文档编号：AN2020

上海东软载波微电子有限公司

应用笔记

ES32F36xx_Bootloader

修订历史

版本	修改日期	更改概要
V1.0	2020-01-17	初版
V1.1	2020-09-22	1. 修改基于 USB 模式的 Bootloader 方案的升级文件格式 2. 修改基于 USB 模式的 Bootloader 方案的启动流程

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/

目 录

内容目录

第1章 Bootloader简介

Bootloader 是在系统运行前执行的一小段程序，通过这一小段程序，可以初始化硬件设备，建立内存空间的映射表，从而建立适合的系统软硬件环境，为最终引导系统内核运行做好准备。由于 Bootloader 具有引导加载的功能，在上电复位时，可以加载指定的程序运行。因此提供了几种具有固件烧录功能的 Bootloader 程序来满足客户的升级需求。

一个完整的 Bootloader 程序大致包含以下四个部分:

- 复位初始化

  此部分主要负责初始化相关外设资源，为接下来的固件传输做准备。

- 固件传输

  此部分主要负责传输功能，用来接收一个完整的固件。

- CRC 校验

  此部分对接收的固件进行校验，检测传输过程是否存在错误。

- 加载引导

  按照烧录的固件信息引导到相应的启动位置。

下面章节将详细介绍这几种 Bootloader 升级方案。

第2章 基于 1K-Xmode协议的Bootloader方案

2. 1 Xmode协议

Xmodem 协议是一种使用拨号调制解调器的个人计算机通信中广泛使用的异步文件运输协议。Bootloader 采用 1k-Xmodem 协议进行 bin 文件传输。校验方式采用 CRC。

在工程中可以通过以下宏定义来选择使用 Xmodem 或 1k-Xmodem：

XMODE_128：数据传输采用标准 Xmodem 协议；

XMODE_1K：数据传输采用 1k-Xmodem 协议。

2. 2 地址划分

2. 2. 1 程序运行于FLASH

FLASH：

0x00000000----0x00007FFF Bootloader 程序空间；

0x00008000----0x0007FFFF 用户程序空间；

SRAM：

0x20000000---0x20017FFF Bootloader 程序和用户程序共用；

2. 2. 2 程序运行于SRAM

FLASH：

0x00000000----0x00007FFF Bootloader 程序空间；

0x00008000----0x0007FFFF 用户程序暂存空间；

SRAM：

0x20000000----0x20003FFF Bootloader 程序和用户程序共用的 SRAM 空间；

0x20004000----0x20018000 用户程序代码段，在用户程序运行前，Bootloader 会将用户程序代码段搬运到此地址空间。

2. 3 Bootloader命令行

Bootloader 命令行提供如下命令：

1. version: 显示 Bootloader 版本号，初始版本为“01-00-00-00”
2. reboot: 重启 MCU；
3. update: 启动 bin 文件更新规程；
4. run_flash: 引导运行于 FLASH 上的用户程序，此后 MCU 控制权移交给用户程序；
5. run_sram: 引导运行于 SRAM 上的用户程序，此后 MCU 控制权移交给用户程序；

注：run_sram 命令在引导用户程序之前，Bootloader 会将暂存在 FLASH 的用户程序代码段搬运至 SRAM 中。

2. 4 Bootloader运行流程

1. 上电复位，MCU 首先会运行 Bootloader；
2. Bootloader 首先检测 MCU 中是否有用户程序；
3. 若无，则等待用户启动 bin 文件更新规程(updata 命令)；
4. 若有，则等待 3 秒；
5. 在此期间命令行无输入，则自动引导用户程序。至于引导至 FLASH 还是 SRAM 上运行，由宏

“RUN_MODE_DEFAULT”控制，默认引导至 FLASH 上运行。

6. 在此期间命令行有输入，则放弃自动引导用户程序，进入“接收命令--解析命令--执行命令”规程。

2.5 工程介绍

工程位置：

ES32F36xx：~\ES32_SDK\Projects\ES32F36xx\Applications\Bootloader\01_uart_1k_xmode

ES32F336x：~\ES32_SDK\Projects\ES32F336x\Applications\Bootloader\01_uart_1k_xmode

2.6 工程演示

上位机串口工具请选择支持 1K-Xmode 和命令行操作的串口软件，本例使用 ECOM 串口助手，它自带 1K Xmode 串口协议，可进行 CRC 校验，并且支持重传及命令行输入等操作。当烧录好 Bootloader 程序，并且正确设定串口参数后复位芯片，会看到如下图所示的打印信息：

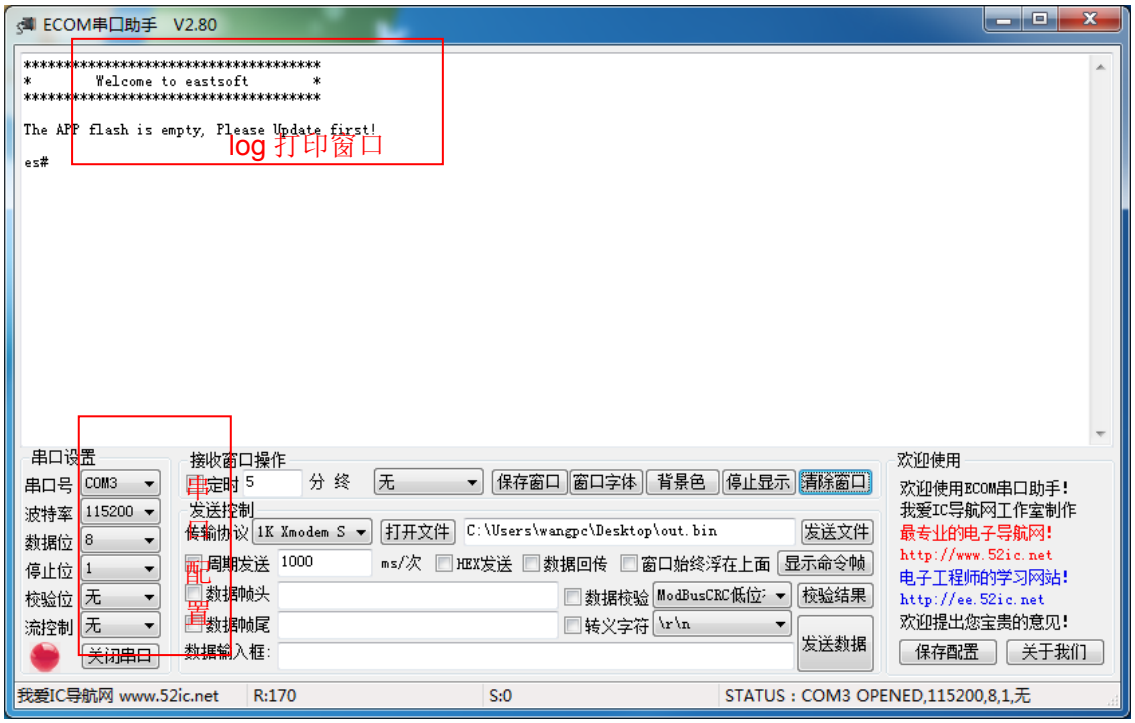

图 2-1 Bootloader 上电复位 log 图

参照 Bootloader 运行流程，在上电复位后，Bootloader 程序会自动检测 App 区域 FLASH 里是否为空，若为空则提示用户需要下载更新固件，因此用户需在命令行中输入更新固件命令—“update”，之后点击“发送文件”按钮，等待烧录开始，如下图所示。

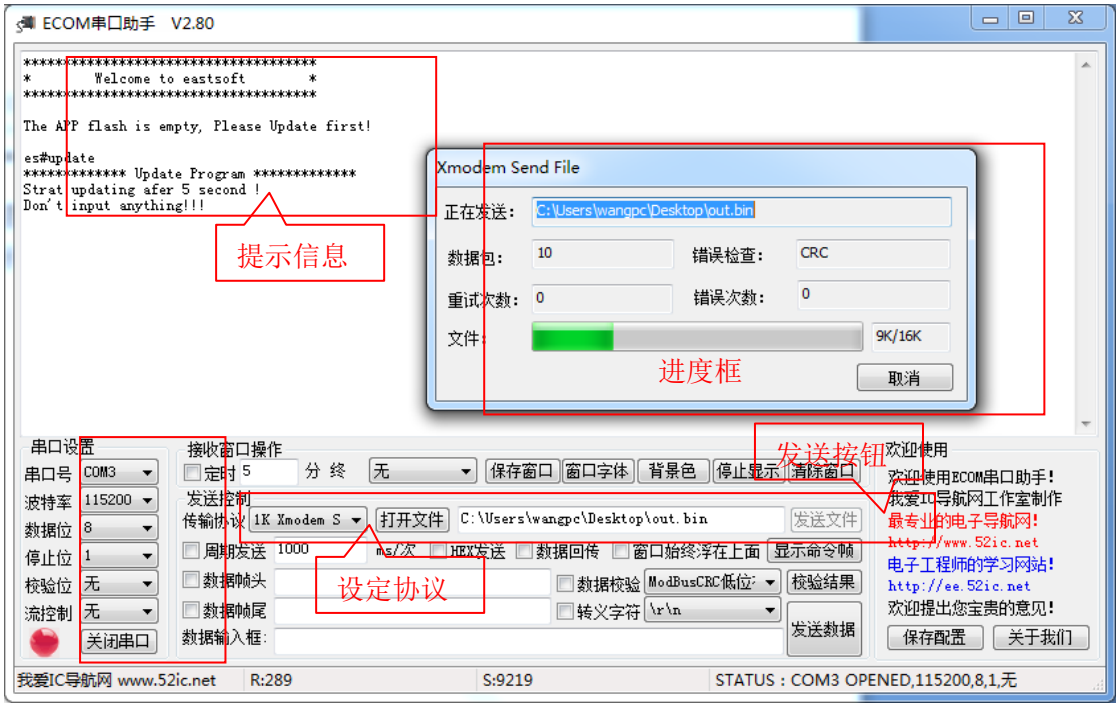

图 2-2 固件更新操作图

若固件更新成功，用户可以按照烧录的 bin 文件，选择正确的引导命令：run_flash(引导至 FLASH)；run_sram(引导至 SRAM)；reboot(重新启动 Bootloader，让其按默认方式自动引导)。

