找到对应芯片型号的 Xxx.ld 链接文件，复制到项目中，本例程使用的是 KF32350MQV 所以复制 KF32F350MQV.ld。

2、 修改 flash 的起始地址和大小。复制到项目中后，双击打开 Xxx.ld 文件，修改 MEMORY 中的 Flash 值。本例程设置 APP 程序的偏移地址是 0x0000 8000，所以响应的 Flash 大小也减去偏移地址。

```
7 MEMORY
8 {
9   flash      : ORIGIN =  0x00008000, LENGTH = 0x00078000
```

图 12.Flash 偏移地址设置

3、 生效当前链接设置。具体操作：选中项目->右键->属性->C/C++构建->设置->通用设定->芯片脚本文件写入 -T"../KF32F350MQV.ld" ->应用->确定。

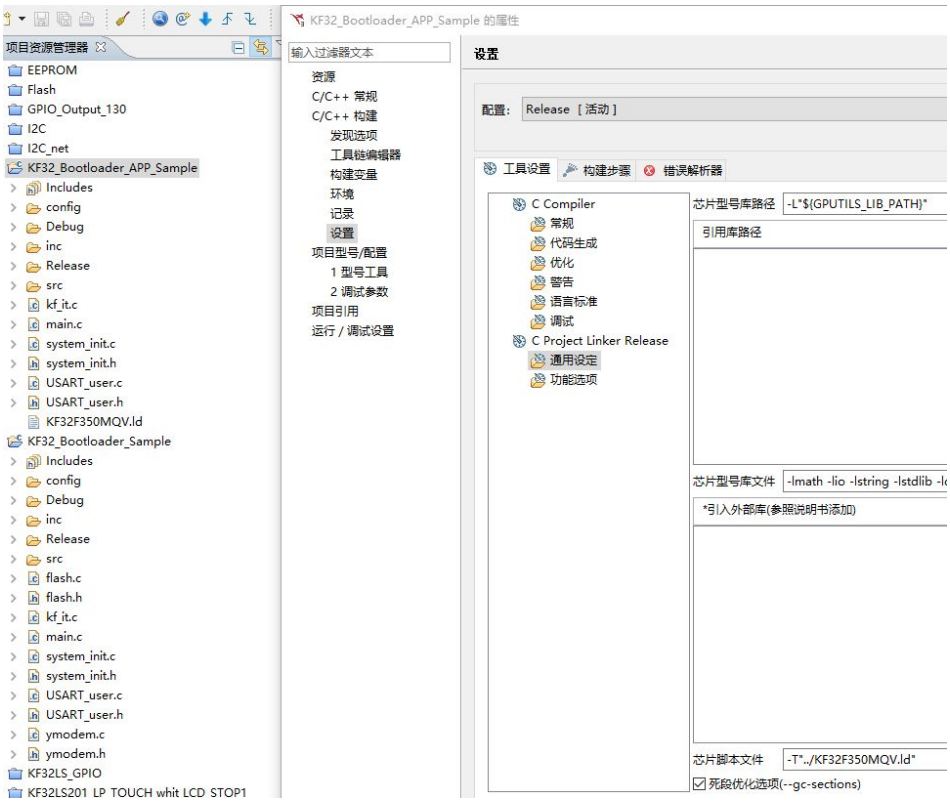

图 13.生效 Flash 偏移地址配置

4、 编译程序。

5、 确认编译结果。打开项目中 Release 文件夹->打开 xxx.map->搜索 startup，可以看到 startup 编译地址已经被分配到 0x0000 8918 地址，说明偏移地址编译已经生效。

