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

78K0S/Kx1+

8 位单片微控制器

EEPROM™ 模拟

μPD78F9200
μPD78F9201
μPD78F9202
μPD78F9210
μPD78F9211
μPD78F9212
μPD78F9221
μPD78F9222
μPD78F9232
μPD78F9234
μPD78F9500
μPD78F9501
μPD78F9502
μPD78F9510
μPD78F9511
μPD78F9512

文档编号： U17379CA2V0AN00 (第二版)
发行日期： 2007 年 12 月 NS

[备忘录]

CMOS设备的注释

① 输入引脚处的电压波形

输入噪音或一个反射波引起的波形失真可能导致错误发生。如果由于噪音等的影响使CMOS设备的输入电压范围保持在V_{IL}（MAX）和V_{IH}（MIN）之间，设备可能发生错误。在输入电平固定时以及输入电平从V_{IL}（MAX）过渡到V_{IH}（MIN）时的传输期间，要防止散射噪声影响设备。

② 未使用的输入引脚的处理

CMOS设备的输入端保持开路可能导致误操作。如果一个输入引脚未被连接，则由于噪音等原因可能会产生内部输入电平，从而导致误操作。CMOS设备的操作特性与Bipolar或NMOS设备不同。CMOS设备的输入电平必须借助上拉或下拉电路固定在高电平或低电平。每一个未使用引脚都应该通过附加电阻连接到VDD或GND。如果有可能尽量定义为输出引脚。对未使用引脚的处理因设备而异，必须遵循与设备相关的规定和说明。

③ ESD防护措施

如果MOS设备周围有强电场，将会击穿氧化栅极，从而影响设备的运行。因此必须采取措施，尽可能防止静电产生。一旦有静电，必须立即释放。对于环境必须有适当的控制。如果空气干燥，应当使用增湿器。建议避免使用容易产生静电的绝缘体。半导体设备的存放和运输必须使用抗静电容器、抗静电屏蔽袋或导电材料容器。所有的测试和测量工具包括工作台和工作面必须良好接地。操作员应当佩戴静电消除手带以保证良好接地。不能用手直接接触半导体设备。对于装配有半导体设备的PW板也应采取类似的静电防范措施。

④ 初始化之前的状态

在上电时MOS设备的初始状态是不确定的。在刚刚上电之后，具有复位功能的MOS设备并没有被初始化。因此上电不能保证输出引脚的电平，I/O设置和寄存器的内容。设备在收到复位信号后才进行初始化。具有复位功能的设备在上电后必须立即进行复位操作。

⑤ 电源开关顺序

在一个设备的内部操作和外部接口使用不同的电源的情况下，按照规定，应先在接通内部电源之后再接通外部电源。当关闭电源时，按照规定，先关闭外部电源再关闭内部电源。如果电源开关顺序颠倒，可能会导致设备的内部组件过电压，产生异常电流，从而引起内部组件的误操作和性能的退化。

对于每个设备电源的正确开关顺序必须依据设备的规范说明分别进行判断。

⑥ 电源关闭状态下的输入信号

不要向没有加电的设备输入信号或提供I/O上拉电源。因为输入信号或提供I/O上拉电源将引起电流注入，从而引起设备的误操作，并产生异常电流，从而使内部组件退化。

每个设备电源关闭时的信号输入必须依据设备的规范说明分别进行判断。

EEPROM 是 NEC Electronics Corporation 的注册商标。
SuperFlash®是 Silicon Storage Technology,Inc（SST 超捷）的注册商标,在包括美国日本的多个国家注册.

注意事项：该产品使用的 SuperFlash®技术获得了 Silicon Storage Technology, Inc.公司的授权。

- **本文档信息先于产品的生产周期发布。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。

引言

读者对象

本应用笔记适用于那些希望了解具有片上 flash 存储器的 78K0/Kx1+的功能，并设计开发应用系统的工程师。

目的

本应用笔记用来告诉用户关于 78K0S/Kx1+ flash 存储器自编程功能的用途，和在 flash 存储器 EEPROM 模拟期间存储数据(用于常量数据的写入)的方法。

组件

本手册通常包括以下部分。

- EEPROM 模拟功能
- EEPROM 模拟编程
- 例子程序列表

手册使用方法

在阅读本手册前，读者应掌握电子工程、逻辑电路和微控制器等方面的一般知识。

☐ 学习更多关于 78K0S/Kx1+的硬件功能：

→ 可参阅 78K0S/Kx1+ 产品的用户手册。

☐ 学习更多关于 78K0S/Kx1+的 flash 存储器自编程功能：

→ 可参阅 78K0S/Kx1+ 产品用户手册的 flash 存储器章。

规定

数据规则:	数据的高位部分在左边，低位部分在右边
有效低电平表示法:	$\overline{\text{xxx}}$ (在引脚和信号名称上划一条线)
注:	文中用**注**标注的相关术语的脚注
注意事项:	需要特别关注的信息
备注:	补充信息
数的表示法:	二进制..... xxxx 或 xxxxB
	十进制..... xxxx
	十六进制..... xxxxH

相关文档

本手册中指出的相关文档可能包括了初级的版本，但未注明。

Document Name	文档编号
78K0/KB1+ 用户手册	U17446E
78K0/KA1+用户手册	U16898E
78K0/KY1+用户手册	U16994E
78K0/KU1+用户手册	U18172E
μPD78F9510, 78F9511, 78F9512 微控制器用户手册	U18372E
μPD78F9500, 78F9501, 78F9502 微控制器用户手册	U18681E
78K0S/Kx1+ EEPROM 模拟应用笔记	本手册
78K/0S Series 系列指令用户手册	U11047E

目录

第一章 FLASH 存储器自编程概述

78K0S/Kx1+ 可以将应用程序写入到 flash 存储器中 (也就是“flash 存储器自编程 ”)。

本应用笔记用来说明，在 flash 存储器 EEPROM 模拟期间，用自编程功能来存储数据 (当作 EEPROM 来读取或者写入) 的方法。

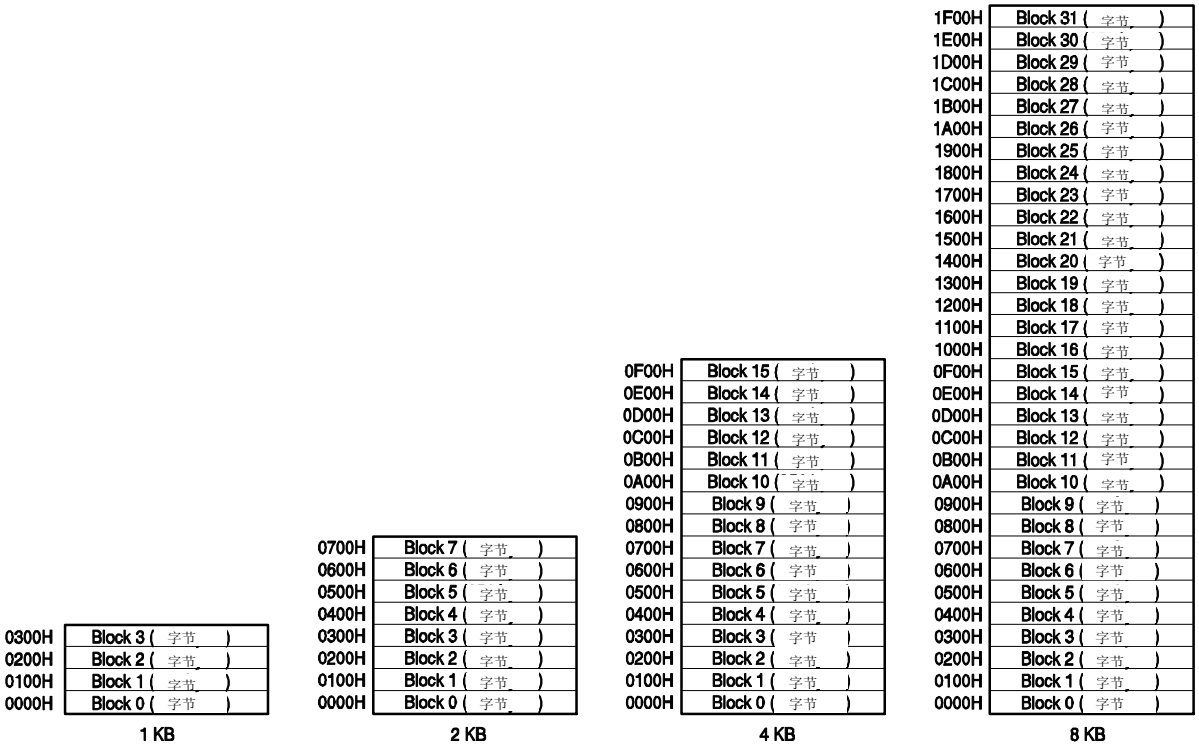
备注 flash 存储器自编程更详细的描述，参阅 78K0S/Kx1+每个产品的用户手册 flash 存储器章节。

1.1 自编程 Flash 存储器区域

flash 存储器控制操作期间的擦除、空白检测和校验区域都是以 block 为单元指定的。Block 编号详细列表如图 1-1。

注意事项 不可访问除了产品 flash 存储器区域之外的任何区域。

图 1-1. Block 编号分配



<R>

第二章 EEPROM 模拟功能 (定长单数据方法)

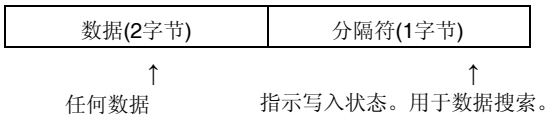
2.1 EEPROM 模拟的主要规范

EEPROM 模拟是一个通过使用 flash 存储器的自编程功能，用于 flash 存储器作为重写数据 RAM 的部分功能。如果和用户程序执行读或写进程中使用提供的例子程序(参见 **附录 A 例子程序列表(定长单数据方法)**)，用作为 EEPROM。

注意数据长度和数的改写是有限制的，因为内部 flash 存储器只能被改写有限的次数。接下来，描述了提供的例子程序和计算改写次数的主要规范。

提供的例子程序和计算改写次数的主要规范。

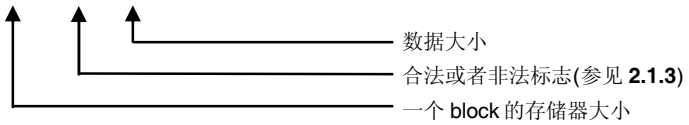
- 存储的数据格式



备注 数据的大小是可以设定的，从 1 字节开始。上限大小取决于 RAM 大小。

- flash 存储器 block 重写次数

(256 − 2) / 3 = 84 次(取最近的整数)



- 用于 EEPROM 区域的 block 数

使用两个 block(MIN.)。

备注 这些必需的 block 用于防止数据的丢失导致产生问题，例如在 block 擦除时切断电源或者电源中断。

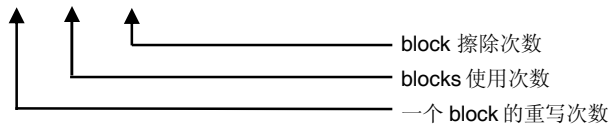
- 一个 block 擦除的次数

100 次

备注 微控制器规范中这个是 1000 次在；然而，提供的例子程序中是 100 次，这是为了降低 EEPROM 模拟控制程序大小。

- 最大重写次数

84 × 2 × 100 = 16,800 次

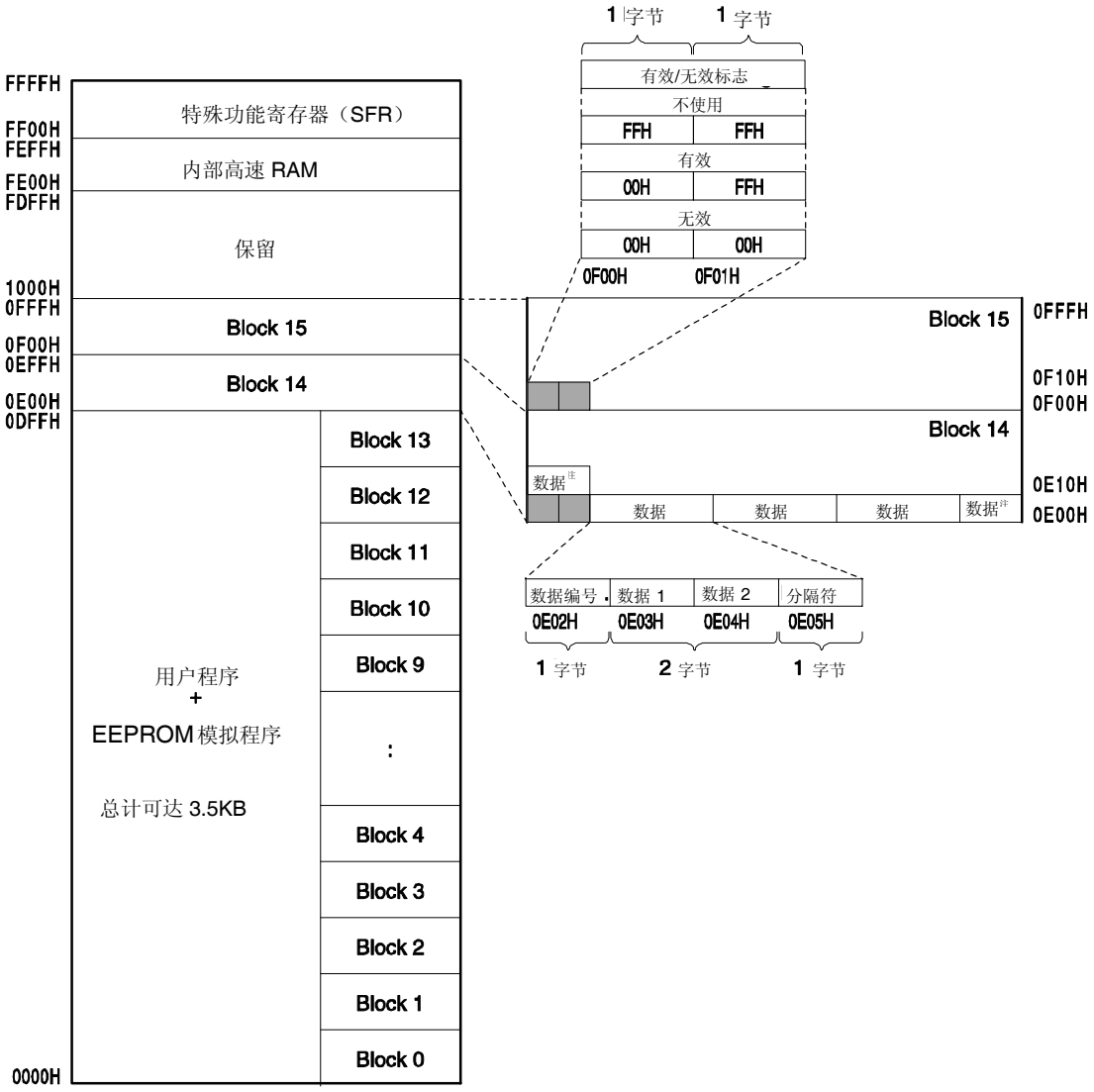


提供的例子程序中，数据按 2 字节单位处理，每个 block(256 字节)可以执行 84 次重写，和一个分隔符(1 字节)指示数据的结尾。此外，一区域需要至少两个连续的 block 来避免丢失目标，例如，产生一个不期望的掉电时，使用两个 block 总共可以执行 168 次重写。另外，在提供的例子程序中一个 block 可以擦除 100 次，使用两个 block 数据可以重写 16,800 次。

在 flash 存储器存储数据时至少要分配 2 个连续的 block。用户可以自由设定这些 block。

图 2-1 显示了一个存储器映射表和程序大小是 3.5KB 并且 block14 和 15 被用作 EEPROM 模拟时的数据结构示例。

图 2-1. 存储器映射表和数据结构示例
(当用户程序大小是 3.5 KB 并且设置 Block 14 和 15 被设置为用作 EEPROM 模拟)



注 数据连续存储。

备注 图 2-1 中的数据结构显示了当使用提供例子程序的实例。

2.1.1 EEPROM 模拟数据 block

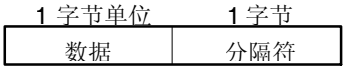
EEPROM 模拟程序至少需要 2 个连续 block 来存储数据。

只要这些 block 不和用户程序区域重叠，就可以由用户自由设定。

2.1.2 数据结构

存储在 EEPROM 模拟部分的数据由数据(1 字节单位)和分隔符(1 字节)组成。

图 2-2. 数据结构



(1) 数据

可以设置从 00H 到 FFH 的任何数据。这个容量可以设定，从 1 字节开始。需要注意的是，容量越大，可以重写的次数会越少。

(2) 分隔符

分隔符的值固定为 00H。写入分隔符是为了能发现写入的失败数据，例如在数据写入过程中万一电源掉电或发生其他问题。通过在数据最后写入分隔符来判断数据写入是否被正常的完成。当写入数据时，若不能正确读取分隔符(00H)，则在数据写入时可能出现了问题，所以相应的数据将不能使用。

若在最新的数据中搜索到一个异常分隔符，则此数据前写入的数据(紧跟着这个数据的下一个接近终点的)将被当作最新的数据读取。

(3) 数据存储的标准流程和搜索操作

数据存储的标准流程和搜索操作描述如下(在此例中，block 14 和 15 用作 EEPROM 模拟指定 blocks)。

(状态 1) 设置 Block 14 为有效 block。

Block 14	+0	
有效标志	00H	0E00H
无效标志	FFH	0E01H
数据	FFH	0E02H

(状态 2) 写入数据(11H 和 22H)。

Block 14	+0	+1	+2	
有效标志	00H			0E00H
无效标志	FFH			0E01H
数据	11H	22H	00H	0E02H
:	FFH			0E06H

(状态 3) 写入数据(22H 和 33H)。

Block 14	+0	+1	+2	
有效标志	00H			0E00H
无效标志	FFH			0E01H
数据	11H	22H	00H	0E02H
数据	22H	33H	00H	0E06H
:	FFH			0E0AH

(状态 4) 写入数据(20H 和 30H)。

Block 14	+0	+1	+2	
有效标志	00H			0E00H
无效标志	FFH			0E01H
数据	11H	22H	00H	0E02H
数据	22H	33H	00H	0E06H
数据	20H	30H	00H	0E0AH
:	FFH			0E0EH

(状态 5) 读取数据(20H 和 30H)。

Block 14	+0	+1	+2	
有效标志	00H			0E00H
无效标志	FFH			0E01H
数据 a	11H	22H	00H	0E02H
数据 b	22H	33H	00H	0E06H
数据 c	20H	30H	00H	0E0AH
数据 d	FFH			0E0EH
	•			
	•			
	•			
擦除状态	FFH			0EFEH

读取方法

<1> 因为数据 a 有不同的数据编号，转向下一数据操作。

<2> 因为数据 b 有匹配的数据编号，其分隔符被检测到。并且因为分隔符的值为 00H (正常)，2 字节数据被存储为最新的数据，转向下一数据操作。

<3> 因为数据 c 有匹配的数据编号，其分隔符被检测到。并且因为分隔符的值为 00H (正常)，2 字节数据被存储为最新的数据，转向下一数据操作。

<4> 检测 block 是否擦除到结尾。

<5> 读取值变成最新的存储数据 (数据 c)。

以下描述了在存储数据时发生问题，例如发生电源掉电时的操作流程(在此例中，block 14 和 15 作为 EEPROM 模拟指定 block)。

(状态 1) 设置 Block 14 为有效 block。

Block 14	+0	
合法标志	00H	0E00H
非法标志	FFH	0E01H
数据	FFH	0E02H

(状态 2) 写入数据编号 1 (对应于数值 11H, 22H)。

Block 14	+0	+1	+2	
合法标志	00H			0E00H
非法标志	FFH			0E01H
数据	11H	22H	00H	0E02H
:	FFH			0E06H

(状态 3) 当写入数据编号 1(对应于数值 22H, 33H) 并且不能正常写入(写入非 00H 值)分隔符时发生电源掉电。

Block 14	+0	+1	+2	
合法标志	00H			0E00H
非法标志	FFH			0E01H
数据	11H	22H	00H	0E02H
数据	22H	33H	01H	0E06H
:	FFH			0E0AH

(状态 4) 读数据编号 1 。

Block 14	+0	+1	+2	
合法标志	00H			0E00H
非法标志	FFH			0E01H
数据 a	11H	22H	00H	0E02H
数据 b	22H	33H	01H	0E06H
数据 c	FFH			0E0AH
	●			
	●			
	●			
擦除状态	FFH			0EFEH

读取方法

<1> 因为数据 a 有匹配数据编号，其分隔符被检测到。因为分隔符值是 00H (正常)，数据以 2 字节存储到最新的数据，操作转向下一数据。

<2> 因为数据 b 有匹配数据编号，其分隔符被检测到。因为分隔符值是 01H (异常), 操作转向下一数据。

<3> 检测 block 是否擦除到结尾。

<4> 读取值变为最新的存储数据 (数据 a)。

2.1.3 有效和无效标志

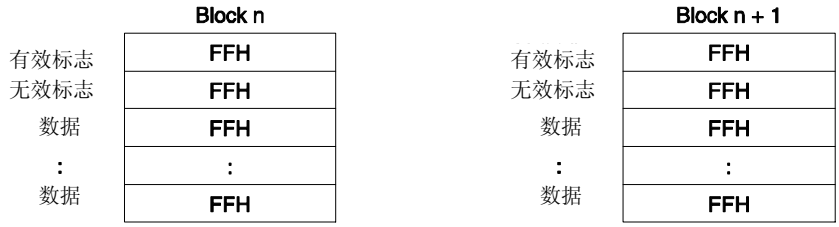
有效和无效标志以 1 字节为单位被放置在 block 的起始位置，总计 2 字节的数据。同样地，有效和无效标志指示存储在相对应 block 中的数据处于有效或无效状态。

当有效标志的值是 00H 并且无效标志的值为 FFH，相对应的 block 是有效的。其他情况，block 是无效的。

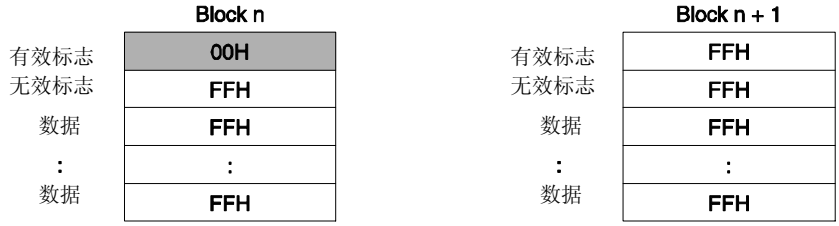
数据被连续存储到有效的 block，如果这个 block 已满，设置有效/无效标志使下一 block 有效，并使先前的 block 无效。在传输数据给下一 block 时，发生下一 block 已满或电源掉电或发生其他问题的情况下，为了防止数据丢失，本操作可使发生故障前的数据得以保存。