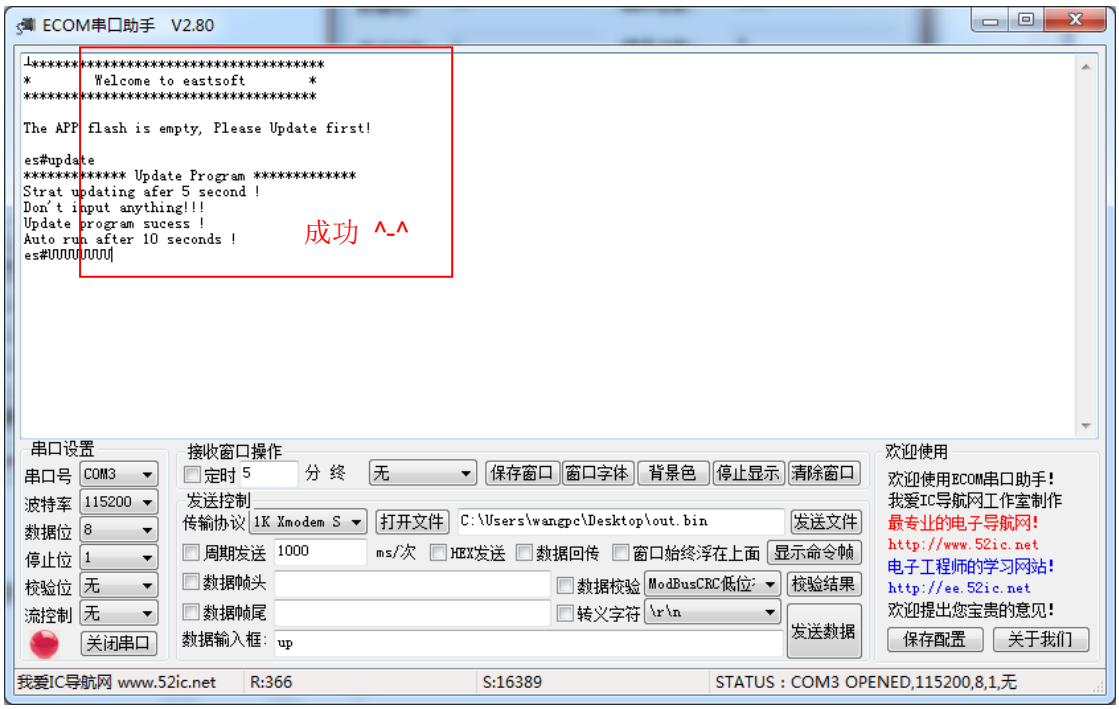

图 2-3 固件更新成功示例图

2.7 注意事项

2.7.1 串口设置

本例中串口使用 UART0：
TX----PB10(GPIO_FUNC_3)
RX----PB11(GPIO_FUNC_3)
波特率：115200
奇偶校验：无
数据位宽：8
停止位：1
自动流控：无
请以此信息正确配置 PC 端的串口软件。