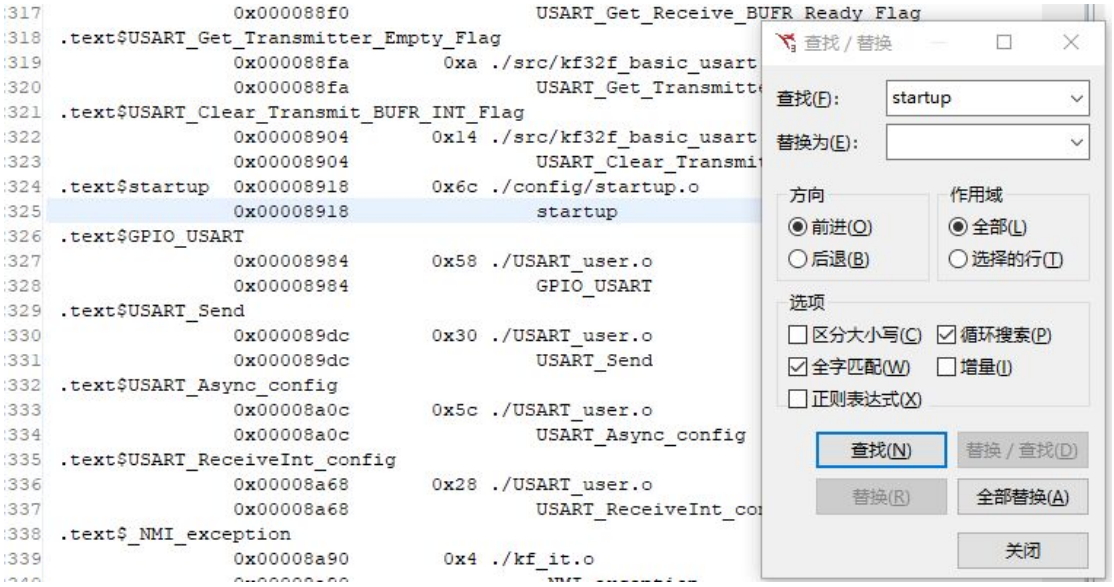

图 14.查看 xxx.map 文件

三、获取.bin 文件

IDE 编译产生 xxx.hex 文件后，我们需要使用 ChipON PRO KF32 软件将它转换成 xxx.bin 文件，以用于最终的 Bootloader。

使用 PRO 转换 bin 文件的步骤：

1、 加载 hex 文件。打开 PRO 软件，文件->加载文件->选择项目文件夹中->Release 文件夹->选择 xxx.hex->确定。

2、 转换 hex 文件。工具->代码转存 bin 文件->输入文件名 xxx ->确定。可以发现产生 2 个 bin 文件：xxx_Flash.bin 和 xxx_Data.bin。xxx_Flash.bin 是我们需要的 Bootloader 目标文件。

四、Bootloader 程序

Bootloader 的主要功能分为三个部分：1、通过通信口接收 APP 程序；2、将 APP 程序文件以 Flash 自写的方式写入目标 Flash 区域；3、引导 APP 程序启动。Bootloader 程序的流程图如下图所示。