有效和无效标志的操作流程描述如下。

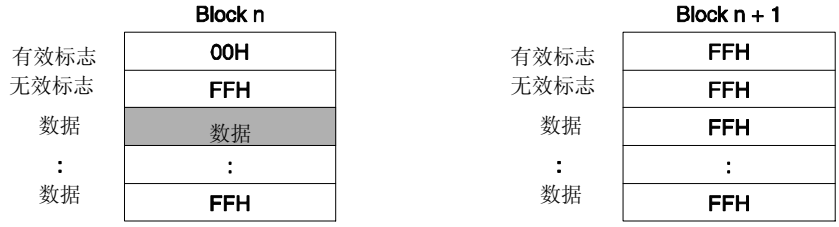
(状态 1) 初值



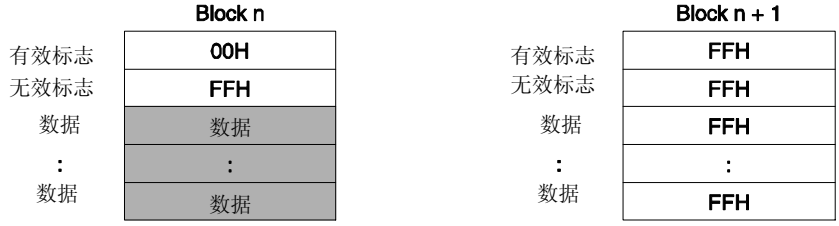
(状态 2) block n 的有效标志写入 00H



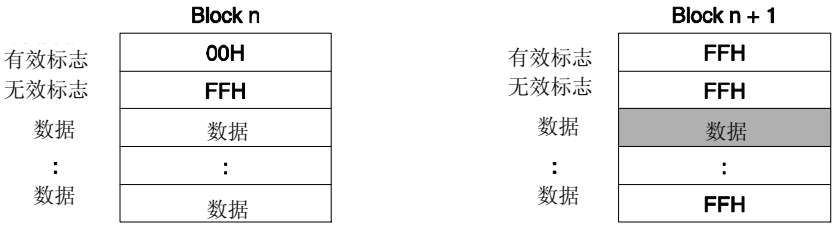
(状态 3) 写入数据到 block n



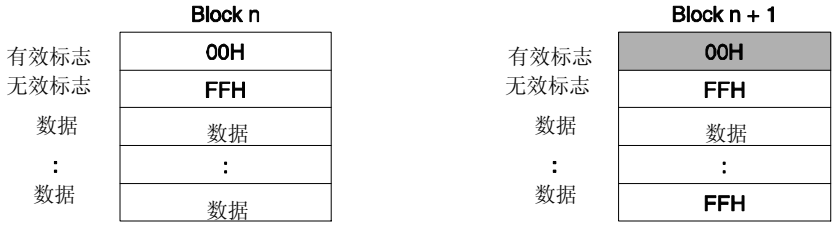
(状态 4) block n 中的数据已满



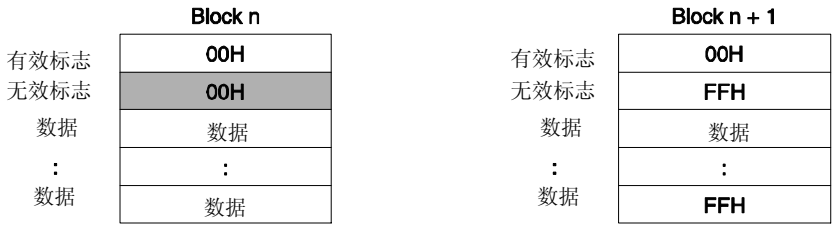
(状态 5) 传输最新的数据到 block n + 1



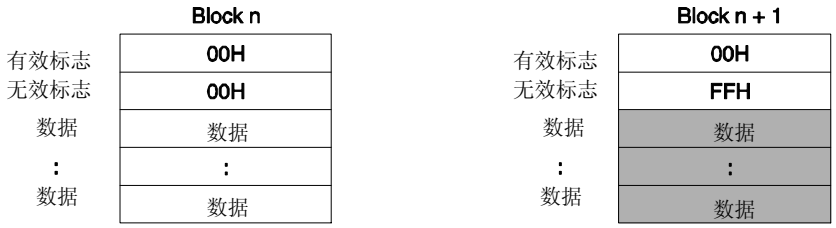
(状态 6) block n + 1 有效标志写入 00H



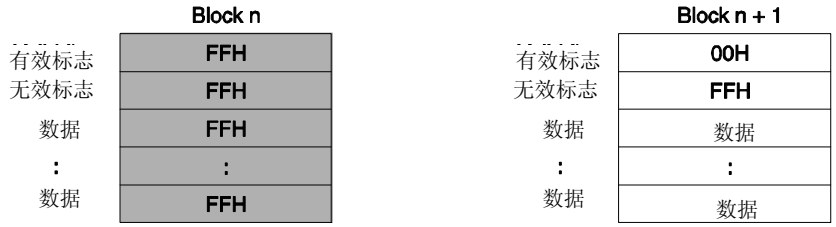
(状态 7) block n 无效标志写入 00H



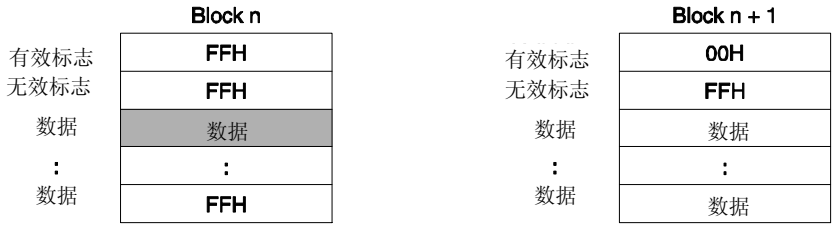
(状态 8) block n + 1 数据已满



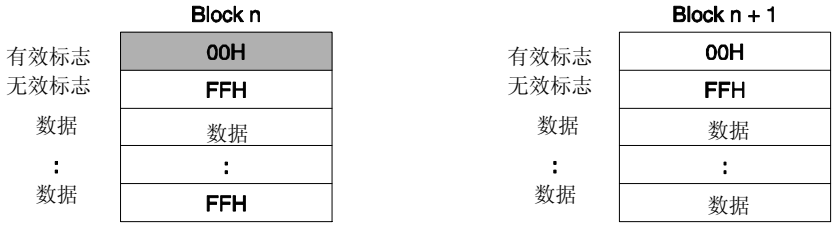
(状态 9) 擦除 block n



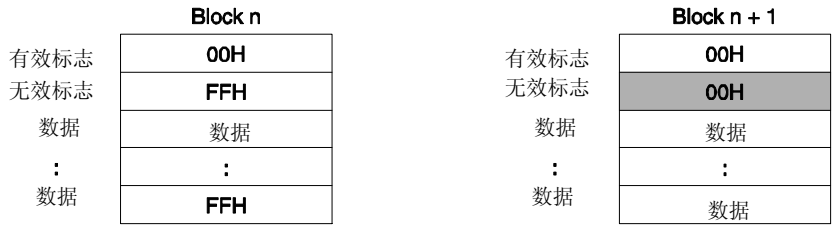
(状态 10) 传输最新数据到 block n



(状态 11) block n 有效标志写入 00H



(状态 12) block n + 1 无效标志写入 00H



2.2 EEPROM 模拟程序执行条件

执行 EEPROM 模拟程序前确保符合表 2-1 所列的所有条件。

表 2-1. EEPROM 模拟操作条件

项目	描述
安全推栈区域 (汇编语言: 22 字节)	EEPROM 模拟程序操作期间，用户程序使用的推栈可以继承并继续使用。另外，EEPROM 模拟程序执行一开始时就需要左边描述的堆栈空间。参阅 **3.2 EEPROM 模拟程序使用的资源**对堆栈的进一步描述。
EEPROM 模拟程序 RAM (汇编语言: 8 字节)	该区域的 RAM 必须专用于 EEPROM 模拟，其用来临时存储读取和写入的数据。确保左边所描述的区域是在内部高速 RAM 区并用作数据缓冲区。
看门狗定时器操作 (WDT)	因为在 EEPROM 模拟程序执行过程中，当 flash 存储器控制处理被执行时没有指令能被执行，则 flash 存储器控制处理对 WDT 计数器清零。此时，设置 WDT 的溢出时间为 10ms 或更长以便不会发生溢出。
禁止复位	不要在 EEPROM 模拟程序操作过程中对微控制器复位。当发生复位时，flash 存储器中正在被访问的任何数据都变得不确定。
禁止电源掉电或中断	在 EEPROM 模拟程序操作过程中确保微控制器具有稳定的电源供应。当电源掉电时，flash 存储器中正在被访问的任何数据都变得不确定。

注意事项 在执行 EEPROM 模拟程序的过程中需禁止所有的中断。在 EEPROM 模拟程序执行完成后，中断屏蔽状态返回 EEPROM 模拟程序执行前的状态，并且允许中断。

备注 对于看门狗定时器操作和禁止电源掉电或中断的详细描述，请参见 78K0S/Kx1+ 产品用户手册 flash 存储器章**自编程功能注意事项**。

<R>

第三章 EEPROM 模拟程序(定长单数据方法，汇编语言)

这个应用程序使用 flash 存储器的自编程功能，将 flash 存储器作为 EEPROM 存储数据等。

3.1 EEPROM 模拟程序的结构

表 3-1 列出了包含这个程序的文件。

表 3-1. 文件结构

文件名称	功能	类型
EEPROM.asm	EEPROM 模拟处理 这个处理不仅包括 EEPROM 模拟的读写，而且还包括数据搜索和 block 传输过程。	汇编源

3.2 EEPROM 模拟程序所需的资源

这个程序所需的资源如表 3-2 所示。

表 3-2. 资源

资源	描述	
RAM	EEPROM 模拟需要的 RAM	8 字节
	堆栈	22 字节
	EEPROM 写入处理	22 字节
	EEPROM 读取处理	12 字节
ROM	EEPROM 模拟处理	295 字节
	Flash 存储器控制处理	173 字节
	总共	468 字节

3.3 EEPROM 模拟程序的使用

本章描述的 EEPROM 模拟至少使用 flash 存储器的 2 个 block。EEPROM 模拟过程需在 flash 存储器中使用和更新存储 2 字节 数据。

当用户应用程序中嵌入 EEPROM 模拟程序，且条件符合 (参阅 **2.2 EEPROM 模拟程序执行条件**)，并执行指定的程序，可以执行 EEPROM 模拟。

以下说明如何使用“定长数据方法，汇编语言”程序来操作 EEPROM 模拟。

3.3.1 用户设置初值

用户必须设置以下所列项目用于 EEPROM 模拟程序的初值。

- 用于 EEPROM 的 block 起始编号
- 用于 EEPROM 的 block 编号

这些初值都包括在 EEPROM.asm 中。

(1) 用于 EEPROM 的 block 数量

指定用于 EEPROM 模拟的 block 数量。设置的 block 必须是超过 2 个连续的 block。设置的 block 不能与用户程序区域重叠。

通过增加 EEPROM 的 block 编号可以增加 EEPROM 重写次数。不管 EEPROM 模拟的数据总量，我们都推荐除了用于程序之外的任意区域作为 EEPROM 模拟。

示例 **1.** 当使用 2 个 block (block 14 和 15) 作为 EEPROM block

```
EEPROM_BLOCK EQU (14)
EEPROM_BLOCK_NO EQU (2)
```

2. 当使用 4 个 block (block 12, 13, 14, 和 15) 作为 EEPROM block

```
EEPROM_BLOCK EQU (12)
EEPROM_BLOCK_NO EQU (4)
```

(2) 用户使用的数据长度

用户必须设定存储在 EEPROM 中的数据长度。

由数据容量和分隔符(1 字节)来设定数据长度，因为 EEPROM 模拟需要一个分隔符。

示例 当存储数据为 2 字节时

```
LENG  EQU  (3)      ; 数据长度(包括分隔符)
```

(3) 重试擦除的次数

设定重试的次数和 flash 存储器 block 需要执行擦除的次数一致。

在本文档的例子程序中 (参见 **附录 A 例子程序列表(定长单数据方法)**), 在 T_A = –10 to +85°C, 4.5 V ≤ V_{DD} ≤ 5.5 V, 和 NERASE ≤ 100 次条件下设定(0.4 秒)。

在不同条件下设定它，要设定的比 block 擦除时间大 8.5ms。一个擦除的时间是 8.5 ms。

备注 在本文档的例子程序中，最大重试次数指定为 225 次(大约 2.16 秒)。如果擦除时间超过 2.1 秒，则必需要改变 SelfFlashBlockErase 函数。

3.3.2 调用 EEPROM 模拟的用户处理

当在用户程序中执行 EEPROM 模拟时，可使用 2 种类型的处理：EEPROM 读和写处理。

<EEPROM 读和写处理>
EEPROM 读和写处理通过设置数据地址的特殊自变量，并调用它们，使 EEPROM 模拟更便利。
EEPROM 读和写处理的汇编版本和 C 语言版本分别包括在 main.asm 和 main.c 中。
下列变量和结构体(RAM)在 EEPROM 模拟中都用于读和写。

用于 main.asm (汇编版本), 用变量 EEPROM_DATA 定义如下。

```
EEPROM_DATA:       DS      2       ; 数据
EEPROM_DELIMITER:  DS      1       ; 分隔符
```

用于 main.c (C 语言版本), 用结构体 eeprom_data 定义如下。

```
Struct eeprom_data{
  unsigned char uc_eeprom_data[2]       ; 数据
  unsigned char uc_delimiter            ; 分隔符
};
```

(1) EEPROM 读处理 (__eeprom_read): 从 EEPROM 区域读取设定容量的数据。

用于 main.asm (汇编版本)

- 自变量:
  存储 EEPROM_DATA 地址到 AX 寄存器并且执行子程序调用_eeprom_read 函数。
- 返回值(CY 标志):
  返回值是 CY=0 指示正常完成读数据或者 CY=1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

用于 main.c (C 语言版本)

- 自变量:
  通过使用 eeprom_data 结构体的地址作为自变量来执行_eeprom_read 函数。
- 返回值(错误标志):
  返回值是 0 指示正常完成读数据或者是 1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

(2) EEPROM 写处理(EEPROMWrite)：写入设定容量的数据到 EEPROM 区域

用于 main.asm (汇编版本)

- 自变量:

  存储 EEPROM_DATA 地址到 AX 寄存器并且在设定写入到 EEPROM_DATA 的数据和写入到 EEPROM_DELIMITER 的分隔符之后执行_eeprom_write 函数。

- 返回值(CY 标志):

  返回值是 CY=0 指示正常完成读数据或者 CY=1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

用于 main.c (C 语言版本)

- 自变量:

  通过使用结构体 eeprom_data 的地址作为自变量来执行_eeprom_write 函数。

- 返回值 (错误标志):

  返回值是 0 指示正常完成读数据或者 1 指示异常完成。

3.4 EEPROM 模拟程序描述

3.4.1 EEPROM 模拟的用户访问处理

表 3-3 和 3-4 列出了用户访问的处理和用来执行 EEPROM 模拟的读写操作。

表 3-3. EEPROM 读处理

(a) 汇编版本

处理名称	__eeprom_read (用户访问函数)
ROM 容量	29 字节
堆栈容量	5 层(10 字节)
输入	AX: 变量地址
返回值	正常完成: CY=0 异常完成: CY=1
操作描述	从 EEPROM 读取指定地址的最后数据到存储地址 1: 搜索用作 EEPROM 的 block。 2: 搜索有效 block 中的最后数据的地址。 3: 从搜索地址中读取最后数据。

(b) C 语言版本

处理名称	_eeprom_read (用户访问函数)
ROM 容量	29 字节
堆栈容量	5 层 (10 字节)
输入	AX: 结构体的指针
返回值	正常完成: 错误标志=0 异常完成: 错误标志=1
操作描述	从 EEPROM 读取指定地址的最后数据到存储地址 1: 搜索用作 EEPROM 的 block。 2: 搜索有效 block 中的最后数据的地址。 3: 从搜索地址中读取最后数据。

表 3-4. EEPROM 写处理

(a) 汇编版本

处理名称	__eeprom_write (用户访问函数)
ROM 容量	58 字节
堆栈容量	11 层(22 字节)
输入	AX: 变量地址
返回值	正常完成: CY=0 异常完成: CY=1
操作描述	数据从存储地址写入到 EEPROM。 1: 搜索用作 EEPROM 的 block。 2: 如果没有有效 block 的话，设定第一个指定的有效 block。 3: 搜索可以写入的有效 block 的地址。 4: 如果有效 block 已满或者无法写入，执行切换到下一个 block 的操作。 5: 创建写入数据。 6: 写入有效 block。

(b) C 语言版本

处理名称	_eeprom_write (用户访问函数)
ROM 容量	58 字节
堆栈容量	11 层(10 字节)
输入	AX: 结构体的指针
返回值	正常完成: 错误标志=0 异常完成: 错误标志=1
操作描述	数据从存储地址写入到 EEPROM。 1: 搜索用作 EEPROM 的 block。 2: 如果没有有效 block 的话，设定第一个指定的有效 block。 3: 搜索可以写入的有效 block 的地址。 4: 如果有效 block 已满或者无法写入，执行切换到下一个 block 的操作。 5: 创建写入数据。 6: 写入有效 block。

3.4.2 EEPROM 模拟的控制处理 (用于内部处理)

表 3-5 到表 3-9 列出了用于控制 EEPROM 模拟使用的处理。

表 3-5. EEPROM Block 搜索处理

处理名称	EEPROMUseBlockSearch
ROM 容量	28 字节
堆栈容量	2 层(4 字节)
输入	无
输出	正常完成: CY=0, A=Block 表编号 (01H ～ FEH) 异常完成: CY=1, A=下一个结束 block
使用寄存器	A
操作描述	搜索分配给 EEPROM 的 flash 存储器中当前正在使用的 block。

表 3-6. EEPROM Block 初始化处理

处理名称	EEPROMBlockInit
ROM 容量	17 字节
堆栈容量	6 层(12 字节)
输入	无
输出	正常完成: CY=0 异常完成: CY=1
使用寄存器	A, X, B, C, D, E
操作描述	若 EEPROM 指定 block 中无有效 block，设置最先指定的 block 为当前使用 block (有效)。 如果正常安全则返回 CY = 0。 如果不正常安全则返回 CY = 1。

表 3-7. EEPROM Block 更改处理

处理名称	EEPROMUseBlockChange
ROM 容量	59 字节
堆栈容量	6 层(12 字节)
输入	A=当前使用的 block 编号
输出	正常完成: CY=0, C=Block 表编号(02H ～ FEH), 零标志(Z)=1 异常完成: CY=1, 零标志(Z)=0
使用寄存器	A, X, B, C, D, E
操作描述	如果当前使用 block 数据已满，此功能搜索可用的下一 block ，并将数据复制给这个 block 。 1: 设置下一可用 block。 2: 擦除下一可用 block。 3: 从有效 block 传输最新的数据给下一 block。 4: 设置下一可用 block 为有效的。 5: 设置当前合法 block 为无效的。 6: 存储到新的 block 编号“CurrentB_No”。

表 3-8. EEPROM Block 数据写入从头搜索处理

处理名称	EEPROMWriteTopSearch
ROM 容量	44 字节
堆栈容量	3 层(6 字节)
输入	A: 当前搜索的 block 表编号
输出	搜索成功: CY = 0, 设置写入地址到 AX. 搜索失败: CY = 1
使用寄存器	A, X, B, D, E
操作描述	搜索指定 block 写入地址。 仅当数据区域在 0FFH 中适合 block 时正常完成。

表 3-9. EEPROM 最新数据搜索处理

处理名称	EEPROMDataSearch
ROM 容量	33 字节
堆栈容量	3 层(6 字节)
输入	A: 当前使用的 block 表编号
输出	正常完成: CY=0, DE=最新数据的地址 异常完成: CY=1, E=0
使用寄存器	A, X, D, E
操作描述	读取最新数据的存储地址

3.4.3 Flash 存储器的控制处理

表 3-10 ～表 3-17 列出了用于控制 flash 存储器作为 EEPROM 模拟的处理。

表 3-10. Block 擦除

处理名称	SelfFlashBlockErase
ROM 容量	27 字节
堆栈容量	3 层(6 字节)
输入	A: 被擦除的 block 编号
输出	正常完成: CY=0 异常完成: CY=1
使用寄存器	B
操作描述	擦除指定 block 并执行空白检测。

表 3-11. 模式转换处理 (从自编程模式到正常模式)

处理名称	SelfFlashModeOff
ROM 容量	31 字节
堆栈容量	1 层(2 字节)
输入	无
输出	无
使用寄存器	A, X
操作描述	释放自编程模式。

表 3-12. 模式转换处理 (从正常模式到自编程模式)

处理名称	SelfFlashModeOn
ROM 容量	35 字节
堆栈容量	1 层(2 字节)
输入	无
输出	无
使用寄存器	A, X
操作描述	设置自编程模式。

表 3-13. Block 擦除处理

处理名称	FlashBlockErase
ROM 容量	15 字节
堆栈容量	1 层(2 字节)
输入	A: Block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	擦除指定 block。

表 3-14. Flash 自编程函数调用处理

处理名称	SubFlashSelfPrg
ROM 容量	12 字节
堆栈容量	1 层(2 字节)
输入	无
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	调用 falsh 自编程函数。

表 3-15. Block 空白检测处理

处理名称	FlashBlockBlankCheck
ROM 容量	17 字节
堆栈容量	1 层(2 字节)
输入	A: 指定 block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	执行指定 block 的空白检测 。

表 3-16. 设定 block 有效处理

处理名称	SetValid
ROM 容量	9 字节
堆栈容量	1 层(2 字节)
输入	A: Block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A, X, B, C, D, E
操作描述	设定使用的 block 有效。

表 3-17．EEPROM 数据写入处理

处理名称	FlashEEPROMWrite
ROM 容量	54 字节
堆栈容量	5 层(10 字节)
输入	DE: 写入开始地址 C: 写入数据量 AX: 写入数据存储地址
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	D, E
操作描述	写入数据到 EEPROM 并内部校验数据。