2.7.2 bin文件生成方式

2.7.2.1 MDK

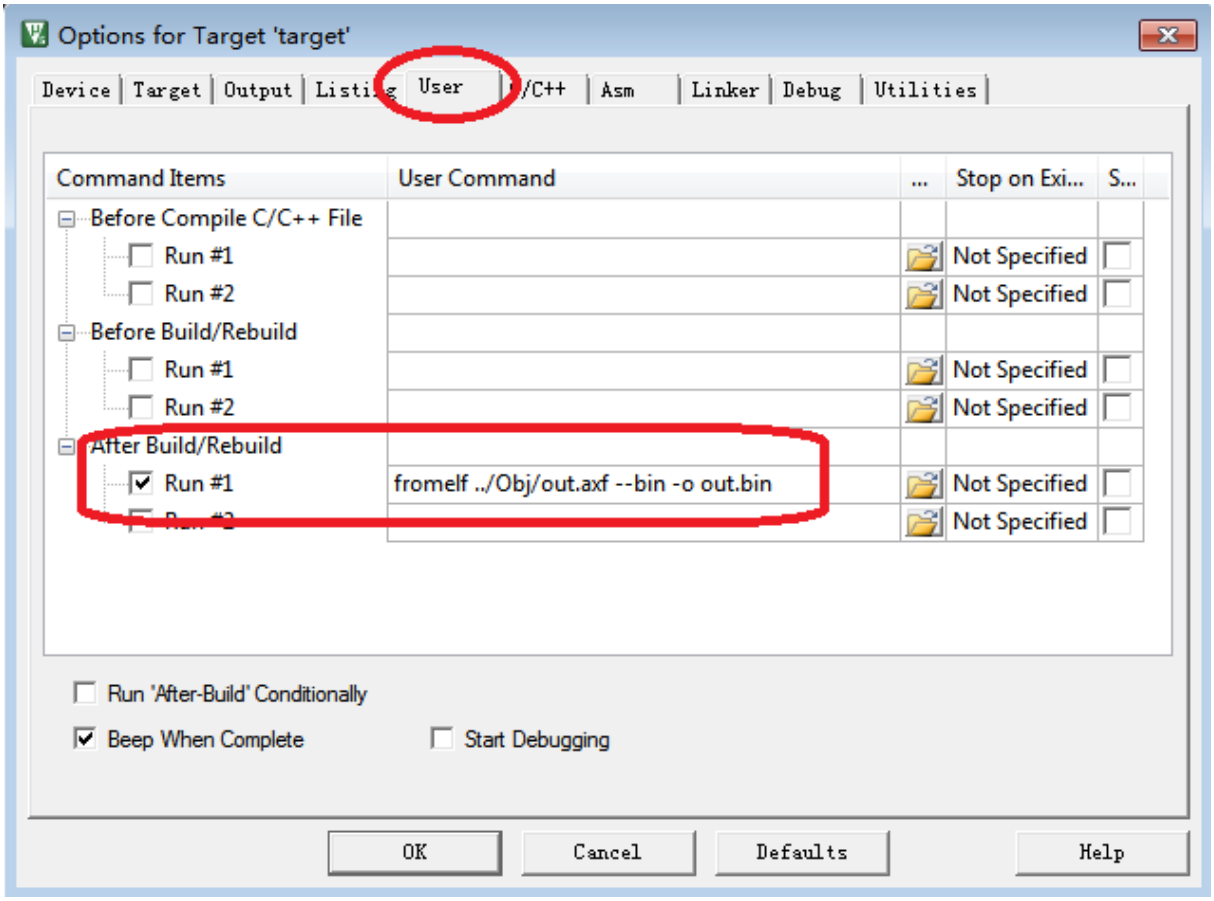

图 2-4 MDK 输出 bin 文件配置

其中 out.axf 根据用户自己设定的可执行文件名确定。与下图中的名称保持一致：

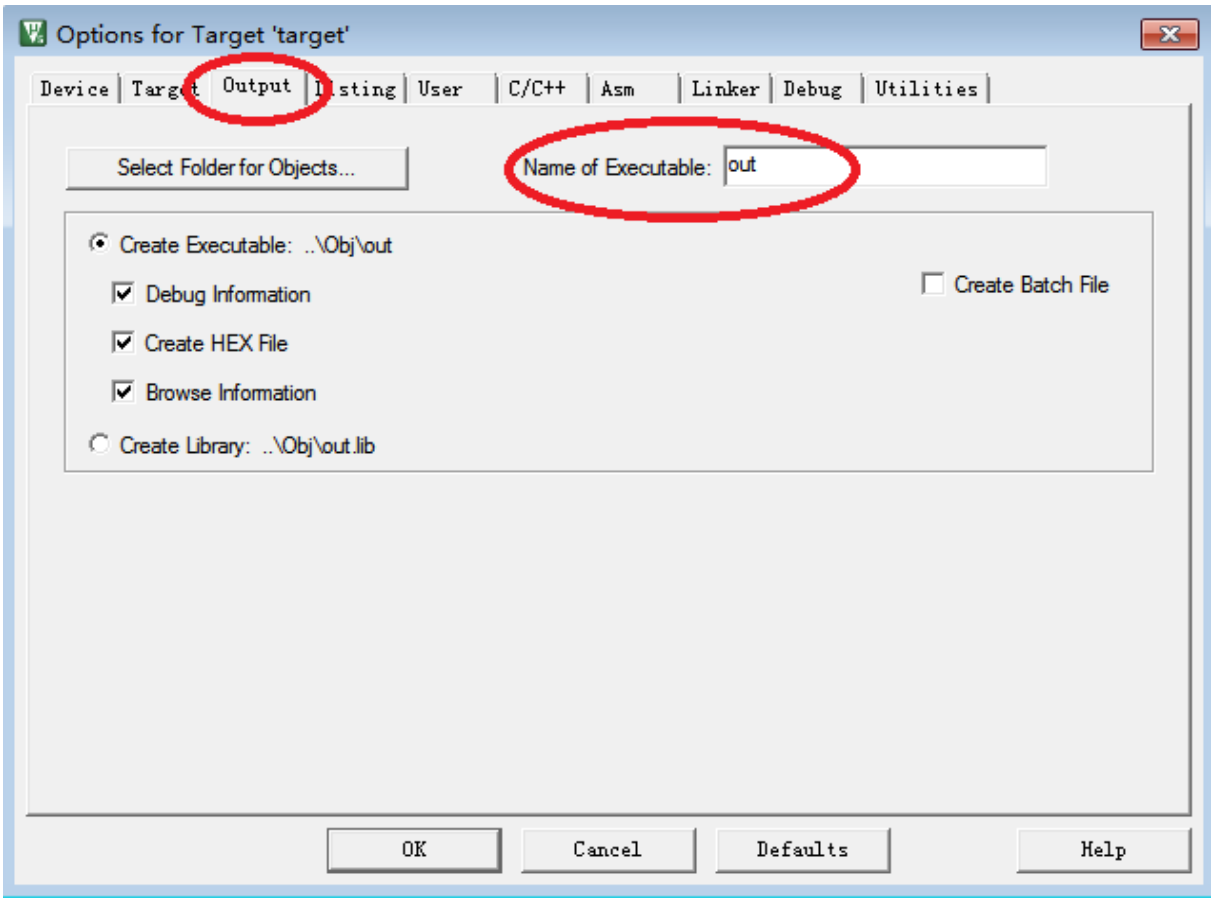

图 2-5 MDK 可执行文件名

配置好后，重新编译，将会在“.uvprojx”同级目录下生成相应 bin 文件。

2.7.2.2 IAR

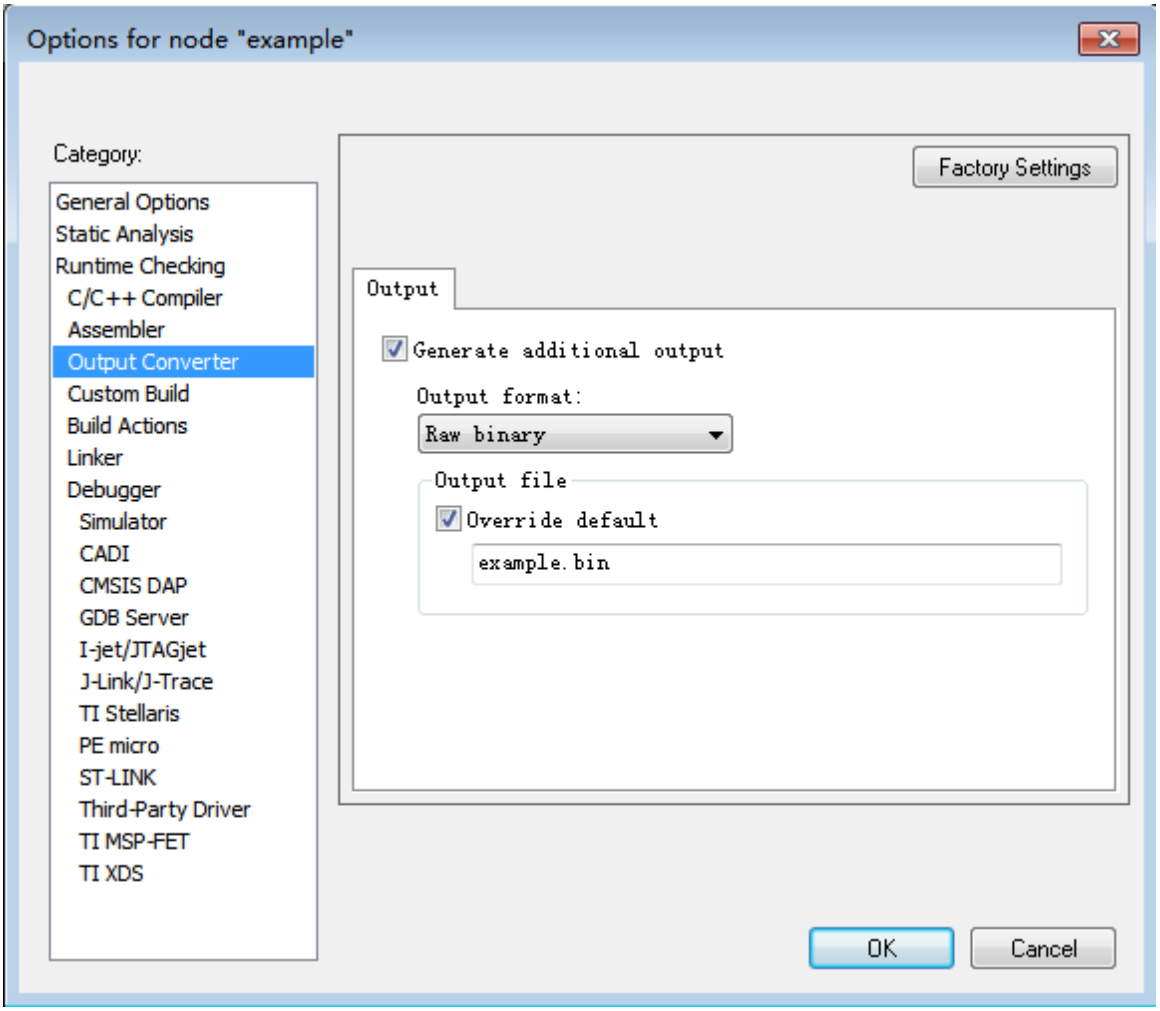

图 2-6 IAR 输出 bin 文件配置

按如图方式配置后，重新编译，在~/Debug/Exe/文件夹下会生成相应 bin 文件。

2.7.2.3 GCC

在 Makefile 文件中添加：

$(OBJCOPY) $(TARGET) -O binary $(TARGET).bin

其中 OBJCOPY 为交叉编译链的 objcopy，一般为：arm-none-eabi-objcopy

第3章　基于USB从机模式的Bootloader方案

3. 1　实现原理

以 ES2F3696LX 芯片为例，它的 512K FLASH 和 96K SRAM 被人为划分为 BOOT 区、FATFS 区和 APP 区， BOOT 区放置 Bootloader 运行的相关资源，FATFS 区放置虚拟的文件系统，APP 区放置用户程序运行的相关资源（如下图所示）。产品上电复位后可选择烧录新的固件，或是运行原有的固件。

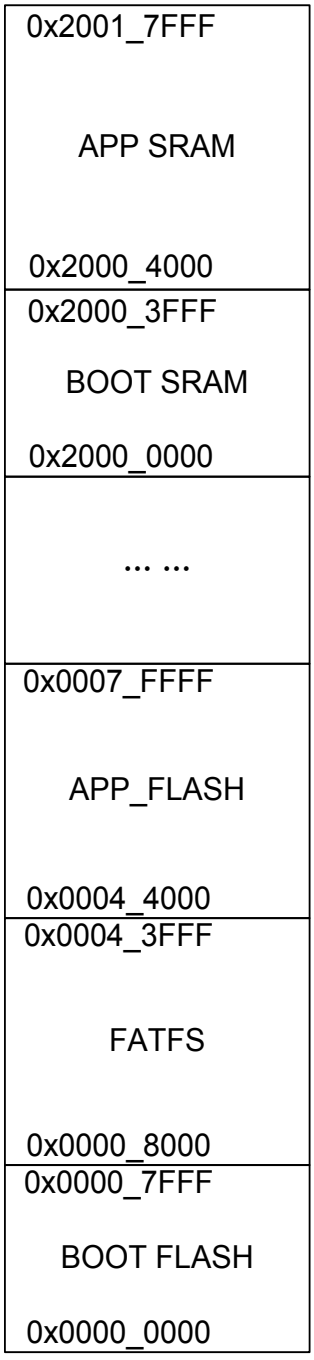

图 3-1 存储区域划分图

3.2 启动流程

产品在烧录 Bootloader 程序后，上电复位会延时 5S 检查 USB 口是否连接到电脑上，若检查到连接，则会在电脑上虚拟出一个“可移动磁盘”，等待用户将 APP 程序的 bin 文件写入“可移动磁盘”，并根据 Bootloader 程序配置的运行参数在 FLASH 或 RAM 上自动运行新固件；若没有检查到连接，5S 过后将自动运行原有的 APP 程序，具体流程如下图所示。