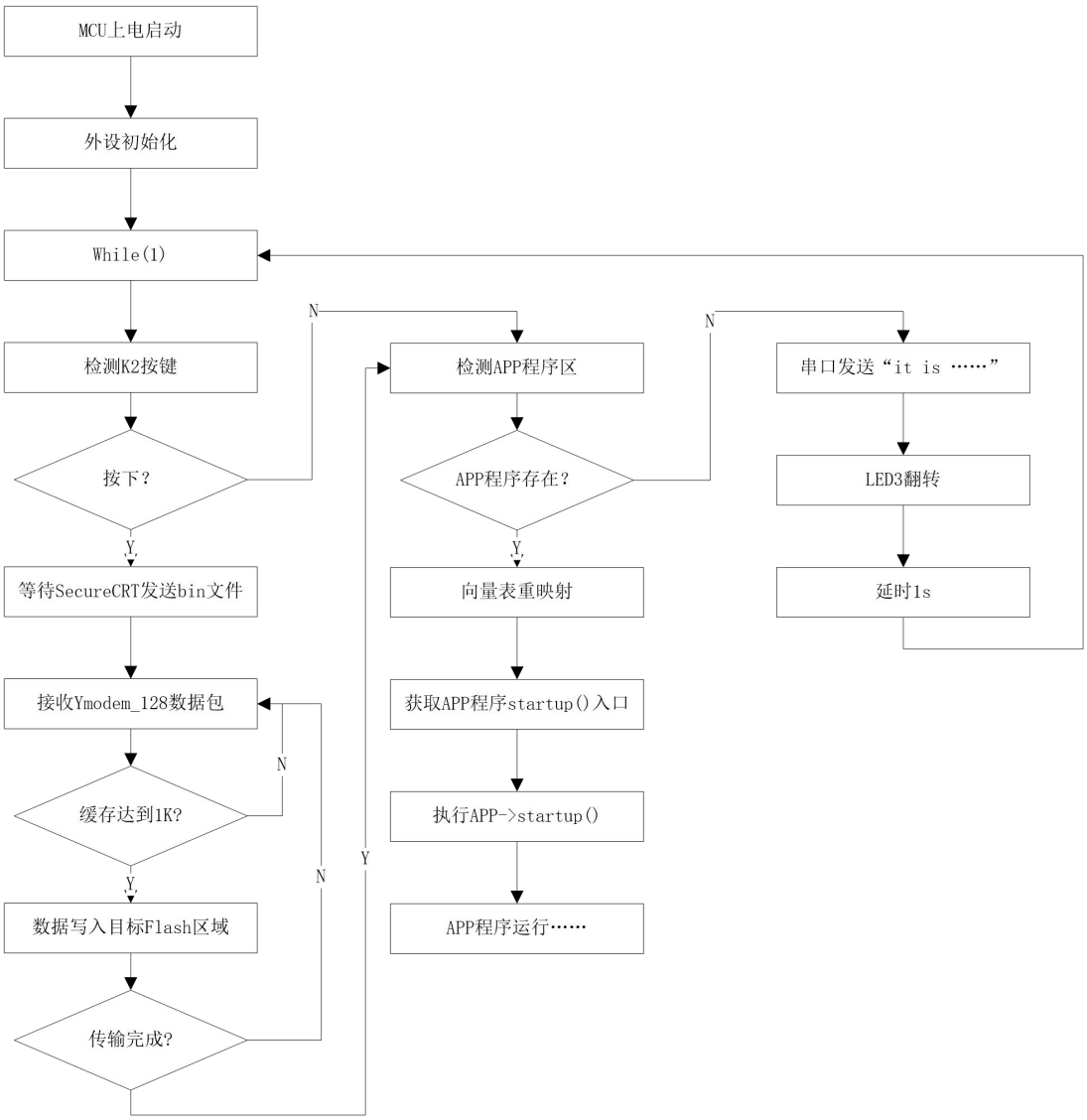

图 15.Bootloader 例程流程图

实际上，APP.bin 文件最大可能达到 500K+，如果先把 APP.bin 完整接收到 RAM 暂存，再写入 Flash，这种方法是不现实的，因为 MCU 的 RAM 空间总是小于 Flash。Bootloader 程序总是一边接收 APP.bin 数据，一边把数据写入目标 Flash 区域的，二者总是交叉进行的。下面详细介绍一下如何按照 ymodem_128 协议接收 APP.bin 以及写入 Flash。