3.5 EEPROM 模拟程序流程图

3.5.1 EEPROM 模拟访问处理流程图

图 3-1 和 3-2 显示了用户调用访问处理的流程图，作为执行 EEPROM 模拟读或写操作。

图 3-1. EEPROM 读处理流程图

[概述]
从指定存储地址读取定义为结构体的数据

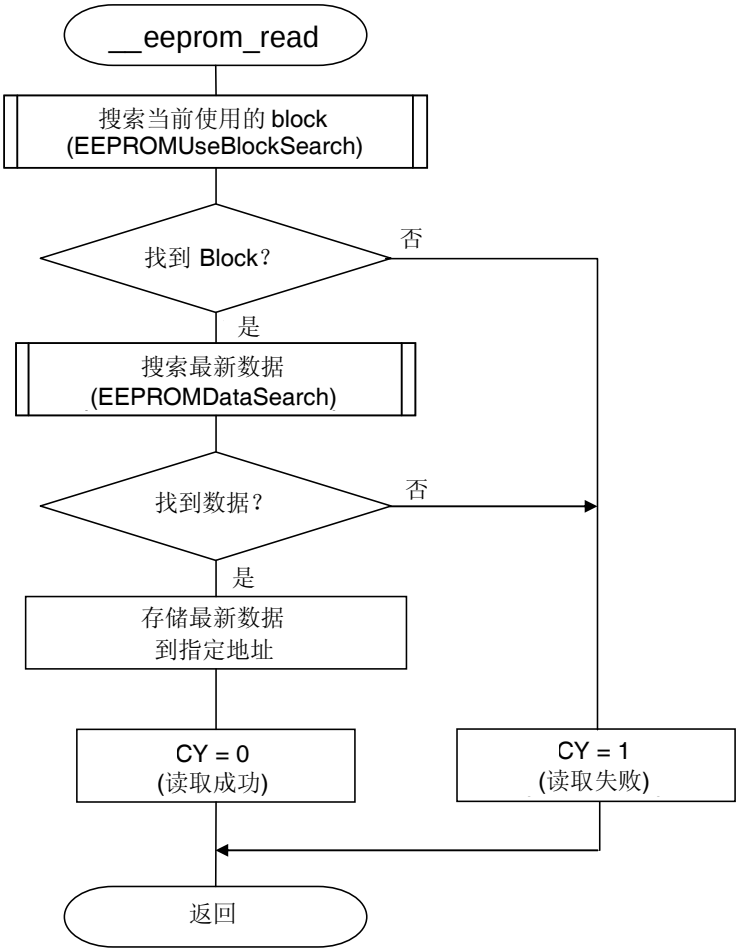
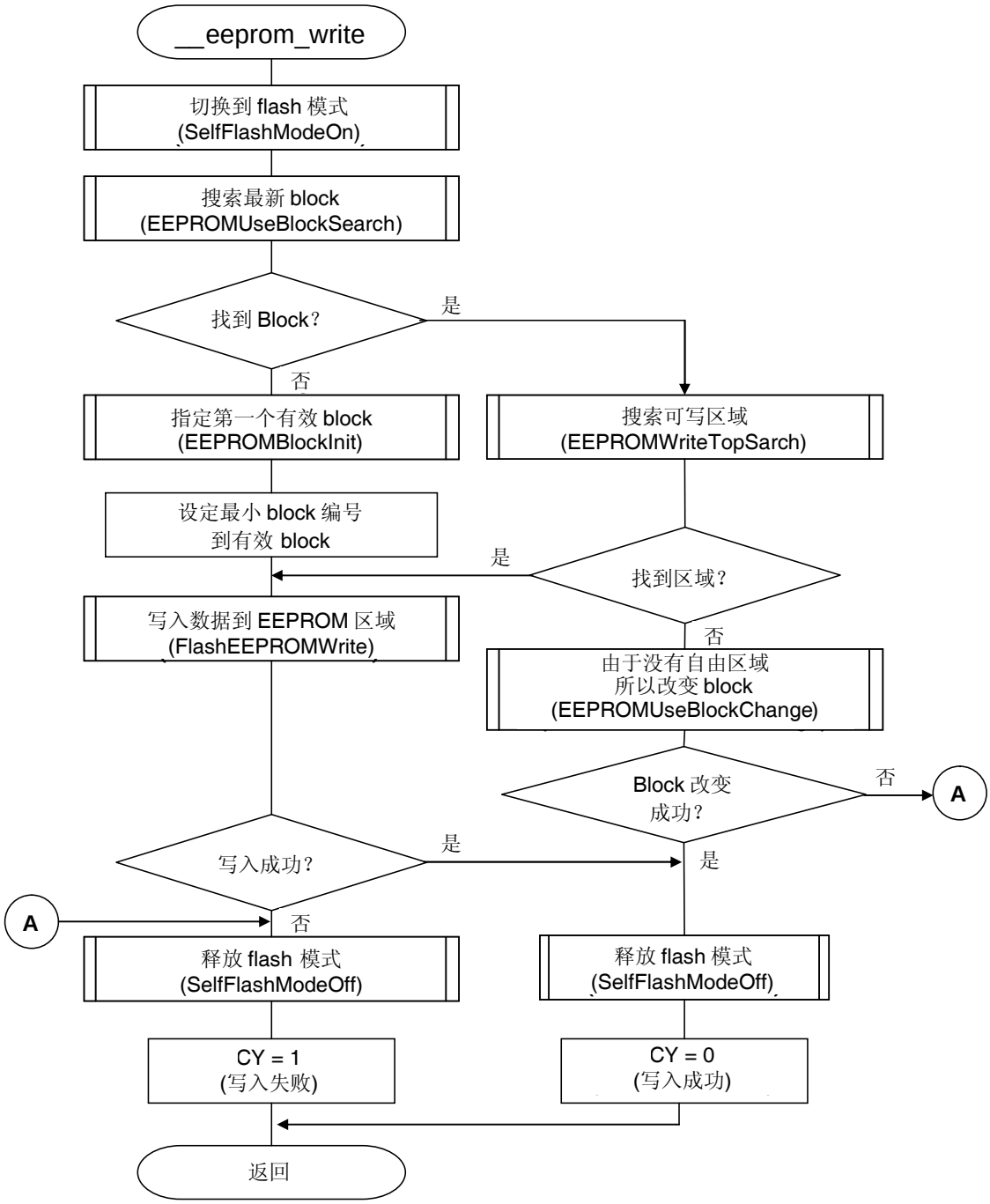


图 3-2. EEPROM 写入处理流程图

[概述]

指定编号的数据从存储地址写入到有效 block。



3.5.2 EEPROM 模拟控制处理流程图

图 3-3 ～3-7 显示了 EEPROM 模拟期间用于模拟控制处理的流程图。

图 3-3. 当前使用的 EEPROM Block 搜索功能流程图

[概述]

搜索当前使用指定为 EEPROM 的 flash 存储器的的 block。

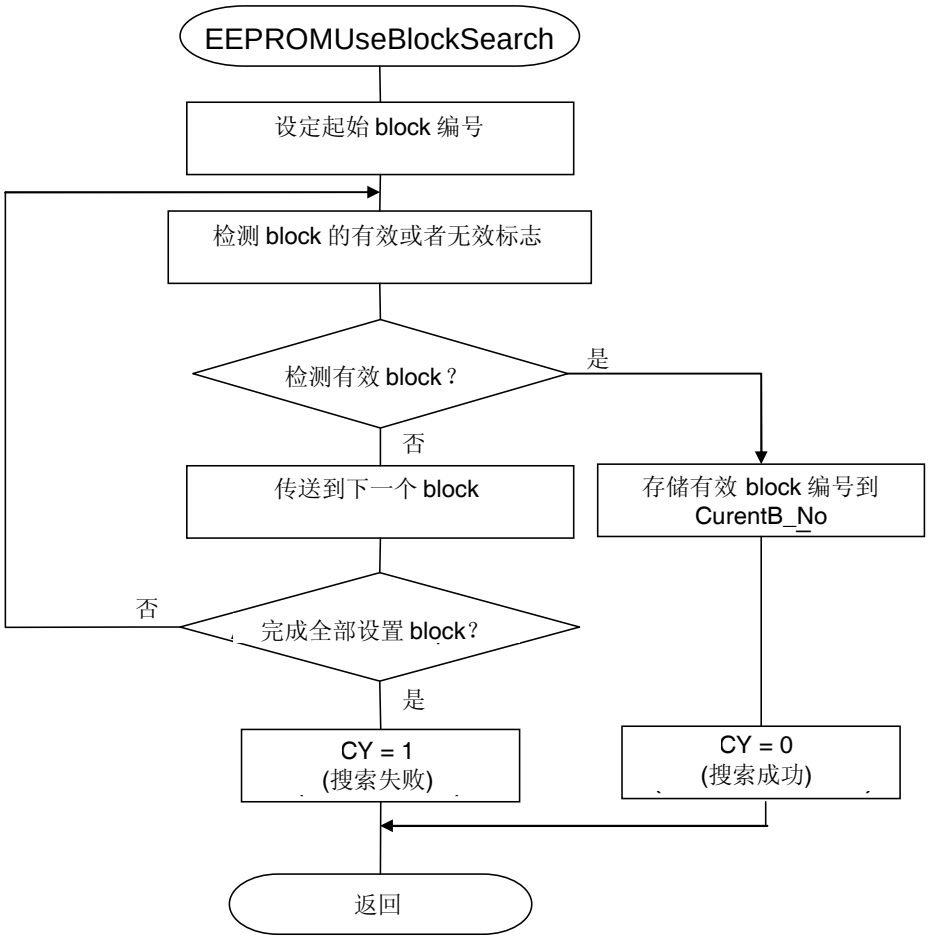


图 3-4. EEPROM Block 初始化处理流程图

[概述]

如果指定用于 EEPROM 的 block 中没有有效的 block，设置第一个制定的 block 为有效的。

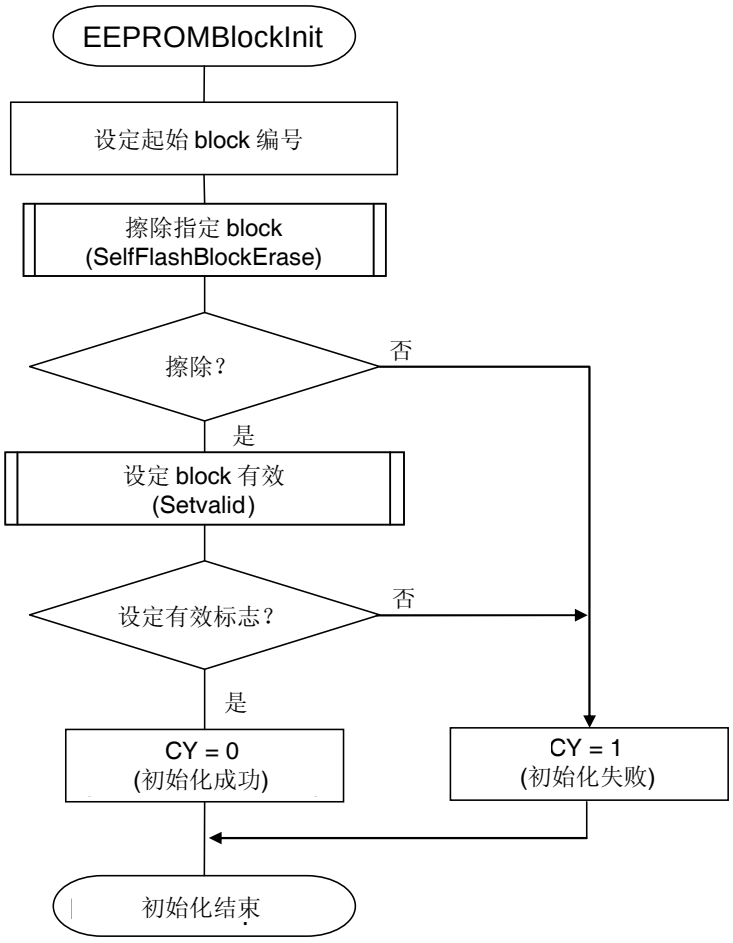


图 3-5. 更改 EEPROM 使用 Block 处理流程图

[概述]

如果当前使用的 block 数据已满，这个函数搜索下一个可用来复制数据新的 block。

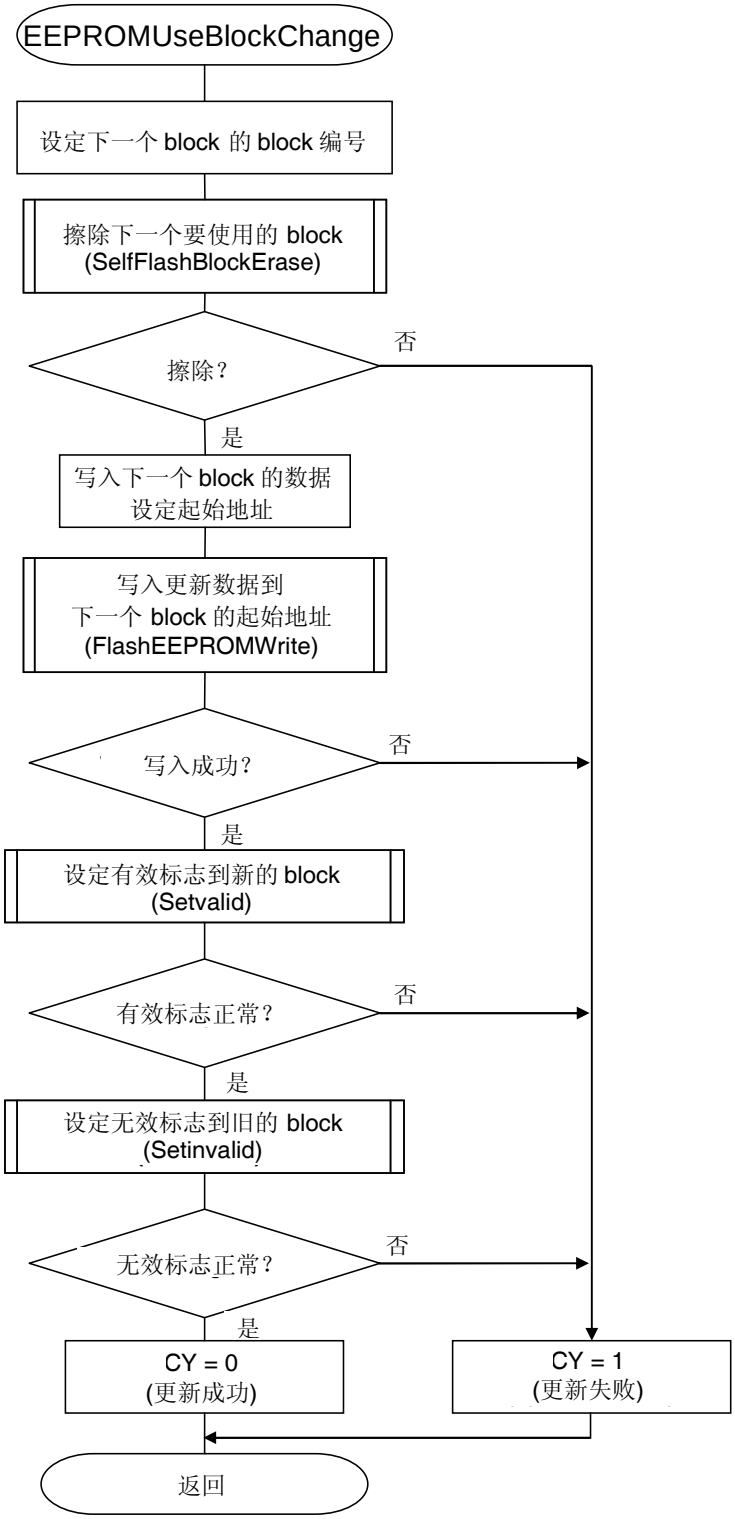


图 3-6. EEPROM 使用 Block 数据写入从头搜索处理流程图

[概述]

搜索指定 block 写入区域。仅当数据区域在 0FFH 中适合 block 时正常完成。

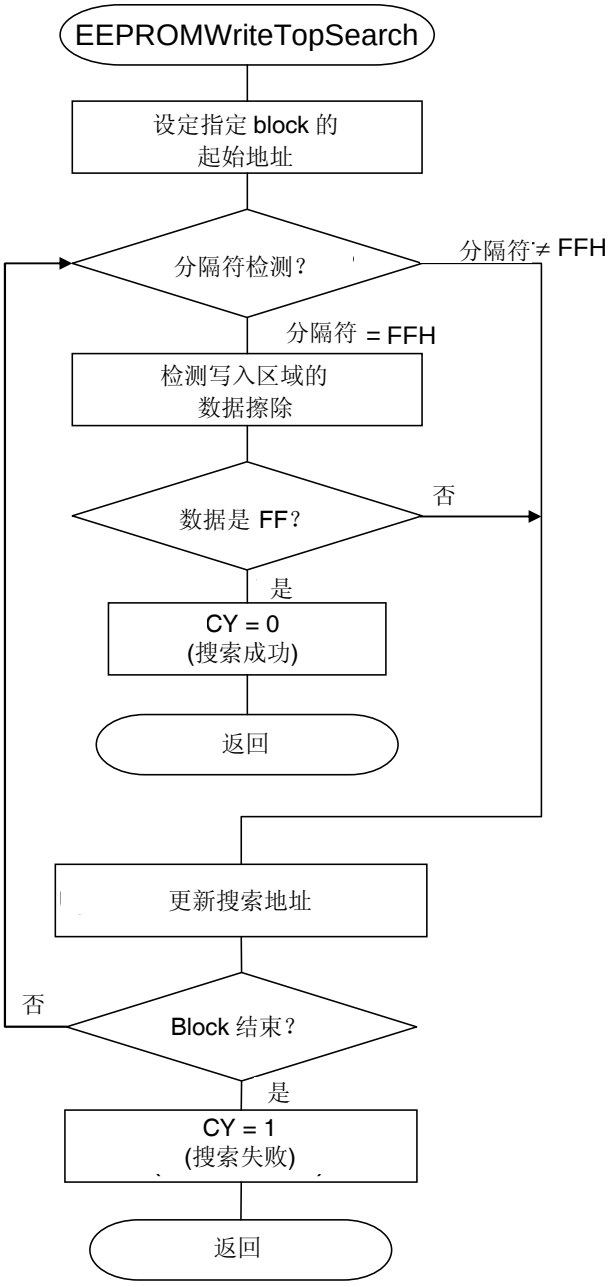
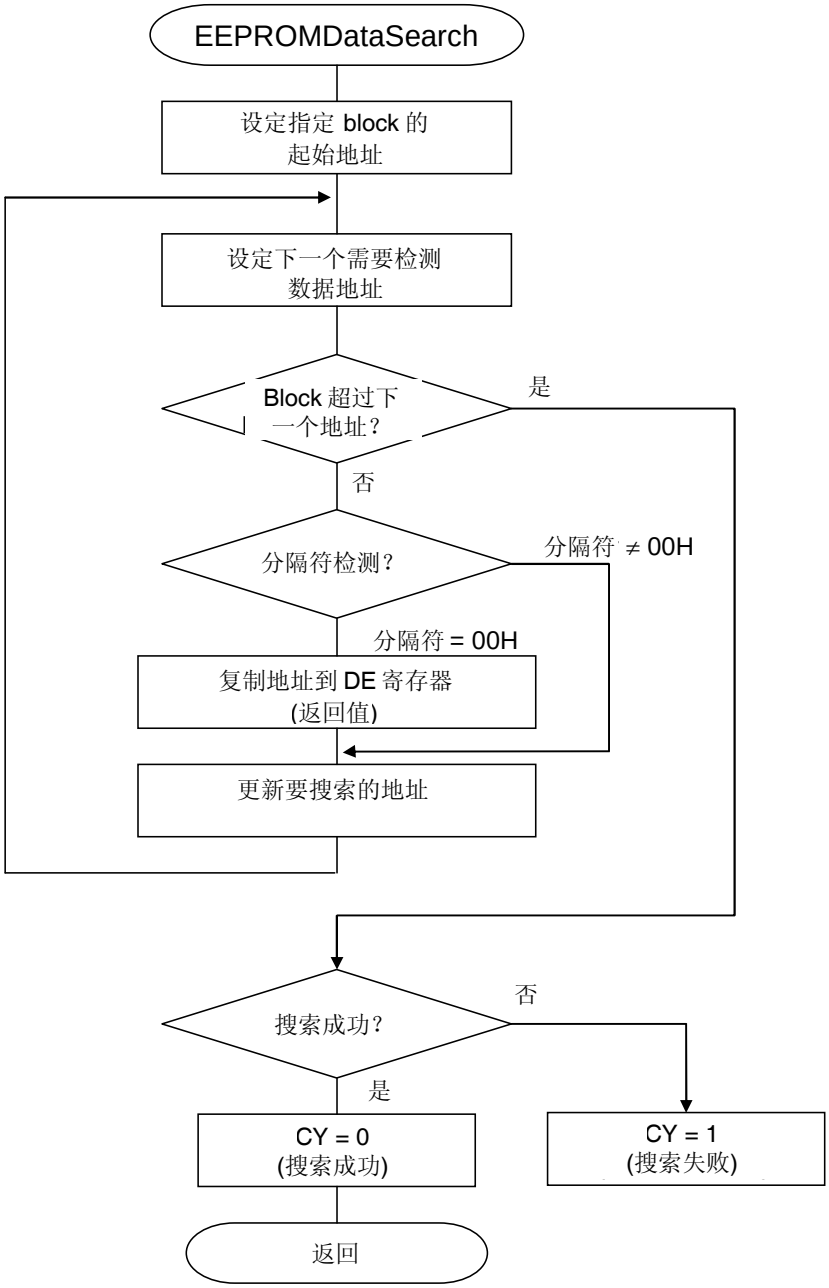


图 3-7. 搜索 EEPROM 最新数据处理流程图

[概述]