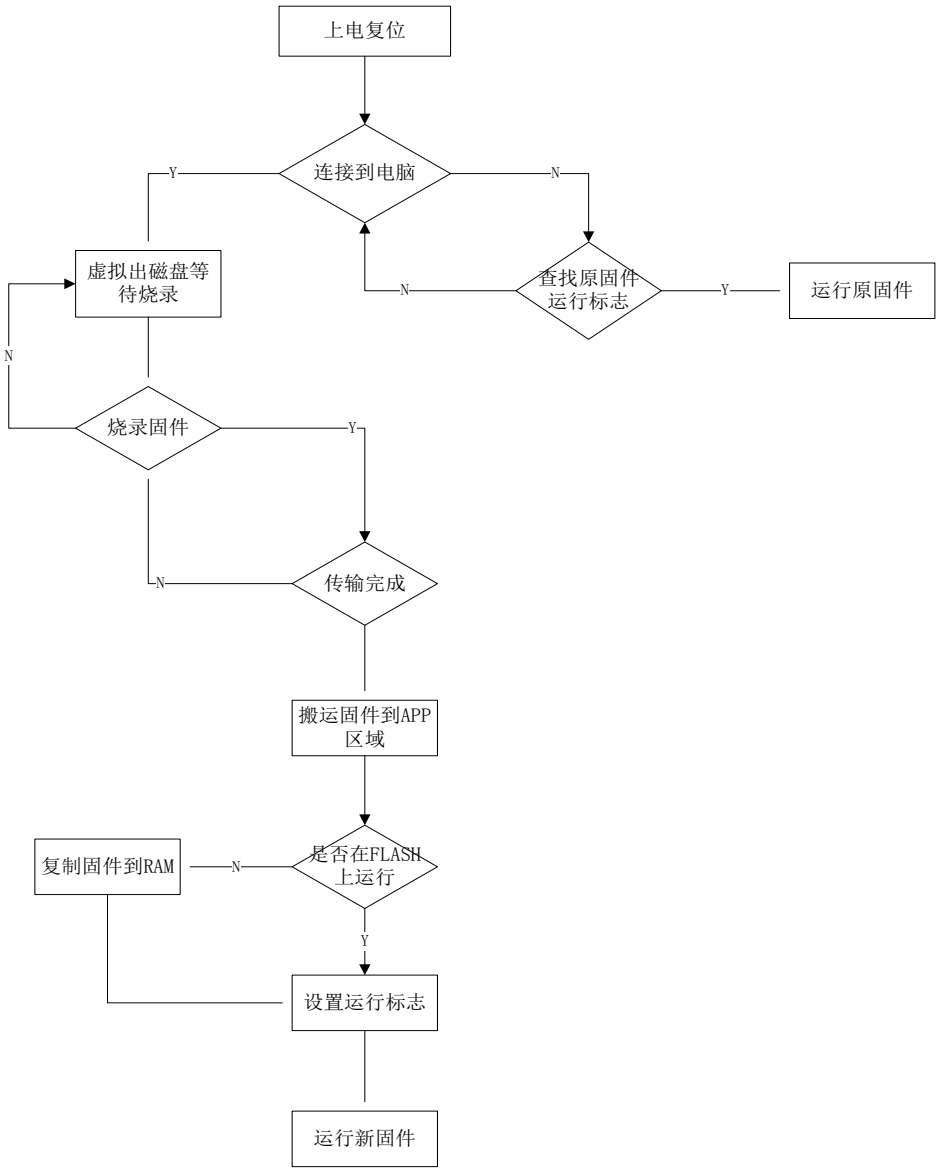

图 3-2 Bootloader 启动流程图

3.3 工程介绍

工程位置：

ES32F36xx：~\ES32_SDK\Projects\ES32F36xx\Applications\Bootloader\02_usb_dev_fatfs

ES32F336x：~\ES32_SDK\Projects\ES32F336x\Applications\Bootloader\02_usb_dev_fatfs

3.4 用户程序链接文件配置

3.4.1 程序运行于FLASH

FLASH：0x00044000~0x0007FFFF
SRAM：0x20000000~0x20017FFF

3.4.2 程序运行于SRAM

FLASH：0x20004000~0x20017FFF
SRAM：0x20000000~0x20003FFF

3.5 工程演示

烧录 Bootloader 固件，使用 USB 数据线将开发部与电脑连接，此时在电脑上会虚拟出一个“可移动磁盘”，如下图所示。

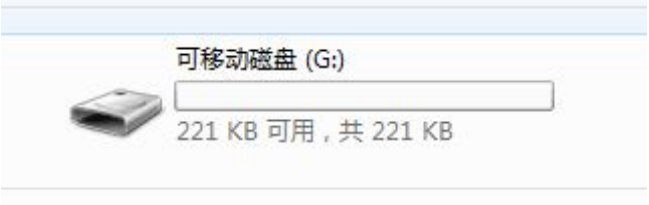

图 3-3 虚拟磁盘图

参照图 3-2 所示的流程图，在虚拟出一个磁盘后，我们只需将要烧录的固件拖入这个磁盘，即可将固件烧录到板子上，如下图所示。

图 3-4 固件更新图

在固件烧录到板子上后，Bootloader 程序会根据配置的运行参数，在 FLASH 或 RAM 区域自动运行新固件，如图 3-5 所示。若连接超时 Bootloader 程序会自动根据原固件的运行标志运行原程序，若无运行标志则会一直检查 USB 的连接情况，等待固件烧录。

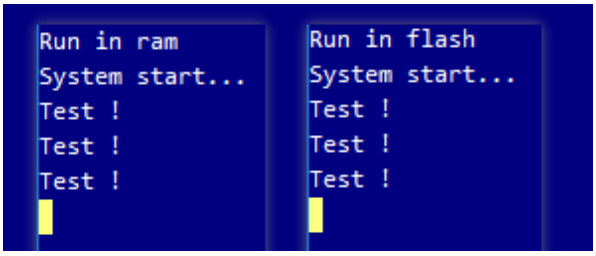

图 3-5 固件更新成功示例图

3.6 注意事项

- Bootloader 检索固件文件的默认名字为“image.bin”，更新的固件要与此名字一致
- Keil、IAR、GCC 等编译器生成的 hex 文件无法直接使用，需参照 2.7.2 小节生成对应的 bin 文件。
- 烧录固件需放在根目录下，Bootloader 程序暂不支持文件夹递归查找功能。

第4章　基于USB主机模式的Bootloader方案

4. 1　实现原理

以 ES2F3696LX 芯片为例，它的 512K FLASH 和 96K SRAM 被人为划分为 BOOT 区和 APP 区， BOOT 区放置 Bootloader 运行的相关资源，APP 区放置用户程序运行的相关资源（如下图所示）。产品上电复位后可选择烧录新的固件，或是运行原有的固件。

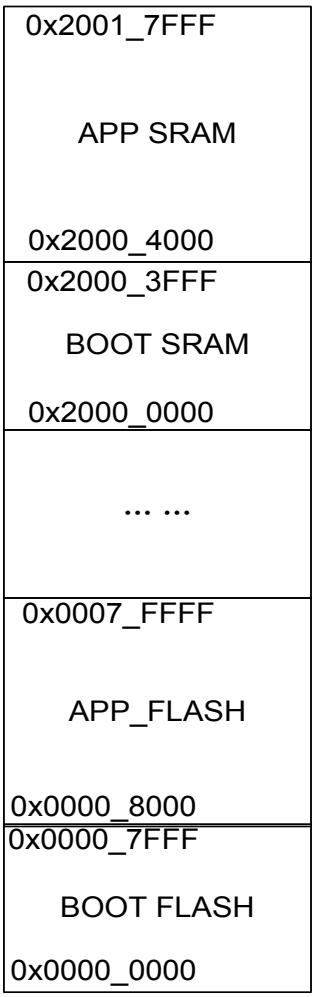

图 4-1 存储区域划分图

4. 2　启动流程

产品在烧录 Bootloader 程序后，上电复位后会检测是否有 U 盘等从机插入，若检查到从机插入，则会自动检索要烧录的固件；若没有检测到从机插入，则会检测原有固件是否完整，运行原有固件；若原有固件不完整则会一直等待从机的连接，具体流程如下图所示。

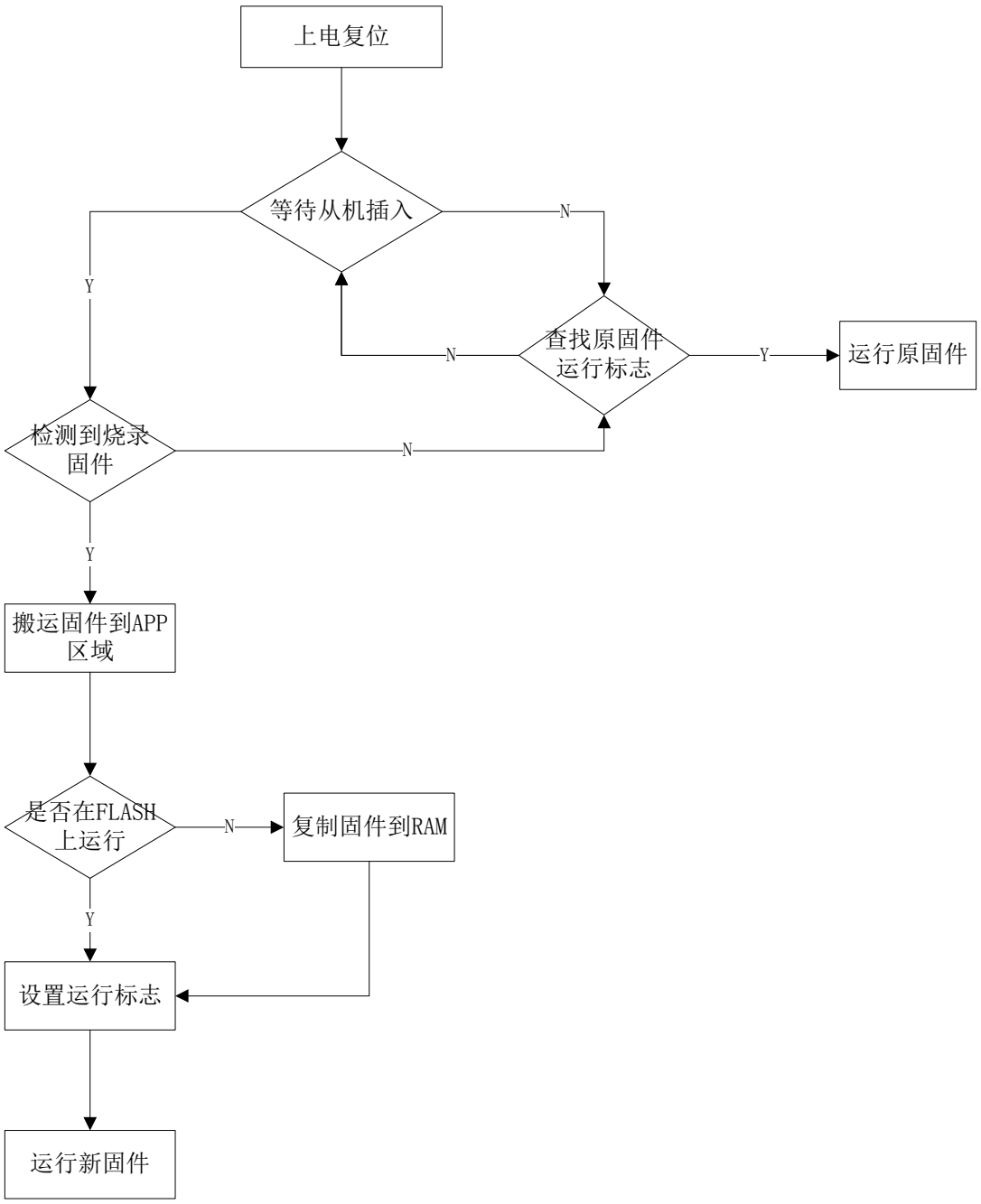

图 4-2 Bootloader 启动流程图

4.3 工程介绍

工程位置：

ES32F36xx：~\ES32_SDK\Projects\ES32F36xx\Applications\Bootloader\03_usb_host_fatfs

4.4 用户程序链接文件配置

4.4.1 程序运行于FLASH

FLASH：0x00008000~0x0007FFFF

SRAM：0x20000000~0x20017FFF

4. 4. 2 程序运行于SRAM

FLASH：0x20004000~0x20017FFF
SRAM：0x20000000~0x20003FFF

4. 5 工程演示

参照图 4-2 所示的流程图，烧录 Bootloader 程序后，将存放更新固件的 U 盘插入板载的 USB 接口。Bootloader 程序会自动检测到更新固件并搬运到 APP 区域，根据 Bootloader 配置的运行标志，选择在 FLASH 或 RAM 区域运行，如图 4-3 所示。若从机连接超时，Bootloader 程序会自动根据原固件的运行标志运行原程序，若无运行标志则会一直检查 USB 的连接情况，等待固件烧录。

