To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

RENESAS

应用笔记

自编程实例

适用于 **NEC 78K0/Kx2** 微控制器

文档编号. U18291CA1V0AN00

本文档信息于 2006 年 3 月开始使用。将来可能未经预先通知而更改。在实际进行生产设计时，请参阅各产品最新的数据表或数据手册等相关资料以获取本公司产品的最新规格。并非所有的产品和/或型号都向每个国家供应。请向本公司销售代表查询产品供应及其他信息。

未经本公司事先书面许可，禁止复制或转载本文件中的内容。本文件所登载内容的错误，本公司概不负责。

本公司对于因使用本文件中列明的本公司产品而引起的，对第三者的专利、版权以及其它知识产权的侵权行为概不负责。本文件登载的内容不应视为本公司对本公司或其他人所有的专利、版权以及其它知识产权作出任何明示或默示的许可及授权。

本文件中的电路、软件以及相关信息仅用以说明半导体产品的运作和应用实例。用户如在设备设计中应用本文件中的电路、软件以及相关信息，应自行负责。对于用户或其他人因使用了上述电路、软件以及相关信息而引起的任何损失，本公司概不负责。

虽然本公司致力于提高半导体产品的质量及可靠性，但用户应同意并知晓，我们仍然无法完全消除出现产品缺陷的可能。为了最大限度地减少因本公司半导体产品故障而引起的对人身、财产造成损害（包括死亡）的危险，用户务必在其设计中采用必要的安全措施，如冗余度、防火和防故障等安全设计。

本公司产品质量分为：

“标准等级”、“专业等级”以及“特殊等级”三种质量等级。

“特殊等级”仅适用于为特定用途而根据用户指定的质量保证程序所开发的日电电子产品。另外，各种日电电子产品的推荐用途取决于其质量等级，详见如下。用户在选用本公司的产品时，请事先确认产品的质量等级。

“标准等级”： 计算机，办公自动化设备，通信设备，测试和测量设备，音频・视频设备，家电，加工机械以及产业用机器人。

“专业等级”： 运输设备（汽车、火车、船舶等），交通用信号控制设备，防灾装置，防止犯罪装置，各种安全装置以及医疗设备（不包括专门为维持生命而设计的设备）。

“特殊等级： 航空器械，宇航设备，海底中继设备，原子能控制系统，为了维持生命的医疗设备、用于维持生命的装置或系统等。

除在本公司半导体产品的数据表或数据手册等资料中另有特别规定以外，本公司半导体产品的质量等级均为“标准等级”。如果用户希望在本公司设计意图以外使用本公司半导体产品，务必事先与本公司销售代表联系以确认本公司是否同意为该项应用提供支持。

注：

1. 本声明中的“本公司”是指日本电气电子株式会社（NEC Electronics Corporation ）及其控股公司。
2. 本声明中的“本公司产品”是指所有由日本电气电子株式会社或为日本电气电子株式会社（定义如上）开发或制造的产品。

M8E 02.10

版本历史

日期	版本	章节	描述
03-2006	—	—	首次发布

目录

1. 绪论

78K0/Kx2 系列微控制器(MCU)可以对它们内部的 flash 进行编程。该应用笔记提供了以μP78F0537 微控制器为目标的 flash 自编程能力的概述。

为了使开发具有自编程能力的应用更为方便，NEC 提供了能实现所有必需功能的示例库。你可以获得该自编程示例库二进制格式代码，在该应用笔记的附录里也提供了源代码可供参考。该示例代码包括 bootloader 程序以及自编程应用例子。

更多关于 78K0/Kx2 自编程信息请参照 *78K0/Kx2 Flash 自编程用户手册* (文档编号 U17516EJxV0UM).

1.1 概述

为了对 flash 存储器编程，78K0/Kx2 MCU 需要执行位于隐藏 ROM 中的代码。可以调用这些隐藏的函数来检查没有使用的区域（空白检查），对 MCU 内任何可用的存储单元擦除或写代码，为了实现该功能，应用程序中必需包含能够访问这些隐藏固件的代码，因此为了能够自编程必需预先对 MCU 的 flash 写入这些代码。

1.2 自编程特点

- 自编程功能利用了 NEC 的单电压 flash 存储器技术，仅仅需要 MCU 的 Vcc 电源。
- 可以对 MCU 内所有 flash 存储器进行自编程。
- 可以采用 MCU 的任何接口方式输入数据，只要通过内部 RAM 即可。
- 最小的编程单位为 4 字节。
- 一次最大编程单位为 256 字节。
- 最小擦除或空白检查的单位是 1flash 存储器块 (1 KB).
- MCU 提供两个 4-KB 簇拥来存储 bootloader 程序，内置的启动交换功能可以自己更新 bootloader。
- 在执行自编程函数时可以响应中断。
- 软性和硬件条件监控防止对存储器的误操作。
- 当执行隐藏 ROM 内的函数时，CPU 使用内部 8-MHz 振荡器。

1.3 需求

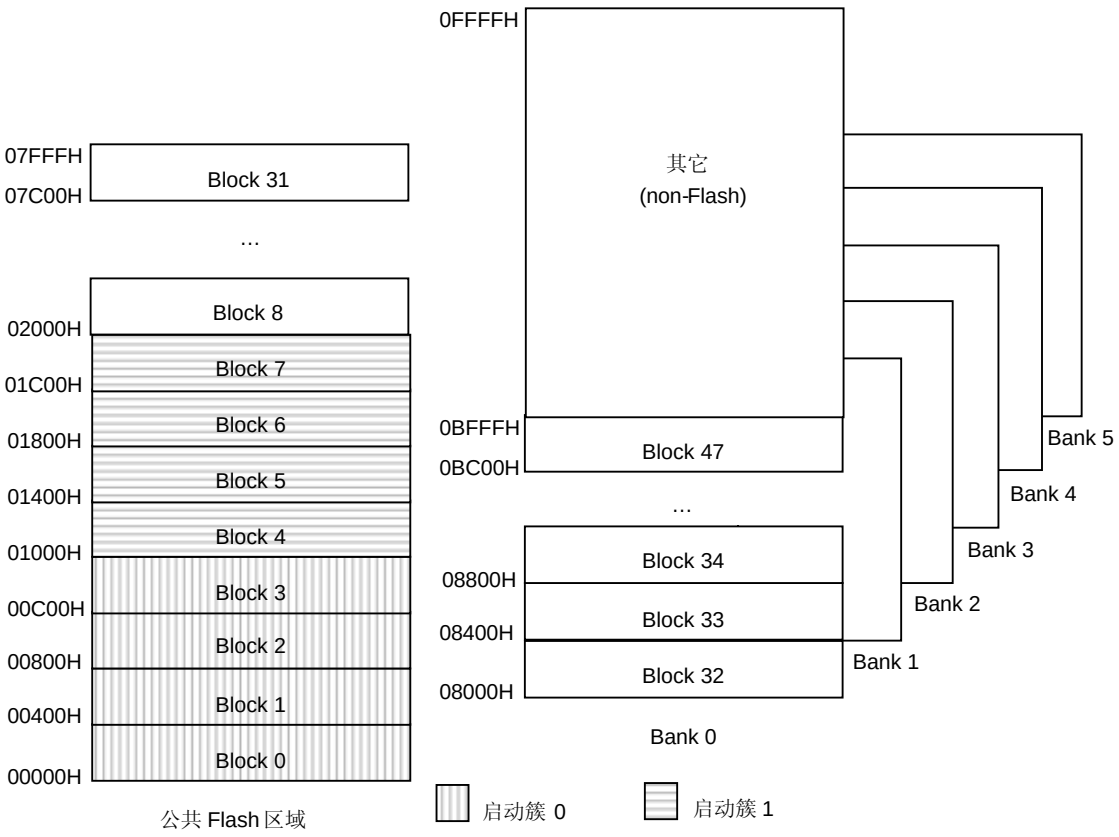
- 使用通用寄存器组 3
- 使用 100 字节 RAM (入口 RAM) 作为隐藏 ROM 中函数的工作区。
- 使用引脚 FLMD0 用来设置 MCU 工作在自编程模式。
- 4 至 256 字节 RAM 作为数据缓冲区。
- 最大 39 字节 RAM 作为隐藏 ROM 函数的堆栈。
- 隐藏 ROM 中的函数被 0000H 至 7FFFH 中的应用程序调用。

2. 微控制器 Flash 体系结构

2.1 Flash 块

如下图所示, μPD78F0537 MCU 的 flash 存储器按 1-KB 分块（block）。这是空白检查、擦除和校验的最小单位。

图 1.　　μPD78F0537 的 Flash 存储器结构



2.2 启动簇

Flash 存储器的前 8 块组成了两个 4-KB 大小的启动簇 (启动簇 0 和启动簇 1)。这两个簇一起完成 MCU 的启动交换功能，允许 bootloader 自我更新。

2.3 Flash 字

4 字节的字是能够写到数据缓冲区的最小单位，最大是 64 个字（256 字节）。

2.4 Flash 页

78K0/Kx2 系列产品的 flash 存储器的容量范围从 8 至 128KB，flash 容量大于 60KB 的产品采用页结构，小于 60KB 的没有页。

有些自编程操作，比如写、擦除或者空白检查，必须同时指定块号和页号。

2.5 启动交换功能

78K0/Kx2 自编程支持启动交换功能，在即使在电气噪声或者掉电的情况下也可以替换缺省的启动区域的函数（启动簇 0）。在自编程的典型应用中，启动簇 0 中包含了执行自编程所有必要操作的 bootloader 代码。为了更新在自编程中使用的 bootloader，先要将新的 bootloader 写入启动簇 1 中，然后使用启动交换功能使 CPU 从新的代码启动。

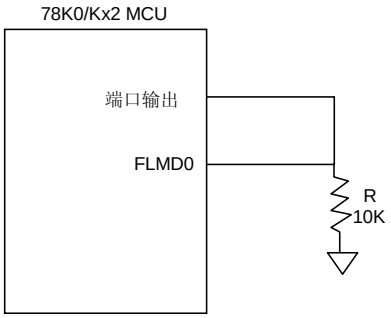
2.6 中断处理

MCU 的中断逻辑可以中断写或擦除等自编程操作。与正常处理方式不同的是中断响应时间还包含响应中断前为了完成自编程处理的延迟时间。通过自编程函数的返回值，可以确定是否再次执行被中断打断德操作。

2.7 硬件需求 (FLMD0 引脚)

Flash 自编程功能使用单电压操作，仅仅需要 MCU 的 V_{CC} 电源。应用程序控制 FLMD0 引脚 (用来设置 MCU 进入自编程模式) 作为 flash 自编程处理的一部分。引脚 FLMD0 通常情况下为低，当编程时必须拉高。如下图所示，可以将 FMD0 接到一个输出口上，并通过一个电阻接地。采用该方式，可以通过设置或清除输出引脚将 FLMD0 设为高电平或低电平。

图 2.　　FLMD0 连接



3. 自编程示例库

表 1. NEC 自编程示例库程序

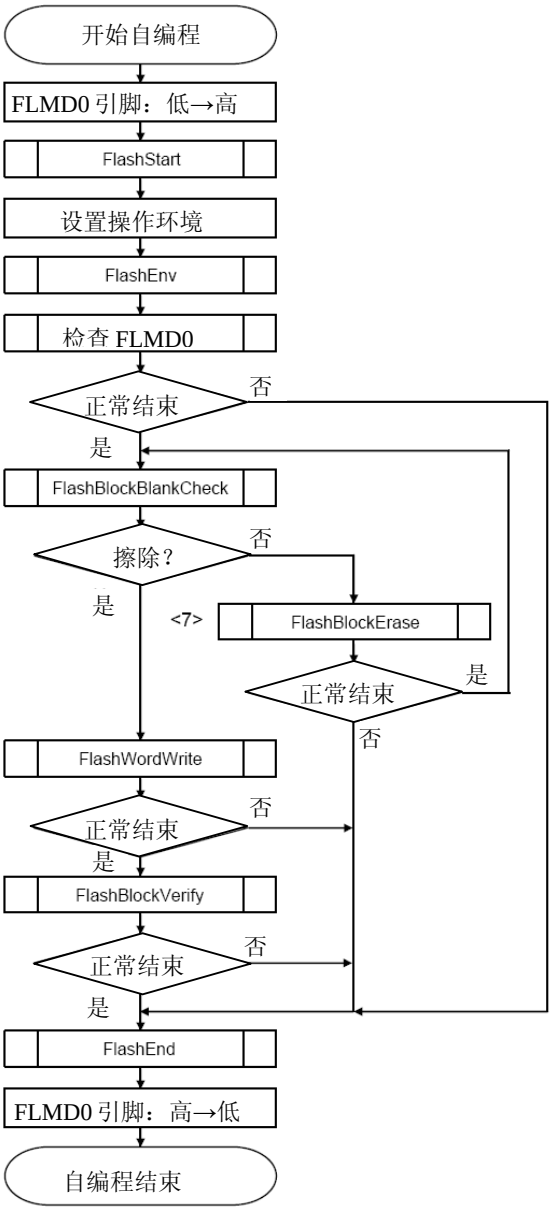
库名称	调用举例(C 语言) 调用举例 (汇编语言)	注
自编程开始库	FlashStart(); CALL !_FlashStart	1
初始化库	FlashEnv(&EntryRAM[0]); CALL !_FlashEnv	2
模式检查库	Status = CheckFLMD( ); CALL !_CheckFLMD	3
空白块检查库	Status = FlashBlockBlankCheck(BlankCheckBANK,BlankCheckBlock); CALL !_FlashBlockBlankCheck	4
块擦除库	Status = FlashBlockErase(EraseBANK, EraseBlock); CALL !_FlashBlockErase	5
写字库	Status = FlashWordWrite(&WordAddr, WordNumber,&DataBuffer); CALL !_FlashWordWrite	6
块校验库	Status = FlashBlockVerfy(VerifyBANK, VerifyBlock); CALL !_FlashBlockVerify	7
自编程结束库	FlashEnd( ); CALL !_FlashEnd	8
获取信息库	Status = FlashGetInfo(&GetInfo, &DataBuffer); CALL !_FlashGetInfo	9
设置信息库	Status = FlashSetInfo(SetInfoData) CALL !_FlashSetInfo	10
写 EEPROM 库	Status = FlashEEPROMWrite(&WordAdder,WordNumber, &DataBuffer); CALL !_EEPROMWrite	11

注：

1. 声明自编程开始
2. I 初始化入口 RAM
3. 检查 FLMD0 电压电平
4. 检查指定块是否擦除(1KB)
5. 擦除指定块(1KB)
6. 向指定地址写 1- 64 字数据
7. 校验指定块(1KB) (内部校验)
8. 声明自编程结束
9. 读取 flash 信息
10. 改变 flash 设置信息
11. 向指定地址写 1- to 64 字数据 (模拟 EEPROM 模式)

3.1 使用示例库的自编程流程

图 3. 自编程流程



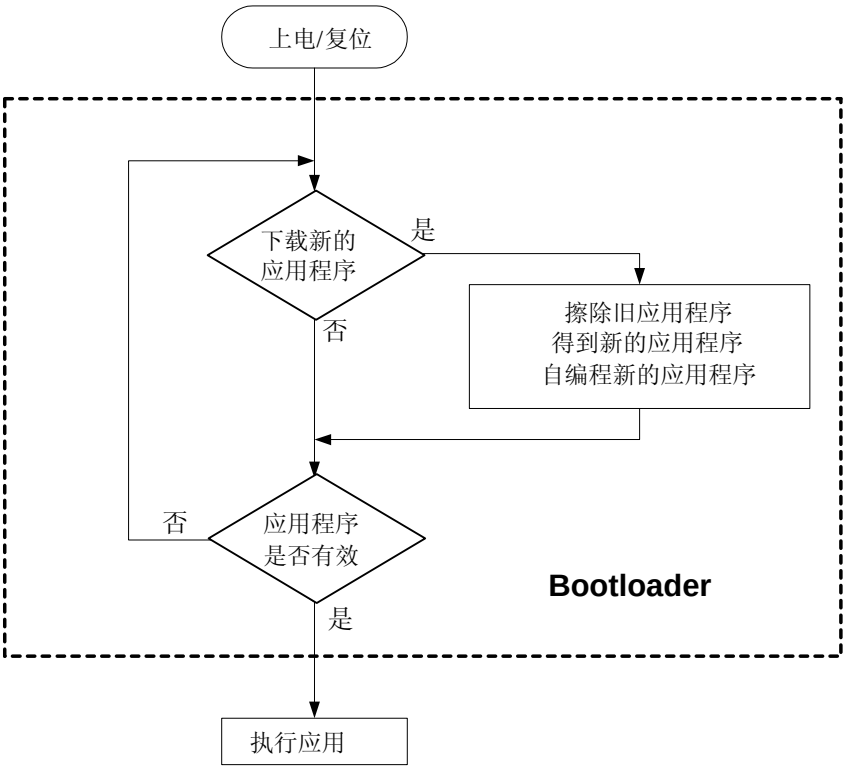
4. 自编程举例

该节描述 NEC 电子美国为演示 78K0/Kx2 自编程主要性能而开发的程序。该例程使用了 μPD78F0537 微控制器并包含了允许通过 PC 主机下载应用例程更新 flash 存储器内容的 bootloader 程序。