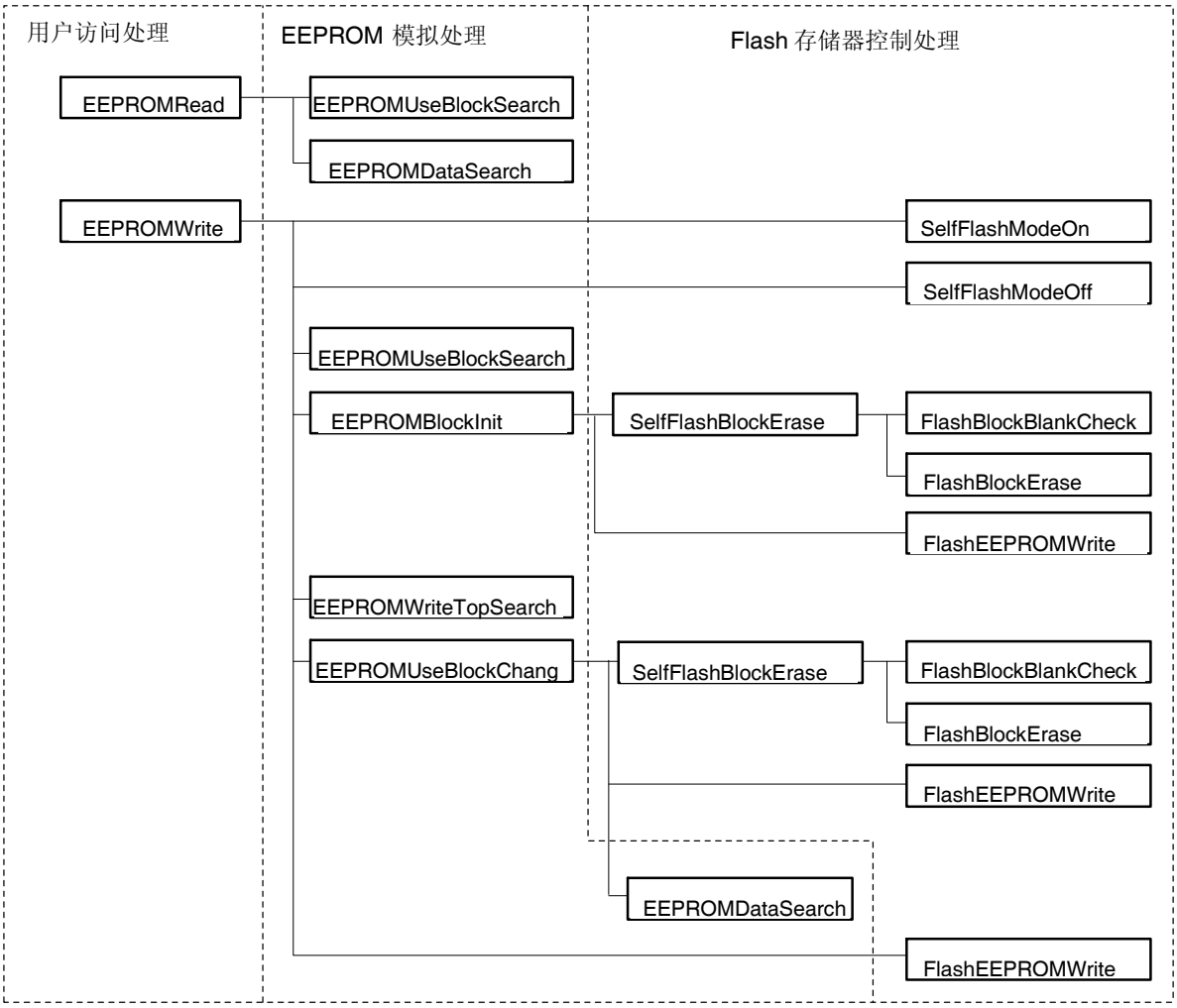
读取最新数据的存储地址。



3.6 EEPROM 模拟处理列表

EEPROM 模拟处理调用树如下所示。

图 3-8. 调用树



<R>

第四章　EEPROM 模拟功能　(定长多数据方法)

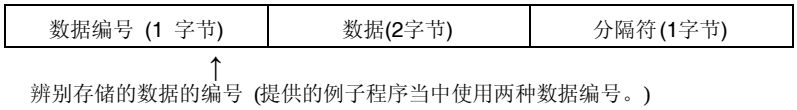
4.1　EEPROM 模拟的主要规范

EEPROM 模拟是一个通过使用 flash 存储器的自编程功能，用于 flash 存储器作为重写数据 RAM 的部分功能。如果和用户程序执行读或写进程中使用提供的例子程序(参见 **附录 B　例子程序列表(定长多数据方法)**)，用作为 EEPROM。

注意数据长度和数的改写是有限制的，因为内部 flash 存储器只能被改写有限的次数。接下来，描述了提供的例子程序和计算改写次数的主要规范。

提供的例子程序和计算改写次数的主要规范。

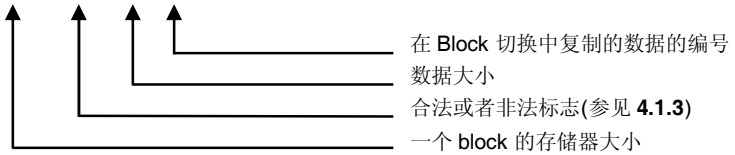
•存储的数据格式



备注　数据的大小是可以设定的，从 1 字节开始。上限大小取决于 RAM 大小。

• flash 存储器 block 重写次数

(256 – 2) /　4 – 1　=　62 次(取最近的整数)



•用于 EEPROM 区域的 block 数

使用两个 block(MIN.)。

备注　这些必需的 block 用于防止数据的丢失导致产生问题，例如在 block 擦除时切断电源或者电源中断。

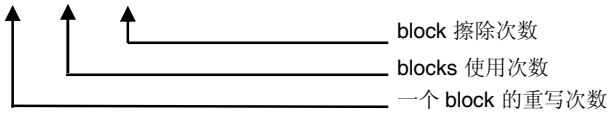
•一个 block 擦除的次数

100 次

备注　微控制器规范中这个是 1000 次在；然而，提供的例子程序中是 100 次，这是为了降低 EEPROM 模拟控制程序大小。

•最大重写次数

62 × 2 × 100 = 12,400 次



提供的例子程序中，数据按 2 字节单位处理，每个 block(256 字节)可以执行 62 次重写，和一个分隔符(1 字节)指示数据的结尾。此外，一区域需要至少两个连续的 block 来避免丢失目标，例如，产生一个不期望的掉电时，使用两个 block 总共可以执行 124 次重写。另外，在提供的例子程序中一个 block 可以擦除 100 次，使用两个 block 数据可以重写 12,400 次。

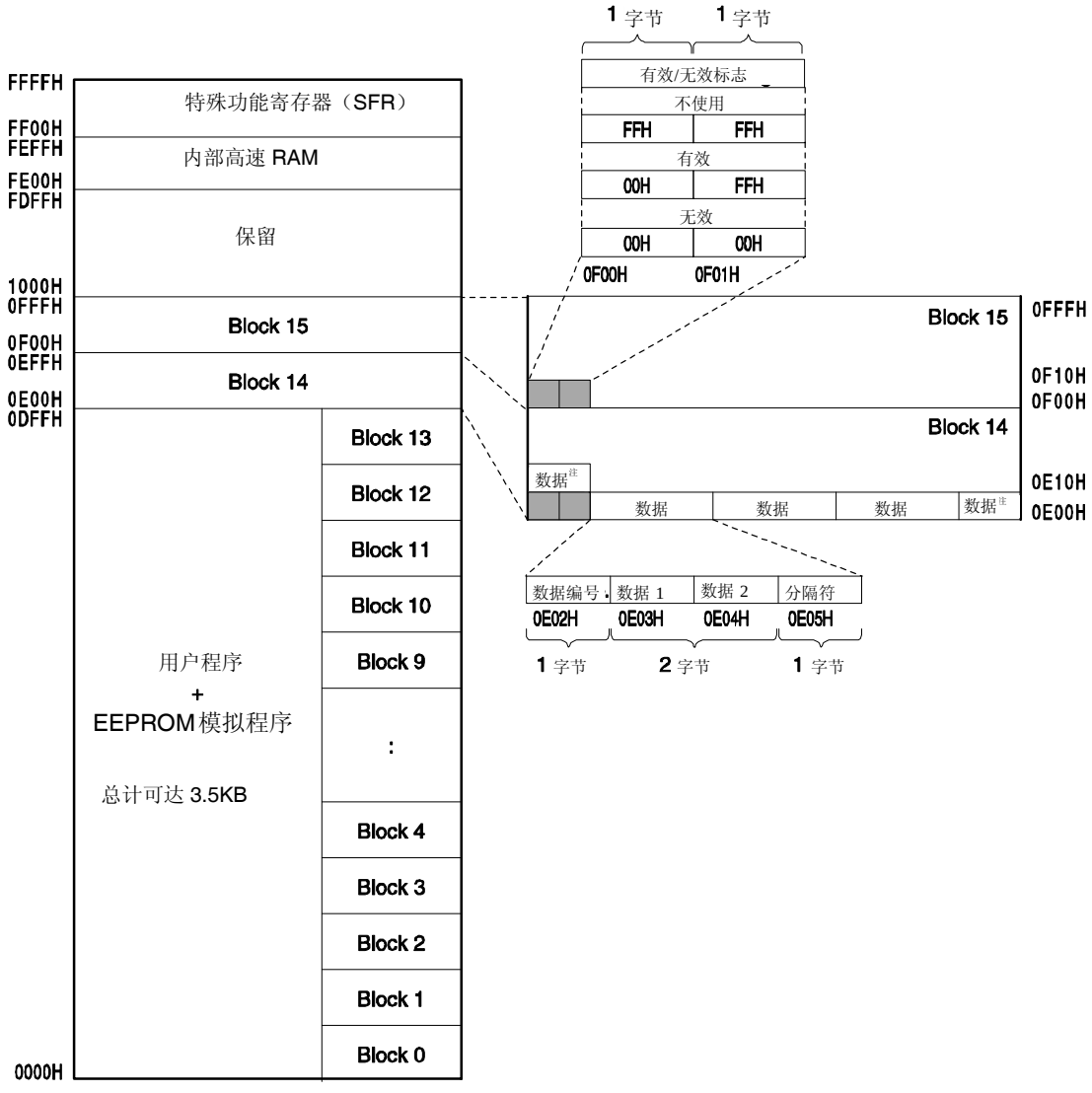
然而，如果全部 EEPROM 存储数据(数据+分隔符)的字节总数种类大到足以超过存储 block 的容量(例如，数据的字节的全部编号大于半个 block 容量)，EEPROM 模拟可能不正常运转。特别是，block 之间的传送数据可能经常出现并且重写数据的编号可能严重减少。

在 flash 存储器存储数据时至少要分配 2 个连续的 block。用户可以自由设定这些 block。

图 4-1 显示了一个存储器映射表和程序大小是 3.5KB 并且 block14 和 15 被用作 EEPROM 模拟时的数据结构示例。

图 4-1.　存储器映射表和数据结构示例

(当用户程序大小是 3.5 KB 并且设置 Block 14 和 15 被设置为用作 EEPROM 模拟



注　　数据连续存储。

备注　　图 4-1 中的数据结构显示了当使用提供例子程序的实例。

4.1.1　EEPROM 模拟数据 block

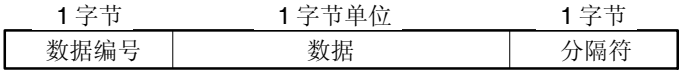
EEPROM 模拟程序至少需要 2 个连续 block 来存储数据。

只要这些 block 不和用户程序区域重叠，就可以由用户自由设定。

4.1.2　数据结构

存储在 EEPROM 模拟部分的数据由数据编号(1 字节)，数据(1 字节单位)，和分隔符(1 字节)组成。

图 2-2.　数据结构



(1)　数据编号

数据编号用来辨别读取或写入的数据。数据编号值从 00H 到 FEH 有效。

在使用数据之前，用户应先给每种类型的数据分配一个数据编号。

基本上，每次执行写入程序时，从 block 的起始部分，以 4 个字节为单元写入数据。

要读取最新的数据，从 block 的起始部分以 4 个字节为单元开始搜索，判断是否有相同的数据编号存在。如果数个场合的数据发现具有相同的数据编号，则将最接近数据终点的数据认作最新的数据。

如果数据编号是 FFH，则被认为是数据终点。Flash 存储器擦除后，所有数据都被指定为 FFH，所以当读到数据编号为 FFH 时，将认为是数据终点并结束搜索。因此，数据编号仅从 00H 到 FEH 有效。

备注　在提供的例子程序当中使用两种数据编号。

(2)　数据

可以设置从 00H 到 FFH 的任何数据。这个容量可以设定，从 1 字节开始。需要注意的是，容量越大，可以重写的次数会越少。在提供的例子程序中指定它为 2 字节(数据长度(LENG)定义为 4)。

(3)　分隔符

分隔符的值固定为 00H。写入分隔符是为了能发现写入的失败数据，例如在数据写入过程中万一电源掉电或发生其他问题。通过在数据最后写入分隔符来判断数据写入是否被正常的完成。当写入数据时，若不能正确读取分隔符(00H)，则在数据写入时可能出现了问题，所以相应的数据将不能使用。

若在最新的数据中搜索到一个异常分隔符，则此数据前写入的数据(紧跟着这个数据的下一个接近终点的)将被当作最新的数据读取。

(4)　数据存储的标准流程和搜索操作

数据存储的标准流程和搜索操作描述如下(在此例中，block 14 和 15 用作 EEPROM 模拟指定 blocks)。

(状态 1) 设置 Block 14 为有效 block。

Block 14	+0	
有效标志	00H	0E00H
无效标志	FFH	0E01H
数据	FFH	0E02H

(状态 2) 写入数据编号 1(对应数值 11H 和 22H)。

Block 14	+0	+1	+2	+3	
有效标志	00H				0E00H
无效标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
:	FFH				0E06H

(状态 3) 写入数据编号 2(对应数值 22H, 33H)。

Block 14	+0	+1	+2	+3	
有效标志	00H				0E00H
无效标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
数据	02H	22H	33H	00H	0E06H
:	FFH				0E0AH

(状态 4) 写入数据编号 2 (对应数值 20H, 30H)。

Block 14	+0	+1	+2	+3	
有效标志	00H				0E00H
无效标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
数据	02H	22H	33H	00H	0E06H
数据	02H	20H	30H	00H	0E0AH
:	FFH				0E0EH

(状态 5) 读数据编号 2。

Block 14	+0	+1	+2	+3	
有效标志	00H				0E00H
无效标志	FFH				0E01H
数据 a	01H	11H	22H	00H	0E02H
数据 b	02H	22H	33H	00H	0E06H
数据 c	02H	20H	30H	00H	0E0AH
数据 d	FFH				0E0EH

<1> 因为数据 a 有不同的数据编号，转向下一数据操作。

<2> 因为数据 b 有匹配的数据编号，其分隔符被检测到。并且因为分隔符的值为 00H (正常)，2 字节数据被存储为最新的数据，转向下一数据操作。

<3> 因为数据 c 有匹配的数据编号，其分隔符被检测到。并且因为分隔符的值为 00H (正常)，2 字节数据被存储为最新的数据，转向下一数据操作。

<4> 数据 d 的数据编号为 FFH (终点)，因此读取操作结束。

<5> 读取值变成最新的存储数据 (数据 c)。

以下描述了在存储数据时发生问题，例如发生电源掉电时的操作流程(在此例中，block 14 和 15 作为 EEPROM 模拟指定 block)。

(状态 1) 设置 Block 14 为有效 block。

Block 14	+0	
合法标志	00H	0E00H
非法标志	FFH	0E01H
数据	FFH	0E02H

(状态 2) 写入数据编号 1(对应于数值 11H, 22H)。

Block 14	+0	+1	+2	+3	
合法标志	00H				0E00H
非法标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
:	FFH				0E06H

(状态 3) 当写入数据编号 1 (对应于数值 22H, 33H) 并且不能正常写入(写入非 00H 值)分隔符时发生电源掉电。

Block 14	+0	+1	+2	+3	
合法标志	00H				0E00H
非法标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
数据	01H	22H	33H	01H	0E06H
:	FFH				0E0AH

(状态 4) 读数据编号 1 。

Block 14	+0	+1	+2	+3	
合法标志	00H				0E00H
非法标志	FFH				0E01H
数据	01H	11H	22H	00H	0E02H
数据	01H	22H	33H	01H	0E06H
:	FFH				0E0AH

读取方法

<1> 因为数据 a 有匹配数据编号，其分隔符被检测到。因为分隔符值是 00H (正常)，数据以 2 字节存储到最新的数据，操作转向下一数据。

<2> 因为数据 b 有匹配数据编号，其分隔符被检测到。因为分隔符值是 01H (异常)，操作转向下一数据。

<3> 因为数据 c 的数据编号是 FFH (终点)，读取操作结束。

<4> 读取值变为最新的存储数据 (数据 a)。

4.1.3　有效和无效标志

有效和无效标志以 1 字节为单位被放置在 block 的起始位置，总计 2 字节的数据。同样地，有效和无效标志指示存储在相对应 block 中的数据处于有效或无效状态。

当有效标志的值是 00H 并且无效标志的值为 FFH，相对应的 block 是有效的。其他情况，block 是无效的。

数据被连续存储到有效的 block，如果这个 block 已满，将执行搜索操作来查找新的 block (需要至少 2 个 block)来存储后续数据，搜索完后数据被传输给这些 block(这个数据仅是每个数据编号的最新数据)。数据传输完成后，设置有效/无效标志使下一 block 有效，并使先前的 block 无效。在传输数据给下一 block 时，发生下一 block 已满或电源掉电或发生其他问题的情况下，为了防止数据丢失，本操作可使发生故障前的数据得以保存。

有效和无效标志的操作流程描述如下。

(状态 1) 初值

	Block n		Block n + 1
有效标志	FFH	有效标志	FFH
无效标志	FFH	无效标志	FFH
数据	FFH	数据	FFH
:	:	:	:
数据	FFH	数据	FFH

(状态 2) block n 的有效标志写入 00H

	Block n		Block n + 1
有效标志	00H	有效标志	FFH
无效标志	FFH	无效标志	FFH
数据	FFH	数据	FFH
:	:	:	:
数据	FFH	数据	FFH

(状态 3) 写入数据到 block n

	Block n		Block n + 1
有效标志	00H	有效标志	FFH
无效标志	FFH	无效标志	FFH
数据	数据	数据	FFH
:	:	:	:
数据	FFH	数据	FFH

(状态 4) block n 中的数据已满

	Block n		Block n + 1
有效标志	00H	有效标志	FFH
无效标志	FFH	无效标志	FFH
数据	数据	数据	FFH
:	:	:	:
数据	数据	数据	FFH

(状态 5) 在最新数据的数据编号还没有更新之后写入需要更新的最新的数据到 block n + 1。

	Block n		Block n + 1
有效标志	00H	Valid flag	FFH
无效标志	FFH	Invalid flag	FFH
数据	数据	Data	Data
:	:	:	:
数据	数据	Data	FFH

(状态 6) block n + 1 有效标志写入 00H

	Block n		Block n + 1
有效标志	00H	有效标志	00H
无效标志	FFH	无效标志	FFH
数据	数据	数据	数据
:	:	:	:
数据	数据	数据	FFH

(状态 7) block n 无效标志写入 00H

	Block n		Block n + 1
有效标志	00H	有效标志	00H
无效标志	00H	无效标志	FFH
数据	数据	数据	数据
:	:	:	:
数据	数据	数据	FFH

(状态 8) block n + 1 数据已满

	Block n		Block n + 1
Valid flag	00H	Valid flag	00H
Invalid flag	00H	Invalid flag	FFH
Data	Data	Data	Data
:	:	:	:
Data	Data	Data	Data

(状态 9) 擦除 block n

	Block n		Block n + 1
有效标志	FFH	有效标志	00H
无效标志	FFH	无效标志	FFH
数据	FFH	数据	数据
:	:	:	:
数据	FFH	数据	数据

(状态 10) 在最新数据的数据编号还没有更新之后写入需要更新的最新的数据到 block n。

	Block n		Block n + 1
有效标志	FFH	Valid flag	00H
无效标志	FFH	Invalid flag	FFH
数据	数据	Data	Data
:	:	:	:
数据	FFH	Data	Data

(Status 11) Write 00H to valid flag for block n

	Block n		Block n + 1
Valid flag	00H	有效标志	00H
Invalid flag	FFH	无效标志	FFH
Data	Data	数据	数据
:	:	:	:
Data	FFH	数据	数据

(状态 12) block n + 1 无效标志写入 00H

	Block n		Block n + 1
有效标志	00H	有效标志	00H
无效标志	FFH	无效标志	00H
数据	数据	数据	数据
:	:	:	:
数据	FFH	数据	数据

4.2 EEPROM 模拟程序执行条件

执行 EEPROM 模拟程序前确保符合表 4-1 所列的所有条件。

表 4-1. EEPROM 模拟操作条件

项目	描述
安全推栈区域 (汇编语言: 22 字节)	EEPROM 模拟程序操作期间，用户程序使用的推栈可以继承并继续使用。另外，EEPROM 模拟程序执行一开始时就需要左边描述的堆栈空间。参阅 **5.2 EEPROM 模拟程序使用的资源**对堆栈的进一步描述。
EEPROM 模拟程序 RAM (汇编语言: 11 字节)	该区域的 RAM 必须专用于 EEPROM 模拟，其用来临时存储读取和写入的数据。确保左边所描述的区域是在内部高速 RAM 区并用作数据缓冲区。除了左边所描述的用于程序的 RAM 之外，EEPROM 模拟程序仅使用堆栈区域。
看门狗定时器操作 (WDT)	因为在 EEPROM 模拟程序执行过程中，当 flash 存储器控制处理被执行时没有指令能被执行，则 flash 存储器控制处理对 WDT 计数器清零。此时，设置 WDT 的溢出时间为 10ms 或更长以便不会发生溢出。
禁止复位	不要在 EEPROM 模拟程序操作过程中对微控制器复位 。当发生复位时，flash 存储器中正在被访问的任何数据都变得不确定。
禁止电源掉电或中断	在 EEPROM 模拟程序操作过程中确保微控制器具有稳定的电源供应。当电源掉电时，flash 存储器中正在被访问的任何数据都变得不确定。

注意事项 在执行 EEPROM 模拟程序的过程中需禁止所有的中断。在 EEPROM 模拟程序执行完成后，中断屏蔽状态返回 EEPROM 模拟程序执行前的状态，并且允许中断。

备注 对于看门狗定时器操作和禁止电源掉电或中断的详细描述，请参见 78K0S/Kx1+ 产品用户手册 flash 存储器章**自编程功能注意事项**。

<R>

第五章 EEPROM 模拟程序(定长多数据方法，汇编语言)

这个应用程序使用 flash 存储器的自编程功能，将 flash 存储器作为 EEPROM 存储数据等。

5.1 EEPROM 模拟程序的结构

表 5-1 列出了包含这个程序的文件。

表 5-1. 文件结构

文件名称	功能	类型
EEPROM.asm	EEPROM 模拟处理 这个处理不仅包括 EEPROM 模拟的读写，而且还包括数据搜索和 block 传输过程。	汇编源