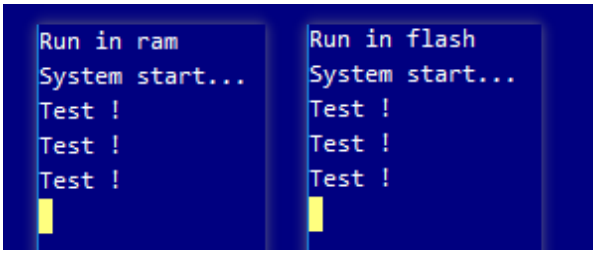

图 4-3 固件更新成功示例图

4. 6 注意事项

- Bootloader 检索固件文件的默认名字为“image.bin”，更新的固件要与此名字一致。
- Keil、IAR、GCC 等编译器生成的 hex 文件无法直接使用，需参照 2.7.2 小节生成对应的 bin 文件。
- 烧录固件需放在根目录下，Bootloader 程序暂不支持文件夹递归查找功能。
- 请将 U 盘格式化为 FAT32 文件系统。

第5章 用户程序连接文件设计

以下介绍均以 ES32F3696 为例，其他型号芯片，可根据 FLASH 和 SRAM 大小自行调整。

5. 1 程序运行于FLASH(基于UART-XMode、主USB模式)

5. 1. 1 MDK

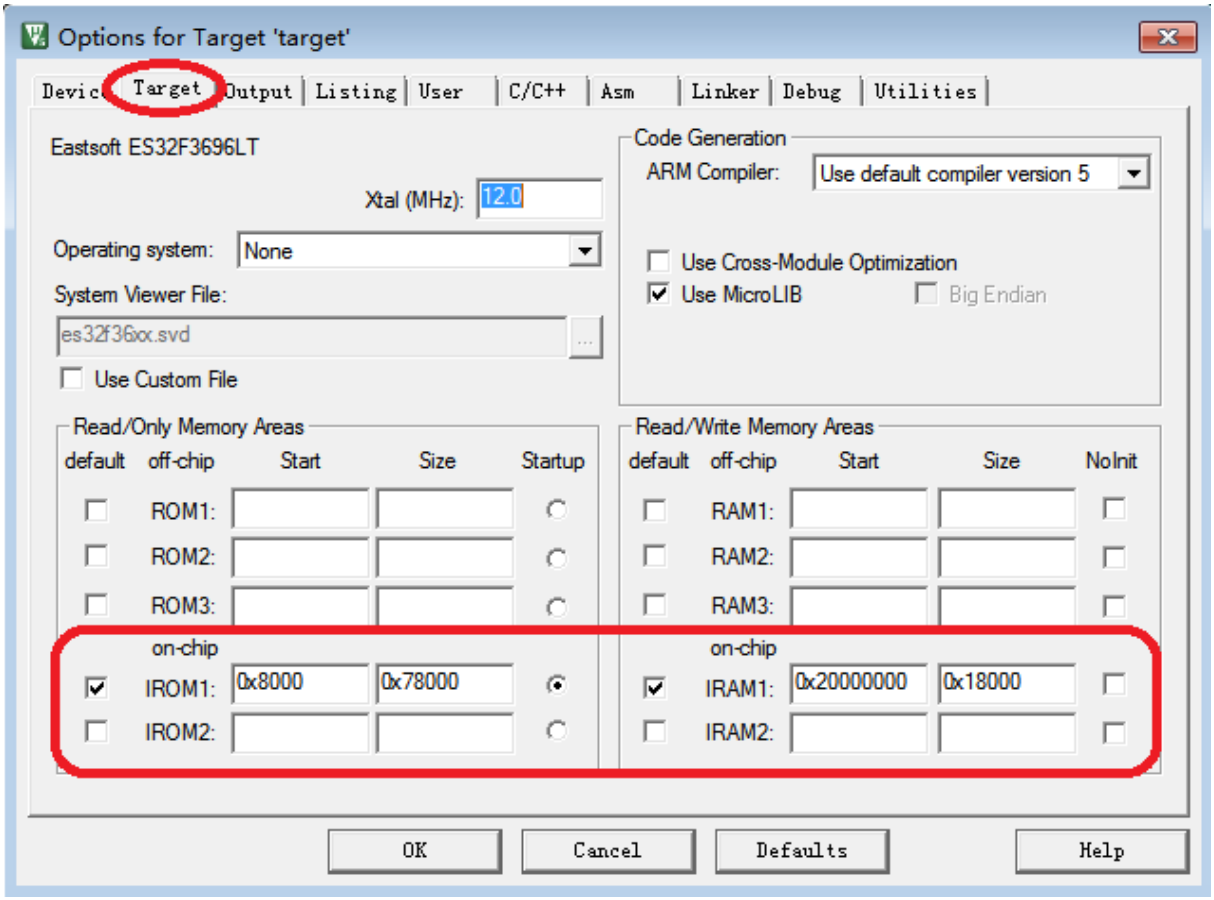

图 5-1 MDK 链接地址

5. 1. 2 IAR

修改 Program Files (x86)/IAR Systems/Embedded Workbench 8.0/arm/config/linker/Eastsoft 文件夹下的 ES32F3696x.icf 文件：

```
5  define symbol __ICFEDIT_intvec_start__ = 0x00008000;
6  /*-Memory Regions-*/
7  define symbol __ICFEDIT_region_ROM_start__ = 0x00008000;
8  define symbol __ICFEDIT_region_ROM_end__   = 0x0007FFFF;
9  define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
10 define symbol __ICFEDIT_region_RAM_end__   = 0x20017FFF;
11 /*-Sizes-*/
12 define symbol __ICFEDIT_size_cstack__ = 0x400;
13 define symbol __ICFEDIT_size_heap__   = 0x400;
14 /**** End of ICF editor section. ###ICF###*/
```

图 5-2 IAR 链接地址

5. 1. 3 GCC

修改 ld 文件：

```
/* Highest address of the user mode stack */
_estack = 0x20018000;    /* end of 96K RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
  FLASH (rx)       : ORIGIN = 0x00008000, LENGTH = 480K
  RAM (xrw)        : ORIGIN = 0x20000000, LENGTH = 96K
  MEMORY_B1 (rx)   : ORIGIN = 0x20018000, LENGTH = 0K
}
```

图 5-3 GCC 链接地址

5. 2 程序运行于FLASH(基于从USB模式)

5. 2. 1 MDK

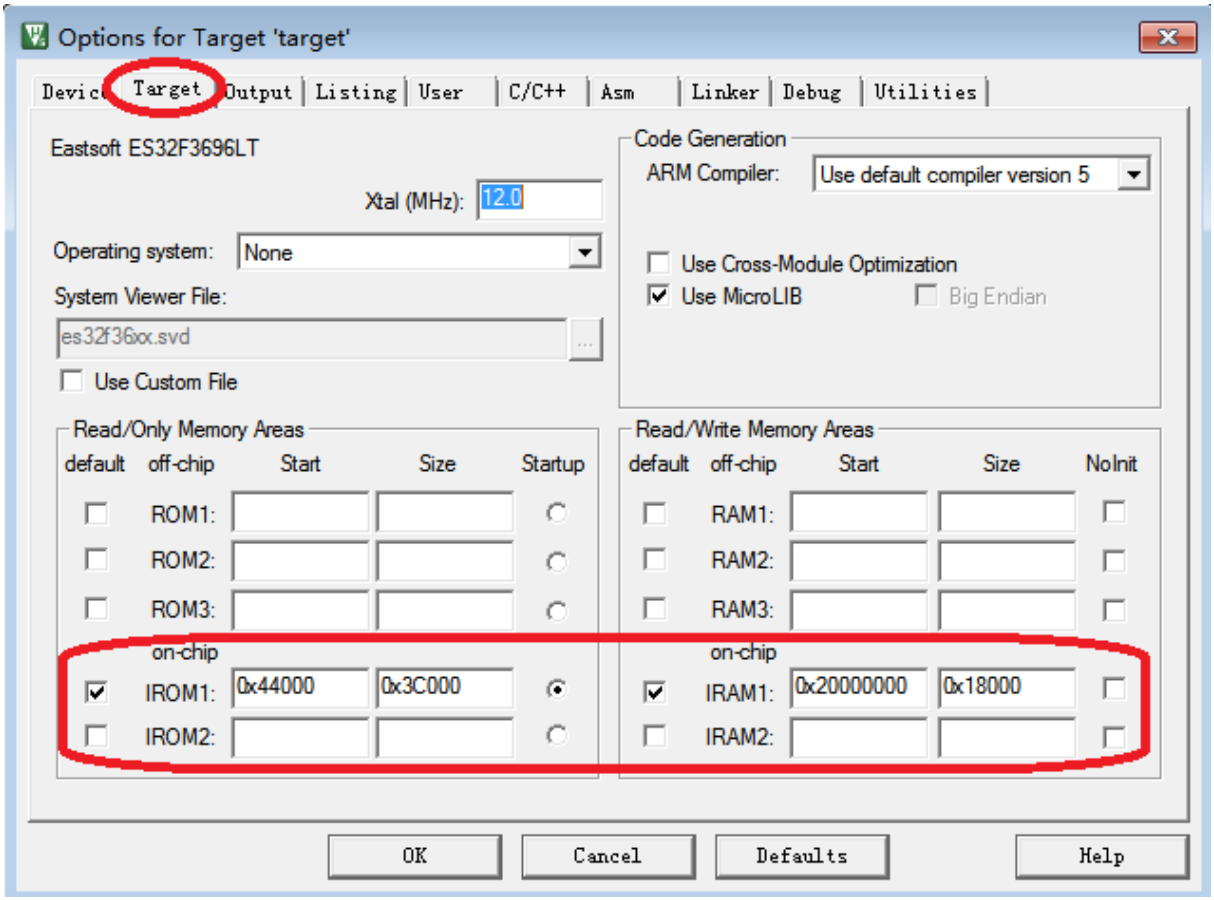

图 5-4 MDK 链接地址

5. 2. 2 IAR

修改 Program Files (x86)/IAR Systems/Embedded Workbench 8.0/arm/config/linker/Eastsoft 文件夹下的 ES32F3696x.icf 文件：

```
define symbol __ICFEDIT_intvec_start__ = 0x00044000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x00044000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0007FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20017FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x400;
/**** End of ICF editor section. ###ICF###*/
```