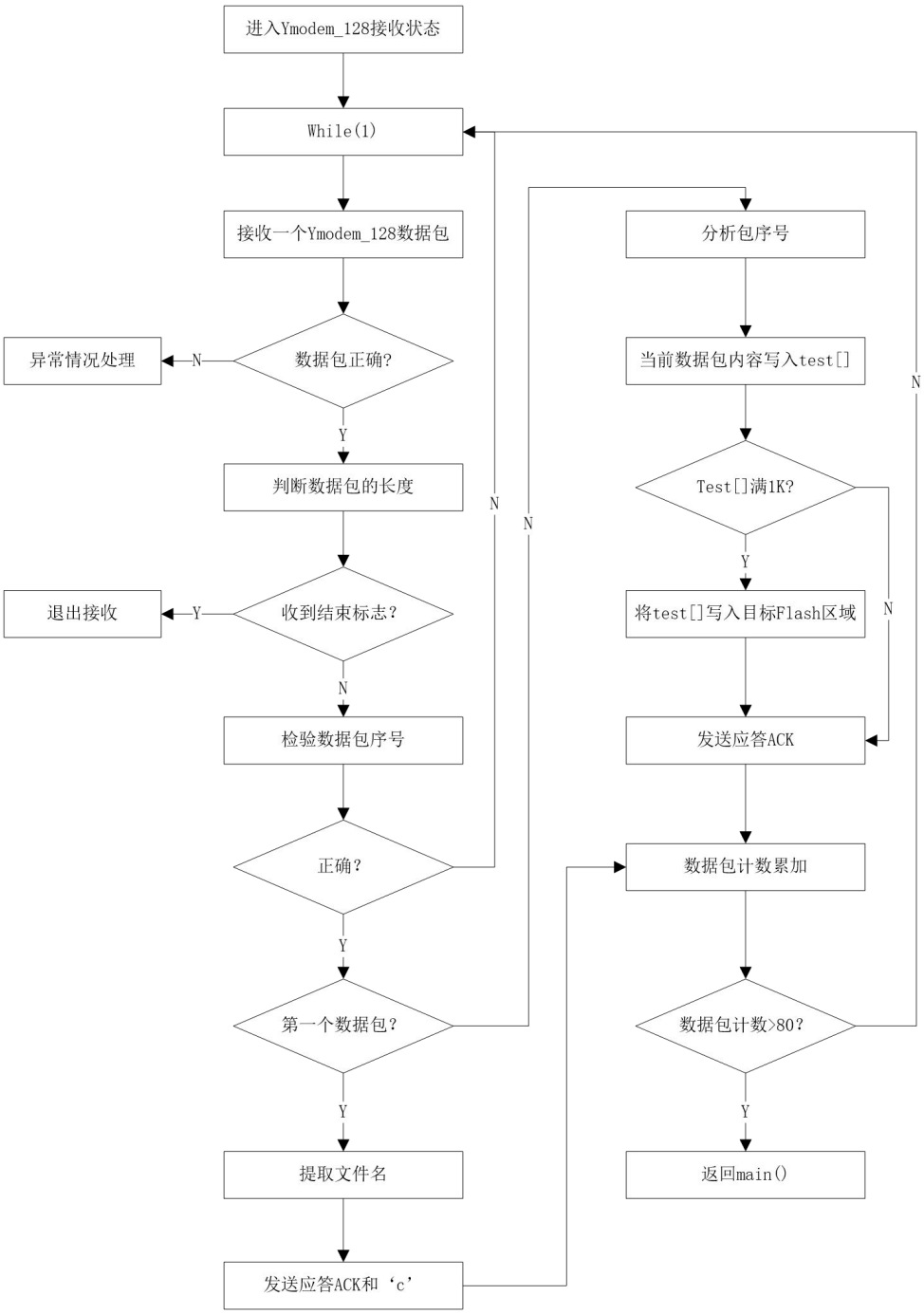

图 16.Ymodem 协议接收和 Flash 写入

关于 Ymodem 协议接收更多细节请参考源代码。需要注意的是，接收超时的阈值设置非常关键，因为 SecureCRT 运行在 Windows 系统上，而 Windows 并非实时操作系统，因此 SecureCRT 通常需要几十个 ms 才能响应 MCU 返回的应答信息。接收超时具体应当设置多长，应当视实际情况而定，不同的 PC 机响应时间也有可能不同，这一点可以通过逻辑分析仪捕捉到。例程中设置超时时间为 30ms 左右，实测可行。

另外，需要注意 PRO 软件转换得到 bin 文件是按照 1k Byte 为单位进行的对齐，也就是说文件结尾处不满 1KByte 的单位也会被填充无效数据。这样处理的好处就是，Flash 是以 1k Byte 为页的，这样方便 Bootloader 接收数据以及填写 Flash。