4.1 Bootloader 介绍

Boot 代码由 MCU 启动时执行的指令组成。这里的 loader 指向 MCU 的 flash 中写入新的应用程序。下面就是使用自编程写 MCU 中 flash 的的简单 bootloader 流程图。

图 4.　　基本 Bootloader



bootloader 允许不使用外部编程器来更新或替代应用程序码，并使通过电话线或 Internet 远程更新成为可能。

例如，如果有 5000 个基于 MCU 的投币式电话需要固件更新，电话公司的服务人员可以手动的对每一个电话重新编程——消耗大量时间，或者使用 bootloader 从控制中心远程对 5000 个电话重新编程。

4.2 Bootloader 元素

一个简单的 bootloader 包含以下 5 个基本元素：

- 一个启动 bootloader 程序的信号
- 一个执行 bootloader 的信号
- 将新的代码传送给 MCU
- Flash 新代码的自编程
- 将控制权转移给有效的应用程序。

4.2.1 启动信号

如果在 Internet 上连接有上百台自动售货机需要更新硬件，你需要产生一个信号触发这些 MCU 开始 bootloader 程序。该信号可以是中断，通过串口传送的一个命令字节，或者其它别的能使程序复位执行 bootloader 代码的信号。

4.2.2 执行信号

启动时，MCU 是装载新的应用程序还是执行已经存在的程序取决于外部的信号。 该信号可以是上电时的一个端口信号，用来控制 MCU 装载新程序还是执行旧程序。 也可以基于从 UART 接收到的字符或者从 A/D 转换器读到的信号。

4.2.3 传送代码

可以通过 RS-232、I^2C、串口或并口传送编程数据。因为要传送的代码一般会超过 MCU 的 RAM 容量，因此需要一些控制数据流的措施。对于 RS-232 串口可以使用较低的波特率给出 MCU 处理数据和自编程的时间而不溢出。或者使用硬件握手信号清除发送 (CTS)和请求发送 (RTS) 线来控制数据流。另外一个方法是使用 XON/XOFF 软件握手协议。

你可以自己选择代码的格式，但必须地址信息还有为错误处理的校验和。大多数应用采用标准格式，比如 Intel hex 格式。

4.2.4 Flash 自编程

每次 MCU 接收到一批新的数据，就要将它们编程到正确的 flash 地址。如果该地址非空白，MCU 在编程前必须先擦除。一般在编程中或者编程后还需要检查存储器的内容。

4.2.5 将控制权传给有效的应用程序

在接收和编程了新的代码，bootloader 写一个校验和或者其它唯一字节序列到一个固定的存储单元。Bootloader 检测该值，如果该值存在，bootloader 就将控制权传给应用程序。

4.3 Bootloader 配置和操作

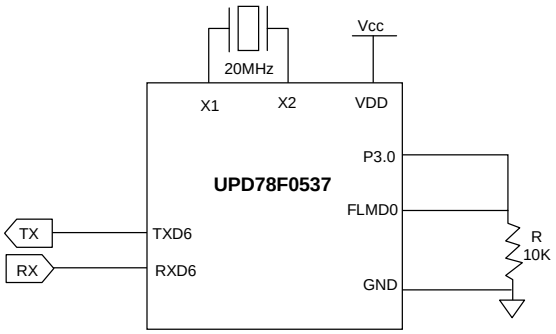
该节通过一个例程描述了当使用 bootloader 装载新的应用代码时的配置和操作。

4.3.1 演示平台

用来演示 bootloader 的硬件平台包括：

- µPD78F0537 MCU
- 20-MHz 时钟
- FLMD0 引脚通 10KΩ 电阻接地
- P3.0 引脚为输出，控制 FLMD0 引脚
- UART6 串行通信

图 5. 演示硬件



4.3.2 Bootloader 通信

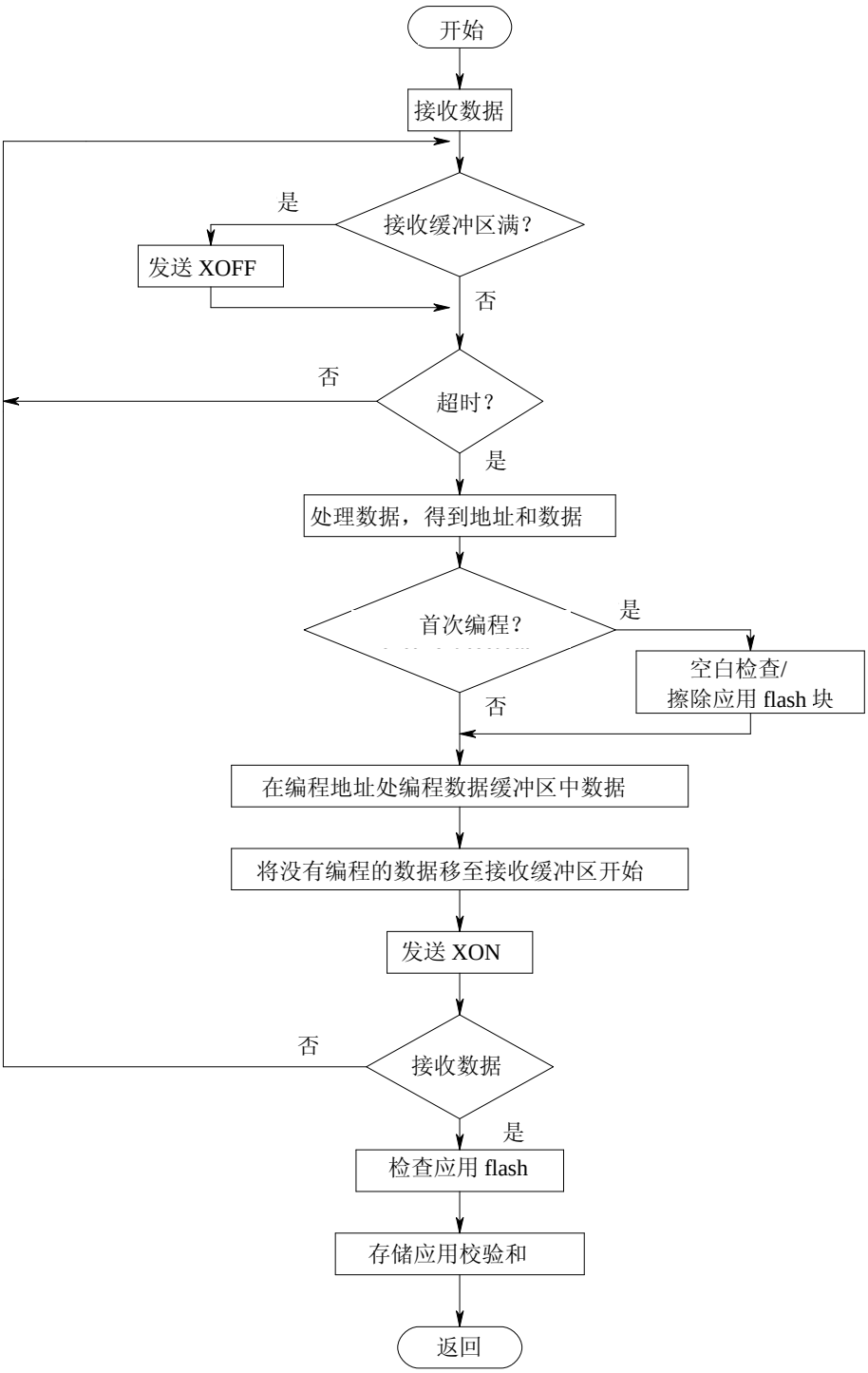
MCU 使用 UART6 发送和接收串行数据，配置如下：

- 115200 bps
- 8 位数据
- 无校验
- 1 位停止位
- XON/XOFF 流控制

4.3.3 Bootloader 操作

下图给出了使用 XON/XOFF 数据流控制协议下载自编程地址和数据的应用代码中**擦除、编程、校验**命令的流程。

图 6. Bootloader 流程



4.3.4 装载应用程序的启动提示符

当 MCU 启动时， 不管是上电还是复位，设备通过另外终端显示下列提示信息。

```
NEC Electronics America Bootloader Ver. 1
Load Y/N?
```

如果输入 **Y**，设备提示下载新的引用文件。设备认为该文件为标准的 Inter 16 进制文件（细节见附录 C）， 如果输入 **N**，或者几秒内没有输入，MCU 检测有效的应用程序执行。

4.3.5 装载应用程序

如果启动的时候输入 **Y** ，bootloader 会提示发送文件：

```
Send File
```

可以使用终端中的 **Send** 或者 **Transfer File** 命令发送文件。

4.3.6 接收数据

bootloader 将输入的数据载入到接收缓冲区中。当缓冲区满时，bootloader 会发送一个 XOFF 字符暂停数据流继续处理剩余的数据直到再也接收不到数据。超时后 MCU 从接收循环中退出开始处理数据。值得注意的是在接收缓冲区满以前接收数据就停止的情况下也有相同的超时机制。

4.3.7 处理接收数据

Bootloader 检查接收缓冲区中的数据并执行以下任务：

- 检查每一行的检验和是否正确
- 检查行地址是否中断（不连续的地址）
- 设置编程地址
- 从待编程的数据中提取数据到数据缓冲区中
- 编程数据
- 将没有编程的数据移至接收缓冲区的开始
- 检查是否文件的结尾
- 在返回接收循环前发送 XON

Bootloader 处理接收缓冲区并存储开始编程地址。Bootloader 先将连续地址的数据存到数据缓冲区中。当处理完接收缓冲区中的所有数据或者遇到一个地址中断时会对数据缓冲区中的数据进行编程。 接着将所有没有编程的数据移至接收缓冲区的开始。然后发送一个 XON 字符并回到原先的接收循环中。 这种 XON/XOFF 接收/编程时序继续直到 bootloader 检测到文件的结尾。

注意事项: NEC 提供的该示例 bootloader 仅能处理 Intel Hex 16 进制文件中的‘00’ (数据) 格式。有关 Intel Hex 16 进制文件格式的细节请参照附录 C。其它记录格式将导致误操作。

4.3.8 空白检查和擦除

当处理完输入文件的第一批数据后，bootloader 在把应用代码写入 flash 前要执行**空白检查/擦除**命令。如果此时在该过程中出现了错误引起 bootloader 复位，应用 flash 仍然有效。Bootloader 首先检查 flash 是否空白，且仅仅擦除非空白的块。如果该块被擦除了，bootloader 以 16 进制显示擦除的块：

```
Erasing 02
Erasing 03
Erasing 1D
```

4.3.9 编程

Bootloader 通过指定地址和要编程的字数对缓冲区的数据进行编程。Bootloader 每次编程都会输出开始地址以便可以监控进展，如下：

```
Programming at ..
101C
102C
112C
Etc.
```

4.3.10 校验

接收和编程整个文件无误后，bootloader 校验 flash 存储器并输出信息：

```
Verifying..
```

4.3.11 存储有效应用程序校验和

应用代码成功校验后 bootloader 在 flash 存储器的末尾存储一个 4 字节的校验和。该校验和是整个 flash 的校验和，从 FIRST_FLASH_BLOCK_C 的开始至 LAST_FLASH_BLOCK_B 的结束, 但不包括存储校验和的 4 个字节。因为μPD78F0537 微控制器有 128KB flash 存储器，第五页的最后一块（47）存储校验和：

```
#define FIRST_FLASH_BLOCK_C      4            // 第一个应用块
#define LAST_FLASH_BLOCK_B       47           // 最后一个应用块
```

以正常模式编程校验和后，bootloader 检查该块：

```
Checksum 00DC8195 at..
BFFC
Verifying..
```

注意在该例中，校验和是通过把从 1000H 到 7FFFH (公共区域), 第 0 页至第 4 页从 8000H 到 BFFFH 以及第 5 页的从 8000H 到 BFFB 加起来计算得来的。Bootloader 将 4 字节的校验和存在第 5 页 47 块的最后 4 个单元中 (BFFCH, BFFDH, BFFEH, BFFFH)。

4.3.12 执行应用程序

MCU 成功装载新的代码后，立即执行新代码。不管该代码是刚刚下载还是上电或者复位，bootloader 在执行应用程序前总是先检查应用程序的校验和是否有效。Bootloader 计算 flash 应用程序存储器的校验和并把它和存储的检验和比较，然后以下面的格式输出信息：

```
Stored checksum    = 00DC8195
Calculated checksum = 00DC8195
```

如果两值相等，MCU 执行代码，否则程序复位提示你下载新的应用程序。有效程序检查可以防止执行非法代码。例如当使用 bootloader 对 MCU 编程但还没有收到应用代码这种情况就可能发生。这种情况还可能出现在由于噪声或者掉电引起的 MCU 复位，或者当下载过程这出现这种情况。

4.3.13 Flash 自编程错误

如果 bootloader 遇到错误，它将输出下列信息：

```
SFP: Function = 04   Return = 05
```

然后 bootloader 进入死循环允许看门狗时间溢出强制复位。

4.3.14 行校验和错误

bootloader 处理接收缓冲区中的字符时，要检查每一行的校验和。如果发现错误的校验和，bootloader 会输出下面的信息：

```
Line checksum!
```

然后 bootloader 进入死循环允许看门狗时间溢出强制复位。

4.3.15 启动区域改写错误

装入新应用程序而对接收数据编程之前，bootloader 检查每一个地址确认在启动簇 0(0000H–0FFFH)之外。在该区域编程将破坏 bootloader 程序。如果地址在在该范围内，bootloader 将输出以下信息：

```
Boot Area!
```

然后 bootloader 进入死循环允许看门狗时间溢出强制复位。

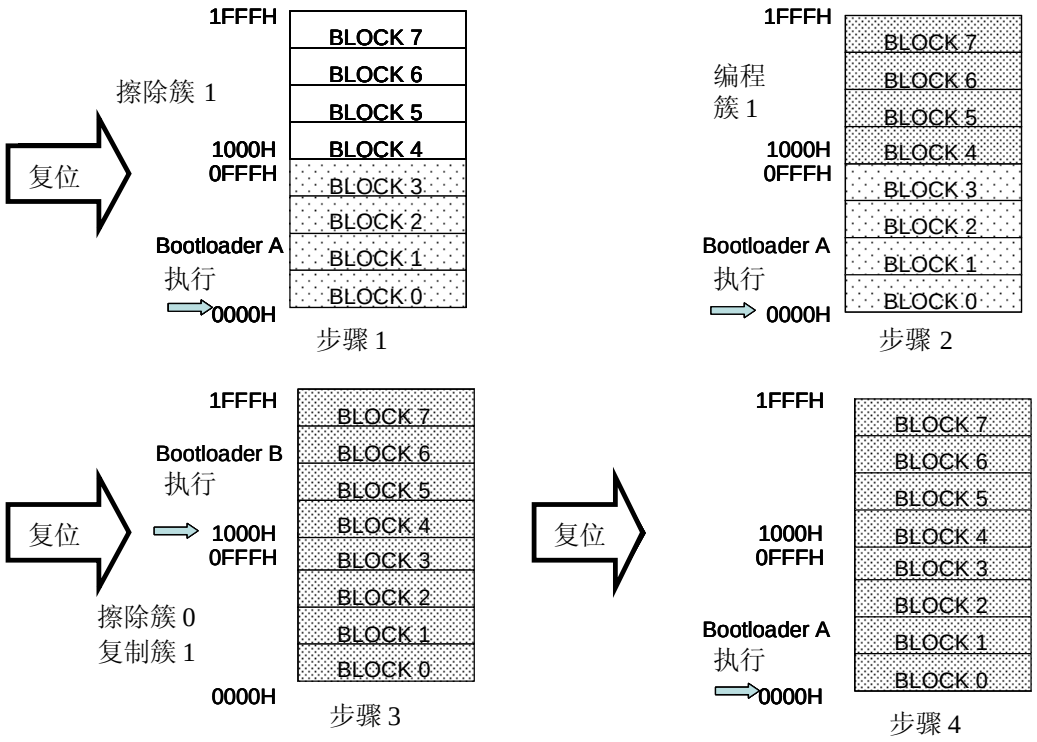
4.3.16 引导交换

替换微控制器中的 bootloader 需要使用一个叫引导交换的程序。该程序涉及两个 bootloader——一个已经在 MCU 的启动簇内，还有一个新的需要存在启动簇内。在下面的启动交换步骤中分别称这两个为 bootloader A 和 bootloader B。

- Bootloader A 检查发送文件的首地址，如果该地址是复位地址 0000H，bootloader 就知道该文件中包含新的 bootloader 代码而非应用程序， Bootloader A 擦除启动簇 1 (块 4 到 7).
- 尽管 Bootloader B (新的 bootloader) 地址在 0000H–0FFFH 的范围(启动簇 0)，Bootloader A 必须把新代码存到 1000H–1FFFH 的范围 (启动簇 1)。 Bootloader A 在编程前对所有的地址加上一个偏移量 1000H。
- 对区域 1000H-1FFFH 编程后，Bootloader A 提示你替换 bootloader 代码。如果决定替换 Bootloader A，将设置启动交换标志并通过看门狗时间溢出强制 MCU 复位。
- 复位时 MCU 看到设置了启动交换标志便从启动簇 1 开始执行。
- Bootloader B 检查启动交换标志。如果该标志被设置，Bootloader B 擦除启动簇 0 (Bootloader A)并将自身复制到该区域。
- 将启动簇 1 复制到启动簇 0 后，Bootloader B 清除启动交换标志并允许看门狗溢出再次强制复位。
- 在此次新的复位时，MCU 看到清除了启动交换标志，便从启动簇 0 (Bootloader B)开始执行。