图 5-5 IAR 链接地址

5. 2. 3 GCC

修改 ld 文件：

```
/* Highest address of the user mode stack */
_estack = 0x20018000;    /* end of 96K RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
  FLASH (rx)       : ORIGIN = 0x00044000, LENGTH = 240K
  RAM (xrw)        : ORIGIN = 0x20000000, LENGTH = 96K
  MEMORY_B1 (rx)   : ORIGIN = 0x20018000, LENGTH = 0K
}
```

图 5-6 GCC 链接地址

5. 3 程序运行于SRAM

5. 3. 1 MDK

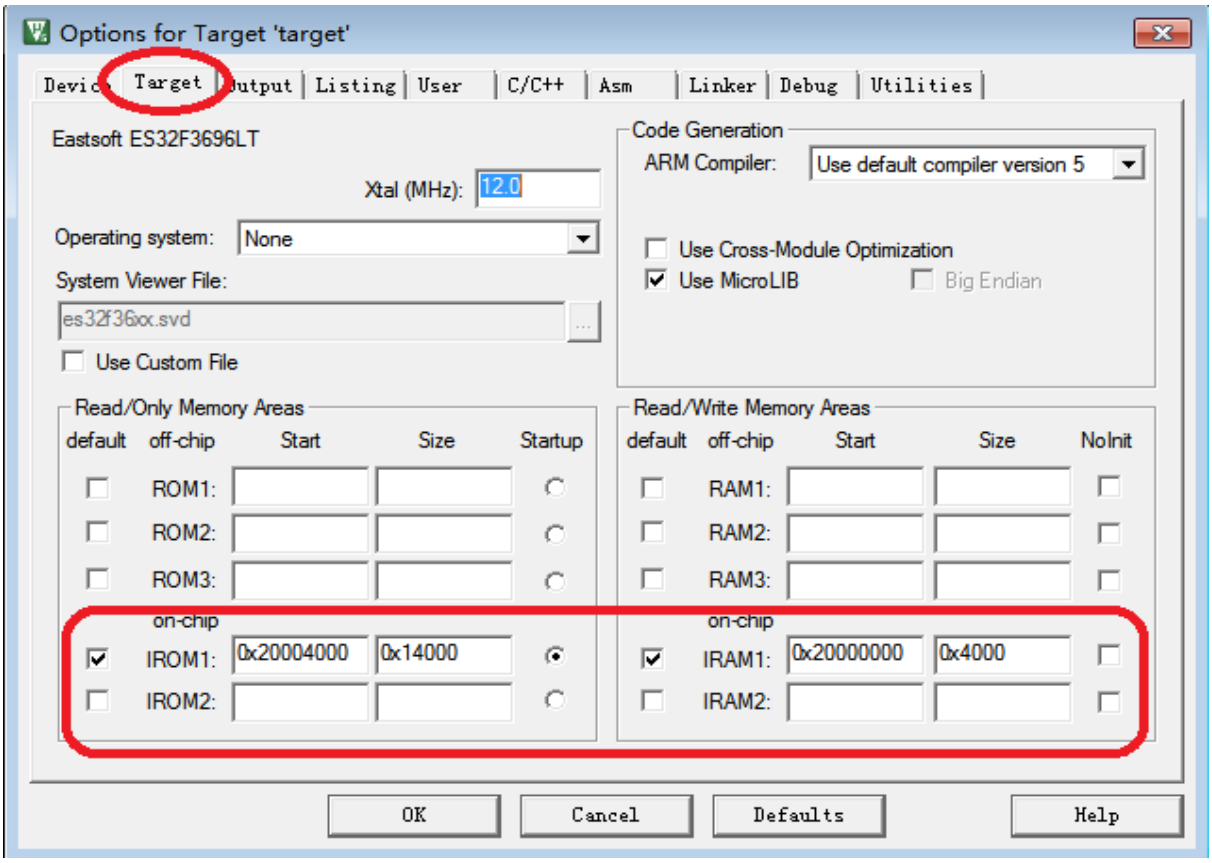

图 5-7 MDK 链接地址

5. 3. 2 IAR

修改 Program Files (x86)/IAR Systems/Embedded Workbench 8.0/arm/config/linker/Eastsoft 文件夹下的 ES32F3696x.icf 文件：

```
define symbol __ICFEDIT_intvec_start__ = 0x20004000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x20004000;
define symbol __ICFEDIT_region_ROM_end__   = 0x00017FFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20003FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x400;
/**** End of ICF editor section. ###ICF###*/
```

图 5-8 IAR 链接地址

5. 3. 3 GCC

修改 ld 文件：

```
/* Highest address of the user mode stack */
_estack = 0x20004000;    /* end of 16K RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
  FLASH (rx)       : ORIGIN = 0x20004000, LENGTH = 80K
  RAM (xrw)        : ORIGIN = 0x20000000, LENGTH = 16K
  MEMORY_B1 (rx)   : ORIGIN = 0x20018000, LENGTH = 0K
}
```

图 5-9 GCC 链接地址

第6章 FAT文件系统简介

6.1 FAT文件系统概述

FAT（File Allocation Table，文件分配表）文件系统是 windows 操作系统所使用的一种文件系统，它的发展过程经历了 FAT12、FAT16、FAT32 三个阶段。

每个 FAT 文件系统由 4 部分组成，这些基本区域按如下顺序排列：

- 保留区（Reserved Region）
- FAT 区（FAT Region）
- 根目录区（Root Directory Region， FAT32 卷无此域）
- 文件和目录数据区（File and Directory Data Region）

FAT 文件系统用“簇”作为数据单元。一个“簇”由一组连续的扇区组成，簇所含的扇区数必须是 2 的整数次幂。簇的最大值为 64 个扇区。所有簇从 2 开始进行编号，每个簇都有一个自己的地址编号。用户文件和目录都存储在簇中。

FAT 文件系统的数据结构中有两个重要的结构：文件分配表和目录项：

- 文件和文件夹内容储存在簇中，如果一个文件或文件夹需要多余一个簇的空间，则用 FAT 表来描述如何找到另外的簇。FAT 结构用于指出文件的下一个簇，同时也说明了簇的分配状态。FAT12、FAT16、FAT32 这三种文件系统之间的主要区别在与 FAT 项的大小不同。
- FAT 文件系统的每一个文件和文件夹都被分配到一个目录项，目录项中记录着文件名、大小、文件内容起始地址以及其他一些元数据。

在 FAT 文件系统中，文件系统的数据记录在“引导扇区中（DBR）”中。引导扇区位于整个文件系统的 0 号扇区，是文件系统隐藏区域（也称为保留区）的一部分，我们称其为 DBR（DOS Boot Recorder——DOS 引导记录）扇区，DBR 中记录着文件系统的起始位置、大小、FAT 表个数及大小等相关信息。

在 FAT 文件系统中，同时使用“扇区地址”和“簇地址”两种地址管理方式。这是因为只有存储用户数据的数据区使用簇进行管理（FAT12 和 FAT16 的根目录除外），所有簇都位于数据区。其他文件系统管理数据区域是不以簇进行管理的，这部分区域使用扇区地址进行管理。文件系统的起始扇区为 0 号扇区。

6.2 FAT文件系统整体布局

图 6-1 FAT 文件系统整体布局

说明：

【1】 保留区含有一个重要的数据结构——系统引导扇区（DBR）。FAT12、FAT16 的保留区通常只有一个扇区，而 FAT32 的保留扇区要多一些，除 0 号扇区外，还有其他一些扇区，其中包括了 DBR 的备份扇区。

【2】 FAT 区由来年各个大小相等的 FAT 表组成——FAT1、FAT2，FAT2 紧跟在 FAT1 之后。

【3】 FAT12、FAT16 的根目录虽然也属于数据区，但是他们并不由簇进行管理。也就是说 FAT12、

FAT16 的根目录是没有簇号的，他们的 2 号簇从根目录之后开始。而 FAT32 的根目录通常位于 2 号簇。

6. 3 FAT文件系统的保留区

引导扇区是 FAT 文件系统的第一个扇区，也称为 DBR 扇区。它包含这样一些文件系统的基本信息：

【1】 每扇区字节数
【2】 每簇扇区数
【3】 保留扇区数
【4】 FAT 表个数
【5】 文件系统大小（扇区数）
【6】 每个 FAT 表大小（扇区数）
【7】 根目录起始簇号
【8】 其他一些附加信息

（DBR 扇区中记录文件系统参数的部分也称为 BPB（BIOS Parameter Block））

我们可以通过每个 FAT 表的大小扇区数乘以 FAT 表的个数得到 FAT 区域的大小；通过保留扇区数和 FAT 区域的大小就可以得知数据区的起始位置，也就得到了文件系统第一簇的位置。由根目录的簇号和第一簇的位置就可以得到根目录的位置。