更多关于通信时序中错误的处理细节这里不再一一赘述，请参考代码。

五、实验验证

本实验用到的器材有 KF32F350 开发板、KF32 编程器、逻辑分析仪，涉及的软件有 KF32_IDE、KF32_PRO、SecureCRT、逻辑分析仪上位机。

编译、下载 Bootloader 程序到开发板。连接开发板的 USB1 接口到 PC 的 USB 接口，开发板使用 USB-UART 芯片是 CH340E，需要安装对应的驱动。打开 SecureCRT 软件（V8.7，其他版本也可以），建立快速连接（File->Quick Connect），选择 Serial 协议，串口参数设置如下图所示。

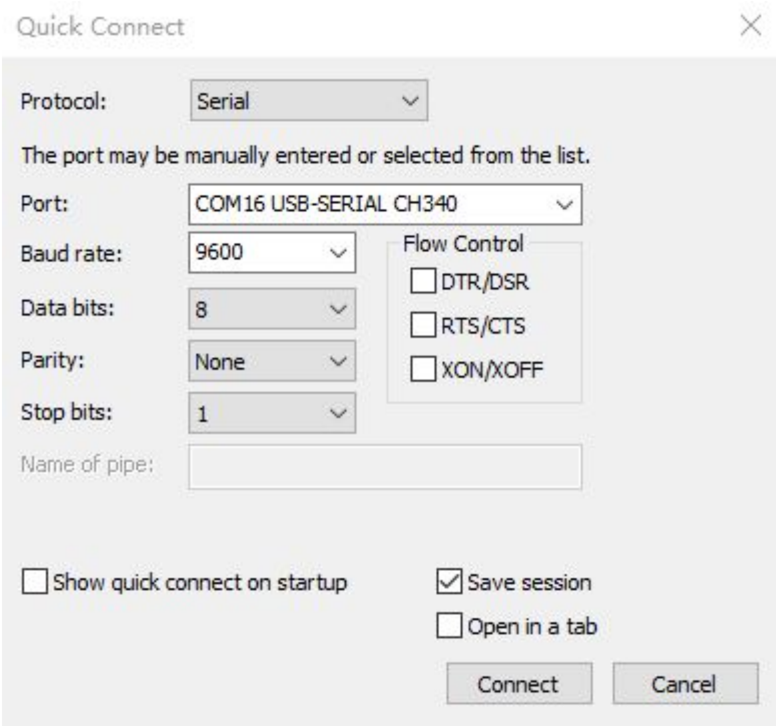

图 17.串口参数设置

按下复位键 K1 后，Bootloader 程序开始运行，LED3 闪烁，串口打印的数据如下图所示。

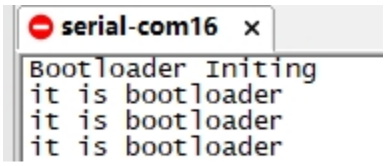

图 18.Bootloader 运行时发送的数据

按下 K2 键 1s 左右松开，串口不停打印‘c’，说明开发板在等待接收 APP.bin。

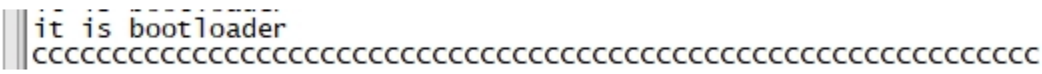

图 20.Bootloader 等待接收 APP.bin 文件

SecureCRT 下载 APP.bin 到 Bootloader。在 SecureCRT 的菜单栏 Transfer->Send Ymodem->选择目标文件 xxx_Flash.bin-> Add ->Ok，文件开始传输。