- 根据启动交换标志，程序从 0000H 或者 1000H 启动。
- 该例子中不包含任何中断服务程序。

图 7.　　启动交换步骤

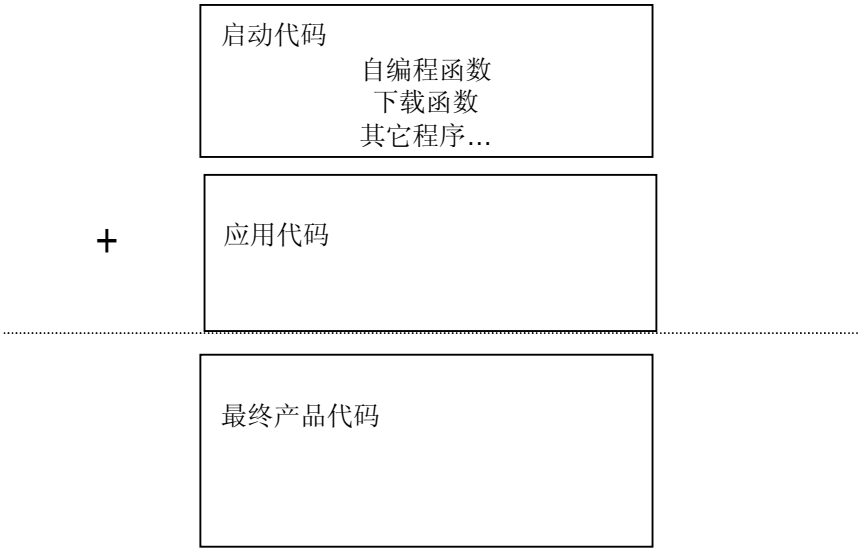


5. 使用 NEC 的工具开发自编程的代码

5.1 代码结构

为了使用 flash 自编程功能，需要将你的代码分为两部分：启动区域和应用区域。这种划分允许更新应用代码而不用破坏 bootloader 程序。另外，也可以通过 MCU 的启动交换功能实现 bootloader 自我更新。

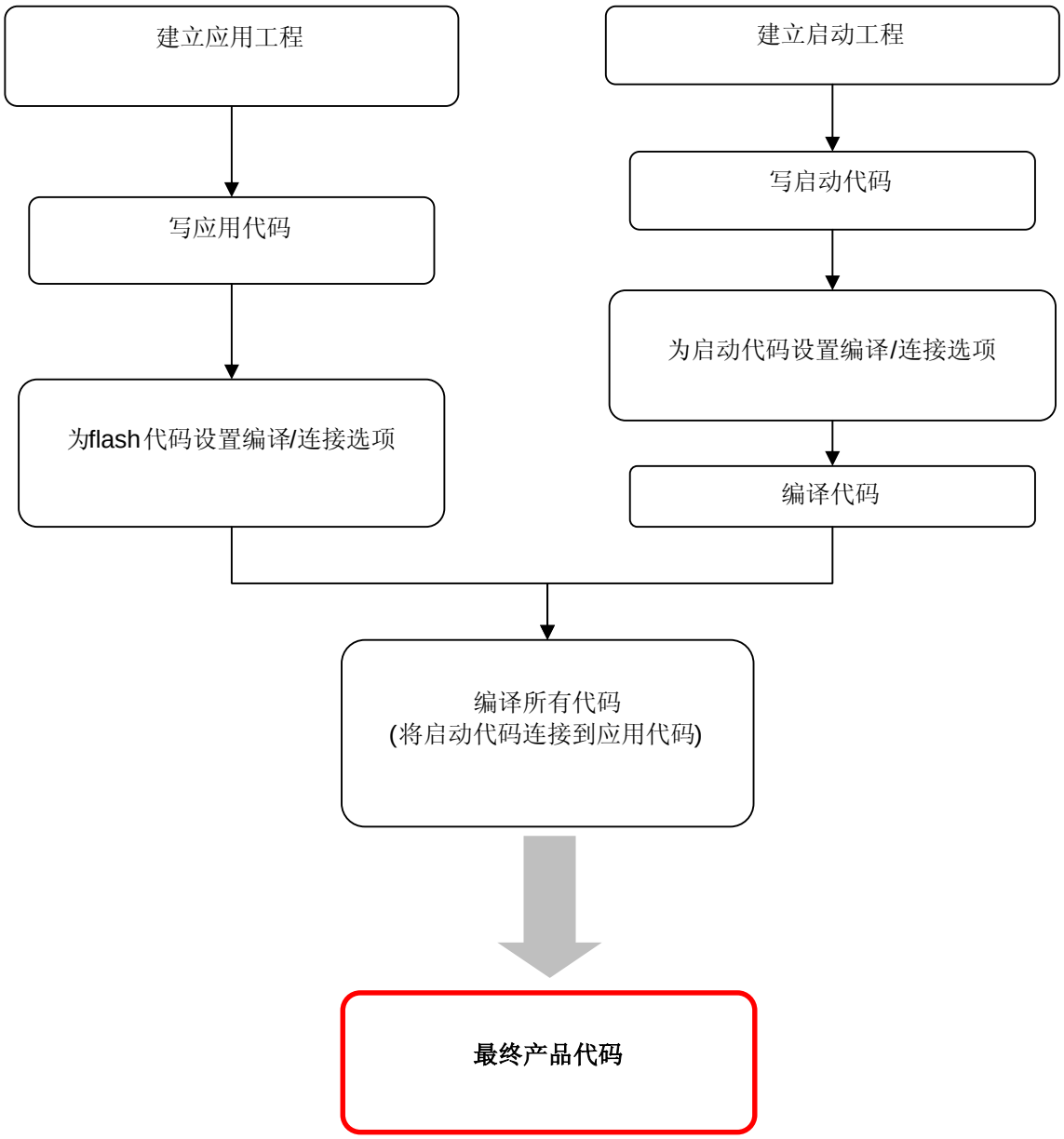
图 8. 自编程代码结构



5.2 开发流程

使用 NEC 工具开发具有自编程功能的代码典型开发流程如下：

图 9.　　代码开发流程



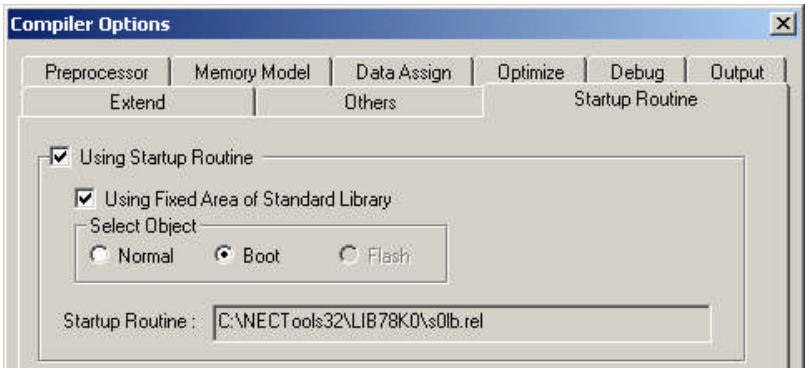
5.3 工具设置

NEC 提供了开发具有自编程功能代码所需要的一整套工具。 利用 NEC 的集成开发环境（IDE）PM+，你可以很容易使用编译器、连接器产生自编程代码以及设置目标转换选项。下面就是使用 NEC 开发工具默认选项来产生启动代码和应用代码的最少的步骤。有关此的详细步骤请参照 CC78K0 和 RA78K0 的语言和操作手册。

5.3.1 产生启动代码的步骤

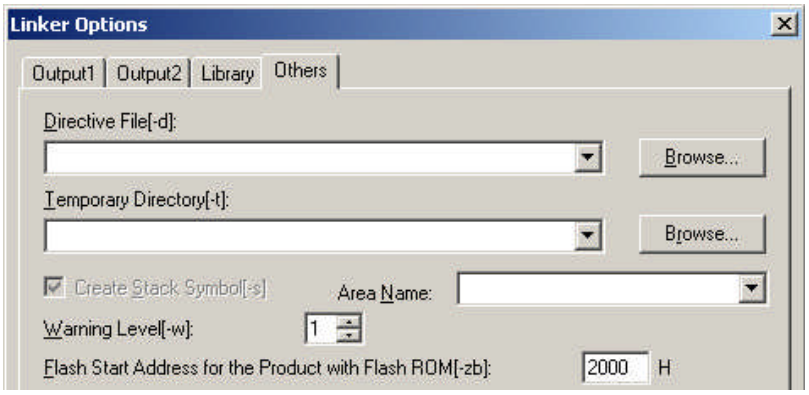
1. 将 Main()函数命名为 **boot_main()**.
2. 在 C 编译选项中为 **Boot** 指定为 **C Startup Routine**。

图 10. 为启动代码选择编译选项



3. 指定应用代码的起始地址(缺省值为 2000H).

图 11. 为启动代码设置连接选项



注：

- 如果应用程序代码开始地址不是 2000H，必须重新编译 NEC 的 run-time 库。更多细节请参照 *CC78K0 语言用户手册*。
- NEC 美国开发的应用程序代码的起始地址为 1000H。

5.3.2 产生应用代码步骤(Flash)

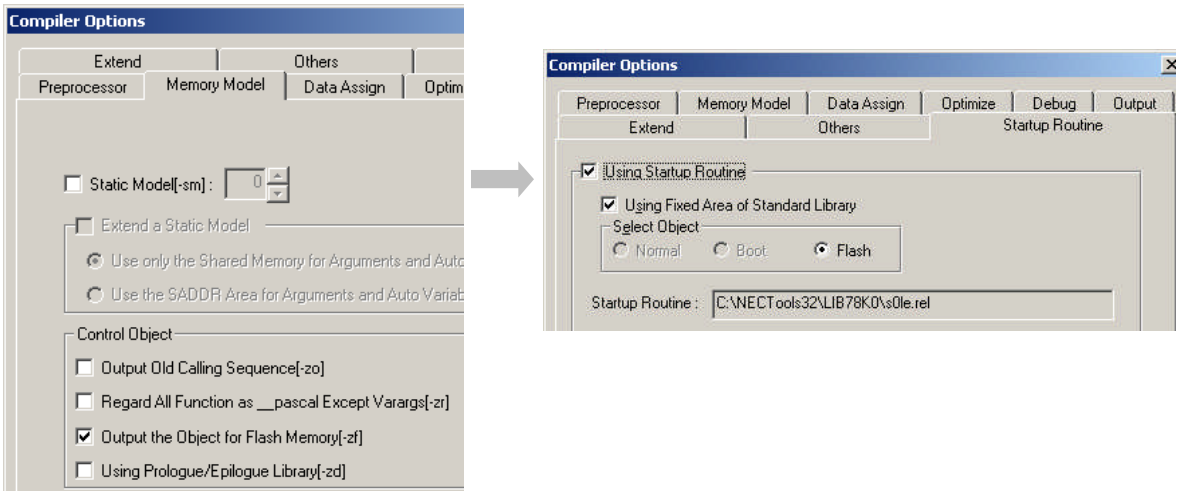
1. 指定(在代码中 code) 应用程序代码首地址。在 C 语言源程序的最开始写下面一句话：

   ```
   #pragma ext_table 0x2000          //使用为首地址 2000H
   ```

   注意该指令定义了应用程序的首地址，用于启动程序和中断函数。

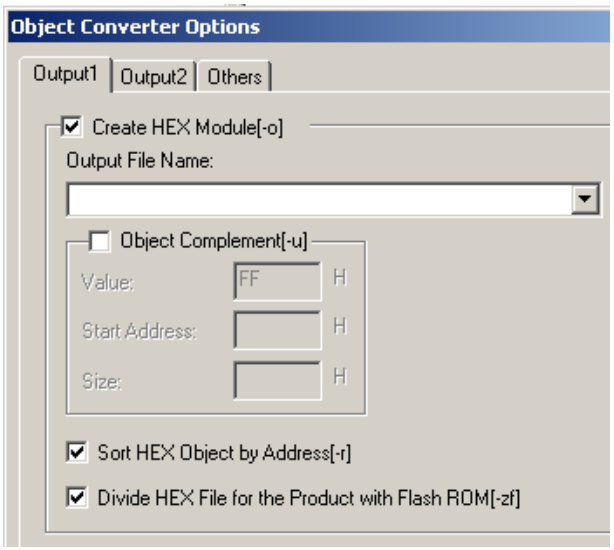
2. 选择设置 PM+为 flash 存储器输出目标代码 (应用代码)。该设置将为 flash 代码自动选择 C 启动程序，如下图所示：

图 12.　　应用代码编译选项



3. 在**目标转换**选项中,，指定工具产生两个分开的 HEX 文件。一个是启动代码，另一个是应用代码（flash）。

图 13.　　为应用程序选择目标转换选项



文件扩展名分别是：

- *.HXB (启动代码 hex 文件)
- *.HXF (应用代码/flash hex 文件)

注意*.HXB 应该和从启动代码编译产生的*.HEX 文件相匹配。

6.示例代码

该演示程序由下表中文件组成的软件模块组成。下表表明了哪些文件是由 Applilet 产生的，哪些需要修改。

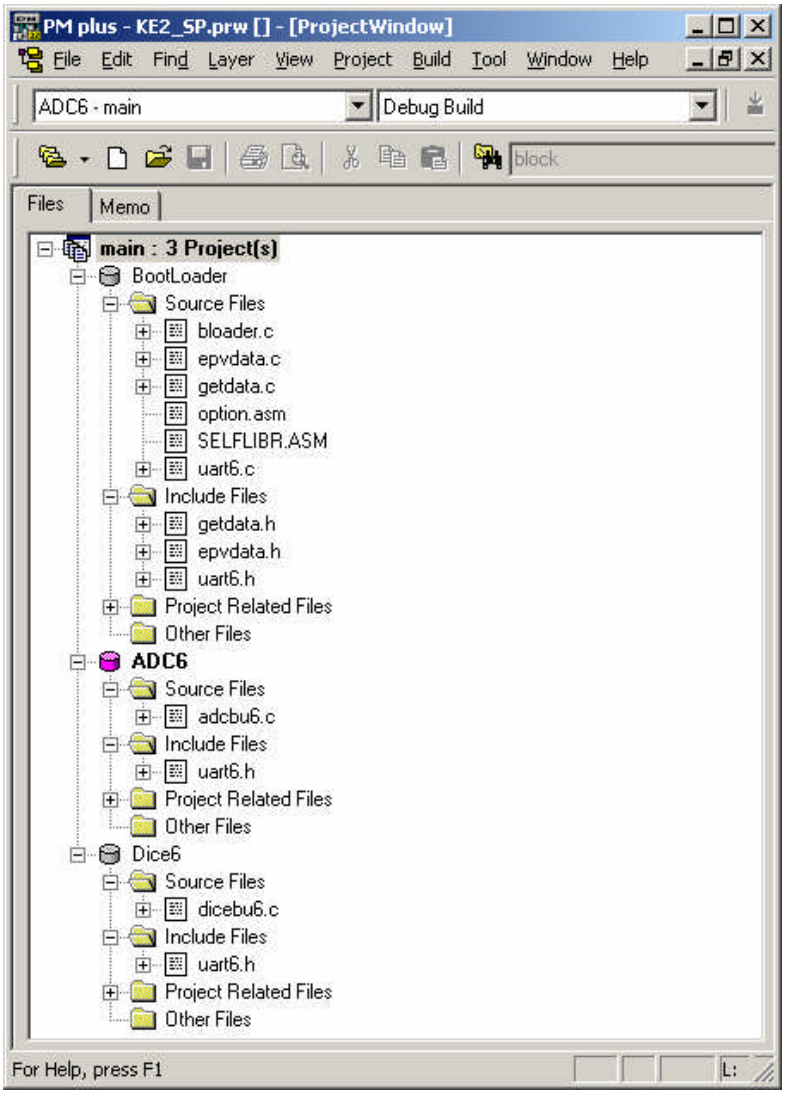
这些文件的程序清单在附录中。

表 2. 演示程序软件模块

文件	目的	由 Applilet 产生	由用户修改
bloader.c	启动装载相关函数	否	是
evpdata.c	擦除、校验和编程函数	否	是
getdata.c	从用户程序中输入数据函数	否	是
uart6.c	RS232 与主机通讯函数	否	是
adcbu6.c	A/D 转换演示程序	否	是
dicebu6.c	骰子显示程序	否	是
uart6.h	RS232 与主机通讯声明	否	是
epvdata.h	擦除、校验和编程声明	否	是
getdata.h	从用户程序中输入数据声明	否	是
SELFLIBR.ASM	自编程库汇编函数	否	是
option.asm	选项字节设置	否	是

有三个子工程文件和该应用相关。为了配置这三个文件应在主目录下为 dice6、ADC6 以及 bootloader 分别创建三个子目录。将 adcbu6.c 和 uart6.h 复制到 dice 目录下，将 adcbu6.c 和 uart6.h 复制到 ADC 目录下，复制其它的文件 (包括 uart6.h) 到 bootloader 目录下。在主目录下创建主工程，并分别将新工程加到主工程中。工程建好后，工程文件设置如下：