引导扇区数据结构及实例讲解：

这个小节将通过讲解一个 Kingston 2GB 的 SD 卡的 DBR（FAT32 文件系统），来详细说明引导扇区数据结构各个参数的含义，先给出几张图片：

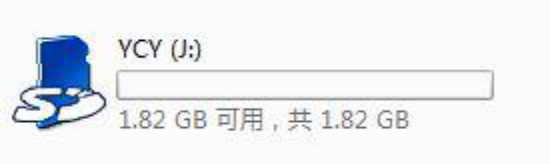

图 6-2 U 盘示意图

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	EB	58	90	4D	53	44	4F	53	35	2E	30	00	02	08	C0	02	ëX.MSDOS5.0...À.
00000010	02	00	00	00	00	F8	00	00	3F	00	FF	00	89	00	00	00	.....ø..?.ÿ.‰...
00000020	77	9F	3A	00	A0	0E	00	00	00	00	00	00	02	00	00	00	w.:. .....
00000030	01	00	06	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000040	80	00	29	C8	6C	9F	54	4E	4F	20	4E	41	4D	45	20	20	.)Èl.TNO NAME
00000050	20	20	46	41	54	33	32	20	20	20	33	C9	8E	D1	BC	F4	FAT32 3É.Ñ¼ô
00000060	7B	8E	C1	8E	D9	BD	00	7C	88	4E	02	8A	56	40	B4	41	{.Á.Ù½.|.N..V@´A
00000070	BB	AA	55	CD	13	72	10	81	FB	55	AA	75	0A	F6	C1	01	»ªUÍ.r..ûUªu.öÁ.
00000080	74	05	FE	46	02	EB	2D	8A	56	40	B4	08	CD	13	73	05	t.þF.ë-.V@´.Í.s.
00000090	B9	FF	FF	8A	F1	66	0F	B6	C6	40	66	0F	B6	D1	80	E2	¹ÿÿ.ñf.¶Æ@f.¶Ñ.â
000000A0	3F	F7	E2	86	CD	C0	ED	06	41	66	0F	B7	C9	66	F7	E1	?÷â.ÍÀí.Af.·Éf÷á
000000B0	66	89	46	F8	83	7E	16	00	75	38	83	7E	2A	00	77	32	f‰Fø.~..u8.~*.w2
000000C0	66	8B	46	1C	66	83	C0	0C	BB	00	80	B9	01	00	E8	2B	f.F.f.À.».€¹..è+
000000D0	00	E9	2C	03	A0	FA	7D	B4	7D	8B	F0	AC	84	C0	74	17	.é,. ú}´}.ð¬.Àt.
000000E0	3C	FF	74	09	B4	0E	BB	07	00	CD	10	EB	EE	A0	FB	7D	<ÿt.´.».. Í.ëî û}
000000F0	EB	E5	A0	F9	7D	EB	E0	98	CD	16	CD	19	66	60	80	7E	ëå ù}ëà.Í.Í.f`.~
00000100	02	00	0F	84	20	00	66	6A	00	66	50	06	53	66	68	10	...„ .fj.fP.Sfh.
00000110	00	01	00	B4	42	8A	56	40	8B	F4	CD	13	66	58	66	58	...´B.V@.ôÍ.fXfX
00000120	66	58	66	58	EB	33	66	3B	46	F8	72	03	F9	EB	2A	66	fXfXë3f;Før.ùë*f
00000130	33	D2	66	0F	B7	4E	18	66	F7	F1	FE	C2	8A	CA	66	8B	3Òf.·N.f÷ñþÂ.Êf.
00000140	D0	66	C1	EA	10	F7	76	1A	86	D6	8A	56	40	8A	E8	C0	ÐfÁê.÷v.†Ö.V@.èÀ
00000150	E4	06	0A	CC	B8	01	02	CD	13	66	61	0F	82	75	FF	81	ä..Ì¸..Í.fa.‚uÿ.
00000160	C3	00	02	66	40	49	75	94	C3	42	4F	4F	54	4D	47	52	Ã..f@Iu”ÃBOOTMGR
00000170	20	20	20	20	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	0D	0A	52	65	.....Re
000001B0	6D	6F	76	65	20	64	69	73	6B	73	20	6F	72	20	6F	74	move disks or ot
000001C0	68	65	72	20	6D	65	64	69	61	2E	FF	0D	0A	44	69	73	her media.ÿ..Dis
000001D0	6B	20	65	72	72	6F	72	FF	0D	0A	50	72	65	73	73	20	k errorÿ..Press
000001E0	61	6E	79	20	6B	65	79	20	74	6F	20	72	65	73	74	61	any key to resta
000001F0	72	74	0D	0A	00	00	00	00	00	AC	CB	D8	00	00	55	AA	rt.....¬ËØ..Uª

图 6-3 DBR 数据

第一张图片就是使用的 SD 卡截图，第二张图片显示的就是通过 WinHex 获取的 DBR 数据截图。现在我们来着重分析 DBR，具体分析如下：

（首先说明，数据的存储是以小端模式存储的）

【1】0x00~0x02：3 个字节，跳转指令。

【2】0x03~0x0A：8 个字节，文件系统标志和版本号，这里为 MSDOC5.0。

【3】0x0B~0x0C：2 个字节，每扇区字节数，512（0X02 00）。

【4】0x0D~0x0D：1 个字节，每簇扇区数，8（0x08）。

【5】0x0E~0x0F：2 个字节，保留扇区数，704（0x02 C0）。

【6】0x10~0x10：1 个字节，FAT 表个数，2。

【7】0x11~0x12：2 个字节，根目录最多可容纳的目录项数，FAT12/16 通常为 512。FAT32 不使用此处值，置 0。

【8】0x13~0x14：2 个字节，扇区总数，小于 32MB 时使用该处存放。超过 32MB 时使用偏移 0x20~0x23 字节处的 4 字节存放。此处 SD 卡容量为 2GB，所以不使用该处，置 0.

【9】0x15~0x15：1 个字节，介质描述符，0xF8 表示本地硬盘。

【10】0x16~0x17：2 个字节，每个 FAT 表的大小扇区数（FAT12/16 使用，FAT32 不使用此处，置 0）。

【11】0x18~0x19：2 个字节，每磁道扇区数，63（0x00 3F）。

【12】0x1A~0x1B：2 个字节磁头数，255（0x00 FF）。

【13】0x1C~0x1F：4 个字节，分区前已使用扇区数，137（0x00 00 00 89）。（这个数据要尤其的

重视，文件系统初始化的第一步要找的就是这玩意儿。因为我们的 SD 卡没有分区，默认就是一个分区，这个数据就是相对于 MBR 的地址偏移量，MBR 的扇区地址才是整个 SD 卡的物理扇区号为 0 的那个地址，也就是说文件系统并不是处在整个 SD 卡最开始的地方，它处在 MBR 所处的保留区之后，于是我们可以对使用 FAT32 文件系统的 SD 卡整体布局给出如下图示）

图 6-4 SD 卡整体布局

【14】0x20~0x23：4 个字节，文件系统大小扇区数，3841911（0x 00 3A 9F 77）。
【15】0x24~0x27：4 个字节，每个 FAT 表的大小扇区数，3744（0x 00 00 0E A0）。
【16】0x28~0x29：2 个字节，标记。
【17】0x2A~0x2B：2 个字节，版本号。
【18】0x2C~0x2F：4 个字节，根目录簇号，2。（虽然在 FAT32 文件系统下，根目录可以存放在数据区的任何位置，但是通常情况下还是起始于 2 号簇）
【19】0x30~0x31：2 个字节，FSINFO（文件系统信息扇区）扇区号，1。（上图的标注即用黄色条纹的标注有误，请读者注意）该扇区为操作系统提供关于空簇总数及下一可用簇的信息。
【20】0x32~0x33：2 个字节，备份引导扇区的位置，6。（上图的标注即用黄色条纹的标注有误，请读者注意）备份引导扇区总是位于文件系统的 6 号扇区。
【21】0x34~0x3F：12 个字节，未使用。
【22】0x40~0x40：1 个字节，BIOS INT 13H 设备号，0x80。（这个我也不知道什么意思 ☺）
【23】0x41~0x41：1 个字节，未用。
【24】0x42~0x42：1 个字节，扩展引导标志。0x29。
【25】0x43~0x46：1 个字节，卷序列号。通常为一个随机值。
【26】0x47~0x51：11 个字节，卷标（ASCII 码），如果建立文件系统的时候指定了卷标，会保存在此。
【27】0x52~0x59：8 个字节，文件系统格式的 ASCII 码，FAT32。
【28】0x5A~0x1FD：410 个字节，未使用。该部分没有明确的用途。
【29】0x1FE~0x1FF：签名标志“55 AA”。

6. 4 FAT表简介

6. 4. 1 FAT表的概述

位于保留区后的是 FAT 区，有两个完全相同的 FAT（File Allocation Table， 文件分配表）表组成，FAT 文件系统的名字也是因此而来。

重要说明：

1. 对于文件系统来说，FAT 表有两个重要作用：描述簇的分配状态以及标明文件或目录的下一簇的簇号。
2. 通常情况下，一个 FAT 把文件系统会有两个 FAT 表，但有时也允许只有一个 FAT 表，FAT 表的具体个数记录在引导扇区的偏移 0x10 字节处。
3. 由于 FAT 区紧跟在文件系统保留区后，所以 FAT1 在文件系统中的位置可以通过引导记录中偏移 0x0E~0x0F 字节处的“保留扇区数”得到。
4. FAT2 紧跟在 FAT1 之后，它的位置可以通过 FAT1 的位置加上 FAT 表的大小扇区数计算出来。

6. 4. 2 FAT表的特性

FAT 表由一系列大小相等的 FAT 表项组成，总的说来 FAT 表有如下特性：

1. FAT32 中每个簇的簇地址，是有 32bit（4 个字节）记录在 FAT 表中。FAT 表中的所有字节位置以 4 字节为单位进行划分，并对所有划分后的位置由 0 进行地址编号。0 号地址与 1 号地址被系统保留并存储特殊标志内容。从 2 号地址开始，每个地址对应于数据区的簇号，FAT 表中的地址编号与数据区中的簇号相同。我们称 FAT 表中的这些地址为 FAT 表项，FAT 表项中记录的值称为 FAT 表项值。

2. 当文件系统被创建，也就是进行格式化操作时，分配给 FAT 区域的空间将会被清空，在 FAT1 与 FAT2 的 0 号表项与 1 号表项写入特定值。由于创建文件系统的同时也会创建根目录，也就是为根目录分配了一个簇空间，通常为 2 号簇，所以 2 号簇所对应的 2 号 FAT 表项也会被写入一个结束标记，如下图所示：