5.2 EEPROM 模拟程序所需的资源

这个程序所需的资源如表 5-2 所示。

表 5-2. 资源

资源	描述	
RAM	EEPROM 模拟需要的 RAM	11 字节
	堆栈	22 字节
	EEPROM 写入处理	22 字节
	EEPROM 读取处理	12 字节
ROM	EEPROM 模拟处理	303 字节
	Flash 存储器控制处理	173 字节
	总共	476 字节

5.3 EEPROM 模拟程序的使用

本章描述的 EEPROM 模拟至少使用 flash 存储器的 2 个 block。EEPROM 模拟过程需在 flash 存储器中使用和更新存储 2 字节 数据。

当用户应用程序中嵌入 EEPROM 模拟程序，且条件符合 (参阅 **4.2 EEPROM 模拟程序执行条件**)，并执行指定的程序，可以执行 EEPROM 模拟。

以下说明如何使用“定长多数据方法，汇编语言”程序来操作 EEPROM 模拟。

5.3.1 用户设置初值

用户必须设置以下所列项目用于 EEPROM 模拟程序的初值。

- 用于 EEPROM 的 block 起始编号(定义为常数 EEPROM_BLOCK)
- 用于 EEPROM 的 block 编号(定义为常数 EEPROM_BLOCK_NO)
- 用户的数据数量，数据长度 (分别定义为常数 DATA_NO_MAX 和 LENG)

这些初值都包括在 EEPROM.asm 中。

(1) 用于 EEPROM 的 block 数量和 block 编号

指定用于 EEPROM 模拟的 block 数量。设置的 block 必须是超过 2 个连续的 block。设置的 block 不能与用户程序区域重叠。

通过增加 EEPROM 的 block 编号可以增加 EEPROM 重写次数。不管 EEPROM 模拟的数据总量，我们都推荐除了用于程序之外的任意区域作为 EEPROM 模拟。

示例 **1.** 当使用 2 个 block (block 14 和 15) 作为 EEPROM block

```
EEPROM_BLOCK EQU (14)
EEPROM_BLOCK_NO EQU (2)
```

2. 当使用 4 个 block (block 12, 13, 14, 和 15) 作为 EEPROM block

```
EEPROM_BLOCK EQU (12)
EEPROM_BLOCK_NO EQU (4)
```

(2) 用户的数据数量和数据长度

用户必须设定数据数量和数据长度存储在 EEPROM 中。

根据数据容量来设定数据长度，数据编号(1 字节)，和分隔符(1 字节)，因为 EEPROM 模拟需要数据编号和一个分隔符。

示例 当存储数据为 2 字节时

```
DATA_NO_MAX  EQU (2) ; 当使用两个数据单位(数据编号 0 和 1)时
LENG  EQU (4)        ; 数据长度(包括数据编号和分隔符的容量(总共 2 字节))
```

(3) 重试擦除的次数

设定重试的次数和 flash 存储器 block 需要执行擦除的次数一致。

在本文档的例子程序中 (参见 **附录 B 例子程序列表(定长多数据方法)**), 在 T_A = −10 to +85°C, 4.5 V ≤ V_{DD} ≤ 5.5 V, 和 NERASE ≤ 100 次条件下设定(0.4 秒)。

在不同条件下设定它，要设定的比 block 擦除时间大 8.5ms。一个擦除的时间是 8.5 ms。

备注 在本文档的例子程序中，最大重试次数指定为 225 次(大约 2.16 秒)。如果擦除时间超过 2.1 秒，则必需要改变 SelfFlashBlockErase 函数。

5.3.2 调用 EEPROM 模拟的用户处理

当在用户程序中执行 EEPROM 模拟时，可使用 2 种类型的处理：EEPROM 读和写处理。

<EEPROM 读和写处理>
EEPROM 读和写处理通过设置数据地址的特殊自变量，并调用它们，使 EEPROM 模拟更便利。
EEPROM 读和写处理的汇编版本和 C 语言版本分别包括在 main.asm 和 main.c 中。
下列变量和结构体(RAM)在 EEPROM 模拟中都用于读和写。

用于 main.asm (汇编版本), 用变量 EEPROM_DATA 定义如下。

```
EEPROM_DO:          DS      1       ; 数据编号
EEPROM_DATA:        DS      2       ; 数据
EEPROM_DELIMITER:   DS      1       ; 分隔符
```

用于 main.c (C 语言版本), 用结构体 eeprom_data 定义如下。

```
Struct eeprom_data{
  unsigned char uc_data_no              ; 数据编号
  unsigned char uc_eeprom_data[2]       ; 数据
  unsigned char uc_delimiter            ; 分隔符
};
```

(1) EEPROM 读处理 (__eeprom_read): 从 EEPROM 区域读取设定容量的数据。

用于 main.asm (汇编版本)

- 自变量:
  在设定到 EEPROM_NO 需要读取的数据编号和数据之后，存储 EEPROM_DATA 地址到 AX 寄存器并且执行子程序调用_eeprom_read 函数。
- 返回值(CY 标志):
  返回值是 CY=0 指示正常完成读数据或者 CY=1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

用于 main.c (C 语言版本)

- 自变量:
  在设定 eeprom_data 结构体的 uc_data_no 需要读取的数据编号和数据之后，通过使用 eeprom_data 结构体的地址作为自变量来执行_eeprom_read 函数。

- 返回值(错误标志):
  返回值是 0 指示正常完成读数据或者是 1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

(2) EEPROM 写处理(__eeprom_write): 写入设定容量的数据到 EEPROM 区域

用于 main.asm (汇编版本)

- 自变量:

  在设定要分别写入到 EEPROM_NO，EEPROM_DATA，和 EEPROM_DELIMITER 数据的数据编号，数据和分隔符之后，存储 EEPROM_NO 地址到 AX 寄存器并且执行__eeprom_write 函数。

- 返回值(CY 标志):

  返回值是 CY=0 指示正常完成读数据或者 CY=1 指示异常完成。如果结果是异常完成，确保检测数据编号是否在设定的范围内。指定的编号的数据如果一次都没有写入的话产生一个错误。

用于 main.c (C 语言版本)

- 自变量:

  在设定要分别写入到 uc_data_no，eeprom_data 结构体，uc_eeprom_data，和 uc_delimiter 数据的数据编号，数据和分隔符之后，存储 EEPROM_NO 地址到 AX 寄存器并且执行__eeprom_write 函数。

- 返回值 (错误标志):

  返回值是 0 指示正常完成读数据或者 1 指示异常完成。

5.4 EEPROM 模拟程序描述

5.4.1 EEPROM 模拟的用户访问处理

表 5-3 和 5-4 列出了用户访问的处理和用来执行 EEPROM 模拟的读写操作。

表 5-3. EEPROM 读处理

(a) 汇编版本

处理名称	__eeprom_read (用户访问函数)
ROM 容量	31 字节
堆栈容量	6 层(12 字节)
输入	AX: 变量地址
返回值	正常完成: CY=0 异常完成: CY=1
操作描述	从 EEPROM 读取指定地址的最后数据到存储地址 1: 搜索用作 EEPROM 的 block。 2: 搜索有效 block 中的最后数据的地址。 3: 从搜索地址中读取最后数据。

(b) C 语言版本

处理名称	_eeprom_read (用户访问函数)
ROM 容量	31 字节
堆栈容量	6 层(12 字节)
输入	AX: 结构体的指针
返回值	正常完成: 错误标志=0 异常完成: 错误标志=1
操作描述	从 EEPROM 读取指定地址的最后数据到存储地址 1: 搜索用作 EEPROM 的 block。 2: 搜索有效 block 中的最后数据的地址。 3: 从搜索地址中读取最后数据。

表 5-4. EEPROM 写处理

(a) 汇编版本

处理名称	__eeprom_write (用户访问函数)
ROM 容量	63 字节
堆栈容量	11 levels (22 字节)
输入	AX: 变量地址
返回值	正常完成: CY=0 异常完成: CY=1
操作描述	数据从存储地址写入到 EEPROM。 1: 搜索用作 EEPROM 的 block。 2: 如果没有有效 block 的话，设定第一个指定的有效 block。 3: 搜索可以写入的有效 block 的地址。 4: 如果有效 block 已满或者无法写入，执行切换到下一个 block 的操作。 5: 创建写入数据。 6: 写入有效 block。

(b) C 语言版本

处理名称	_eeprom_write (用户访问函数)
ROM 容量	63 字节
堆栈容量	11 levels (10 字节)
输入	AX: 结构体的指针
返回值	正常完成: 错误标志=0 异常完成: 错误标志=1
操作描述	数据从存储地址写入到 EEPROM。 1: 搜索用作 EEPROM 的 block。 2: 如果没有有效 block 的话，设定第一个指定的有效 block。 3: 搜索可以写入的有效 block 的地址。 4: 如果有效 block 已满或者无法写入，执行切换到下一个 block 的操作。 5: 创建写入数据。 6: 写入有效 block。

5.4.2 EEPROM 模拟的控制处理 (用于内部处理)

表 5-5 到表 5-10 列出了用于控制 EEPROM 模拟使用的处理。

表 5-5. EEPROM 参数采集处理

处理名称	getprara
ROM 容量	8 字节
堆栈容量	1 层 (2 字节)
输入	AX: 结构体的指针
输出	A=数据编号。复制到 RQDATA_No。 HL=数据地址
操作描述	读取用户调用函数的参数(指针)，获得结构体的内容和需要的参数。

表 5-6. EEPROM Block 搜索处理

处理名称	EEPROMUseBlockSearch
ROM 容量	27 字节
堆栈容量	2 层 (4 字节)
输入	无
输出	正常完成: CY=0, A=Block 表编号 (01H ～ FEH) 异常完成: CY=1, A=下一个结束 block
使用寄存器	A
操作描述	搜索分配给 EEPROM 的 flash 存储器中当前正在使用的 block。

表 5-7. EEPROM Block 初始化处理

处理名称	EEPROMBlockInit
ROM 容量	19 字节
堆栈容量	6 层 (12 字节)
输入	无
输出	正常完成: CY=0 异常完成: CY=1
使用寄存器	A, X, B, C, D, E
操作描述	若 EEPROM 指定 block 中无有效 block，设置最先指定的 block 为当前使用 block (有效)。 如果正常安全则返回 CY = 0。 如果不正常安全则返回 CY = 1。

表 5-8. EEPROM Block 更改处理

处理名称	EEPROMUseBlockChange
ROM 容量	88 字节
堆栈容量	6 层 (12 字节)
输入	A=当前使用的 block 编号
输出	正常完成: CY=0 异常完成: CY=1
使用寄存器	A, X, B, C, D, E
操作描述	如果当前使用 block 数据已满，此功能搜索可用的下一 block ，并将数据复制给这个 block 。 1: 设置下一可用 block。 2: 擦除下一可用 block。 3: 从有效 block 传输最新的数据给下一 block。 4: 设置下一可用 block 为有效的。 5: 设置当前合法 block 为无效的。 6: 存储到新的 block 编号“CurrentB_No”。

表 5-9. EEPROM Block 数据写入从头搜索处理

处理名称	EEPROMWriteTopSearch
ROM 容量	26 字节
堆栈容量	3 层(6 字节)
输入	A: 当前搜索的 block 表编号
输出	搜索成功: CY = 0, 设置写入地址到 AX. 搜索失败: CY = 1
使用寄存器	A, AX
操作描述	搜索指定 block 写入地址。 仅当数据区域在 0FFH 中适合 block 时正常完成。

表 5-10. EEPROM 最新数据搜索处理

处理名称	EEPROMDataSearch
ROM 容量	41 字节
堆栈容量	2 levels (4 字节)
输入	A: 当前使用的 block 表编号, X: 搜索数据编号 5
输出	正常完成: CY=0, DE=最新数据的地址 异常完成: CY=1, E=0
使用寄存器	A, D, E
操作描述	读取符合指定编号的最新数据的存储地址

5.4.3 Flash 存储器的控制处理

表 5-11 ～表 5-18 列出了用于控制 flash 存储器作为 EEPROM 模拟的处理。

表 5-11. Block 擦除

处理名称	SelfFlashBlockErase
ROM 容量	27 字节
堆栈容量	3 层 (8 字节)
输入	A: 被擦除的 block 编号
输出	正常完成: CY=0 异常完成: CY=1
使用寄存器	B
操作描述	擦除指定 block 并执行空白检测。

表 5-12. 模式转换处理 (从自编程模式到正常模式)

处理名称	SelfFlashModeOff
ROM 容量	31 字节
堆栈容量	1 层 (2 字节)
输入	无
输出	无
使用寄存器	A, X
操作描述	释放自编程模式。

表 5-13. 模式转换处理 (从正常模式到自编程模式)

处理名称	SelfFlashModeOn
ROM 容量	35 字节
堆栈容量	1 层 (2 字节)
输入	无
输出	无
使用寄存器	A, X
操作描述	设置自编程模式。

表 5-14. Block 擦除处理

处理名称	FlashBlockErase
ROM 容量	15 字节
堆栈容量	1 层(2 字节)
输入	A: Block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	擦除指定 block。

表 5-15. Flash 自编程函数调用处理

处理名称	SubFlashSelfPrg
ROM 容量	12 字节
堆栈容量	1 层(2 字节)
输入	无
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	调用 falsh 自编程函数。

表 5-16. Block 空白检测处理

处理名称	FlashBlockBlankCheck
ROM 容量	17 字节
堆栈容量	1 层(2 字节)
输入	A: 指定 block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A
操作描述	执行指定 block 的空白检测 。

表 5-17. 设定 block 有效处理

处理名称	SetValid
ROM 容量	9 字节
堆栈容量	1 层(2 字节)
输入	A: Block 编号
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	A, X, B, C, D, E
操作描述	设定使用的 block 有效。

表 5-18．EEPROM 数据写入处理

处理名称	FlashEEPROMWrite
ROM 容量	54 字节
堆栈容量	5 层(10 字节)
输入	DE: 写入开始地址 C: 写入数据量 AX: 写入数据存储地址
输出	正常完成: 零标志(Z)=1 异常完成: 零标志(Z)=0
使用寄存器	D, E
操作描述	写入数据到 EEPROM 并内部校验数据。

5.5 EEPROM 模拟程序流程图

5.5.1 EEPROM 模拟访问处理流程图

图 5-1 和 5-2 显示了用户调用访问处理的流程图，作为执行 EEPROM 模拟读或写操作。

图 5-1. EEPROM 读处理流程图

[概述]
从指定存储地址读取定义为结构体的数据。

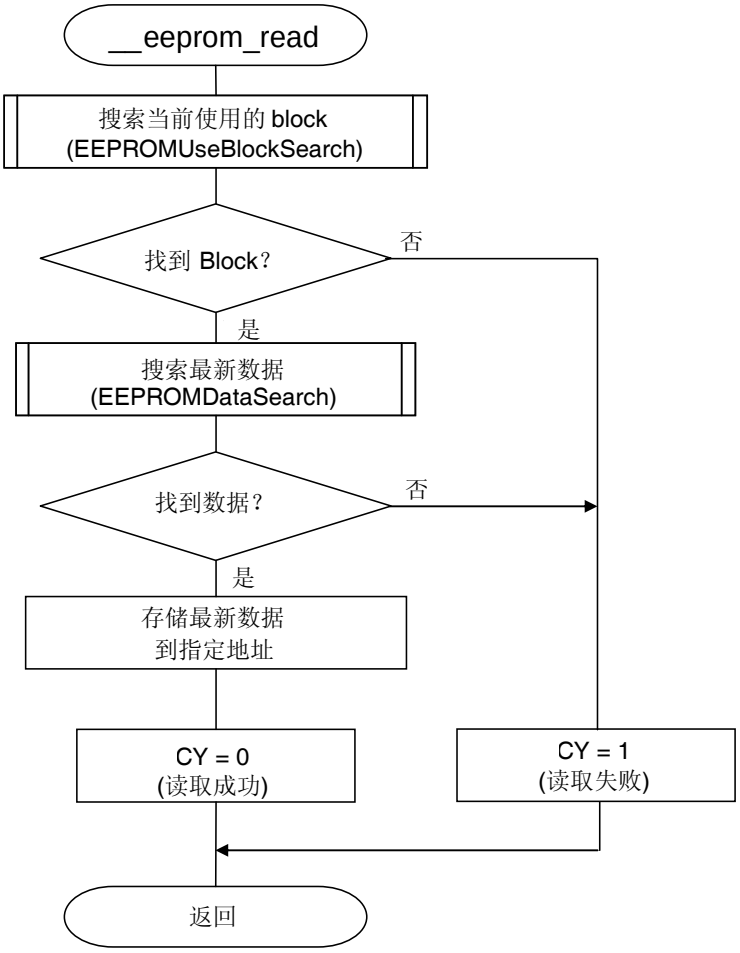
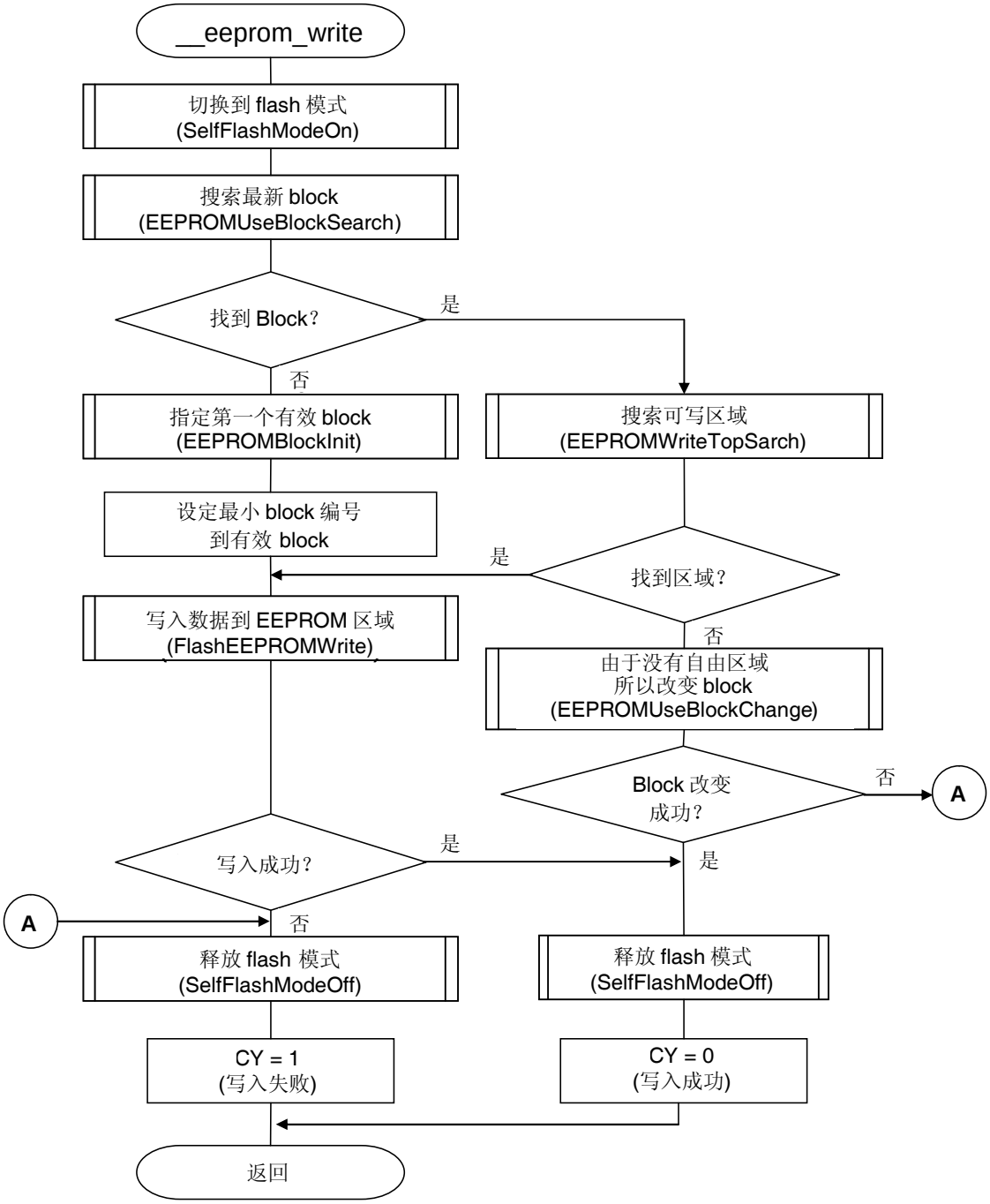


图 5-2. EEPROM 写入处理流程图

[概述]

指定编号的数据从存储地址写入到有效 block。



5.5.2 EEPROM 模拟控制处理流程图

图 5-3 ～5-7 显示了 EEPROM 模拟期间用于模拟控制处理的流程图。

图 5-3. 当前使用的 EEPROM Block 搜索功能流程图

[概述]

搜索当前使用指定为 EEPROM 的 flash 存储器的的 block。

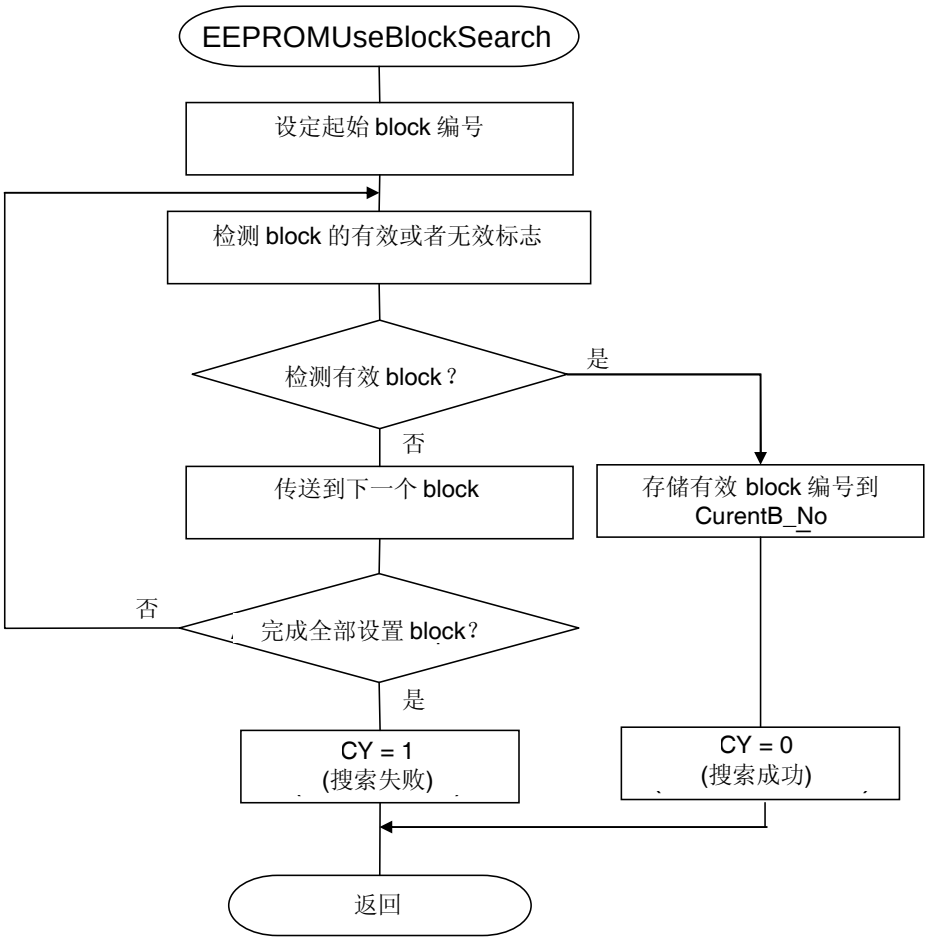


图 5-4. EEPROM Block 初始化处理流程图

[概述]