图 14.　　项目文件设置


PM plus - KE2_SP.prw [] - [ProjectWindow]
File Edit Find Layer View Project Build Tool Window Help
ADC6 - main
Debug Build
block
Files
Memo
main : 3 Project(s)
BootLoader
Source Files
bloader.c
epvdata.c
getdata.c
option.asm
SELFLIBR.ASM
uart6.c
Include Files
getdata.h
epvdata.h
uart6.h
Project Related Files
Other Files
ADC6
Source Files
adcbu6.c
Include Files
uart6.h
Project Related Files
Other Files
Dice6
Source Files
dicebu6.c
Include Files
uart6.h
Project Related Files
Other Files
For Help, press F1
L:

7. 附录 A — Flash 自编程例子程序

7.1 bloader.c

```
/*****************************************************************************
*
* 文件                    : bloader.c
* 日期                    : 2006 .2
* 描述                    : M-78F0537 的引导文件
* CPU 类型                : 78K0/KE2 - uPD78F0537D
*
* 注意:          :
*
*****************************************************************************/

#pragma sfr
#pragma NOP
#pragma EI
#pragma DI

// 文件
#define VERSION 3
#define SSEG_L  0xc7

/*=======================================
; 头文件
;=======================================*/

#include "getdata.h"
#include "epvdata.h"
#include "uart6.h"

unsigned long storedAppWord;      // 存储应用程序校验和
unsigned int wordData;                    // 通用整型变量
const unsigned char
led_array[16]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x08,0x03,0x46,0x21,0x06,0x0e};

/*****************************************************************************/
// void hdwinit(void) – 设置初始化硬件
//         注意- 此函数在 boot_main() 之前被启动代码调用
void hdwinit ()
{
        /* 注意选项字节(option.asm)的 WDTON 位是 1，因此 */
        /* 看门狗定时器在复位时被使能 */

        DI();               // 禁止中断

        WDTE=0xac;                          // 清除看门狗定时器

        IXS     = 0x00;         /* 使能 6144 字节 IXS RAM (uPD78F0537) */
        IMS     = 0xCC;         /*设置 uPD78F0537 的存储器容量寄存器 (128KB)*/
```

```
                                        /* 32K 共用区 + 5 x 16K 页 */
        BANK = 0x00;    /* 设置 0 页*/


        /* 启动时钟发生器, 切换到 X1/X2 时钟 */
        /* 设置: X1/X2 晶振  EXCLK (OSCCTL.7) = 0, OSCSEL (OSCCTL.6) = 1 */
        /*     XT1/XT2 晶振 EXCLKS (OSCCTL.5) = 0, OSCSELS (OSCCTL.4) = 1 */
        /*     AMPH (OSCCTL.0) = 0 for X1/X2 clock > 10 MHz */
        OSCCTL = 0x50;

        /*设置 MSTOP (MOC.7) = 0 以启动 X1/X2 震荡器 */
        MSTOP = 0;

        /* 等待 X1 震荡器稳定 */
        while (OSTC.0 == 0){
                WDTE=0xac;                              // 清除看门狗定时器
                ;
        }

        /* 设置 X1 时钟输入以替代内部震荡器 */
        MCM = 0x05;             /* XSEL (MCM.2) = 1, MCM0 (MCM.0) = 1 */

        /* 等待时钟切换到 X1 时钟 */
        while (MCS == 0){
                WDTE=0xac;                              // 清除看门狗定时器
                ;
        }


        LVIM = 0x00;            //禁止 LVI 检测
        LVIS = 0x00;

        PCC = 0x00;             /* 设置 CPU 高速运行 */

}


/*****************************************************/
void boot_main(void)            // 保留字"boot_main"
{
        unsigned char delay;


        WDTE=0xac;                                      // 清除看门狗定时器

        P4=0x03;                                        // "b"
        P5=0x08;
        P7=led_array[VERSION];          // bootloader 例子程序版本

        PM4=0x00;
        PM5=0x00;
        PM7=0x00;       // 设置所有端口输出
```

```
InitUART6_8N1(BAUD_115200_20);      // 初始化 UART6
Enable _SFP();                              // 使能自编程

bootSwapFlag=CheckBootSwapFlag();   // 读取引导交换标志
if(bootSwapFlag)                    // 如果设置引导交换标志
{
        Tx_String("\n\rBootSwapping..");  // 发送引导交换信息
        performBootSwap();                  // 执行引导交换
        ToggleBootSwapFlag();       // 设置标志到 0
        disable_SFP();                      // 禁止自编程
        while(1);                           // 无限循环以迫使看门狗溢出
}
// 发送程序描述
Tx_String("\n\r\n\rNEC Electronics America Bootloader Ver. ");
Tx_Character(VERSION+0x30);         // 发送版本
Tx_Character(X_ON);                 // 发送 XON
timeOut=MAX_COUNT;
while(timeOut>0)
{
        if(SRIF6)
        {
                rx_char=RXB6;
                SRIF6=0;
                timeOut=MAX_COUNT;
        }
        timeOut--;
}
Tx_String("\n\rLoad Y/N? ");       // 询问用户是否加载文件

if(ConfirmOnPrompt())               // 如果加载
{
        P7=SSEG_L;                          // 在 m-station 的 第二个发光二极管上显示 L
        GetFile();                          // 引导加载文件
}
Disable_SFP();                      // 禁止自编程

P7=led_array[VERSION];              // 在 m-station 上再次显示版本
P4=0x03;
P5=0x08;

WDTE=0xac;                          // 清除看门狗定时器

Tx_CRLF();

BANK=CS_FLASH_BANK;                 //设置 Flash 页到 #5，用于校验和处理
validApp_ptr=(unsigned long *)CS_ADDRESS;       // 指向校验和地址
storedAppWord=*validApp_ptr;                    // 获得存储校验和

BANK=0;                             //设置 Flash 页为却省值

Tx_String("\n\rStored checksum = ");        // 发送校验和
wordData=(unsigned int)(storedAppWord>>16);
```

```
        Tx_Word(wordData);
        wordData=(unsigned int)(storedAppWord&0x0000ffff);
        Tx_Word(wordData);

        validAppWord=CalculateAppChecksum ();                    // 计算校验和
        Tx_String("\n\rCalculated checksum = ");          // 发送计算校验和
        wordData=(unsigned int)(validAppWord>>16);
        Tx_Word(wordData);
        wordData=(unsigned int)(validAppWord&0x0000ffff);
        Tx_Word(wordData);


        if(validAppWord!=storedAppWord)                          //如果校验和不匹配
        {
                P4=0x06;                                                        // "C"
                P5=0x0C;
                P7=led_array[0x0e];                              // "E"

                while(1);                                        // 无限循环迫使看门狗溢出
        }
        else
        {
                for(delay=0;delay<0x00ff;delay++) WDTE=0xac;     // 应用程序启动之前增加延时
        }
}
```

7.2 epvdata.c

```
/*****************************************************************************
*
* 文件                  : epvdata.c
* 日期                  : 2006 .2
* 描述          : M-78F0537 的引导文件
* CPU 类型              : 78K0/KE2 - uPD78F0537D
*
* 注意:                 : 支持擦除校验程序
*                       : 加 校验和的计算与存储
*
*****************************************************************************/
#define EPVDATA_C
#pragma sfr
#pragma NOP                             /*关键字 NOP 指令 */
#pragma DI
#include "epvdata.h"
#include "uart6.h"

//下面是自编程库文件需要的两个数据结构
struct stWordAddress{
        USHORT WriteAddress;
        UCHAR WriteBank;
};

struct stGetInfo{
        UCHAR OptionNumber;
        UCHAR GetInfoBank;
        UCHAR GetInfoBlock;
};

struct stWordAddress WordAddr;
struct stGetInfo GetInfo;

//NEC selflibr.rel ( 自编程库)的库函数
extern  void    FlashStart(void);
extern  void    FlashEnd(void);
extern  void    FlashEnv(USHORT EntryRAM);
extern  UCHAR FlashBlockErase(UCHAR EraseBank,UCHAR EraseBlock);
extern  UCHAR FlashWordWrite(struct stWordAddress *ptr, UCHAR WordNumber, USHORT myDataBuffer);
extern  UCHAR FlashBlockVerify(UCHAR VerifyBank, UCHAR VerifyBlock);
extern  UCHAR FlashBlockBlankCheck(UCHAR BlankCheckBank,UCHAR BlankCheckBlock);
extern  UCHAR FlashGetInfo(struct stGetInfo *ptr, USHORT myDataBuffer);
extern  UCHAR FlashSetInfo(UCHAR SetInfoData);
extern  UCHAR CheckFLMD(void);


unsigned int gp_i;
unsigned char myEntryRam[ENTRY_RAM_SIZE];
unsigned char returnValue;

unsigned short startAddress;
```

```
unsigned char *bootswap_ptr;

void programDataBuffer(void)
{
        WDTE=0xac;              // 清除看门狗定时器

        if(firstProgram)        // 如果是第一次自编程
        {
                firstProgram=FALSE;     // 复位首次自编程标志

                if(programmingAddress==0)                       // 如果首次自编程地址是 0000H
                {
                        bootDownLoad=TRUE;                      // 引导加载标志设置为真
                        EraseFlashBlocks(0,0,FIRST_BOOT_BLOCK,LAST_BOOT_BLOCK); // 空白检测和
擦除共用区的 4-7 块
                }
                else
                {
                    EraseFlashBlocks(0,0,FIRST_FLASH_BLOCK_C,LAST_FLASH_BLOCK_C); //空白检测和
擦除共用区的 4-31 块
                    EraseFlashBlocks(0,5,FIRST_FLASH_BLOCK_B,LAST_FLASH_BLOCK_B); //空白检测和
擦除 0-5 页的 32-47 块
                }

                Tx_String("\n\r\n\rProgramming at.."); // 编程的开始

        }
        startAddress=(unsigned short)(programmingAddress);      // 得到编程的开始地址

        numberOfWords=db_index/4;       // 忽略余数字节
        Tx_CRLF();                                      // 在新行写地址
        if(bootDownLoad)                        //如果是引导区加载
        {
                WDTE=0xac;                              // 清除看门狗定时器

                startAddress+=0x1000;   // 开始地址加 1000H
                Tx_Word(programmingAddress);  // 发送 "from" 地址
                Tx_Character('-');
                Tx_Character('>');
        }
        else                                            // 如果不是引导区加载
        {
                if(programmingAddress<0x1000)  //地址是引导区
                {
                        Tx_String("\n\rBoot Area!");    // 发送错误
                        while(1);                                               // 迫使看门狗溢出
                }
        }
        Tx_Word((unsigned int)startAddress);                            // 发送数据开始地址

        WriteDataBufferToFlash(0,startAddress,numberOfWords); //写缓冲区数据
```

```
}

void disable_SFP(void)
{
        FLMD0_CONTROL_IO=OUTPUT;                        // 设置 FLMD0 控制引脚到输出
        FLMD0_CONTROL_PIN=1;                            // 设置 FLMD0 为高

        FlashStart();
        FlashEnv((unsigned short)&myEntryRam);

        // check mode is ok
        returnValue=CheckFLMD();

        if(returnValue!=0)                                      // 如果不成功
        {
                Tx_String("\n\rSFP Error: FLMD0 CHECK: ");      // 发送功能返回代码
                Tx_Byte(returnValue);                           // 发送返回功能号
        }

}

void disable_SFP(void)
{
                FLMD0_CONTROL_PIN=0;                            // 设置 FLMD0 引脚为低
                FlashEnd();
}

void EraseFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char lastBlock)
{
        unsigned char blocknumber, banknumber;
        WDTE=0xac;              // 清除看门狗定时器

        // 空白检测和擦除可能被编程的块        for(banknumber=firstBank;banknumber<lastBank+1;banknumber++)
        {
                for(blocknumber=firstBlock;blocknumber<lastBlock+1;blocknumber++)
                {
                        WDTE=0xac;
                        returnValue=FlashBlockBlankCheck(banknumber,blocknumber);

                        if(returnValue!=0)                              // 如果不成功
                        {
                                if(returnValue==NOT_BLANK)              // 如果没有空白
                                {
                                        Tx_String("\n\rErasing block ");    // 发送擦除

                                        returnValue=FlashBlockErase(banknumber,blocknumber); // 擦除块

                                        if(returnValue!=0)                  // 如果擦除不成功
                                        {
                                                Tx_String("\n\rSFP Error: BLOCK ERASE: ");      // 发送功能返
回代码
                                                Tx_Byte(returnValue);                           // 返回功能号
                                        }
```

```
                                        else                                    // 否则如果擦除成功
                                        {
                                                if(!bootSwapFlag)               // 如果不是引导交换
                                                {
                                                        Tx_Byte(blocknumber);   // 发送块号
                                                }
                                                else                            // 发送 0,1 以替代 2,3
                                                {                                       // 作为内部地
址切换
                                                        Tx_Byte(blocknumber-2); //发送块号
                                                }

                                        }
                                }
                                else                                            // 如果空白核查错误
                                {
                                        Tx_String("\n\rSFP Error: BLOCK BLANK CHECK: ");        // 发送功能返
回代码
                                        Tx_Byte(returnValue);                   // 返回功能号

                                }
                        }
                }
        }
}

void WriteDataBufferToFlash(unsigned char banknumber,unsigned short writeAddress,unsigned char wordCount)
{
        WDTE=0xac;              // 清除看门狗定时器

        WordAddr.WriteAddress = writeAddress;
        WordAddr.WriteBank = banknumber;

        returnValue=FlashWordWrite(&WordAddr,wordCount,&myDataBuffer);          // 写数据到 flash 存储器

        if(returnValue!=0)                                                      // 如果不成功
        {
                Tx_String("\n\rSFP Error: WORD WRITE: ");       // 发送功能返回代码
                Tx_Byte(returnValue);                           //返回功能号

        }
}

void VerifyFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char lastBlock)
{
        unsigned char blocknumber, banknumber;
        WDTE=0xac;              // 清除看门狗定时器
        Tx_String("\n\rVerifying..\n\r");

                // for all specified blocks
                for(banknumber=firstBank;banknumber<=lastBank;banknumber++)
                {
                        for(blocknumber=firstBlock;blocknumber<=lastBlock;blocknumber++)
```

```
                {
                        WDTE=0xac;              // 清除看门狗定时器

                        returnValue=FlashBlockVerify(banknumber,blocknumber);    // 核查块

                        if(returnValue!=0)                                              // 如果
不成功
                        {
                                Tx_String("\n\rSFP Error: BLOCK VERIFY: ");      // 发送功能返
回代码
                                Tx_Byte(returnValue);                           // 返回功能
                        }
                }
        }
}

unsigned long CalculateAppCheckSum(void)
{
        unsigned long appCheckSum;
        unsigned char *address_ptr, bank;
        unsigned int firstaddr,lastaddr;

        WDTE=0xac;              // 清除看门狗定时器

        appCheckSum=0;                                          // 清除变量以存储应用校验和

        //整个存储器校验和
        firstaddr = 0x0000;
        lastaddr = 0xBFFF;

        for(bank=0;bank<6;bank++){

                BANK=bank;

                if (bank==1) firstaddr = 0x8000;
                else if (bank==5) lastaddr=CS_ADDRESS-1;

                for(address_ptr=(unsigned char *)firstaddr;address_ptr<(unsigned char *)lastaddr;address_ptr++){
                        appCheckSum+=*address_ptr;              // 加数据到校验和
                        WDTE=0xac;                                      // 清除看门狗定时器
                }

        }

        BANK = 0;

        return appCheckSum;                                     //返回计算校验和
}

void WriteVerifyAppCheckSum(void)
{
        WDTE=0xac;                                                              // 清除看门狗
定时器
```

```
        validApp_ptr=(unsigned long *)myDataBuffer;                    // 指向数据缓冲器的开始地址
        *validApp_ptr=validAppWord;                                                    // 存放校验和到缓冲器中
        Tx_Word((unsigned int)CS_ADDRESS);                                    // 发送数据开始地址
        WriteDataBufferToFlash(CS_FLASH_BANK,CS_ADDRESS,1);        //编程校验和
        VerifyFlashBlocks(CS_FLASH_BANK,CS_FLASH_BANK,CS_FLASH_BLOCK,CS_FLASH_BLOCK); // 核查
块
        WDTE=0xac;                                                                    // 清除看门狗定时器
}

unsigned char ConfirmOnPrompt(void)
{

        for(i=0;i<25;i++)
        {
                WDTE=0xac;              // 清除看门狗定时器

                timeOut=MAX_COUNT;
                while(timeOut>0)// 等候用户确认
                {
                        if(SRIF6)
                        {
                                rx_char=Rx_Character();  // 获得字符串
                                Tx_Character(rx_char);    // 回放字符串
                                if((rx_char=='y')||(rx_char=='Y'))    // 如果是
                                {
                                        return(TRUE);                                        //返回真
                                }
                                else
                                {
                                        if((rx_char=='n')||(rx_char=='N'))    // if no
                                        {
                                                return(FALSE);                                // 返回假
                                        }
                                        else                // 如果输入不是 yes，也不是 no
                                        {
                                                i=0;      // 重启动超时循环
        }
                                }
                        }

                        timeOut--;                                // 消耗超时次数
                }
        }

        return(FALSE);  // 如果时间超出之后没有输入，则返回假

}

unsigned char CheckBootSwapFlag(void)
{
        // 核查引导交换标志
```

```
	GetInfo.OptionNumber=0x04;
	GetInfo.GetInfoBank=0;		//当 Option = 4 时忽略
	GetInfo.GetInfoBlock=0;		//当 Option = 4 时忽略

	returnValue=FlashGetInfo(&GetInfo,&myDataBuffer);

	if(returnValue!=0)					// 如果不成功
	{
		Tx_String("\n\rSFP Error: GET INFORMATION: ");// 发送功能返回代码
		Tx_Byte(returnValue);				// 返回功能号
	}
	return(myDataBuffer[0]);  // 返回信息
}

void performBootSwap(void)
{
	unsigned char blocknumber,j;

	EraseFlashBlocks(0,0,FIRST_BOOT_BLOCK,LAST_BOOT_BLOCK);		// 擦除目标块
	WDTE=0xac;					// 清除看门狗定时器

	bootswap_ptr=(unsigned char *)0x0000;	// 设置指针指向引导代码
	startAddress=(unsigned long)0x1000;		// 设置开始地址
	for(int_i=0;int_i<256;int_i++)			// 256 * 16 次= 4096
	{
		for(j=0;j<16;j++)
		{
			myDataBuffer[j]=*bootswap_ptr++;	// 填充数据缓冲区（16 字节）
		}
		WriteDataBufferToFlash(0,startAddress,4);  // write 4 words
		startAddress+=16;				// 增加地址（16）
		WDTE=0xac;					// 清除看门狗定时器

	}
}

void ToggleBootSwapFlag(void)
{
	returnValue=FlashSetInfo(0xFE);   // 置引导交换标志为 0 (bit 0)
```

```
        if(returnValue!=0)                                          //如果没有成功
        {

                Tx_String("\n\rSFP Error: SET INFORMATION: "); //发送功能返回代码
                Tx_Byte(returnValue);                               // 返回功能号

        }
}
```