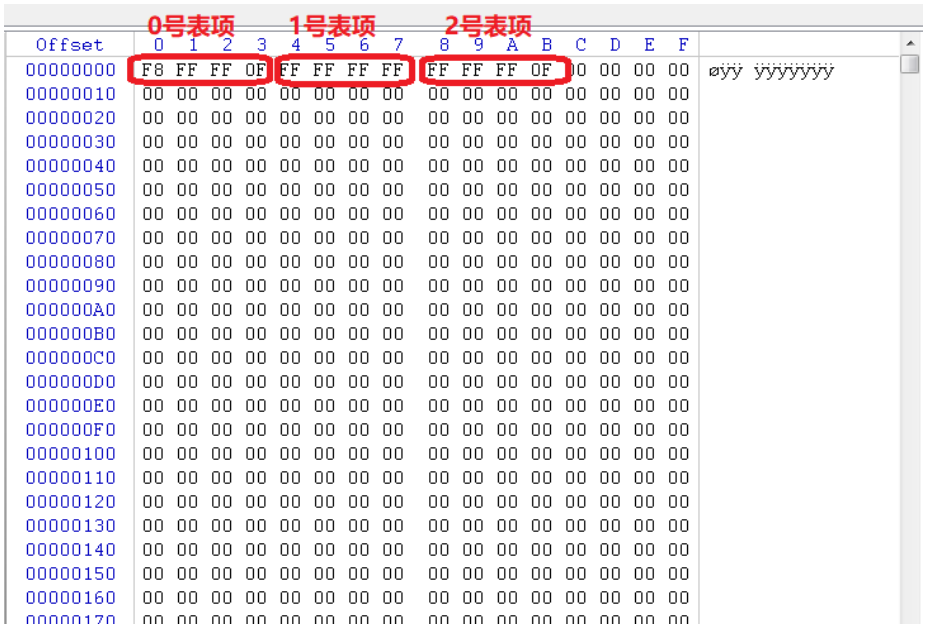

图 6-5 FAT 表项示意图

3. 如果某个簇未被分配使用，它所对应的 FAT 表项内的 FAT 表项值即用 0 进行填充，表示该 FAT 表项所对应的簇未被分配。

4. 当某个簇已被分配使用时，则它对应的 FAT 表项内的 FAT 表项值也就是该文件的下一个存储位置的簇号。如果该文件结束于该簇，则在它的 FAT 表项中记录的是一个文件结束标记，对于 FAT32 而言，代表文件结束的 FAT 表项值为 0x0FFFFFFF。

5. 如果某个簇存在坏扇区，则整个簇会用 FAT 表项值 0xFFFFFF7 标记为坏簇，不再使用，这个坏簇标记就记录在它所对应的 FAT 表项中。

6. 由于簇号起始于 2 号，所以 FAT 表项的 0 号表项与 1 号表项不与任何簇对应。FAT32 的 0 号表项值总是“F8FFFF0F”。如上图所示。

7. 1 号表项可能被用于记录脏标志，以说明文件系统没有被正常卸载或者磁盘表面存在错误。不过这个值并不重要。正常情况下 1 号表项的值为“FFFFFFFF”或“FFFFFF0F”。

8. 在文件系统中新建文件时，如果新建的文件只占用一个簇，为其分配的簇对应的 FAT 表项将会写入结束标记。如果新建的文件不只占用一个簇，则在其所占用的每个簇对应的 FAT 表项中写入为其分配的下一簇的簇号，在最后一个簇对应的 FAT 表象中写入结束标记。

9. 新建目录时，只为其分配一个簇的空间，对应的 FAT 表项中写入结束标记。当目录增大超出一个簇的大小时，将会在空闲空间中继续为其分配一个簇，并在 FAT 表中为其建立 FAT 表链以描述它所占用的簇情况。

10. 对文件或目录进行操作时，他们所对应的 FAT 表项将会被清空，设置为 0 以表示其所对应的簇处于未分配状态。

6.5 FAT目录结构

FAT 目录其实就是一个由 32-bytes 的线性表构成的“文件”。根目录是一个特殊的目录。它存在于每一个 FAT 卷中。对于 FAT12/16，根目录存储在磁盘固定的地方，它紧跟在最后一个 FAT 表后。根目录的扇区数也是固定，可以根据 DBR 中的参数计算出，对于 FAT12/16，根目录的扇区号是相对该 FAT 卷第一个扇区的偏移量。

FAT32 的根目录由簇链组成，其扇区数不定，这点和普通的文件相同，根目录的第一个扇区号存储在 DBR 中，根目录不同于其他的目录，没有日期和时间戳，也没有目录名（“/”并不是其目录名），同时根目录里没有“.”和“..”这两个目录项，根目录另一个特殊的地方在于，根目录中有一个设置了 ATTR_VOLUME_ID 位（如下表所示）的文件，这个文件在整个 FAT 卷中是唯一的。

名称	Offset（Byte）	大小（Byte）	描述
DIR_Name	0	11	短文件名
DIR_Attr	11	1	ATTR_READ_ONLG 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 ATTR_DIRECTORY 0x10 ATTR_ARCHIVE 0x20 ATTR_LONG_NAME ATTR_READ | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID 前两个属性位为保留位，在文件创建时应该把这两位设为 0，在以后的使用中不能读写和更改。
DIR_NTRes	12	1	NT 保留位
DIR_CrtTimeTeenth	13	1	文件创建时间的毫秒级时间戳，由于 DIR_CrtTime 的精度为 2 秒，所以此域的有效值在 0-199 间
DIR_CrtTime	14	2	文件创建时间
DIR_CrtData	16	2	文件创建日期
DIR_LastAccData	18	2	最后访问日期，写操作会同步更新 DIR_WrtTime
DIR_FstClusHI	20	2	该目录项簇号的高位字（FAT12/16 为 0）
DIR_WrtTime	22	2	最后写的时间，文件创建被认作写
DIR_WrtData	24	2	最后写的日期，文件创建被认作写
DIR_FstClusL0	26	2	该目录项簇号的低位字
DIR_FileSize	28	2	文件大小，由 32-Byte 双子组成

表 6-1 FAT 目录结构

DIR_Name 域实际由两部分组成：8 个字符的主文件名和 3 个字符的扩展名，两部分不够的字符用空格（0x20）填充。此处特别注意 DIR_Name 的第一个字节（DIR_NAME[0]）。

- 如果 DIR_NAME[0] == 0xE5，则此目录为空，目录项不包含文件和目录。
- 如果 DIR_NAME[0] == 0x00，则此目录为空（同 0xE5），并且此后的不在分配有目录项（此

后所有的 DIR_Name[0]均为 0）。

- 如果 DIR_NAME[0] == 0x05，则此文件名在给位的实际值为 0xE5，0xE5 为日文合法字符
- 0x22、0x2A~0x2C、0x2E、0x2F、0x3A~0x3F、0x5B~0x5D、0x7C 为非法字符，禁止出现在 DIR_Name 的任意位置。

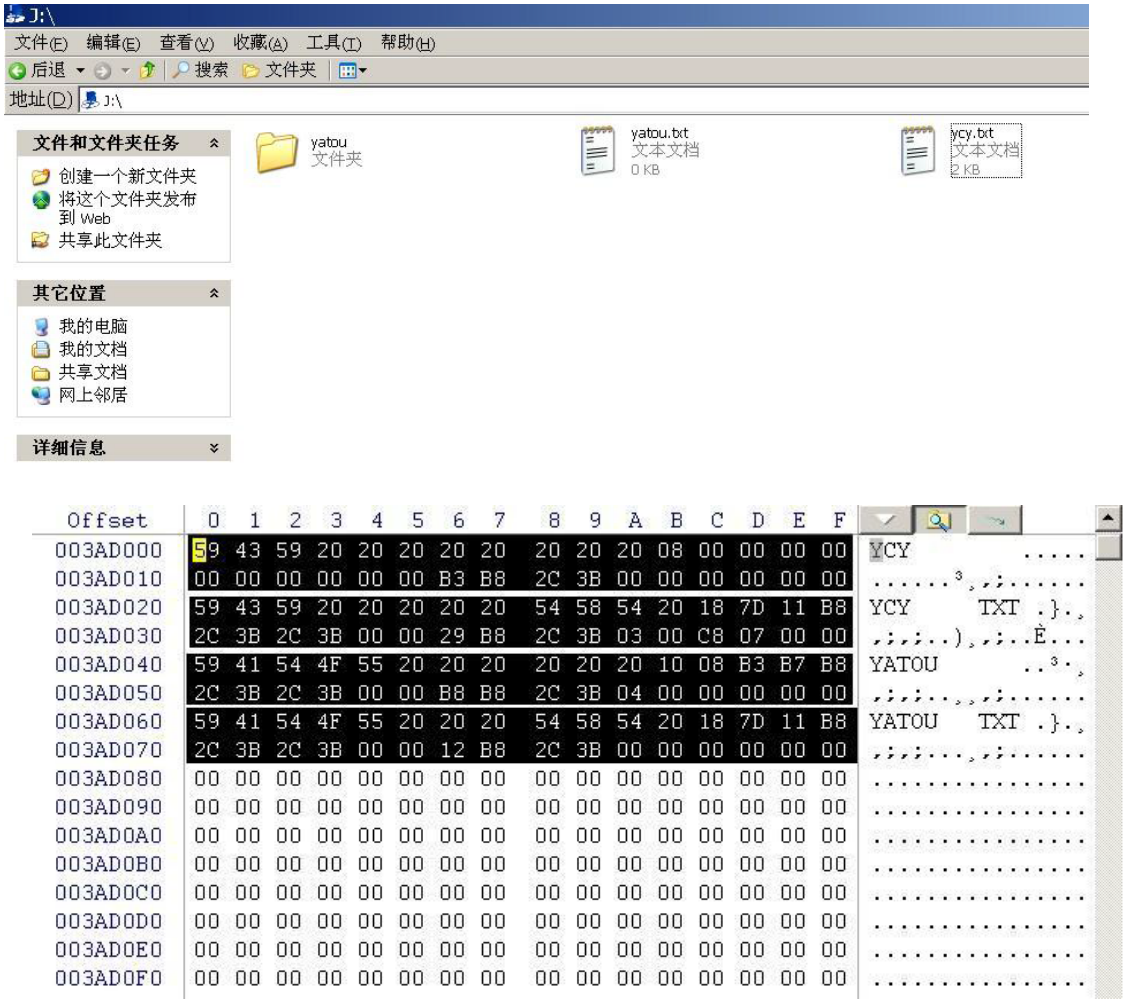

图 6-6 FAT 表项示意图