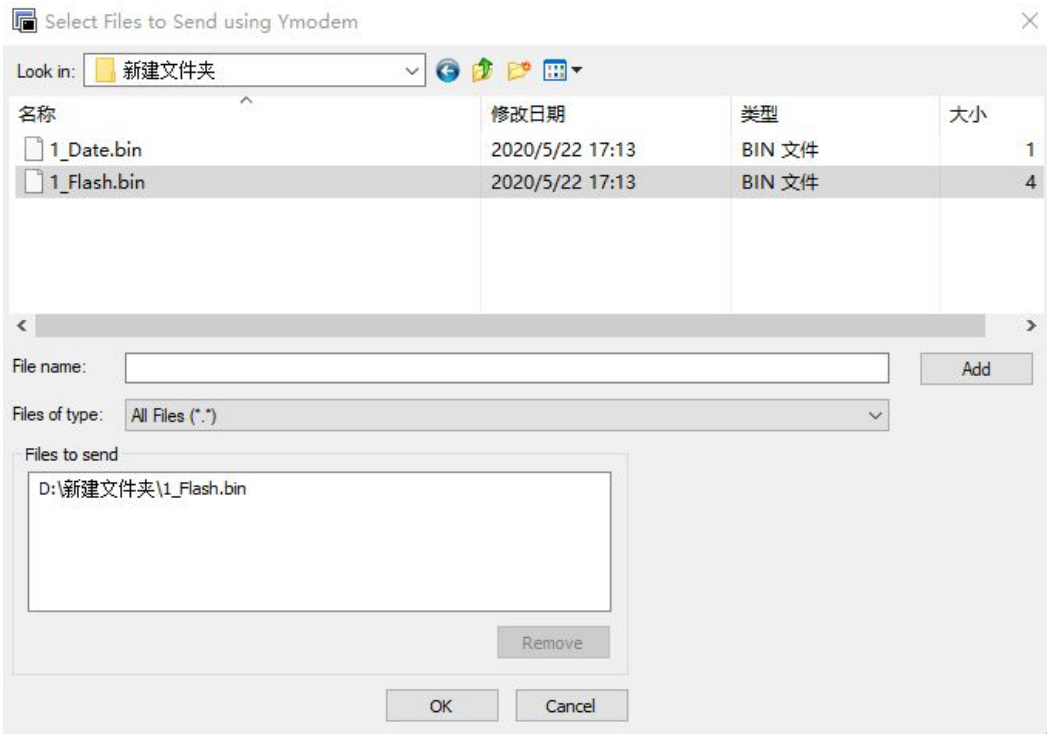

图 21.SecureCRT 中选择目标文件

文件传输完毕后（进度达到 100%），按下 Ctrl+C 组合键，SecureCRT 退出文件传输状态，返回串口通信状态。此时，可以看到 LED1、LED2、LED3 同时闪烁，串口打印“APP is running”，说明 Bootloader 成功，APP 成功启动。

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Starting ymodem transfer.  Press Ctrl+C to cancel.
Transferring 1_Flash.bin...
  100%       4 KB    1024 bytes/sec 00:00:04       0 Errors