7.3 getdata.c

```
/*****************************************************************************
*
* 文件                  : getdata.c
* 日期                  : 2006.2
* 描述          : M-78F0537 的引导文件
* CPU 类型              : 78K0/KE2 - uPD78F0537D
*
* 注意:         :
*
*****************************************************************************/
#define GETDATA_C
#pragma sfr
#pragma NOP
#pragma DI
#include "getdata.h"
#include "epvdata.h"
#include "uart6.h"


// 标志
unsigned char firstXOFF;        // 第一个 XOFF 标志
unsigned char endOfFile;        // 文件结束标志
unsigned char breakFlag;        // 地址中断标志

// 接收数据
unsigned char rx_buffer[RB_SIZE];       // 接收数据缓冲器
unsigned int  rb_index;                 // 接收缓冲器索引
unsigned char flowState;        // 流程状态

// 记录
unsigned char recordCount;              // 每行的数据字节数目
unsigned char msAddress;        // 行地址的高字节
unsigned char lsAddress;        //行地址的低字节
unsigned char recordType;       // 记录字节 (00=数据, 01=文件结束)
unsigned char CheckSum;                 // 每行校验和

// 处理数据
unsigned int  pr_index;                 // 处理数据的索引
unsigned int  pr_end;                   // 处理循环的结束索引
unsigned char charPosition;             // 每行的字符串位置
unsigned char nextByte;                 // 缓冲区的下个字节
unsigned char sumOfBytes;               // 用于核查校验和的字节和
unsigned char dataCount;        // 每行的保留数据字节计数
unsigned char sob_whole;        // 最后整个自编程字数
unsigned int rb_whole;                  // 最好整个自编程字索引
unsigned char dc_whole;                 // 最后自编程的字数据计数

// 地址
unsigned char oddAddress;               // 大于自编程字边界的字节
```

```
unsigned int currentAddress;        // 当前数据地址
unsigned int lineAddress;           // 接收当前行地址


// 通用
unsigned int int_j;                         // 整型 j


void GetFile(void)
{
        WDTE=0xac;              // 清除看门狗定时器

        //初始化变量
        firstProgram=TRUE;              // 首次编程为真
        firstXOFF=TRUE;                         // 首个 XOFF 为真
        endOfFile=FALSE;                // 文件结束为假
        breakFlag=FALSE;                // 地址中断标志为假
        bootDownLoad=FALSE;             // 引导加载为假
        rb_index=0;                             // 接收缓冲区索引为 0
        charPosition=0;                 // 行字符串位置为 0
        Tx_Character(X_ON);             // 确定在 XON 状态
        flowState=X_ON;                         // 设置流程状态为 XON
        Tx_String("\n\rSend file\n\r");     // 要求用户发送文件

        while(!SRIF6)                   // 等候用户发送文件
        {
                WDTE=0xac;                      // 清除看门狗定时器
        }


        while(!endOfFile)               // 主编程循环
        {
                WDTE=0xac;                      // 清除看门狗定时器
                timeOut=MAX_COUNT;              // 设置超时参数为最大

                while(timeOut!=0)       // 获得数据循环
                {
                        WDTE=0xac;              // 清除看门狗定时器
                        timeOut--;              // 消耗延时参数计数器
                        if(SRIF6)               // 如果接收到字符串
                        {
                                rx_buffer[rb_index++]=RXB6;     // 放置字符在缓冲区里
                                                                // 并消减索引数
                                SRIF6=0;                                        // 清除中断标志
                                timeOut=MAX_COUNT;                              // 设置延时参数为最大
                                if(rb_index>=RB_DATA_SIZE)  // 如果数据限制
                                {
                                        TXB6=X_OFF;                             // 发送 XOFF 字符
                                flowState=X_OFF;                // 设置流程状态为 XOFF
                                }
                        }
```

```
}

// 流程状态为 XOFF (或 XON 如果数据已经结束发送)
programmingAddress=currentAddress;      // 分配新的编程地址
db_index=0;                 // 复位数据缓冲索引
for(int_i=0;int_i<MDB_SIZE;int_i++)
{
        myDataBuffer[int_i]=0x00;       // 清除数据缓冲
}
pr_index=0;                             // 复位处理接收缓冲区索引

if(breakFlag)                   // 如果有一个地址中断
{
        breakFlag=FALSE;        // 清除地址中断标志
        oddAddress=(unsigned char)(programmingAddress%4);       // 获得余数地址计数
        while(oddAddress!=0)                    // 当余数地址不是为 0
        {
                programmingAddress--;           // 消减编程地址
                myDataBuffer[db_index++]=0xff;  // 放衬垫字节到数据缓冲取中
                oddAddress--;                                   // 消减余数地址计数

        }
}

// 初始化处理循环
if(rb_index>=RB_DATA_SIZE)    // 如果数据大于等于限制值
{
        pr_end=RB_DATA_SIZE-2;          // 设置结束索引减二
}
else
{
        pr_end=rb_index-1;      // 减一以防止文件结束字符
}

while(pr_index<pr_end)    // 处理接收缓冲区数据循环
{
        WDTE=0xac;                      // 清除看门狗定时器

        switch(charPosition)// 主分支结构
        {
                case 0:
                                nextByte=rx_buffer[pr_index++];   // 获得下个字节
                                if(nextByte==':')                 // 如果是一个冒号
                        {
                                        charPosition=1;                 // 下个字符位置为 1

                                }
                                else
                                {
                                        if(firstXOFF)                   // 如果文件没有头冒号
                                        {
```

```
                                        while(1);                       // 等候看门狗
超时
                                    }
                                }
                                break;

                        case 1:
                                recordCount=GetHexByte();       // 获得记录计数
                                charPosition=3;                         // 下个字符位置为 3
                                sumOfBytes=recordCount;         // 开始计算每行字节
                        dataCount=recordCount;          // 初始化数据计数
                                break;

                        case 3:
                                msAddress=GetHexByte();                 // 获得地址高字节
                                charPosition=5;                         // 下个位置为 5
                                sumOfBytes+=msAddress;          // 增加到字节和
                                break;

                        case 5:
                                lsAddress=GetHexByte();         // 获得地址低字节
                                charPosition=7;                         // 下个字符位置为 7
                                sumOfBytes+=lsAddress;          // 增加到字节和
                                lineAddress=(unsigned int)msAddress<<8;  // 放置高字节到行地址
中
                                lineAddress|=(unsigned int)lsAddress;       // 放置低字节到行地址
中
                                if(firstXOFF)                           // 如果是第一个 XOFF
                                {

                                        firstXOFF=FALSE;                // 清除 XOFF 标志
                                        programmingAddress=lineAddress;// 初始化编程地址
                                        currentAddress=lineAddress;             // 初始化当前
地址
                                        oddAddress=(unsigned char)(programmingAddress%4);
        // 获得余数地址计数
                                        while(oddAddress!=0)    // 在自编程地址边界编程
                                        {
                                                programmingAddress--;           // 消减编程地
址
                                                myDataBuffer[db_index++]=0xff;  // 填充缓冲区
                                                oddAddress--;
// 消减余数地址计数
                                        }

                                }
                                else                                    // 如果不是首
个 XOFF
                                {

                                        while((currentAddress<lineAddress)&&(oddBytes!=0))
                                        {
```

```
                                        myDataBuffer[db_index++]=0xff;  // 填充字节
                                        currentAddress++;           // 消减当前地址
                                        oddBytes++;                         // 消减余数字
节
                                        oddBytes=oddBytes%4;// 查看是否满足完整的字

                                }
                                if((oddBytes==0)&& (currentAddress<lineAddress))

                                {
                                        breakFlag=TRUE;         // 设置地址中断标志为
真
                                        currentAddress=lineAddress;         // 设置新的当
前地址为中断地址

        for(int_i=0,int_j=pr_index;int_j<RB_SIZE;int_i++,int_j++)
                                        {
                                                rx_buffer[int_i]=rx_buffer[int_j];   // 移动
缓冲区数据
                                        }
                                        rb_index-=pr_index;         // 移动缓冲区索引
                                        programDataBuffer();
                                        pr_index=pr_end;            // 如果有地址中断则退
出循环

                                }
                        }

                        break;

                case 7:
                        recordType=GetHexByte();            // 获得记录字节
                        if(recordType!=0)endOfFile=TRUE;          // 如果不是 0，然后设
置文件结束标志

                        sumOfBytes+=recordType;                   // 增加大字节和
                        charPosition=9;                           // 下个字符位置为 9
                        break;

                case 9:
                        if(dataCount!=0)            // 如果仍有数据字节保留在此行上
                        {
                                nextByte=GetHexByte();  // 获得下个数据字节
                                sumOfBytes+=nextByte;   // 增加到字节和
                                myDataBuffer[db_index++]=nextByte;        // 放置字节到
数据缓冲区

                                currentAddress++;                 // 消减当前地址
                                dataCount--;                              // 消减数据计
数

                                oddBytes=(unsigned char)(db_index%4);               // 计算
余数字节
```

```
                                        if(oddBytes==0)                         // 如果自编程
字为整数
                                        {
                                                rb_whole=pr_index;              // 存储索引
                                                sob_whole=sumOfBytes;   // 存储字节和
                                                dc_whole=dataCount;             // 存储数据计
数
                                        }

                                }
                                else
                                {
                                        charPosition=11;                // 校验和在位置>=11
                                }
                                break;

                        case 11:
                                CheckSum=GetHexByte();                          // 获得校验和

                                sumOfBytes+=CheckSum;                   // 增加到字节和
                                if(sumOfBytes!=0)                               // 校验和 错误
如果校验和不是 0
                                {
                                        Tx_String("\n\rLine CheckSum!"); // 发送错误
                                        while(1);
        // 等候看门狗复位
                                }
                                else                                            // 否则如果校
验和正确
                                {
                                        charPosition=0;                 // 下个位置为 0
                                }
                                break;


                        default:
                                NOP();                                          // 无操作

                        }       // 结束 switch

                }       // 结束 while(pr_index<pr_end)

                if(!breakFlag)                                                  // 如果不是一个地址中
断
                {
                        if(rb_index>=RB_DATA_SIZE)              // 如果索引 >= 数据限制
                        {
                                currentAddress-=oddBytes;               // 移动当前地址返回到整个字边
界
                                sumOfBytes=sob_whole;                   // 恢复整个字边界的存储和
                                dataCount=dc_whole;                             // 恢复数据计数
                                for(int_i=0,int_j=rb_whole;int_j<RB_SIZE;int_i++,int_j++)
```

```
                                {
                                        rx_buffer[int_i]=rx_buffer[int_j];   // 移动缓冲区数据
                                }
                                rb_index-=rb_whole;                          // 移动缓冲区索引
                                charPosition=9;                              // 移动返回到数据
                                programDataBuffer();                         // 编程数据缓冲区

                        }
                        else                                                 // 否则 如果索
引 < 数据限制
                        {
                                while(oddBytes!=0)                           // 当余数字节不是 0
                                {
                                        myDataBuffer[db_index++]=0xff;  // 放置填充字节
                                        currentAddress++;                    // 消减当前地址
                                        oddBytes++;                                  // 消减余数字
节
                                        oddBytes=oddBytes%4;            // 查看是否满足一个整字节

                                }
                                programDataBuffer();                         // 编程数据缓冲区
                        }
                }
        Tx_Character(X_ON);             // 转变发送返回
        flowState=X_ON;                         // 和设置流程状态到 XON

        } // 结束 while(!endOfFile)


        timeOut=MAX_COUNT;                              // 设置超时参数
        while(timeOut>0)                        //等待文件结束字符
        {
                WDTE=0xac;                                              // 清除看门狗定时器

                timeOut--;                                              // 消减延时参数
                if(SRIF6)                                               // 如果有连续的字符串
                {
                        rx_char=Rx_Character();  // rx_char =接收数据
                        timeOut=MAX_COUNT;                      // 复位超时参数到最大
                }
        }
        WDTE=0xac;              // 清除看门狗定时器

        VerifyFlashBlocks(0,0,FIRST_FLASH_BLOCK_C,LAST_FLASH_BLOCK_C); // verify flash area

                        //(only common area for demonstration purposes)
        if(!bootDownLoad)
        {
                WDTE=0xac;              // 清除看门狗定时器

                Disable_SFP();  // 禁止自编程
                validAppWord=CalculateAppCheckSum();  //计算应用校验校验和
```

```
                Tx_String("\n\rCheckSum ");
                Tx_Word((unsigned int)(validAppWord>>16));
                Tx_Word((unsigned int)(validAppWord&0x0000ffff));
                Tx_String(" at..\n\r");
                Enable_SFP();               // 自编程准备
                WriteVerifyAppCheckSum();       // 写和核查应用校验和
                Disable_SFP();              // 禁止自编程
        }
        else                                        // 否则如果引导程序加载
        {
                WDTE=0xac;                  // 清除看门狗定时器
                Tx_String("\n\rReplace the boot code Y/N? ");       // 发送确认信号
                if(ConfirmOnPrompt())
                {
                        ToggleBootSwapFlag();   // 设置引导交换标志
                        bootSwapFlag=CheckBootSwapFlag(); // 核查引导交换标志状态

                }
                Disable_SFP();              // 禁止自编程

                P4=0x0F;                    // "r"
                P5=0x0A;
                P7=0x86;                    // "E"

                while(1);                   // 等待看门狗超时
        }

}
/*****************************************************************************
*
* 功能:       获得十六进制字节
* 参数:       无
* 返回值:     字符数据的十六进制字节
*
* 日期:       2005.1.24
* 注意:       减小缓冲区索引值为 2
*
*****************************************************************************/

unsigned char GetHexByte(void)
{
        unsigned char hexByte,nibbleValue;

        nibbleValue=rx_buffer[pr_index++]-'0';      // 获得高半位
        if(nibbleValue>9)nibbleValue-=7;            // 转为二进制
```

```
        hexByte=nibbleValue<<4;                                 // 将高字节放入字节中
        nibbleValue=rx_buffer[pr_index++]-'0';      // 获得低半位
        if(nibbleValue>9)nibbleValue-=7;            //转为二进制
        hexByte|=(nibbleValue&0x0f);                        //将低字节放入字节中
        return(hexByte);
}
```