如果指定用于 EEPROM 的 block 中没有有效的 block，设置第一个制定的 block 为有效的。

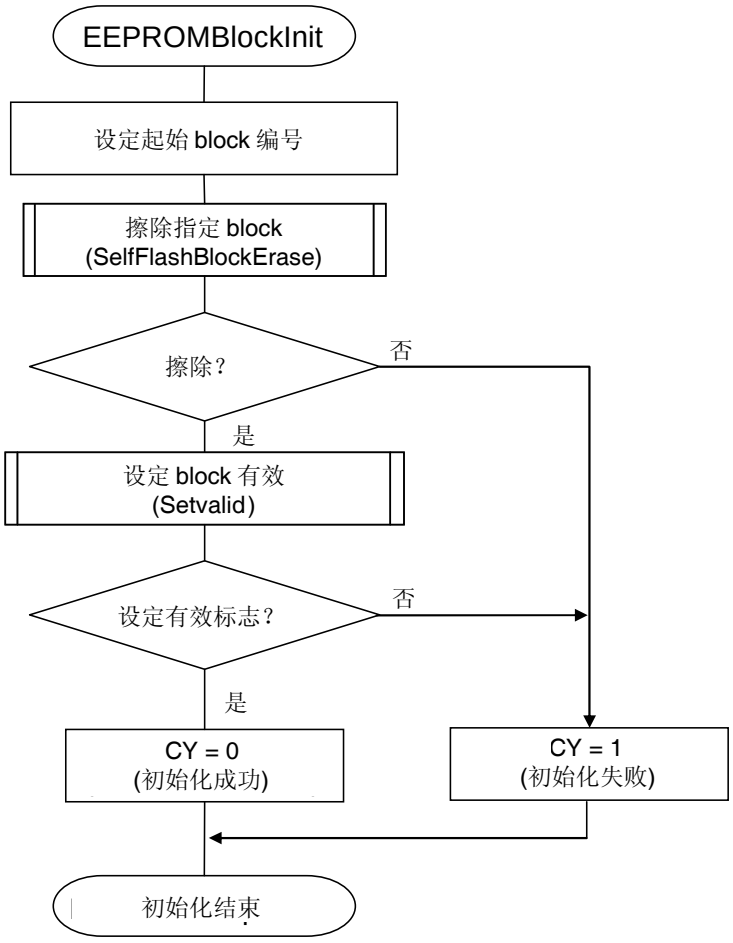


图 5-5. 更改 EEPROM 使用 Block 处理流程图 (1/2)

[概述]

如果当前使用的 block 数据已满，这个函数搜索下一个可用来复制数据新的 block。

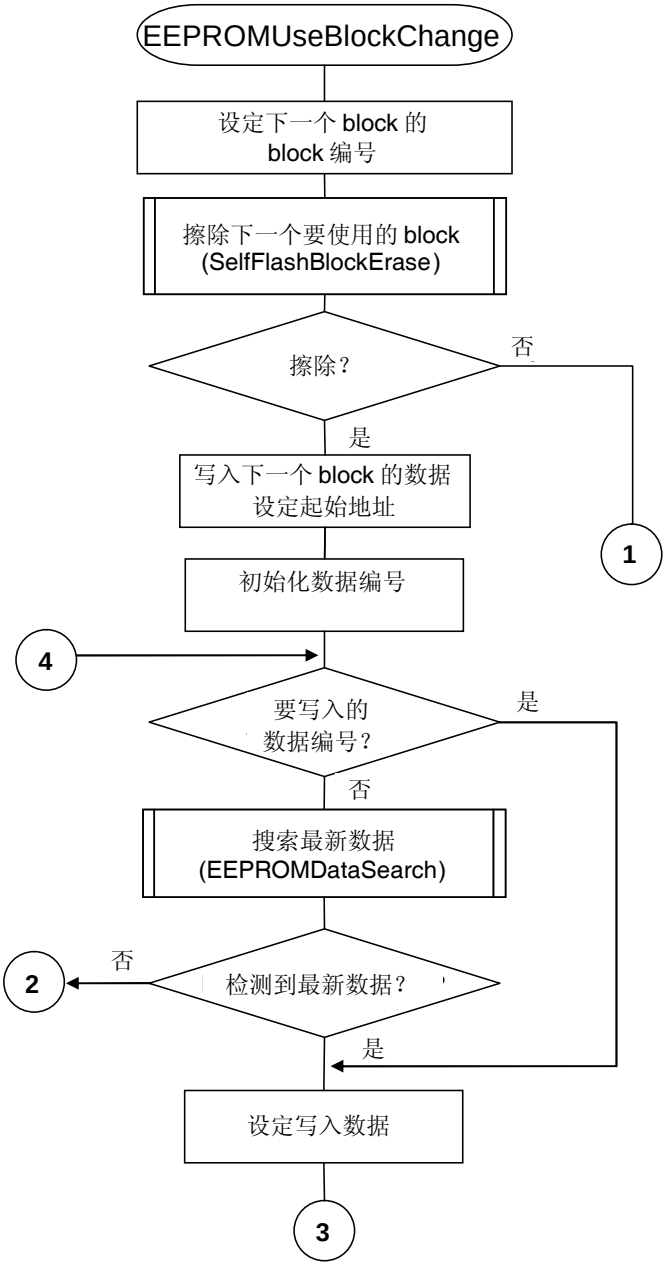


图 5-5. 更改 EEPROM 使用 Block 处理流程图(2/2)

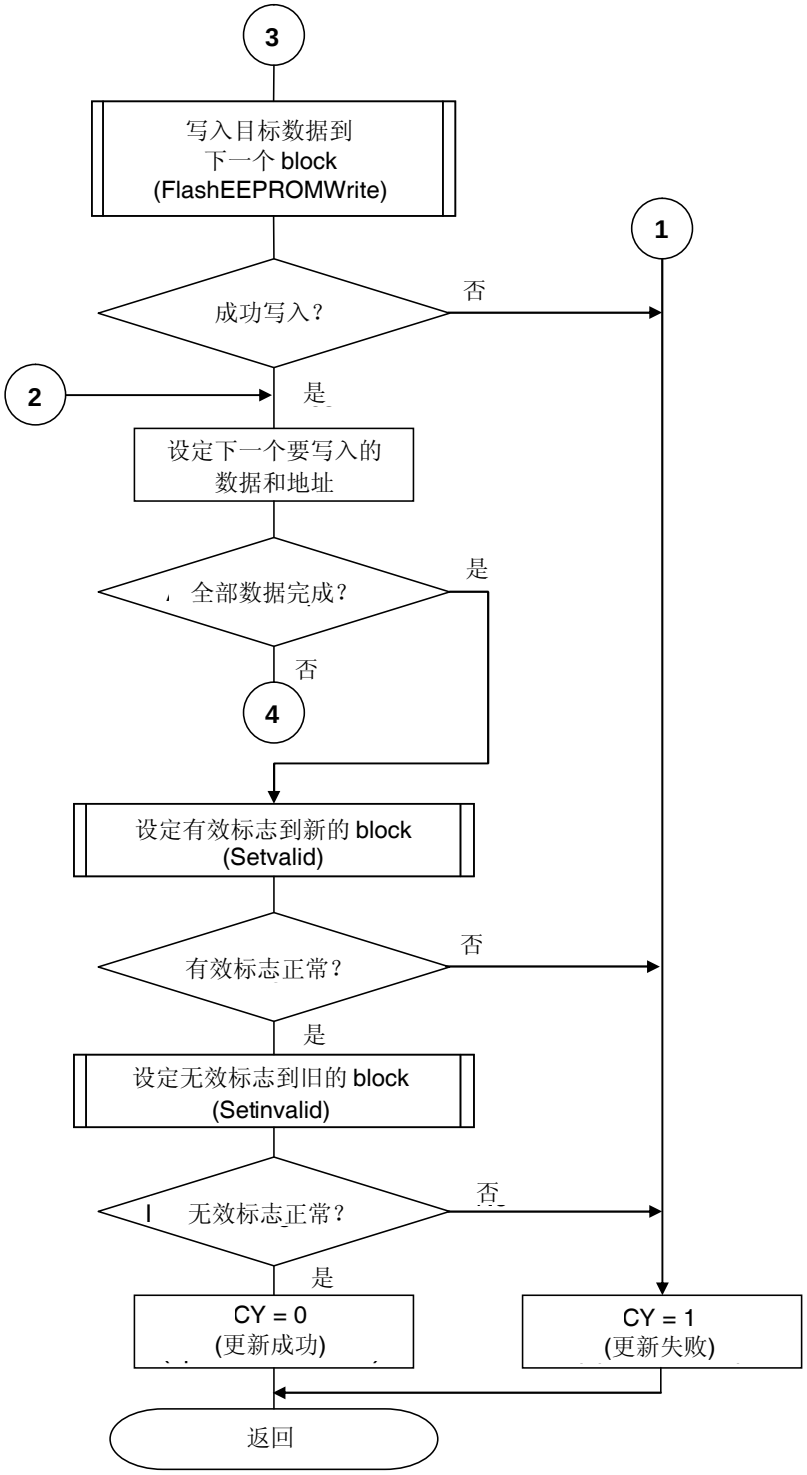


图 5-6. EEPROM 使用 Block 数据写入从头搜索处理流程图

[概述]

搜索指定 block 写入区域。

仅当数据区域在 0xFFH 中适合 block 时正常完成。

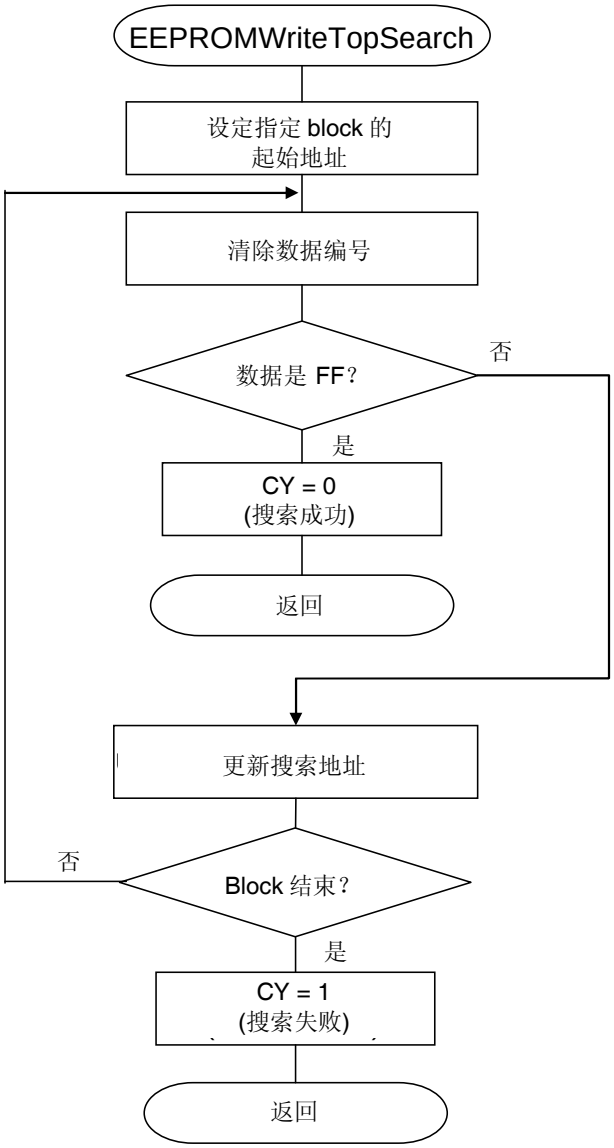
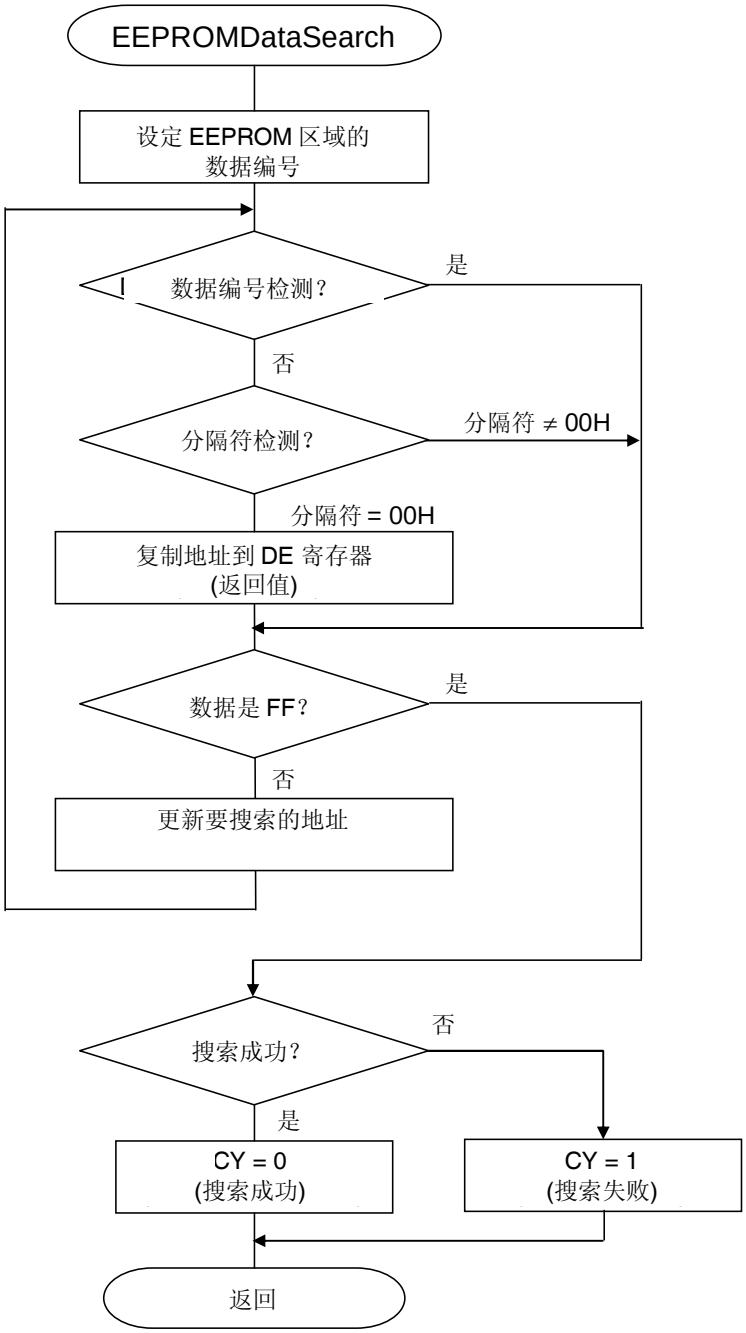


图 5-7. 搜索 EEPROM 最新数据处理流程图

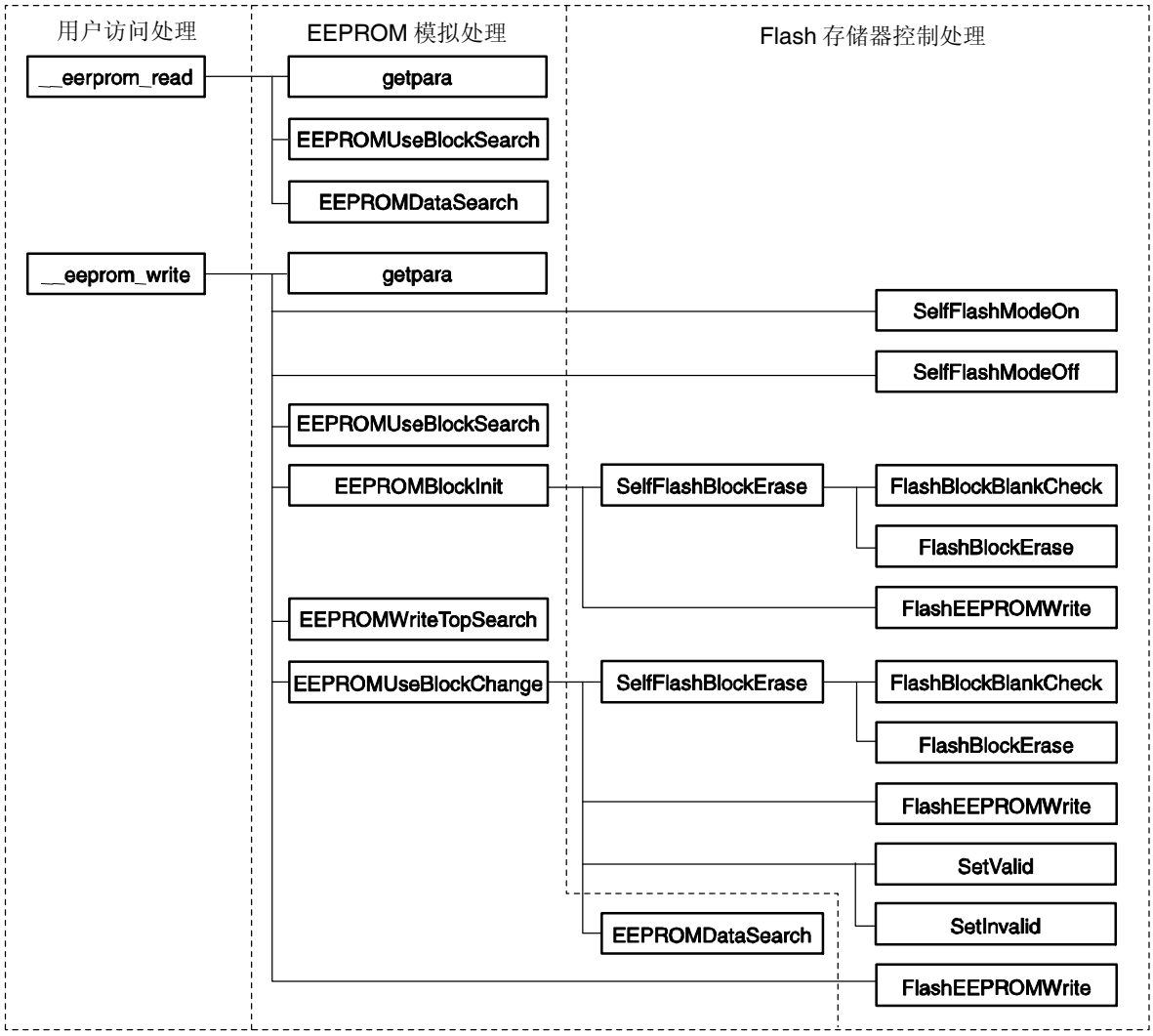
[概述]
读取最新数据的存储地址。



5.6 EEPROM 模拟处理列表

EEPROM 模拟处理调用树如下所示。

图 5-8. 调用树



<R>

A.1 主例子程序 (EEPROM.asm) (定长单数据方法，汇编语言)

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  系统:78K/0S EEPROM 模拟程序
; (定长单数据方法)
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PUBLIC __eeprom_write
PUBLIC __eeprom_read
;-------------------------------------------------------------------------
;       选项字节, 保护字节
;-------------------------------------------------------------------------
OPTBYTE CSEG AT 0080H
        DB 10011100B
         ; ||||||||
         ; |||||||+-- INTOSC  (通过软件来停止允许)
         ; |||||++--- OSCSEL1/0       (高速内部振荡器)
         ; ||||+----- RMCE     (复位)
         ; |||+------ <1>
         ; |++------- DEFOSTS1/0 (最小值)
         ; +--------- <1>

        DB      11111111B       ; 保护字节: 位(6-2) = 11111B
                                ; 保护区域: 无
                                ; 非保护区域: Blocks 0 ～ 15
                                ; 确保要释放 block 的保护
                                ; 设定为 EEPROM 仿真
;-------------------------------------------------------------------------
;       用于 EEPROM 的 Block 编号       **用户设定**
;-------------------------------------------------------------------------
                                        ;例) 当使用 block 14 和 15
EEPROM_BLOCK            EQU     (14)    ; 设定的 block 必须是连续的。
EEPROM_BLOCK_NO         EQU     (2)     ; 使用的 block 编号

;-------------------------------------------------------------------------
;       要写入数据的数据长度              **用户设定**
;-------------------------------------------------------------------------
LENG                    EQU     (3)             ; 数据长度(包括分隔符)

;-------------------------------------------------------------------------
;       其它 EQU 设定
;-------------------------------------------------------------------------
CMD_INTERNAL_VERIFY2            EQU     (00000010B)     ; 内部校验 2
CMD_BLOCK_ERASE                 EQU     (00000011B)     ; Block 擦除
CMD_BLANK_CHECK                 EQU     (00000100B)     ; Blank 检测
CMD_BYTE_WRITE                  EQU     (00000101B)     ; 1 字节写入
```

```
NO_ERR                          EQU     (00000000B)     ; 未发现错误
ERR_FPRERR                      EQU     (00000001B)     ; PFCMD 写入错误
ERR_VCERR                       EQU     (00000010B)     ; 擦除/写入/内部校验错误
ERR_WEPRERR                     EQU     (00000100B)     ; 保护错误
FALSE                           EQU     (0FFH)          ; “FALSE” 结果
TRUE                            EQU     (00H)           ; “TRUE” 结果
DATATOP                         EQU     (2)             ; 数据写入起始地址
LENGB                           EQU     LENG-1          ; 数据长度 – 1
NEXT_BLOCK_WADDR                EQU     DATATOP+LENG    ; 下一 block 写入地址 (仅单数据有效)
ERASE_RETRY                     EQU     (48)            ; 为擦除进行的重新计数(4.0 ～ 5.5 V时 100 次)
EEPROM_END_BLOCK                EQU     EEPROM_BLOCK+EEPROM_BLOCK_NO-1
;---------------------------------------------------------------------
;       RAM 设置
;---------------------------------------------------------------------
EP_RAM1         DSEG    saddr
d2MaskStatus:   DS      2       ; 用于存储掩模数据的 RAM
d2SetAdr:       DS      2       ; 在模块转换过程中用于存储数据传送目的地址的 RAM
CurentB_No:     DS      1       ; 在 EEPROM 读/写进程中用于存储 block 的 RAM
ci:             DS      1       ; 通用目的循环计数器