APP is running
APP is running
```

图 22.SecureCRT 传输 APP.bin 的过程

逻辑分析仪记录下文件传输的过程。通过下图截取的片段可以看出数据包之间的间隔有 10ms，说明我们之前设置的串口接收超时的 30ms 阈值是合理的。

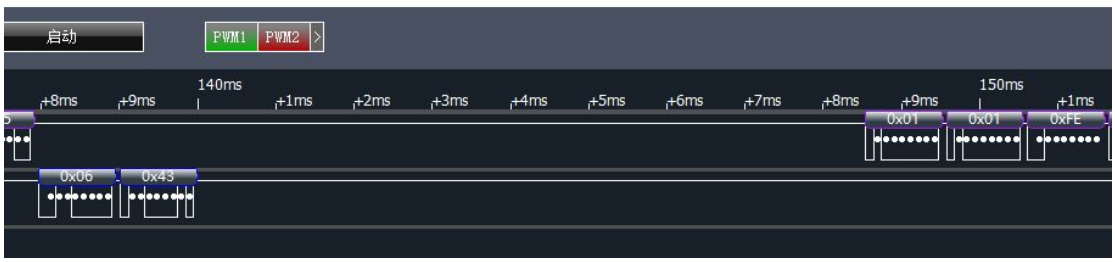

图 23.数据包之间的间隔

在 SecureCRT 的发送窗口中，右键->勾选 Send character，输入小写字母‘c’。可以看到，串口打印“Interrup is ok”。这个现象说明，APP 的串口接收中断工作正常，向量表重映射设置正确。

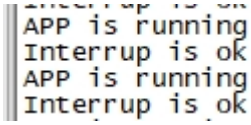

图 24.串口接收中断响应

此时，可以用 PRO 软件读出 MCU 中 Flash 的内容，以直观的检查 Flash 内容。对比 APP.hex 文件发现，从 0x0000 0000 开始是 Bootloader 程序，从 0x0000 8000 开始的区域和 APP.hex 是一致的，完全符合预期。

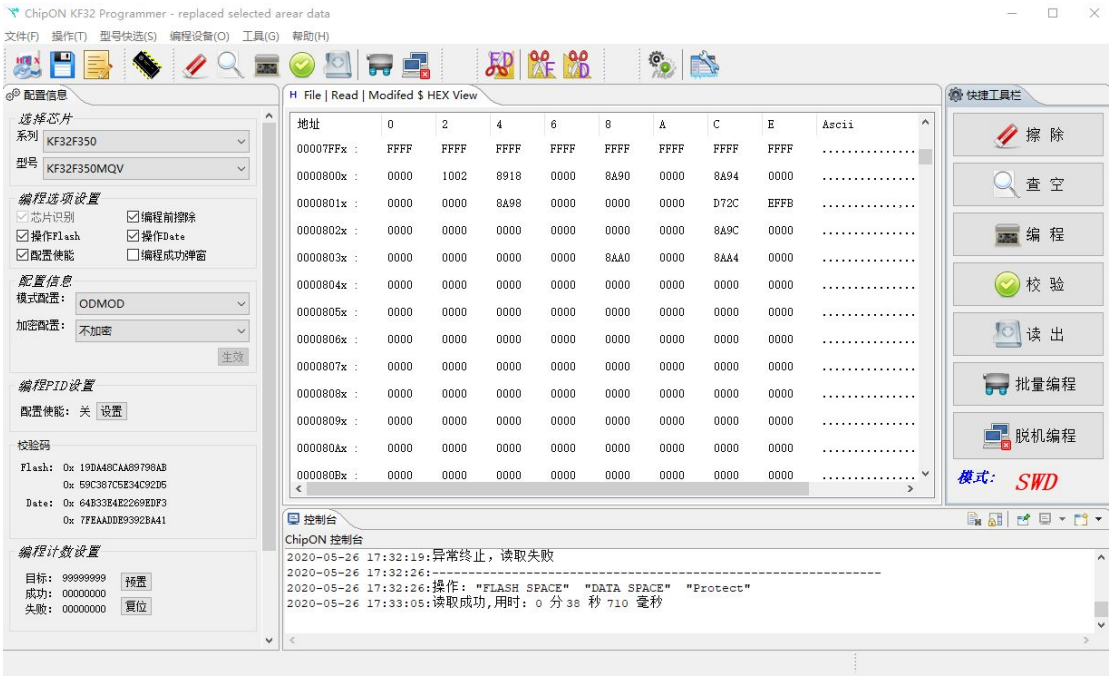

图 25.PRO 读出 Flash 内容

开发板重新上电，由于 Bootloader 程序检测到 APP 程序已经存在，将会继续跳转执行 APP 程序。到此时，Bootloader 程序验证完成。

六、总结

本文以 KF32F350MQV 为例描述了 Bootloader 的实现过程，其他芯片型号需要注意 Flash、RAM 和外设的差异并做相应的修改即可以成功使用 Bootloader。本文描述了 UART 方法的 Boot loader，重点在于介绍 Bootloader 的核心思想和关键方法。只要读者理解了 KF32Bootloader 的核心思想，结合自身的实际需求，并加以灵活应用，应该可以很快的开发出自己的 Bootloader。