7.4 uart6.c

```
/*****************************************************************************
*
* 文件                  : uart6.c
* 日期                  : 2004 .11.17
* 描述          : M-78F0537 的异步串口文件
* CPU 类型              : 78K0/KE2 - 78F0537D
*
* 注意:         : UART6 (8 数据位,无校验位, 1 个停止位)
*
*****************************************************************************/
#define UART6_C
#pragma sfr
#pragma NOP                             /* NOP 指令关键字 */
#include "uart6.h"

const char Hex_Values[]="0123456789ABCDEF";
/*****************************************************************************
*
* 功能:         初始化 UART6
* 参数:         无符号字符值设置波特率
* 返回值:       无
* 日期:         2006.12
* 描述:         初始化 UART6
*                                       (8 数据位,无校验位, 1 个停止位)
* 注意:
*
*
*****************************************************************************/

void InitUART6_8N1(unsigned char baudRate)
{
        unsigned int i;

        PM1.4=1;                /* 设置 RxD6 引脚为输入 */
        P1.3=1;                 // 设置 TxD6 引脚为高
        PM1.3=0;                /* 设置 TxD6 引脚到输出 */

        ASIM6=0x01;             // 停止开始 – 设置到复位状态

        BRGC6=BAUD_115200_20;       /*设置波特率 */

        CKSR6 = 0x00;
        ASIM6 |= 0xE5;

        POWER6=1;               // 使能 UART6
        for(i=0;i<1000;i++)     // 等候 2 BRGC6 时钟
        {
                NOP();
        }
```

```
        STIF6=0;                    // 清除 Tx 中断请求 t
        TXE6=1;                             // 使能 发送

        SRIF6=0;                    // 清除 Rx 中断请求
        RXE6=1;                             // 使能 接收
}
/*****************************************************************************
*
* 功能:          发送字符串
* 参数:          指向无符号字符串数组
* 返回值:        无
* 日期:          2004.2.26
* 描述:          通过 UART 发送字符串
* 调用:          Tx_Character(unsigned char ascii_character)
* 注意:
*
*****************************************************************************/
void Tx_String(const char *puc)
{
        while(*puc != NULL_CHAR)
                Tx_Character(*puc++);
}
/*****************************************************************************
*
* 功能:             发送字
* 参数:             16 位无符号整型
* 无:               无
* 日期:             2004.2.29
* 描述:             转换 16 位数值为 4 个 ASCII 十六进制字符
*                   和通过 UART 发送它
* 日期:
*
*****************************************************************************/
void Tx_Word(unsigned int data_16_bit)
{
        Tx_Byte((unsigned char)(data_16_bit >> 8));                 // 发送高字节
        Tx_Byte((unsigned char)(data_16_bit & 0xff));       // 发送低字节
}
/*****************************************************************************
*
* 功能:          发送字节
* 参数:          8 位数据
* 返回值:        无
* 日期:          2004 .2.27
* 描述:          通过 2 个 ASCII 的十六进制字符发送 8 位值
*
* 日期:
*
*
*****************************************************************************/
void Tx_Byte(unsigned char data_8_bit)
{
```

```
        Tx_Character(Hex_Values[data_8_bit>>4]);// 发送高字节十六进制字符
        Tx_Character(Hex_Values[data_8_bit&0x0f]);        //发送高字节十六进制字符
}

/*****************************************************************************
*
* 功能:            Tx_CRLF
* 参数:            无
* 返回值:                  无
* 日期:                     2004.02.28
* 描述:  Send CR/LF
*
* 注意:
*
*****************************************************************************/
void Tx_CRLF(void)
{
        Tx_Character(CR_CHAR);          // 发送回车
        Tx_Character(LF_CHAR);          // 发送换行
}

/*****************************************************************************
*
* 功能:       发送字符
* 参数:       无符号 ASCII 字符串
* 返回值:     无
* 日期:       2004.2.23
* 描述:       通过 UART 发送字符
*
* 注意:
*
*
*****************************************************************************/
void Tx_Character(char ascii_character)
{
        char char _char;

        if((SRIF6) && (ASIS6!=0))                                //如果状态寄存器有错误
        {
                SRIF6=0;                        // 清除错误中断请求标志
                char _char=RXB6;          // 读取接收寄存器以清除错误
        }

        TXB6=ascii_character;     // 加载字符到发送寄存器
        while(STIF6==0);                  // 发送时等待
        STIF6=0;                                // 清除中断请求


}

/*****************************************************************************
*
```

```
* 功能:          读取字符串
* 参数:          无
* 返回值:        接收字符串
* 日期:          2004.2.23
* 描述:          通过 UART 接收字符
*
* 注意:          如果接收错误设置 INTSER0
*                读 RXB6 清除 ASIS6 错误标志准备下个接收
*
*
*****************************************************************************/
char Rx_Character(void)
{
        char rx_char;


        // 如果接收错误返回 NULL 字符
        // 注意接受寄存器必须被清除以清除错误状态
        if(SRIF6)
        {
                if(ASIS6!=0)                            // 如果状态寄存器有任何错误
                {
                        SRIF6=0;                        // 清除错误中断请求标志
                        rx_char=RXB6;           // 读取接收寄存器以清除错误
                }
                else
                {
                        rx_char=RXB6;           // 获得接收字符
                        SRIF6=0;                        // 清除 错误中请求标志

                }
        }

        return(rx_char);                        // 返回接收字符串

}
```

7.5 adcbu6.c

```
/*****************************************************************************
*
* 文件                    : adcbu6.c
* 日期                    : 2006.2
* 描述                    : M-78F0537 的 ADC 测试文件
* CPU 类型                : 78K0/KE2 - 78F0537
*
* 注意:                   : 使用蓝牙
*                         : 假设引导代码已经初始化单片机和 UART
*
*****************************************************************************/


#pragma ext_table 0x2000
#pragma SFR
#pragma NOP
#pragma DI
#pragma EI
#pragma interrupt INTSR6 UartRx_Interrupt rb1
#include "uart6.h"

#define SWITCH_2 P3.1
#define SWITCH_3 P3.2

#define HEX_DISPLAY 0
#define VOLT_DISPLAY 1

void Delay(unsigned int delay);
const unsigned char
led_values[16]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x08,0x03,0x46,0x21,0x06,0x0e};
unsigned char int_count;
unsigned char led1_value;
unsigned char led2_value;
unsigned char led_index;
unsigned char dp_LED1,dp_LED2;
unsigned int adc_result;
unsigned char adc_msbyte;
unsigned char volts;
unsigned int loop_count;
unsigned char display_state;
char received_char;
unsigned int timedowncount;
unsigned char tcnt;
unsigned char bootRequest;
unsigned char intRx;

unsigned char old_volts=0, old_adcval=0;

void main(){

        unsigned int i;
```

```
DI();                   // 禁止中断
WDTE=0xac;              // 清除看门狗定时器

bootRequest=0;  // bootloader 请求是假
intRx=0;        // 接收请求标志为 0
ADCS=1;                         // 使能 ADC

P4=0xff;        // 关闭 7 段 led1 输出
P5=0xff;
P7=0xff;        // 关闭 7 段 led2 输出

PM4=0x00;               // 设置 7 段 led1 都为输出
PM5=0x00;
PM7=0x00;               // 设置 7 段 led2 都为输出
PM3|=0x06;              // 设置开关 2 和 3 为输入

dp_LED2=0x80; // 设置数值以确保小数点后 2 位关闭
led1_value=0;   // 复位 led1 数值
led2_value=0;   // 复位 led2 数值
display_state=VOLT_DISPLAY;   // 设置为 volt 显示模式

//发送测试头信息
Tx_String("\n\r\n\rNEC Electronics America, Inc. 2006\n\r");
Tx_String("ADC Display Demo on M-Station\n\r\n\r");
Tx_String("Press SW2 to change display to hex value (most significant 8-bits of ADC result)\n\r");
Tx_String("Press SW3 to change display to volts\n\r");
Tx_String("Potentiometer varies voltage from 0.0 to 5.0V \n\r\n\r");
Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");


SRIF6=0;        // 清除读中断请求
SRMK6=0;        // 使能读请求中断

EI();           // 使能中断

while(1){       // 主循环
        WDTE=0xac;              // 清除看门狗定时器
        loop_count++;   // 消减循环计数

        if(ADIF)                // 如果 ADC 准备好
        {
                adc_result=ADCR;        // 获得结果
                adc_msbyte=(unsigned char)(ADCR>>8);    // 获得结果的高字节
                ADIF=0;                 // 清除 ADC 中断标志
        }

        if(display_state==VOLT_DISPLAY)                 // 如果是 volts 显示模式
        {
                volts=adc_msbyte/5;                                     // 接近 volt 值
                led1_value=volts/10;                    // 获得 led1 volt 单元

                P4=(led_values[led1_value]) & 0x0F;
```

```
        P5=(led_values[led1_value]>>4) & 0x0F;
        P5.3=0;                                                 // 显示小数点

        led2_value=volts%10;                    // 获得 led1 volt 的十分之一
        P7=(led_values[led2_value])|dp_LED2;    // 显示 led2 值

        if (volts!=old_volts){
                Tx_Character(CR_CHAR);
                Tx_Character(0x30+led1_value);
                Tx_Character('.');
                Tx_Character(0x30+led2_value);
                Tx_Character('V');
                old_volts=volts;
        }
}

else                                                            // 否则如果显示模式是 hex
{
        led1_value= adc_msbyte>>4;              // 获得 led1 高半位
        P4=(led_values[led1_value]) & 0x0F;
        P5=(led_values[led1_value]>>4) & 0x0F;
        P5.3=1;                                                 // 关闭小数点

        led2_value= adc_msbyte&0x0f;    // 获得 led2 低半位
        P7=(led_values[led2_value])|dp_LED2;    // 显示 led2 值

        if(adc_msbyte!=old_adcval){
                Tx_Character(CR_CHAR);
                Tx_Byte(adc_msbyte);
                Tx_Character('H');
                Tx_Character(' ');
                old_adcval=adc_msbyte;
        }
}

if((SWITCH_2==0)&& (SWITCH_3==1))               // 如果仅有 sw2 按下
{
        Delay(1000);

        if((SWITCH_2==0)&& (SWITCH_3==1))
        {
                display_state=VOLT_DISPLAY;             // 显示模式是 volts
        }

}

if((SWITCH_2==1)&& (SWITCH_3==0))               //如果仅有 sw3 按下
{
        Delay(1000);

        if((SWITCH_2==1)&& (SWITCH_3==0))
        {
                display_state=HEX_DISPLAY;              // 显示模式是 hex
```

```
        }

}

if(intRx)                                                   // 如果字符被接收
{
        DI();                                                       // 禁止中断

        Tx_String("\n\rStart bootloader Y/N?\n\r");  // 要求确认

        for(tcnt=0;tcnt<40;tcnt++)
        {
                WDTE=0xac;              // 清除看门狗定时器

                timedowncount=0xffff;   // 设置超时计数

                while(timedowncount>0)  // 当没有超时
                {
                        if(SRIF6)                       // 如果接收到字符
        {
                                received_char=Rx_Character();     // 获得字符
                                Tx_Character(received_char);      // 回放字符

                                if(received_char=='y' || received_char=='Y')
                                {
                                        bootRequest=1;  // 如果是 yes，则设置引导请求标志
                                        tcnt=40;        // 和设置用于退出的 tcnt
                                }
                                else                            // 如果不是 yes
                                {
                                        if(received_char=='n' || received_char=='N')
                                        {
                                                bootRequest=0;  // 如果没有清除引导请求标志
                                                tcnt=40;        //和设置用于退出的 tcnt
                                        }
                                        else
                                        {
                                                bootRequest=0;  // 清除引导请求标志
                                        }
                                }

                        }
                        timedowncount--;        // 消减超时计数
                }
        }

        if(bootRequest)         // 如果它是一个引导请求
        {                               // 设置 led1 和 led2 显示为 rE
                P4=0x0f;                // "r"
                P5-0x0a;
                P7=0x06;                // "E"

                while(1);               // 无限循环以迫使看门狗定时器超时
```

```
                }
                else    // 如果不是一个引导请求，发送返回应用信息
                {
                        Tx_String("\n\rReturned to ADC Display Demo on M-Station\n\r");
                        Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");
                }

                intRx=0;                                        // 清除中断标志
                EI();                                                   // 使能中断
        }

        WDTE=0xac;              // 清除看门狗定时器

        Delay(5000);    // 延时
    }
}

void Delay(unsigned int delay)
{
    unsigned int i;

    for(i=0;i<delay;i++);
}

/*****************************************************************************
*
* 功能:     UART 发送中断
* 参数:         无
* 返回值:     无
* 日期:       2004.3.12
* 描述:   UART6 接收的中断服务程序
*
* 注意:                 用于 bootloader 请求
*
*****************************************************************************/

__interrupt void UartRx_Interrupt(void)
{

    received_char=RXB6;     // 获得请求字符
    intRx=1;                // 设置中断标志
}
```

7.6 dicebu6.c

```
/*****************************************************************************
*
* 文件                    : dicebu6.c
* 日期                    : 2006 .2
* 描述                    : M-Station M-78F0537 的测试程序片段
* CPU 类型                : 78K0/KE2 (uPD78F0537D)
*
* 注意:                   : 应用的前提是具有 bootloader
*                         : 假设引导代码已经初始化 MCU 和 UART
*                         : 使用中断和库文件
*
*****************************************************************************/

// pragmas
#pragma ext_table 0x2000
#pragma SFR
#pragma NOP
#pragma DI
#pragma EI
#pragma interrupt INTSR6 UartRx_Interrupt rb1
#pragma interrupt INTTM50 TM50_Interrupt rb2

// 头文件
#include "uart6.h"
#include <stdlib.h>

// 定义
#define SWITCH_2 P3.1
#define SWITCH_3 P3.2

// 函数声明
void Init 定时器 50(void);
void Start 定时器 50(void);

// m-station 的 7 段 led 显示代码 (led1 and led2)
const unsigned char led_values[10]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};

// 变量
unsigned char int_count;
unsigned char led1_count;
unsigned char led2_count;
unsigned char led_index;
unsigned char dp_LED2;
unsigned int adc_result;
unsigned char adc_msbyte;
unsigned char ledcycle_speed;
unsigned int gp_int2;
unsigned int gp_int3;
unsigned int rand2;
unsigned int rand3;
```

```
unsigned int loop_count;
unsigned char doublesix;
unsigned char speedreported;
unsigned char highrollspeed;
char received_char;
unsigned char tcnt,bootRequest;
unsigned int timedowncount;
unsigned char intRx;

void main(){

        unsigned int i;
        DI();               // 禁止中断


        intRx=0;                    // 清除接收中断标志
        received_char='Q';          // 设置接收字符 Q
        bootRequest=0;              // 引导加载请求是假
        ADCS=1;                                 // 使能 ADC

        PM3.1 = 1;                      // 使能输入
        PM3.2 = 1;                      // 使能输入

        P4=0xff;
        P5=0xff;
        P7=0x00;                        // 关闭 7 段 led2 输出

        PM4=0x00;                       // 使能 7 段 led1 引脚输出
        PM5=0x00;
        PM7=0x00;                       //使能 7 段 led2 引脚输出

        dp_LED2=0x80;               // 给 led2 加小数点显示值
        led1_count=6;               // 初始化 led1 计数为 6
        led2_count=6;               // 初始化 led2 计数为 6
        doublesix=1;                // 设置两个六标志为真
        speedreported=0; // 设置显示速度值为 0
        highrollspeed=0; // 设置高摆动速度值为 0

        InitTimer50();              // 初始化定时器 50
        StartTimer50();             // 启动定时器 50

        WDTE=0xac;                      // 清除看门狗定时器
        SRIF6=0;                        // 清除 接收中断请求
        ADMK=1;                                 // ADC 中断是被屏蔽

        //发送测试头信息
        Tx_String("\n\r\n\rNEC Electronics America, Inc. 2006\n\r");
        Tx_String("Dice Rolling Demo on M-Station\n\r");
        Tx_String("SW2 rolls die displayed on seven segment display LED1\n\r");
        Tx_String("SW3 rolls die displayed on seven segment display LED2\n\r");
        Tx_String("Potentiometer controls the speed of the roll\n\r\n\r");
        Tx_String("What is the fastest roll you can throw that gets a double six?\n\r\n\r");
```

```
Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");


adc_result=ADCR;                                                    // 获得 ADC 结果
adc_msbyte=(unsigned char)(ADCR>>8);    // 获得高半位字节结果
ledcycle_speed=0xff-adc_msbyte;                     // 计算周期速度

if(ledcycle_speed>0x10)CR50=ledcycle_speed;       // 如果速度 > 10H，把速度值放在定时器比较寄存器
ADIF=0;                                                                     // 清除 ADC 中断标志
SRIF6=0;          // 清除接收中断请求
SRMK6=0;          // 使能接收中断

srand(adc_result);          // 具有 ADC 结果的种子随机数

EI();                               // 使能中断

while(1){
        WDTE=0xac;              // 清除看门狗定时器
        loop_count++;    // 消减循环计数
        {rand2=rand(); gp_int2=rand2;}              // 获得 2 号随机数
        {rand3=rand(); gp_int3=rand3;}              //获得 3 号随机数

        NOP();                      // 无操作

        if(doublesix && (!speedreported))  // 如果是两个六
        {                                                                   // 并且已经显示速度
                DI();                                                   // 禁止中断
                if(adc_msbyte>highrollspeed)        // 如果当前速度 > 高摆动速度
                {
                        highrollspeed=adc_msbyte;          // 设置高摆动速度到当前速度
                }

                Tx_String("Current Roll speed for double six = ");     // 发送摆动速度
                Tx_Byte(adc_msbyte);                        // ADC 结果的高半位值
                Tx_CRLF();                                              // 换行
                Tx_String("Highest Roll speed for double six = ");    // 发送摆动速度
                 Tx_Byte(highrollspeed);                    // 高摆动速度值
                Tx_CRLF();                                              // 换行
                Tx_CRLF();                                              // 换行

                loop_count=0;                                       // 清除循环计数
                speedreported=1;                            // 设置速度报告标志为真
                EI();                                                   // 使能中断
        }

        if(intRx)                                                       // 如果接收到字符
        {
                DI();                                                                   // 禁止中断

                Tx_String("\n\rStart bootloader Y/N?\n\r");  // 请求确认
```

```
for(tcnt=0;tcnt<40;tcnt++)
{
        WDTE=0xac;              // 清除看门狗定时器

        timedowncount=0xffff;   // 设置超时参数计数器

        while(timedowncount>0)  // 如果没有超时
        {
                if(SRIF6)                       // 如果接收到字符
                {
                        received_char=Rx_Character();   // 获得字符
                        Tx_Character(received_char);    // 回放字符

                        if(received_char=='y' || received_char=='Y')
                        {
                                bootRequest=1;  // 如果是‘Y’，则设置引导请求标志
                                tcnt=40;        // 并且设置 tcnt，用于退出
                        }
                        else                            // 如果不是‘Y’
                        {
                                if(received_char=='n' || received_char=='N')
                                {
                                        bootRequest=0;  // 如果没有清除引导请求标志
                                        tcnt=40;        // 并且设置 tcnt，用于退出
                                }
                                else
                                {
                                        bootRequest=0;  // 清除引导请求标志
                                }
                        }
                }

                timedowncount--;        // 消减超时参数计数器
        }
}

if(bootRequest)         // 如果是引导请求标志
{                               // 设置 led1 和 led2 显示为 rE
        P4=0x0f;                // "r"
        P5=0x0a;
        P7=0x06;                // "E"

        while(1);               // 无限循环以迫使看门狗定时器溢出
}

else    // 如果不是引导请求，则发送返回到应用信息
{
        Tx_String("\n\rReturned to Dice Rolling Demo on M-Station\n\r");
        Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");
}

intRx=0;                                        // 清除中断标志
EI();                                                   // 使能中断
```

```
                }

                if(bootRequest)             // 如果不是一个引导请求
                {                                       // 设置 led1 和 led2 显示为 rE
                        P4=0x0f;                // "r"
                        P5=0x0a;
                        P7=0x06;                // "E"

                        while(1);               // 无限循环以迫使看门狗定时器溢出
                }

                WDTE=0xac;              // 清除看门狗定时器
        }
}

/*****************************************************************************
*
* 功能:         UART 接收中断
* 参数:         无
* 返回值:       无
* 日期:         2005.1.21
* 描述:         UART6 接收中断服务程序
*
* 注意:         用于 bootloader 请求
*
*****************************************************************************/

__interrupt void UartRx_Interrupt(void)
{

        received_char=RXB6;     // 获得中断字符
        intRx=1;                // 设置中断标志
}

/*****************************************************************************
*
* 功能:       TM50 中断
* 参数:       无
* 返回值:     无
* 日期:       2005.1.21
* 描述:       定时器 50 中断服务程序
*
* 注意:       用于产生脉冲模式
*
*****************************************************************************/

__interrupt void TM50_Interrupt(void)
{
        int_count++;                // 消减中断计数

        // 如果不是双六，并且速度报告标志非 0
        if((led1_count!=6) || (led2_count!=6)){doublesix=0;speedreported=0;}
```

```
// 如果是双六，既不是开关 2，也不是开关 3 被按下
if((led1_count==6)&&(led2_count==6)&&(SWITCH_2==1)&&(SWITCH_3==1))
{
        doublesix=1;                            // 设置双六标志
        if((int_count&0x03)==0)          // 每到第 4 个时中断
        {
                P4=0xff;                        // 关闭 7 段 led1
                P5=0xff;
                P7=0xff;                        // 关闭 7 段 led2
        }
        else                                                    // 否则对于所有中断中断
        {
                P4=(led_values[6]) & 0x0f;                      // 在 7 段 led1 显示 6
                P5=(led_values[6]>>4) & 0x0f;
                P5.3 = 1;

                P7=(led_values[6])|dp_LED2;     // 在 7 段 led2 显示 6

        }
}
if((int_count&0x07)==0)                 // 每到第 8 个中断
{
        if(SWITCH_2==0)                                 // 如果开关 2 被按下
        {

                led1_count=(unsigned char)gp_int2;      // 获得 led1 计数
                led1_count&=0x07;                               // 使用低 3 位
                if(led1_count>6)led1_count=1;   // 改变 7 为 1
                if(led1_count==0)led1_count=6;  // 改变 0 为 6
                P4=(led_values[led1_count]) & 0x0f;             // 显示 led1 计数
                P5=(led_values[led1_count]>>4) & 0x0f;          // 显示 led1 计数
                P5.3 = 1;

        }
        if(SWITCH_3==0)                                 // 如果开关 3 被按下
        {
                led2_count=(unsigned char)gp_int3;      // 获得 led2 计数器
                led2_count&=0x07;                               // 使用低 3 位
                if(led2_count>6)led2_count=1;   //改变 7 为 1
                if(led2_count==0)led2_count=6;  //改变 0 为 6
                P7=(led_values[led2_count])|dp_LED2;    // 显示 led2 计数
        }


        if(ADIF)                                                // 如果 ADC 已经准备好
        {
                adc_result=ADCR;                // 获得 ADC 结果
                adc_msbyte=(unsigned char)(ADCR>>8);    // 获得结果的高半位
                ledcycle_speed=0xff-adc_msbyte;         // 计算周期速度
                if(ledcycle_speed>0x10)CR50=ledcycle_speed;     // 如果 speed > 10H，放速度在定时器
```