EEPROM_PROG     CSEG
;---------------------------------------------------------------------
; 函数名称:         __eeprom_read (用户访问功能)
; 输入:             AX = 结构体指针
; 返回值:           CY = 0 (正常完成) 或者 CY = 1 (异常终止)
; 摘要:             将指定的数据从 EEPROM 中读取到存储地址。
;                   1. 搜索用做 EEPROM 的 block。
;                   2. 在有效的 block 中寻找最新的数据地址。
;                   3. 在搜索到的地址中读取最新的数据
;---------------------------------------------------------------------
;bit _eeprom_read(struct_eeprom_data *)

__eeprom_read:
        PUSH    HL
        PUSH    AX

        MOVW    HL,AX                   ; 从读取数据地址复制到 HL

        CALL    !EEPROMUseBlockSearch   ; 搜索当前正在用于 EEPROM 的 block
        BC      $EP_READ_FL2            ; 异常终止

        PUSH    DE                      ; 保存寄存器值
        CALL    !EEPROMDataSearch       ; 设定指定数据编号的最新数据的地址到 DE
                                        ; 使用 EEPROMUseBlockSearch 函数搜索 block 编号输入到 A 寄存器
        BC      $EP_READ_FL             ; 如果数据没有找到就退出

        MOV     ci,#LENGB               ; 设定容量，包括分隔符
```

```
EP_READ_LP:
        MOV     A,[DE]                  ; 读取 EEPROM 数据
        MOV     [HL],A                  ; 存储数据
        INCW    DE                      ; 更新读取地址(由于固定长度稍后执行)
        INCW    HL                      ; 更新存储地址
        DBNZ    ci,$EP_READ_LP          ; 复制数据长度
        CLR1    CY                      ; 返回值 = 正常完成(TRUE)

EP_READ_FL:
        POP     DE
EP_READ_FL2:
        POP     AX
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名称:       __eeprom_write (用户访问功能)
; 输入:           AX = 结构体指针
; 返回值:         CY = 0(正常完成) 或者 CY = 1 (异常终止)
; 摘要:           把指定编号的数据从存储地址写入 EEPROM。
;                 1. 搜索用做 EEPROM 的 block。
;                 2. 如果没有有效地 block，则把第一个指定的 block 设定为有效。
;                 3. 搜索用于允许写入到有效 block 的地址。
;                 4. 如果有效的 block 已满或者不能被写入，操作将转移到下一个block。
;                 5. 创建写入数据。
;                 6. 写入到有效的 block。
;---------------------------------------------------------------------
;bit _eeprom_write(struct_eeprom_data *)

__eeprom_write:
        PUSH    HL
        PUSH    AX
        PUSH    DE
        PUSH    BC
        MOVW    HL,AX                   ; 从写入数据地址复制到 HL
        CALL    !SelfFlashModeOn        ; 设定 flash 模式
        CALL    !EEPROMUseBlockSearch   ; 搜索当前用作 EEPROM 的 block
        BNC     $EP_WRITE_4             ; 如果没有找到当前使用的 block 则跳转
;
;   由于没有找到当前使用的 block 开始使用新的 block
;
        CALL    !EEPROMBlockInit        ; 如果没有找到当前使用的 block，指定作为 EEPROM 的
                                        ; 最小编号的 block 为有效 block
        BC      $EP_WRITE_FL            ; 如果保护失败则跳转并且完成

EP_WRITE_2:
        MOVW    AX,#EEPROM_BLOCK*100H+DATATOP
        MOV     CurentB_No,A            ; 要使用的分隔符 block 编号
```

```
;
;  数据写入进程
;

EP_WRITE_3:
        MOVW DE,AX                      ; 设定当前写入地址到 DE
        MOVW    AX,HL                   ; 读取结构体地址
        XCHW    AX,DE                   ; 设定结构体地址到 DE
        MOV     C,#LENG
        CALL    !FlashEEPROMWrite       ; 写入数据到 EEPROM
        BZ      $EP_WRITE_TR

EP_WRITE_FL:
        CALL    !SelfFlashModeOff       ; 释放 flash 模式
        SET1    CY                      ; 返回值 = 异常终止(FALSE)
        BR      $EP_WRITE_E

;
;  如果当前 block 有效
;
EP_WRITE_4:
        CALL    !EEPROMWriteTopSearch   ; 搜索可写入区域(存储写入地址到 AX)
        BNC     $EP_WRITE_3             ; 因为区域没有找到而跳转到数据写入

;
;   因为区域里没有足够的空间而传送下一个 block。
;

        CALL    !EEPROMUseBlockChange   ; 改变使用的 block
        BC      $EP_WRITE_FL            ; 异常终止退出
;
;  正常完成结束
;
EP_WRITE_TR:
        CALL    !SelfFlashModeOff       ; 释放 flash 模式
        CLR1    CY                      ; 返回值 = 正常完成(TRUE)

EP_WRITE_E:
        POP     BC
        POP     DE
        POP     AX
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名称:         EEPROMUseBlockSearch
; 输入:             无
```

```
; 输出:
;    正常完成:      CY = 0, A = block 表编号(00H ～ FEH)
;    异常终止:      CY = 1, A = 下一个结束 block
; 使用的寄存器:     A
; 摘要:  搜索份配给 EEPROM 的 flash 存储器中当前正在使用的 block。
;---------------------------------------------------------------------
EEPROMUseBlockSearch:
        PUSH    HL
        MOV     ci,#EEPROM_BLOCK_NO    ; 循环的数量 = EEPROM 设定 block 的数量
        MOVW    HL,#EEPROM_BLOCK*100H  ; 设定起始 block 编号
EP_BSRC_T:
        MOV     A,[HL+1]               ; 读取无效标志
        INC     A                      ; 如果无效标志 = FF 结果为00
        OR      A,[HL]                 ; 如果有效标志 = 00 结果为00
        ADD     A,#0FFH                ; 如果结果非 00 设定进位标志
        BNC     $EP_BSRC_E             ; 如果 CY = 00 则 block 有效
        INC     H                      ; 如果无效则移动到下一个 block
        DBNZ    ci,$EP_BSRC_T          ; 循环到 block 的数量结束
EP_BSRC_E:
        MOV     A,H                    ; 设定 block 编号
        MOV     CurentB_No,A           ; 存储使用的 block 编号到 RAM
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名称:        EEPROMBlockInit
; 输入:            无
; 输出:
;    正常完成:      CY = 0
;    异常终止:      CY = 1
; 使用的寄存器:    A, X, B, C, D, E
; 摘要:            如果用于 EEPROM 的 block 没有有效的，则第一个被指定的 block 被
;                  设定为当前应用的(有效的)。
;                  安全正常返回 CY = 0
;                  安全异常返回 CY = 1
;---------------------------------------------------------------------
EEPROMBlockInit:
        PUSH HL
        MOV     A,#EEPROM_BLOCK        ; 设定 block 的最小编号
        CALL    !SelfFlashBlockErase   ; 擦除指定 block
        BC      $EP_BINI_FL            ; 异常终止
        CALL    !Setvalid              ; 设定 block 有效
        BZ      $EP_BINI_E             ; 如果正常完成则跳转
        SET1    CY                     ; 如果设定为有效则产生一个错误
EP_BINI_FL:
EP_BINI_E:
        POP     HL
        RET
```

```
;---------------------------------------------------------------------
; 函数名称:        EEPROMUseBlockChange
; 输入:            X = 当前使用的 block 编号
; 输出:
;   正常完成:      CY = 0, block 表编号 (02H ～ FEH)
;                  存储新的 block 编号到 “CurrentB_No”.
;   异常终止:      CY = 1
; 使用的寄存器:    A, X, B, C, D, E
; 摘要:            如果当前使用的 block 已经写满了数据，该函数搜索下一个 block 用于写入数据。
;                  1. 设定下一个被使用的 block。
;                  2. 擦除下一个被使用的 block。
;                  3. 把最新的数据从有效的 block 传送到下一个 block。
;                  4. 把下一个 block 设定为有效。
;                  5. 把当前有效的 block 设定为无效。
;---------------------------------------------------------------------
EEPROMUseBlockChange:
        PUSH    HL
        CMP     A,#EEPROM_END_BLOCK    ; 和最后 block 比较
        INC     A                      ; 如果找到剩余则传送到下一个 block。
        BC      $EP_BCHG_2             ; 如果没有找到剩余
        MOV     A,#EEPROM_BLOCK        ; 选择起始 block
;
;   擦除下一个被使用的 block 并且准备使用
;

EP_BCHG_2:
        CALL    !SelfFlashBlockErase   ; 擦除下一个被使用的 block
        BC      $EP_BCHG_E             ; 如果不能擦除 block 则退出

EP_BCHG_3:
        MOV     X,#DATATOP             ; 设定下一个写入起始地址的 block
        MOVW    d2SetAdr,AX            ; 保存地址到参数区域

;
;   复制处理: 用于直接从旧 block 写入到新 block 的处理
;

EP_BCHG_T:
;
;   复制数据到新 block
;
        PUSH    HL                     ; 复制需要写入的数据到 DE 寄存器
        POP     DE

        MOV     C,#LENG                ; 循环数据长度
        MOVW    AX,d2SetAdr            ; 设定数据写入的地址
        CALL    !FlashEEPROMWrite      ; 写入数据到新 EEPROM 区域
```

```
        SET1    CY                      ; 预先设定错误标志
        BNZ     $EP_BCHG_E              ; 如果异常终止则退出

;
;   因为已经完成数据传送设定新 block 有效
;

        MOV     A,d2SetAdr+1            ; 读取 block 编号
        CALL    !Setvalid               ; 写入有效标志
        SET1    CY                      ; 设定错误标志
        BNZ     $EP_BCHG_E              ; 如果异常终止则退出

;
;   设定旧 block 无效
;

        XCH     A,CurentB_No            ; 设定旧 block 编号
        MOV     X,#1                    ; 设定无效标志地址
        CALL    !Setinvalid             ; 设定无效标志
        SET1    CY                      ; 预先设定错误标志
        BNZ     $EP_BCHG_E              ; 如果异常终止则退出
        CLR1    CY                      ; 清除标志
;
;   正常完成返回新 block 编号(CurentB_No)
;
        MOV     A,CurentB_No            ; 设定新 block 编号
        MOV     X,#NEXT_BLOCK_WADDR     ; 设定新写入地址

EP_BCHG_E:
        POP     HL
        RET
```

```
;-------------------------------------------------------------------
; 函数名称:       EEPROMWriteTopSearch
; 输入:           A = 搜索 block 表编号
; 输出:
;       搜索成功:           CY = 0, 设定写入地址到 AX
;       搜索失败:           CY = 1
; 使用的寄存器:  A,X,B,D,E
; 摘要:          搜索可以在指定 block 写入的区
;                仅当数据编号区域在 0ffH 适合时正常完成
;-------------------------------------------------------------------
EEPROMWriteTopSearch:
        PUSH    HL
        MOV     H,A
        MOV     L,#DATATOP

EP_WTSR_T:
        MOV     A,[HL+LENGB]    ; 读取数据
        CMP     A,#0FFH         ; 分隔符处于不能写入状态 (FFH)?
        BNZ     $EP_WTSR_2      ; 如果不是 FF 检测下一个数据

                                ; 通过电源中断补偿检测是否所有数据是 (FFH)
        PUSH    HL              ; 复制当前搜索位置到 DE
        POP     DE
        MOV     B,#LENGB        ; 设定数据的数量 (循环的数量)

EP_WTSR_1:
        MOV     A,[DE]          ; 读取数据
        CMP     A,#0FFH         ; 没有写入数据 (FFH)?
        BNZ     $EP_WTSR_2      ; 如果没有找到写入数据则检测下一数据
        INC     E               ; 检测地址加一
        DBNZ    B, $EP_WTSR_1   ; 检测全部数据?
        BR      $EP_WTSR_3      ; 如果全部数据都不能写入则完成搜索

EP_WTSR_2:
        MOV     A,L             ; 更新地址
        ADD     A,#LENG
        MOV     L,A
        BC      $EP_WTSR_3      ; 如果 block 没有完成则带错误退出
        ADD     A,#LENG-1       ; 如果剩下的至少 LENG 则继续
        BNC     $EP_WTSR_T      ; 否则设置 CY

EP_WTSR_3:

        MOVW    AX,HL           ; 返回值 = 搜索指针的地址
        POP     HL
        RET

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

```
; 函数名称:       EEPROMDataSearch
; 输入:           A = 当前使用的 block 表编号
;                 (通过用户定义设置用于 EEPROM 的block 编号)
; 输出:
;   正常完成:     CY = 0, DE = 最新数据的地址
;   异常终止:     CY = 1, E = 0
; 使用的寄存器:   A, X, D, E
; 摘要:           读取最新数据的存储地址。
;---------------------------------------------------------------------
EEPROMDataSearch:
        PUSH    HL
        MOV     E,#0            ; 设定结果的初始值为错误
        MOV     H,A             ; 设定 block 的高地址
        MOV     A,#DATATOP      ; 设定低地址

EP_DASR_1:                      ; 因为单数据定长，不能查看数据编号
        MOV     L,A             ; 更新数据地址
        ADD     A,#LENG         ; 检测下一地址是否超出 block
        BC      $EP_DASR_E      ; 如果超出 block 则完成
        XCH     A,X             ; 存储更新地址到 X 寄存器

        MOV     A,[HL+LENGB]    ; 读取分隔符
        CMP     A,#00H          ; 如果分隔符非“00”则读取组成的数据
        BNZ     $EP_DASR_2      ; 以定长单数据，搜索到 block 结尾来忽略在电源中断时的写入数据失败
        PUSH    HL              ; 如果分隔符有效
        POP     DE              ; 复制地址到 DE
EP_DASR_2:
        XCH     A,X             ; 返回下一地址到 A 寄存器
        BR      $EP_DASR_1      ; 读取下一数据

EP_DASR_E:
        MOV     A,E             ; 提取数据的低地址
        CMP     A,#DATATOP      ; 如果错误则设定 CY (如果数据没有找到则 A = 0)
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名称:       SelfFlashBlockErase
; 输入:           A = 擦除 block 编号
; 输出:           CY = 0 (正常完成) 或者 CY = 1 (异常终止)
; 使用的寄存器:   B
; 摘要:           擦除指定 block 并且执行空白检测。
;---------------------------------------------------------------------
SelfFlashBlockErase:
        PUSH    AX
        MOV     X,A                     ; 保存 block 编号
        MOV     B,#ERASE_RETRY          ; 设定擦除时的重试次数
        BR      $SF_BKER_ST             ; 从空白检测开始
```

```
SF_BKER_RE:
        MOV     A,X                     ; 设定擦除 block
        CALL    !FlashBlockErase        ; 擦除处理
        BNZ     $SF_BKER_01             ; 异常终止
        MOV     A,X                     ; 设定空白检测 block

SF_BKER_ST:
        CALL    !FlashBlockBlankCheck   ; 空白检测进程
        BZ      $SF_BKER_TR             ; 空白检测正常完成
SF_BKER_01:
        DBNZ    B,$SF_BKER_RE           ; 检测重试计数
SF_BKER_FL:
        SET1    CY                      ; 返回值 = 异常终止

SF_BKER_TR:
        POP     AX
        RET

;---------------------------------------------------------------------
; 函数名称:         SelfFlashModeOff
; 输入:             无
; 输出:             无
; 使用的寄存器:     A, X
; 摘要:             释放自编程模式处理
;---------------------------------------------------------------------
SelfFlashModeOff:
        MOV     FLCMD,#0
        MOV     PFS,#NO_ERR
        MOV     PFCMD,#0A5H                     ; PFCMD 寄存器控制
        MOV     FLPMC,#00H                      ; FLPMC 寄存器控制 (设定值)
        MOV     FLPMC,#0FFH                     ; FLPMC 寄存器控制 (反向设定值)
        MOV     FLPMC,#00H                      ; 设定正常模式: FLPMC 寄存器控制 (设定值)
        BT      PFS.0,$SelfFlashModeOff         ; 检测写入到指定寄存器是否完成
        EI
        MOVW    AX,d2MaskStatus
        MOV     MK1,A                           ; 使用 78K0S/KY1+ 时删除这行
        XCH     A,X
        MOV     MK0,A                           ; 恢复中断屏蔽标志
        RET

;---------------------------------------------------------------------
; 函数名称:         SelfFlashModeOn
; 输入:             无
; 输出:             无
; 使用的寄存器:     A, X
; 摘要:             设定自编程模式的处理
;---------------------------------------------------------------------
```

```
SelfFlashModeOn:
        MOV     A,MK0
        XCH     A,X
        MOV     A,MK1                   ; 使用 78K0S/KY1+ 时删除这行
        MOVW    d2MaskStatus,AX         ; 保存中断屏蔽标志
        MOV     MK0,#11111111B
        MOV     MK1,#11111110B          ; 屏蔽除 INTFLC 之外的中断，使用 78K0S/KY1+ 时删除这行
        DI
SF_MDON_LP:
        MOV     PFS,#NO_ERR
        MOV     PFCMD,#0A5H             ; PFCMD 寄存器控制
        MOV     FLPMC,#01H              ; FLPMC 寄存器控制 (设定值)
        MOV     FLPMC,#0FEH             ; FLPMC 寄存器控制 (反向设定值)
         MOV    FLPMC,#01H              ; 设定自变成模式: FLPMC 寄存器控制 (设定值)
        NOP
        HALT
        BT      PFS.0,$SF_MDON_LP       ; 检测写入到指定寄存器是否完成
        RET

;---------------------------------------------------------------------
; 函数名称:         FlashBlockErase
; 输入:             A = block 编号
; 输出:             Z (零标志) = 正常完成 (1) 或者异常终止 (0)
; 使用的寄存器:     A
; 摘要:             擦除指定的 block。
;---------------------------------------------------------------------
FlashBlockErase:
        MOV     FLCMD,#CMD_BLOCK_ERASE ; 设定 flash 控制命令 (block 擦除)
        MOV     FLAPH,A                ; 设定 blank 检测 block 编号
        MOV     FLAPL,#00H
        MOV     FLAPHC,A
        MOV     FLAPLC,#00H
;
;  继续执行目前的 flash 自编程
;
;---------------------------------------------------------------------
; 函数名称:         SubFlashSelfPrg
; 输入:             无
; 输出:             Z (零标志) = 正常完成 (1) 或者异常终止 (0)
; 使用的寄存器:     A
; 摘要:             调用 flash 自编程函数。
;---------------------------------------------------------------------
SubFlashSelfPrg:
        MOV     PFS,#NO_ERR             ; 清除 flash 状态寄存器
        MOV     WDTE,#0ACH              ; 清楚和重启 WDT
        HALT                            ; 开始自编程
        MOV     A,PFS
        CMP     A,#NO_ERR
```

```
        RET

;---------------------------------------------------------------------
; 函数名称:        FlashBlockBlankCheck
; 输入:            A = Block 编号
; 输出:            Z(零标志) = 正常完成 (1) 或者异常终止 (0)
; 使用的寄存器:    A
; 摘要:            执行指定 block 的空白检测。
;---------------------------------------------------------------------
FlashBlockBlankCheck:
        MOV     FLCMD,#CMD_BLANK_CHECK ; 设定 flash 控制命令 (block 空白检测)
        MOV     FLAPH,A                ; 设定空白检测 block 编号
        MOV     FLAPL,#00H
        MOV     FLAPHC,A
        MOV     FLAPLC,#0FFH
        BR      $SubFlashSelfPrg

;---------------------------------------------------------------------
; 函数名称:        Setvalid
; 输入:            A = 目标 block 编号
; 输出:            Z (零标志) =正常完成(1) or 或者异常终止 (0)
; 使用的寄存器:    A, X, C, D, E
; 摘要:            设定使用的 block 有效。
;---------------------------------------------------------------------
Setvalid:
        MOV     X,#0                   ; AX 指示有效标志的地址
Setinvalid:
        MOVW    DE,#Setinvalid-1       ; 指示 0 作为要写入的数据
        MOV     C,#1                   ; 写入数据是 1 字节
;
;   下面是写入函数处理
;

;---------------------------------------------------------------------
; 函数名称:        FlashEEPROMWrite
; 输入:            DE = 写入起始地址
;                  C = 要写入的数据 block 的数量
;                  AX = 写入数据存储地址
; 输出:            Z(零标志 ) =正常完成(1) 或者异常终止 (0)
; 使用的寄存器:    D, E
; 摘要:            写入数据到 EEPROM 并且内部校验数据。
;---------------------------------------------------------------------
FlashEEPROMWrite:
        PUSH    HL
        PUSH    AX
        PUSH    BC
        MOVW    HL,AX
        MOV     FLCMD,#CMD_BYTE_WRITE          ; 设定 flash 控制命令 (字节写入)
```

```
FL_WR_W1:
        MOVW    AX,HL
        MOV     FLAPH,A
        XCH     A,X
        MOV     FLAPL,A                         ; 设定写入/校验地址
        MOV     A,[DE]
        MOV     FLW,A                           ; 设定写入数据
        CALL    !SubFlashSelfPrg
        BZ      $FL_WR_W2                       ; 如果正常完成则退出
        POP     BC
        BR      $FL_WR_E
FL_WR_W2:
        INCW    DE                              ; 读地址 + 1
        INCW    HL                              ; 写地址 + 1
        DBNZ    C,$FL_WR_W1                     ; 循环指定字节的数量

        POP     BC                              ; 写入完成
        MOV     FLCMD,#CMD_INTERNAL_VERIFY2     ; 设定 flash 控制命令 (内部校验)
        MOV     FLAPH,A
        XCH     A,X
        MOV     FLAPL,A                         ; 设定校验开始地址
        DEC     C
        ADD     A,C
        XCH     A,X
        ADDC    A,#0
        MOV     FLAPHC,A
        XCH     A,X
        MOV     FLAPLC,A                        ; 设定校验结束地址
        CALL    !SubFlashSelfPrg
FL_WR_E:
        POP     AX
        POP     HL
        RET

          END
```