比较寄存器中

```
                    ADIF=0;                                    // 清除 ADC 中断标志
                }
        }
}
/*****************************************************************************
*
* 功能:       初始化 timer_50
* 参数:         无
* 返回值:       无
* 日期:         2005.1.21
* 描述:    初始化定时器/事件计数器 50
*
* 注意:
*
*****************************************************************************/

void InitTimer50(void)
{
        TMC50=0x00;                         // 停止定时器, 当与 CR50 匹配时清除和 启动
        TCL50=0x07;                         // 设置最慢时钟频率
        CR50=0xff;                          // 设置比较计数器
        PR1L |= 0x02;                   // 设置低优先级中断
        TMMK50=1;                           // 屏蔽中断
        TMIF50=0;                           // 清除中断请求

}
/*****************************************************************************
*
* 功能:       启动定时器 50
* 参数:         无
* 返回值:       无
* 日期:         2005.1.21
* 描述:    启动定时器/事件计数器 50 和使能中断
*
* 注意:
*****************************************************************************/

void StartTimer50(void)
{
        TCE50=1;                            // 启动定时器 50
        TMIF50=0;                           // 清除清除中断请求
        TMMK50=0;                           // 解除屏蔽中断
}
```

7.7 UART6.h

```
/*****************************************************************************
*
* FILE                    : UART6.h
* 日期                    : 2006.2
* 描述                    : M-78F0537 的头文件
* CPU 类型                : 78K0/KE2 - 78F0537D
*
* 注意:           :
*
*****************************************************************************/
#ifndef _UART6_H
#define _UART6_H


#ifdef    UART6_C
#define UART6_GLOBAL
#else
#define UART6_GLOBAL extern
#endif

UART6_GLOBAL char rx_char;
#define BAUD_115200_20 87       // 此值用于 20M 时钟产生 115200 波特率


// 字符定义
#define NULL_CHAR 0x00                  // NULL 字符
#define TAB_CHAR 0x09                   // Tab 字符
#define LF_CHAR 0x0a                    // 换行字符
#define CR_CHAR 0x0d                    // 回车字符
#define ESC_CHAR 0x1B                   // 换码符
#define SP_CHAR 0x20                    // 空格字符
#define PROMPT_CHAR '>'                 // 终端提示字符
#define X_ON 0x11                       // XON 控制
#define X_OFF 0x13                      // XOFF 控制


// 函数原型
void InitUART6_8N1(unsigned char baudRate);
void Tx_String(const char *puc);
void Tx_Word(unsigned int data_16_bit);
void Tx_Byte(unsigned char data_8_bit);
void Tx_CRLF(void);
void Tx_Character(char ascii_character);
char Rx_Character(void);


#endif  /* _UART6_H */
```

7.8 epvdata.h

```
/*****************************************************************************
*
* FILE                    : epvdata.h
* 日期                    : 2006
* 描述                    : M-78F0537 的头文件
* CPU 类型                : 78K0/KE2 - uPD78F0537D
*
* 注意:           :
*
*****************************************************************************/
#ifndef _EPVDATA_H
#define _EPVDATA_H


#ifdef    EPVDATA_C
#define EPVDATA_GLOBAL
#else
#define EPVDATA_GLOBAL extern
#endif

#define MDB_SIZE 136   // 数据缓冲大小
EPVDATA_GLOBAL unsigned char myDataBuffer[MDB_SIZE];// 用于编程的数据缓冲器
EPVDATA_GLOBAL unsigned char db_index;                    // 数据缓冲索引
EPVDATA_GLOBAL unsigned char numberOfWords;         // 被编程的字数
EPVDATA_GLOBAL unsigned short programmingAddress;// 编程数据开始地址
EPVDATA_GLOBAL unsigned char firstProgram;          // 首次编程标志
EPVDATA_GLOBAL unsigned char oddBytes;                    // 字节数需要满足字数
EPVDATA_GLOBAL unsigned long *validApp_ptr;         // 指向有效的应用字
EPVDATA_GLOBAL unsigned long validAppWord; // FLASH 区的应用校验和
EPVDATA_GLOBAL unsigned int timeOut;                      // 超时计数器
EPVDATA_GLOBAL unsigned char i;                           // 字节 i
EPVDATA_GLOBAL unsigned int int_i;                        // 整型 i
EPVDATA_GLOBAL unsigned char bootSwapFlag; // 引导交换标志
EPVDATA_GLOBAL unsigned char bootDownLoad;          // 引导加载标志

// 通用定义
#define FALSE 0
#define TRUE 1
#define OUTPUT 0
#define INPUT 1

typedef  unsigned short    USHORT;
typedef  unsigned char     UCHAR;

// 常量定义
#define MAX_COUNT 0xffff

#define ENTRY_RAM_SIZE 100

// 应用定义
```

```
#define FLMD0_CONTROL_PIN P3.0          // 此端口用于设置 FLMD0 引脚
#define FLMD0_CONTROL_IO PM3.0          // 端口输入输出配置

// 自编程定义
#define FIRST_BOOT_BLOCK 4              // 如果是引导交换，则首个引导块一定是 4
#define LAST_BOOT_BLOCK 7               // 最后引导块一定是 7
#define FIRST_FLASH_BLOCK_C 8                   // 共用区的首个应用块
#define LAST_FLASH_BLOCK_C 31                   //共用区的最后的应用块
#define FIRST_FLASH_BLOCK_B 32                  // 页区的首个应用块
#define LAST_FLASH_BLOCK_B 47                   //页区的最后应用块

#define CS_FLASH_BLOCK      47                  //校验和存储的页和块
#define CS_FLASH_BANK       5

#define CS_ADDRESS    0xBFFC        //校验和存储在第 5 页的地址
#define NOT_BLANK                   0x1b

// 函数原型
void programDataBuffer(void);
void Disable_SFP(void);
void Enable_SFP(void);
void EraseFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char lastBlock);
void WriteDataBufferToFlash(unsigned char Banknumber, unsigned short writeAddress,unsigned char wordCount);
void VerifyFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char
lastBlock);
unsigned long CalculateAppCheckSum(void);
void WriteVerifyAppCheckSum(void);
unsigned char ConfirmOnPrompt(void);
unsigned char CheckBootSwapFlag(void);
void ToggleBootSwapFlag(void);
void performBootSwap(void);

#endif  /* _EPVDATA_H */
```

7.9 getdata.h

```
/*****************************************************************************
*
* FILE                  : getdata.h
* 日期                  : 2005 .1.17
* 描述                  : M-78F0148H 的引导头文件
* CPU 类型              : 78K0/KF1+ - uPD78F0148H
*
* 注意:          :
*
*****************************************************************************/
#ifndef _GETDATA_H
#define _GETDATA_H


#ifdef    GETDATA_C
#define GETDATA_GLOBAL
#else
#define GETDATA_GLOBAL extern
#endif


#define RB_DATA_SIZE 354              // 接收缓冲区数据容量
#define RB_OVER_SIZE  32              // 接收缓冲区超出容量
#define RB_SIZE (RB_DATA_SIZE+RB_OVER_SIZE)      // 总的接收缓冲区容量

#define SWITCH_2 P3.1
#define SWITCH_3 P3.2




// 函数原型
void GetFile(void);
unsigned char GetHexByte(void);


#endif  /* _GETDATA_H */
```

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; 系统     : 自编程库(标准模式)
; 文件名 : SelfLibrary_normal.asm
; 版本     : 2.00
; 目标 CPU   : 78K0/Kx2
; 最后更改日期 : 2005/07/08
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PUBLIC_FlashStart
PUBLIC_FlashEnd
PUBLIC_FlashEnv
PUBLIC_FlashBlockErase
PUBLIC_FlashWordWrite
PUBLIC_FlashBlockVerify
PUBLIC_FlashBlockBlankCheck
PUBLIC_FlashGetInfo
PUBLIC_FlashSetInfo
PUBLIC_CheckFLMD
;PUBLIC          _EEPROMWrite                    ;自编程程序中不需要 (从代码中删除)

;---------------------------------------------------------------------
;       EQU 设置
;---------------------------------------------------------------------
FLASH_ENV                       EQU     00H     ;初始化
FLASH_BLOCK_ERASE               EQU     03H     ;块擦除
FLASH_WORD_WRITE                EQU     04H     ;字写
FLASH_BLOCK_VERIFY                      EQU     06H     ;块校验
FLASH_BLOCK_BLANKCHECK          EQU     08H     ;块页核查
FLASH_GET_INF                           EQU     09H     ;读取 Flash 存储器信息
FLASH_SET_INF                           EQU     0AH     ;设置 Flash 存储器信息
FLASH_CHECK_FLMD                EQU     0EH     ;模式核查
FLASH_EEPROM_WRITE              EQU     17H     ;EEPROM 写

FLASHFIRM_PARAMETER_ERROR       EQU     05H     ;参数错误

BANK_BLC_ERROR                  EQU     0FFH    ;页码错误(块)
BANK_ADDR_ERROR                 EQU     0FFFFH; 页码错误(地址)


SELF_PROG       CSEG

;---------------------------------------------------------------------
; 函数名                : _FlashStart
; 输入                  : 无
; 输出                  : 无
; 破坏寄存器            : 无
; 概要                  : 自编程启动处理
;---------------------------------------------------------------------
_FlashStart:
        MOV     PFCMD,#0A5H             ;PFCMD 寄存器控制
        MOV     FLPMC,#001H             ;FLPMC 寄存器控制(设置值)
        MOV     FLPMC,#0FEH             ;FLPMC 寄存器控制(反向设置值)
```

```
        MOV     FLPMC,#001H             ;FLPMC 寄存器控制(设置值)
        RET

;---------------------------------------------------------------------
; 函数名          : _FlashEnd
; 输入            : 无
; 输出            : 无
; 破坏寄存器      : 无
; 概要            : 自编程结束处理
;---------------------------------------------------------------------
_FlashEnd:
        MOV     PFCMD,#0A5H             ;PFCMD 寄存器控制
        MOV     FLPMC,#000H             ;FLPMC 寄存器控制 (设置值)
        MOV     FLPMC,#0FFH             ;FLPMC 寄存器控制(反向设置值)
        MOV     FLPMC,#000H             ;FLPMC 寄存器控制 (设置值)
        RET

;---------------------------------------------------------------------
; 函数名          : _FlashEnv
; 输入            : AX=入口 RAM 地址
; 输出            : 无
; 破坏寄存器      : 无
; 概要            : 初始化自编程处理
;---------------------------------------------------------------------
_FlashEnv:
;初始化处理
        PUSH    PSW                     ;存储组寄存器到堆栈中
        PUSH    AX
        SEL     RB3                     ;设置寄存器组 3.
        POP     HL                      ;设置入口 RAM 地址到 HL 寄存器
        MOV     C,#FLASH_ENV            ;设置功能号到 C 寄存器
        CALL    !8100H                  ;调用 flash 固件

        MOV     A, #09H
        MOV     [HL+13H], A             ;设置块擦除重试号码
        MOV     [HL+14H], A             ; 设置芯片擦除重试号码

        POP     PSW                     ;从堆栈恢复组寄存器.
        RET

;---------------------------------------------------------------------
; 函数名: _FlashBlockErase
; 输入            : AX=擦除页码
;                           STACK=擦除块码
; 输出   : BC=状态
; 破坏寄存器      : AX,BC
; 概要            : 指定块擦除(1K 字节).
;---------------------------------------------------------------------
_FlashBlockErase:
        PUSH    HL
```

```
;根据块和页码计算擦除块号码
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV  A,[HL+4]              ;读取堆栈数据(=擦除块号码)
        MOV  B,A
        MOV  A,C                   ;A...擦除页, B...擦除块号码
        CALL !ExchangeBlockNum     ; 根据块和页码计算擦除块号码.
                                   ;(返回 A=计算之后擦除的块号码)
        BZ   $FBE_PErr             ;如果页号超出范围，则出错

;块擦除处理
        PUSH PSW                   ;在堆栈中存储组寄存器
        PUSH AX
        SEL  RB3                   ;设置为组寄存器 3.
        POP  AX
        MOV  [HL+3],A              ;设置入口 RAM+3，以擦除计算出的块号码
        MOV  C,#FLASH_BLOCK_ERASE      ;设置功能号到 C 寄存器
        CALL !8100H                ;调用 flash 固件
        POP  PSW                   ;从堆栈中恢复组寄存器

;获得 flash 错误信息
        MOV  A,0FEE3H              ;设置 flash 固件错误信息到返回值中
                                   ;(0FEE3H = 组寄存器 3 的 B 寄存器)
        BR   FlashBlockErase00

;参数错误
FBE_PErr:
        MOV  A,#FLASHFIRM_PARAMETER_ERROR  ;设置参数错误到返回值

FlashBlockErase00:
        MOV  C,A
        MOV  B,#00H
        POP  HL
        RET

;---------------------------------------------------------------------
; 功能号: _FlashWordWrite
; 输入          : AX= 写开始地址结构的地址
;                     (结构成员...写开始地址   写开始地址的页)
;                     STACK1=写数据号码
;                     STACK2=写数据缓冲地址
; 输出          : BC=状态
; 破坏寄存器    : AX,BC,DE
; 概要          : RAM 的数据被写到 flash 存储器
;                 每次写 256 字节 或更少 (每 4 个字节)
;---------------------------------------------------------------------
_FlashWordWrite:
        PUSH HL

;根据写地址和页号寄存器写地址
        MOVW DE,AX
```

```
        MOVW AX,SP
        MOVW HL,AX
        MOV  A,[HL+4]           ;读取堆栈数据(=写数据号码)
        MOV  B,A
        MOV  A,[HL+6]           ;读取堆栈数据(=写数据缓冲地址)
        XCH  A,X
        MOV  A,[HL+7]
        MOVW HL,AX
        MOVW AX,DE              ;AX...写开始地址结构的地址
                                ;B...写数据号码,HL...写数据缓冲区地址
        CALL !ExchangeAddress   ;根据结构成员的写地址和页号计算写地址
                                ;(Return AX=写地址)
        BZ   $FWW_PErr          ;如果页号超出范围，则出错

;字写处理
        PUSH PSW                ;在堆栈中存储组寄存器
        PUSH AX
        PUSH BC
        PUSH HL
        SEL  RB3                ;设置寄存器组 3
        POP  AX
        MOV  [HL+5],A           ;设置入口 RAM+5 到写数据缓冲区的高地址
        MOV  A,X
        MOV  [HL+4],A           ;设置入口 RAM+4 到写数据缓冲区的低地址
        POP  AX
        MOV  [HL+3],A           ;设置入口 RAM+3 到写数据号码
        MOV  A,X
        MOV  [HL+0],A           ;设置入口 RAM+0 到写地址低字节
        POP  AX
        MOV  [HL+2],A           ;设置入口 RAM+2 到写地址最高字节
        MOV  A,X
        MOV  [HL+1],A           ;设置入口 RAM+1 到写地址高字节
        MOV  C,#FLASH_WORD_WRITE    ;设置功能号到 C 寄存器
        CALL !8100H             ;调用 flash 固件
        POP  PSW                ;恢复堆栈中的组寄存器

;获得 flash 固件错误信息
        MOV  A,0FEE3H           ;设置 flash 固件错误信息到返回值
                                ;(0FEE3H=组 3 中的 B 寄存器)
        BR   FlashWordWrite00

;参数错误
FWW_PErr:
        MOV  A,#FLASHFIRM_PARAMETER_ERROR  ;设置参数错误到返回值

FlashWordWrite00:
        MOV  C,A
        MOV  B,#00H
        POP  HL
        RET

;---------------------------------------------------------------------
```

```
; 函数名          : _FlashBlockVerify
; 输入                    : AX=核查页
;                          STACK=核查块号
; 输出            : BC=状态
; 破坏寄存器      : AX,BC
; 概要            : 指定块的内部核查 (1K 字节).
;-------------------------------------------------------------------
_FlashBlockVerify:
        PUSH    HL

;根据块号和页号计算核查块号.
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV     A,[HL+4]                ;读取堆栈数据(=核查块号)
        MOV     B,A
        MOV     A,C                     ;A...核查页, B...核查块号
        CALL    !ExchangeBlockNum       ;根据块号和页计算块号
                                        ;(返回 A=计算后核查块号)
        BZ      $FBV_PErr               ;如果页号超出范围，则出错

;Block verify processing
        PUSH    PSW                     ;存储组寄存器到堆栈中.
        PUSH    AX
        SEL     RB3                     ;选择组寄存器 3
        POP     AX
        MOV     [HL+3],A                ;计算完之后，设置入口 RAM+3 到核查块号
        MOV     C,#FLASH_BLOCK_VERIFY       ;设置功能号到 C 寄存器
        CALL    !8100H                  ;调用 flash 固件
        POP     PSW                     ;从堆栈中恢复组寄存器.

;获得 flash 固件错误信息
        MOV     A,0FEE3H                ;设置 flash 固件错误信息到返回值
                                        ;(0FEE3H = 组 3 的 B 寄存器)
        BR      FlashBlockVerify00

;参数错误
FBV_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;设置参数错误到返回值

FlashBlockVerify00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-------------------------------------------------------------------
;函数名 : _FlashBlockBlankCheck
; 输入                    : AX=空白核查页
;                          STACK=空白核查块号
; 输出            : BC=状态
; 破坏寄存器      : AX,BC
```

```
; 概要          : 指定块的空白核查 (1K 字节).
;---------------------------------------------------------------------
_FlashBlockBlankCheck:
        PUSH    HL

;根据块号和页号计算空白核查块号.
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV     A,[HL+4]                ;读取堆栈数据(=空白核查块号)
        MOV     B,A
        MOV     A,C                     ;A...空白核查页, B...空白核查块号
        CALL    !ExchangeBlockNum       ;块号码是通过块号和页码计算获得.
                                        ;(返回 A=计算之后空白核查块号)
        BZ      $FBBC_PErr              ;如果页号超出范围，则将出错.

;块空白核查处理
        PUSH    PSW                     ;在堆栈中存储组寄存器.
        PUSH    AX
        SEL     RB3                     ;选择组寄存器 3.
        POP     AX
        MOV     [HL+3],A                ;计算之后，设置入口 RAM+3 到空白核查块号
        MOV     C,#FLASH_BLOCK_BLANKCHECK       ;设置功能号 C 寄存器
        CALL    !8100H                  ;调用 flash 固件
        POP     PSW                     ;从堆栈中恢复组寄存器.

;获得 flash 固件错误信息
        MOV     A,0FEE3H                ;设置 flash 固件错误信息到返回值
                                        ;(0FEE3H =组 3 的 B 寄存器)
        BR      FlashBlockBlankCheck00

;参数错误
FBBC_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;设置参数错误到返回值

FlashBlockBlankCheck00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名          : _FlashGetInfo
; 输入                    : AX= flash 信息采集结构地址
;                           (成员结构              ...选项号
;                                                    页号
;                                                    块号)
;                         STACK=获得的数据被存储到缓冲器的第一个地址
; 输出            : BC=状态
; 破坏寄存器      : AX,BC,DE
; 概要            : flash 寄存器的启动信息.
;---------------------------------------------------------------------
```

```
_FlashGetInfo:
        PUSH    HL

;选项号的核查
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV     A,[HL+4]            ;读取堆栈数据(=获得的数据被存储到缓冲器的第一个地址)
        XCH     A,X
        MOV     A,[HL+5]
        XCHW  AX,BC                 ;AX... flash 信息采集结构地址
                                    ;BC... 获得的数据被存储到缓冲器的第一个地址
        MOVW HL,AX
        MOVW AX,BC
        MOVW DE,AX
        MOV     A,[HL+0]            ;从 flash 信息采集结构中读取数据(=选项号)
        CMP     A,#05H              ;选项号= 5？
        BNZ     $FlashGetInfo10     ;否

;根据块号和页寄存器计算块号.
        MOV     X,A
        MOV     A,[HL+2]            ; 从 flash 信息采集结构中读取数据(=块号)
        MOV     B,A
        MOV     A,[HL+1]            ; 从 flash 信息采集结构中读取数据(=页寄存器)
                                    ;A...页, B...块号
        CALL    !ExchangeBlockNum   ; 根据块号和页寄存器计算块号.
                                    ;(返回 A=计算之后的块号)
        BZ      $FlashGetInfo20     ;如果页号超出范围，则将出错.
        XCH     A,X                 ;A...选项号, X...块号

;获得信息处理(当选项号 = 5)
        PUSH    PSW                 ;在堆栈中存储寄存器页号.
        PUSH    DE
        PUSH    AX
        SEL     RB3                 ;选择组寄存器 3.
        POP     AX
        XCH     A,X
        MOV     [HL+0],A            ;设置入口 RAM+0 到块号
        MOV     A,X                 ;A...选项号
        BR      FlashGetInfo40

;选项号错误核查
FlashGetInfo10:
        CMP     A,#03H              ;选项号 = 3？
        BZ      $FlashGetInfo30     ;是
        CMP     A,#04H              ;选项号 = 4？
        BZ      $FlashGetInfo30     ;是
FlashGetInfo20:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;参数错误被返回, 除非选项号是 3, 4 和 5.
        BR      FlashGetInfo50

;获得信息处理(当选项号= 3,4)
```

```
FlashGetInfo30:
        PUSH    PSW                     ;在堆栈中存储寄存器页.
        PUSH    DE
        PUSH    AX
        SEL     RB3                     ;选择寄存器组 3.
        POP     AX
FlashGetInfo40:
        MOV     [HL+3],A                ;设置入口 RAM+3 到选项号
        POP     AX
        MOV     [HL+5],A                ;设置入口 RAM+5 到存储缓冲区高地址
        MOV     A,X
        MOV     [HL+4],A                ;设置入口 RAM+4 到存储缓冲低地址
        MOV     C,#FLASH_GET_INF        ;设置功能号到 C 寄存器
        CALL    !8100H                  ;调用 flash 固件
        POP     PSW                     ;从堆栈中恢复组寄存器.

;计算存储缓冲区和页地址.当选项号=3 或 4 或 Bank = 0 时无操作
;或块号(先前) < 32 或块号(先前) >= 48.
;A=选项号, B=页号, C...块号(先前), DE=存储获得数据的首地址到缓冲区中
        CMP     A,#05H                  ;选项= 5 ?
        BNZ     $ReturnAddress_end      ;否
        MOV     A,B
        CMP     A,#0                    ;页 = 0 ?
        BZ      $ReturnAddress_end      ;是
        XCH     A,C
        CMP     A,#32                   ;块号(先前) < 32 ?
        BC      $ReturnAddress_end      ;是
        CMP     A,#48                   ;块号(先前) >= 48 ?
        BNC     $ReturnAddress_end      ;是
        MOV     A,C

;地址的计算(40H*页来源两个类型的高字节.低地址保持不变.)
        XCHW    AX,DE
        MOVW    HL,AX
        MOV     A,[HL+1]
        MOV     X,A
        MOV     A,[HL+2]                ;A...最高地址, X...较高地址
        XCHW    AX,DE                   ;A...页, D...最高地址, E...较高地址
        MOV     [HL+2],A                ;设置存储缓冲器+2 到页.
        MOV     X,#0
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROLC    A,1
        XCH     A,X
        ROLC    A,1                     ;AX=40H*页
        XCHW    AX,DE
        XCH     A,X
        SUB     A,E
        XCH     A,X
```

```
        SUBC    A,D
        MOV     A,X
        MOV     [HL+1],A            ;设置存储缓冲器+1 到计算地址(高).
ReturnAddress_end:

;获得 flash 固件错误信息
        MOV     A,0FEE3H            ;设置 flash 固件错误信息到返回值
                                    ;(0FEE3H = 组 3 的 B 寄存器)
FlashGetInfo50:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名: _FlashSetInfo
; 输入          : AX=Flash 信息数据
; 输出          : BC=状态
; 破坏寄存器    : A,BC
; 概要          : flash 信息的设置.
;---------------------------------------------------------------------
_FlashSetInfo:
;设置信息处理
        MOV     A,X
        PUSH    AX                  ;在堆栈中存储 Flash 信息数据
        PUSH    PSW                 ;在堆栈中存储组寄存器.
        SEL     RB3                 ;选择寄存器组 3.
        MOVW    AX,SP
        ADDW    AX,#2
        MOV     [HL+5],A            ; 设置入口 RAM+5 到 flash 信息数据高地址，以安全堆栈
        MOV     A,X
        MOV     [HL+4],A            ;设置入口 RAM+4 到 flash 信息数据低地址，以安全堆栈
        MOV     C,#FLASH_SET_INF    ;设置功能号到 C 寄存器
        CALL    !8100H              ;调用 flash 固件
        POP     PSW                 ;从堆栈中恢复组寄存器.
        POP     AX

;获得 flash 固件错误信息
        MOV     A,0FEE3H            ;设置 flash 固件错误信息返回值
                                    ;(0FEE3H = 组 3 的 B 寄存器)
        MOV     C,A
        MOV     B,#00H
        RET

;---------------------------------------------------------------------
; 函数名: _CheckFLMD
; 输入          : 无
; 输出          : BC=状态
; 破坏寄存器    : A,BC
; 概要    : 核查 FLMD 引脚的电压.
;---------------------------------------------------------------------
_CheckFLMD:
```

```
;设置信息处理
        PUSH    PSW                     ;在堆栈中存储组寄存器.
        SEL     RB3                     ;选择寄存器组.
        MOV     C,#FLASH_CHECK_FLMD     ;设置功能号到 C 寄存器
        CALL    !8100H                  ;调用 flash 固件
        POP     PSW                     ;从堆栈中恢复组寄存器

;获得 flash 固件错误信息
        MOV     A,0FEE3H                ;设置 flash 固件错误信息到返回值
                                        ;(0FEE3H = 组 3 的 B 寄存器)
        MOV     C,A
        MOV     B,#00H
        RET

;---------------------------------------------------------------------
; 函数名: ExchangeBlockNum
; 输入          : A=页
;                 B=块号
; 输出    : A=块号(新)
;                 B=页号
;                 C=块号(先前的)
; 概要    : 块号被转变为页信息的真正地址.
;---------------------------------------------------------------------
ExchangeBlockNum:
;从 32 到 47 计算块号.
        XCH     A,B
        CMP     A,#32
        BC      $EBN_end
        CMP     A,#48
        BNC     $EBN_end

;块号的计算(页*16 被加到块号)
        XCH     A,B
        MOV     C,A                     ;C...页
        CMP     A,#6
        BNC     $EBN_error_end
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1                     ;A=16*页
        ADD     A,B
        XCH     A,C
        XCH     A,B
        XCH     A,C                     ;A=计算之后的块号, B=页号,
                                        ;C=计算之前的块号
        BR      EBN_end

;页错误
EBN_error_end:
        MOV     A,#BANK_BLC_ERROR ;返回错误号

EBN_end:
```

```
        CMP     A,#BANK_BLC_ERROR ;Bank error ?
        RET

;---------------------------------------------------------------
; 函数名 : ExchangeAddress
; 输入          : AX=写开始地址结构的地址
;                       (存储器结构...写开始结构
;                                       写开始地址页)
; 输出   : AX=写开始地址(分成两个高地址)
;                C=写开始地址(低地址)
; 概要   : 写结构的开始地址被转变为页信息的真正的地址.
;---------------------------------------------------------------
ExchangeAddress:
        PUSH    HL

;从地址 8000H 到 BFFFH 计算.
        MOVW HL,AX
        MOV     A,[HL+0]                ; 从开始地址结构中读取数据(=写地址)
        MOV     X,A
        MOV     A,[HL+1]
        CMPW    AX,#8000H
        BC      $EA_end
        CMPW    AX,#0C000H
        BNC     $EA_end

;地址的计算(页*40H 被分别加到两个高字节上.低地址保持不变)
        MOV     D,A
        XCH     A,X
        MOV     C,A
        MOV     X,#0
        MOV     A,[HL+2]                ;从开始地址结构中读取数据
                                        ;(=写开始地址的页)
        CMP     A,#6
        BNC     $EA_error_end
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROLC    A,1
        XCH     A,X
        ROLC    A,1                     ;AX=40H*页
        XCH     A,X
        ADD     A,D                     ;较高地址的增加
        XCH     A,X
        ADDC    A,#0                    ;最高地址的增加
                                        ;A.... 计算之后的最高地址
                                        ;X.... 计算之后的较高地址, C... 低地址
        BR      EA_normal_end

;页号错误
EA_error_end:
        MOVW AX,#BANK_ADDR_ERROR
```

```
        BR      EA_normal_end

EA_end:
        XCH     A,X
        MOV     C,A
        MOV     A,#0                    ;A....计算之后的最高地址
                                        ;X....计算之后的较高地址, C...低地址
EA_normal_end:
        POP     HL
        CMPW    AX,#BANK_ADDR_ERROR     ;页号错误?
        RET


        END
```

7.10 option.asm

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; 系统      : 选项字节设置
; 文件名    : option.asm
; 版本      : 0.00
; 目标 CPU  : 78K0/Kx2
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; 为 78F0537 系列指定选项字节
; 选项字节在 0080H 或 1080H (当使用引导交换时)
; OB.7 = 0            第 7 位设置为 0
; OB.6 = WINDOW1      窗口 1/2
; OB.5 = WINDOW2      00 = 25% 打开看门狗窗口周期
;                     01 = 50%打开看门狗窗口周期
;                     10 = 75%打开看门狗窗口周期
;                     11 = 100%打开看门狗窗口周期(却省)
; OB.4 = WDTON        0=看门狗计数器禁止, 1=看门狗计数器使能
; OB.3 = WDCS2        WDCS[2:0] = 看门狗溢出时间
; OB.2 = WDCS1
; OB.1 = WDCS0
; OB.0 = INTERNALOSC      0=可以通过软件禁止, 1=不能被禁止

OPT    CSEG AT 0080H

OPTION:          DB 01111110B   ; 第 7  位 = 0
                                ; 第 6，5 位  = 11  = 100% 打开窗口周期
                                ; 第 4 位   = 1   = 看们狗使能
                                ; 第 3，2，1 位 = 111 = 496 ms.
                                ; 第 0 位   = 0   = 内部震荡器可以通过软件禁止
```