A.2 测试例子程序(定长单数据方法，汇编语言)

```
;*******************************************************************************
;
;       78K0S/Kx1+ EEPROM 模拟例子主程序
;
;*******************************************************************************

;===============================================================================
;
;       EEPROM 外部引用声明
;
;===============================================================================
EXTRN __eeprom_read    ; EEPROM 读函数
EXTRN __eeprom_write   ; EEPROM 写函数

;===============================================================================
;
;       RAM 定义
;
;===============================================================================
XRAM    DSEG    SADDR
EEPROM_DATA:    DS      2       ; EEPROM 数据
DELIMITER:      DS      1       ; 分隔符

;===============================================================================
;
;       可靠的堆栈区域
;
;===============================================================================
XSTK    DSEG    AT      0FEE0H
STACKEND:
        DS      20H                     ; 保护堆栈区域的 32 字节
STACKTOP:                               ; 堆栈区域起始地址 = FF00H

;===============================================================================
;
;       向量表设定
;
;===============================================================================
XVCT    CSEG    AT      0000H
        DW      RESET_START             ;(00)   复位

;*******************************************************************************
;
;       复位释放后的初始化处理
;
;*******************************************************************************
```

```
XMAIN   CSEG    UNIT
RESET_START:

;---------------------------------------------------------------------------
;       堆栈指针设定
;---------------------------------------------------------------------------
        MOVW    AX,     #STACKTOP
        MOVW    SP,     AX                              ; 设定堆栈指针

;---------------------------------------------------------------------------
;       看门狗定时器设定
;---------------------------------------------------------------------------
        MOV     WDTM,   #01110111B      ; 停止看门狗定时器操作

;---------------------------------------------------------------------------
;       时钟频率设定
;---------------------------------------------------------------------------
        MOV     PPCC,   #00000000B      ; 外围硬件提供时钟 fxp = fx (= 8MHz)
        MOV     PCC,    #00000000B      ; CPU 时钟 fcpu = fxp (= 8MHz)
        MOV     LSRCM,  #00000001B      ; 停止低速内部振荡器

;***************************************************************************
;
;       主处理
;
;***************************************************************************
        MOV     DELIMITER, #00H         ; 初始化分隔符

        MOVW AX, #EEPROM_DATA           ; 传送 EEPROM 数据地址到 HL
        MOVW HL, AX

        PUSH AX                         ; 保存 EEPROM 数据地址

        MOV     A, #01H                 ; 设定 EEPROM 存储地址
        MOV [HL], A
        MOV     A, #10H                 ; 设定 EEPROM 存储数据
        MOV [HL+1], A

        POP AX                          ; 恢复 EEPROM 数据地址

        ; 写测试
        CALL !__eeprom_write            ; 调用 EEPROM 写函数
        BC      $LOOP                   ; 如果错误则跳转到 LOOP

        ; 清除测试数据
        PUSH AX                         ; 保存 EEPROM 数据地址

        MOV     A, #00H                 ; 设定 EEPROM 存储数据
```

```
        MOV [HL], A
        MOV     A, #00H                 ; 设定 EEPROM 存储数据
        MOV [HL+1], A

        POP AX                          ; 恢复 EEPROM 数据地址

        ; 读测试
        CALL !__eeprom_read             ; 调用 EEPROM 读函数
        BC      $LOOP                   ; 如果错误则跳转到 LOOP

        NOP                             ; 如果正常完成则执行 NOP

LOOP:
        BR      $LOOP                   ; 完成 EEPROM 处理

end
```

A.3 测试例子程序(定长单数据方法, C 语言)

```
/*****************************************************************
          78K0S/Kx1+ EEPROM 模拟例子主程序
*****************************************************************/

#define TRUE   0x01;
#define FALSE  0x00;

/*************************************
          函数说明
 *************************************/

// 汇编函数
extern bit _eeprom_write(unsigned char *);   // 写入到 EEPROM
extern bit _eeprom_read(unsigned char *);    // 从 EEPROM 读取

/*************************************
          EEPROM 数据结构体说明
 *************************************/
typedef struct eeprom_data{
        unsigned char  uc_eeprom_data[2];     // 数据
        unsigned char  uc_delimiter;          // 分隔符
} struct_eeprom_data;

/*****************************************************************
               主函数
*****************************************************************/
void  main(void)
{
        /*      变量说明 */
        struct_eeprom_data     s_eeprom_data;
        unsigned char          uc_eeprom_func_result;

        /*      使用 hwinit 函数来设置外围设定    */

        /*              初始化 EEPROM 数据       */
        s_eeprom_data.uc_eeprom_data[0] = 0x01;
        s_eeprom_data.uc_eeprom_data[1] = 0x10;
        s_eeprom_data.uc_delimiter = 0;

        /*      写入到 EEPROM           */
        if(_eeprom_write(&(s_eeprom_data.uc_eeprom_data[0])))
        {
                /*      写入失败 */
                uc_eeprom_func_result = FALSE;
        }
     else
```

```
        {
                /*      写入成功 */
                uc_eeprom_func_result = TRUE;
        }

        /*      读取 EEPROM             */
        if(_eeprom_read(&(s_eeprom_data.uc_eeprom_data[0])))
        {
                /*      读取失败 */
                uc_eeprom_func_result = FALSE;
        }
    else
        {
                /*      读取成功 */
                uc_eeprom_func_result = TRUE;
        }

        while(1);
}
```

<R>

B.1 主例子程序(EEPROM.asm) (定长多数据方法，汇编语言)

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  系统:78K/0S EEPROM 模拟程序 (定长数据方法)
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PUBLIC __eeprom_write
PUBLIC __eeprom_read
;-----------------------------------------------------------------------
;        选项字节, 保护字节
;-----------------------------------------------------------------------
OPTBYTE CSEG AT 0080H
        DB 10011100B
         ; ||||||||
         ; |||||||+-- INTOSC   (通过软件来停止允许)
         ; |||||++--- OSCSEL1/0        (高速内部振荡器)
         ; ||||+----- RMCE     (复位)
         ; |||+------ <1>
         ; |++------- DEFOSTS1/0 (最小值)
         ; +--------- <1>

        DB      11111111B       ; 保护字节: 位(6-2) = 11111B
                                ; 保护区域: 无
                                ; 非保护区域: Blocks 0 ～ 15
                                ; 确保要释放 block 的保护
                                ; 设定为 EEPROM 仿真
;-----------------------------------------------------------------------
;        用于 EEPROM 的 Block 编号       **用户设定**
;-----------------------------------------------------------------------
                                ; 例) 当使用 block 14 和 15
EEPROM_BLOCK            EQU     (14)    ; 设定的 block 必须是连续的。
EEPROM_BLOCK_NO         EQU     (2)     ; 使用的 block 编号

;-----------------------------------------------------------------------
;        要写入数据的数据长度            **用户设定**
;-----------------------------------------------------------------------
        DATA_NO_MAX  EQU (2) ;例) 当使用数据的两个单位 (数据编号 0 和 1)
        LENG  EQU (4)        ; 数据长度(包括数据编号和分隔符(总共 2 字节))

;-----------------------------------------------------------------------
;        其它 EQU 设定
;-----------------------------------------------------------------------
CMD_INTERNAL_VERIFY2    EQU     (00000010B)     ; 内部校验 2
CMD_BLOCK_ERASE         EQU     (00000011B)     ; Block 擦除
CMD_BLANK_CHECK         EQU     (00000100B)     ; Blank 检测
CMD_BYTE_WRITE          EQU     (00000101B)     ; 1 字节写入
```

```
NO_ERR                EQU     (00000000B)     ; 未发现错误
ERR_FPRERR            EQU     (00000001B)     ; PFCMD 写入错误
ERR_VCERR             EQU     (00000010B)     ; 擦除/写入/内部校验错误
ERR_WEPRERR           EQU     (00000100B)     ; 保护错误
FALSE                 EQU     (0FFH)          ; “FALSE” 结果
TRUE                  EQU     (00H)           ; “TRUE” 结果
DATATOP               EQU     (2)             ; 数据写入起始地址
LENGB                 EQU     LENG-1          ; 数据长度 – 1
ERASE_RETRY           EQU     (48)            ; 为擦除进行的重新计数(4.0 ～ 5.5 V时 100 次)
EEPROM_END_BLOCK      EQU     EEPROM_BLOCK+EEPROM_BLOCK_NO-1
;---------------------------------------------------------------------
;       RAM 设置
;---------------------------------------------------------------------
EP_RAM1         DSEG    saddr
d2MaskStatus:   DS      2       ; 用于存储掩模数据的 RAM
RWPointer:      DS      2       ; 用于存储结构体的起始地址指针
d2SetAdr:       DS      2       ; 在模块转换过程中用于存储数据传送目的地址的 RAM
ci:             DS      1       ; 通用目的循环计数器
CurentB_No:     DS      1       ; 当前使用 block 的数量
RQDATA_No:      DS      1       ; 保存请求数据编号的工作区域

EEPROM_PROG     CSEG
;---------------------------------------------------------------------
; 函数名称:         __eeprom_read (用户访问功能)
; 输入:             AX = 结构体指针
; 返回值:           CY = 0 (正常完成) 或者 CY = 1 (异常终止)
; 摘要:             将指定的数据从 EEPROM 中读取到存储地址。
;                   1. 搜索用做 EEPROM 的 block。
;                   2. 在有效的 block 中寻找最新的数据地址。
;                   3. 在搜索到的地址中读取最新的数据
;---------------------------------------------------------------------
;bit _eeprom_read(struct_eeprom_data *)

__eeprom_read:
          PUSH   HL
          PUSH   AX
          CALL   !getpara               ; 从结构体中提取参数
          MOV    X,A                    ; 预先保存数据编号到 X
          CALL   !EEPROMUseBlockSearch  ; 搜索当前正在用于 EEPROM 的 block
          BC     $EP_READ_FL2           ; 如果异常终止则跳转并结束
;
; 搜索 A: block 编号和 X: 数据编号
;
          PUSH   DE                     ; 保存工作寄存器
         CALL    !EEPROMDataSearch      ; 设定指定数据编号的最新数据的地址到 DE
          BC     $EP_READ_FL            ; 如果数据没有找到就退出

          MOV    ci,#LENG-2
```

```
;
;  复制搜索数据到结构体的数据区域
;
EP_READ_LP:
         INCW  DE
         MOV   A,[DE]                ; 读取 EEPROM 数据
         MOV   [HL],A                ; 存储数据
         INCW  HL                    ; 更新存储地址
         DBNZ  ci,$EP_READ_LP        ; 复制数据长度

EP_READ_FL:
         POP   DE
EP_READ_FL2:
         POP   AX
         POP   HL
         RET

;
;---------------------------------------------------------------------
; 函数名称:       getpara
; 输入:           AX = 结构体指针
; 返回值:         A = 数据编号。也复制到RQDATA_No。
;                 HL = 数据地址
; 摘要:           从自变量(指针)中读取用户调用的函数，结构体的内容和获得需要的变量
;---------------------------------------------------------------------

getpara:
         MOVW  HL,AX                 ; 设置结构体指针到 HL
         MOVW  RWPointer,AX          ; 复制到工作区域
         MOV   A,[HL]                ; 读取数据编号
         MOV   RQDATA_No,A           ; 复制到工作区域
         INCW  HL                    ; 设置数据地址到 HL
         RET

;---------------------------------------------------------------------
; 函数名称:       __eeprom_write (用户访问功能)
; 输入:           AX =结构体指针
; 返回值:         CY = 0(正常完成) 或者 CY = 1 (异常终止)
; 摘要:           把指定编号的数据从存储地址写入 EEPROM。
;                 1. 搜索用做 EEPROM 的 block。
;                 2. 如果没有有效地 block，则把第一个指定的 block 设定为有效。
;                 3. 搜索用于允许写入到有效 block 的地址。
;                 4. 如果有效的 block 已满或者不能被写入，操作将转移到下一个block。
;                 5. 创建写入数据。
;                 6. 写入到有效的 block。
;---------------------------------------------------------------------
;bit _eeprom_write(struct_eeprom_data *)
```

```
__eeprom_write:
        PUSH   HL
        PUSH   AX
        PUSH   DE
        PUSH   BC
        CALL   !getpara              ; 分解自变量中的变量
        CALL   !SelfFlashModeOn      ; 设定 flash 模式
        CALL   !EEPROMUseBlockSearch ; 搜索当前用作 EEPROM 的 block
        BNC    $EP_WRITE_4           ; 如果没有找到当前使用的 block 则跳转
;
;   由于没有找到当前使用的 block 开始使用新的 block
;
        CALL   !EEPROMBlockInit      ; 如果没有找到当前使用的 block，指定作为 EEPROM 的
                                     ; 最小编号的 block 为有效 block
        BC     $EP_WRITE_FL          ; 如果保护失败则跳转并且完成

EP_WRITE_2:
        MOVW   AX,#EEPROM_BLOCK*100H+DATATOP
        MOV    CurentB_No,A          ; 计算要使用的 block 编号

;
;   数据写入进程
;

EP_WRITE_3:
        MOVW   DE,AX                 ; 设定当前写入地址到 DE
        MOVW   AX,RWPointer          ; 读取结构体地址
        XCHW   AX,DE                 ; 设定结构体地址到 DE
        MOV    C,#LENG
        CALL   !FlashEEPROMWrite     ; 写入数据到 EEPROM
        BZ     $EP_WRITE_TR

EP_WRITE_FL:
        CALL   !SelfFlashModeOff     ; 释放 flash 模式
        SET1   CY                    ; 返回值 = 异常终止(FALSE)
        BR     $EP_WRITE_E

;
;   如果当前 block 有效
;
EP_WRITE_4:
        CALL   !EEPROMWriteTopSearch ; 搜索可写入区域
        BNC    $EP_WRITE_3           ; 因为区域没有找到而跳转到数据写入

;
;   因为区域里没有足够的空间而传送下一个 block。
;
```

```
        CALL  !EEPROMUseBlockChange ; 改变使用的 block
        BC    $EP_WRITE_FL          ; 异常终止退出
;
;   正常完成结束
;
EP_WRITE_TR:
        CALL  !SelfFlashModeOff     ; 释放 flash 模式
        CLR1  CY                    ; 返回值 = 正常完成(TRUE)

EP_WRITE_E:
        POP   BC
        POP   DE
        POP   AX
        POP   HL
        RET

;---------------------------------------------------------------------
; 函数名称:         EEPROMUseBlockSearch
; 输入:             无
; 输出:
;   正常完成:               CY = 0
;                           A, CurrentB_No = block 表编号(00H ～ FEH)
;   异常终止:               CY = 1
;                           A, CurrentB_No =下一个结束 block
; 使用的寄存器:     A
; 摘要:             搜索份配给 EEPROM 的 flash 存储器中当前正在使用的 block。
;---------------------------------------------------------------------
EEPROMUseBlockSearch:
        PUSH  HL
        MOV   ci,#EEPROM_BLOCK_NO   ; 循环的数量 = EEPROM 设定 block 的数量
        MOVW  HL,#EEPROM_BLOCK*100H ; 设定起始 block 编号

EP_BSRC_T:
        MOV   A,[HL+1]              ; 读取无效标志
        INC   A                     ; 如果无效标志 = FF 结果为00
        OR    A,[HL]                ; 如果有效标志 = 00 结果为00
        ADD   A,#0FFH               ; 如果结果非 00 设定进位标志
        BNC   $EP_BSRC_E            ; 如果 CY = 00 则 block 有效
        INC   H                     ; 如果无效则移动到下一个 block
        DBNZ  ci,$EP_BSRC_T         ; 循环到 block 的数量结束
EP_BSRC_E:
        MOV   A,H                   ; 设定 block 编号
        MOV   CurentB_No,A          ; 存储使用的 block 编号到 RAM
        POP   HL
        RET

;---------------------------------------------------------------------
; 函数名称:       EEPROMBlockInit
```

```
; 输入:           无
; 输出:
;   正常完成:     CY = 0
;   异常终止:     CY = 1
; 使用的寄存器:   A, X, B, C, D, E
; 摘要:           如果用于 EEPROM 的 block 没有有效的，则第一个被指定的 block 被
;                 设定为当前应用的(有效的)。
;                 安全正常返回 CY = 0
;                 安全异常返回 CY = 1
;---------------------------------------------------------------------
EEPROMBlockInit:
        PUSH HL
        MOV   A,#EEPROM_BLOCK       ; 设定 block 的最小编号
        CALL  !SelfFlashBlockErase  ; 擦除指定 block
        BC    $EP_BINI_FL           ; 异常终止
        CALL  !Setvalid             ; 设定 block 有效
        BZ    $EP_BINI_E            ; 如果正常完成则跳转
        SET1  CY                    ; 如果设定为有效则产生一个错误
EP_BINI_FL:
EP_BINI_E:
        POP   HL
        RET

;---------------------------------------------------------------------
; 函数名称:       EEPROMUseBlockChange
; 输入:           X =当前使用的 block 编号
; 输出:
;   正常完成:           CY = 0, 零标志 = 1, C = block 表编号 (02H ～ FEH)
;                       存储新的 block 编号到“CurrentB_No”.
;   异常终止:           CY = 1, 零标志= 0
; 使用的寄存器:   A, X, B, C, D, E
; 摘要:           如果当前使用的 block 已经写满了数据，该函数搜索下一个 block 用于写入数据。
;                 1. 设定下一个被使用的 block。
;                 2. 擦除下一个被使用的 block。
;                 3. 把最新的数据从有效的 block 传送到下一个 block。
;                 4. 把下一个 block 设定为有效。
;                 5. 把当前有效的 block 设定为无效。
;---------------------------------------------------------------------
EEPROMUseBlockChange:
        MOV   A,CurentB_No          ; 读取当前使用 block 的编号
        CMP   A,#EEPROM_END_BLOCK   ; 和最后 block 比较
        INC   A                     ; 如果找到剩余则传送到下一个 block。
        BC    $EP_BCHG_2            ; 如果没有找到剩余
        MOV   A,#EEPROM_BLOCK       ; 选择起始 block
;
;   擦除下一个被使用的 block 并且准备使用
;
```

```
EP_BCHG_2:
        CALL  !SelfFlashBlockErase   ; 擦除下一个被使用的 block
        BC    $EP_BCHG_E             ; 如果不能擦除 block 则退出

EP_BCHG_3:
        MOV   X,#DATATOP             ; 设定下一个写入起始地址的 block
        MOVW  d2SetAdr,AX            ; 保存地址到参数区域

;
;   复制处理: 用于直接从旧 block 写入到新 block 的处理
;

        MOV   ci,#DATA_NO_MAX        ; 设定重复数
EP_BCHG_T:
;
; 搜索旧 block 的数据
;
        MOV   A,#DATA_NO_MAX
        SUB   A,ci                   ; 计算目标数据编号
        MOV   X,A                    ; 设定数据编号
        CMP   A,RQDATA_No            ; 数据编和所需要的相同?
        BNZ   $EP_BCHG_5             ; 如果和写入数据相同,
        MOVW  AX,RWPointer           ; 读取结构体指针
        MOVW  DE,AX                  ; 并且设定数据编号到 DE 寄存器
        BR    $EP_BCHG_6

EP_BCHG_5:                           ; 否则, 搜索数据
        MOV   A,CurentB_No           ; 设定 block 编号
        CALL  !EEPROMDataSearch      ; 搜索最新数据编号的地址 并且设定初始数据的地址到 DE
        BC    $EP_BCHG_8             ; 如果数据没有找到则跳转

;
; 复制数据到新的 block
;
EP_BCHG_6:
        MOV   C,#LENG                ; 循环数据长度
        MOVW  AX,d2SetAdr            ; 设定数据要写入的地址
        CALL  !FlashEEPROMWrite      ; 写入数据到新的 EEPROM 区域
        SET1  CY                     ; 预先设定错误标志
        BNZ   $EP_BCHG_E             ; 如果异常终止则退出

;
; 如果完成写入则更新到新的写入地址
;
EP_BCHG_7:
        MOV   A,d2SetAdr             ; 设定写入地址 + 数据长度
        ADD   A,#LENG                ; 作为新的写入地址
        MOV   d2SetAdr,A             ; 来写入下一数据
```

```
EP_BCHG_8:
        DBNZ   ci,$EP_BCHG_T          ; 重复到最大数据编号

;
;   因为已经完成数据传送设定新 block 有效
;

EP_BCHG_9:
        MOV    A,d2SetAdr+1           ; 读取 block 编号
        CALL   !Setvalid              ; 写入有效标志
        SET1   CY                     ; 设定错误标志
        BNZ    $EP_BCHG_E             ; 如果异常终止则退出

;
;   设定旧 block 无效
;

        MOV    A,CurentB_No           ; 设定旧 block 编号
        MOV    X,#1                   ; 设定无效标志地址
        CALL   !Setinvalid            ; 设定无效标志
        ADD    A,#0FFH                ; 如果失败则设置 CY

EP_BCHG_E:
        RET

;---------------------------------------------------------------------
; 函数名称:       EEPROMWriteTopSearch
; 输入:           A = 搜索 block 表编号
; 输出:
;       搜索成功:       CY = 0, 设定写入地址到 AX
;       搜索失败:       CY = 1
; 使用的寄存器:  A,X
; 摘要:           搜索可以在指定 block 写入的区
;                 仅当数据编号区域在 0ffH 适合时正常完成
;---------------------------------------------------------------------
EEPROMWriteTopSearch:
        PUSH   HL
        MOV    H,A
        MOV    L,#DATATOP

EP_WTSR_T:
        MOV    A,[HL]                 ; 读取数据
        CMP    A,#0FFH                ; 分隔符处于不能写入状态 (FFH)?
        BZ     $EP_WTSR_1             ; 搜索完成退出
        MOV    A,L                    ; 更新地址
        ADD    A,#LENG
        MOV    L,A
```

```
        BC      $EP_WTSR_1              ; 如果 block 完成则错误退出
        ADD     A,#LENG-1               ; 如果剩下的至少 LENG 则继续
        BNC     $EP_WTSR_T              ; 否则设置 CY

EP_WTSR_1:

        MOVW    AX,HL                   ; 返回值 = 搜索指针的地址
        POP     HL
        RET

;---------------------------------------------------------------------
; 函数名称:       EEPROMDataSearch
; 输入:           A = 前使用的 block 表编号
;                 X = 搜索表编号
;                 (通过用户定义设置用于 EEPROM 的block 编号)
; 输出:
;    正常完成:     CY = 0, DE =最新数据的地址
;    异常终止:     CY = 1, E = 0
; 使用的寄存器:    A, D, E
; 摘要:           读取最新数据的存储地址。
;---------------------------------------------------------------------
EEPROMDataSearch:
        PUSH    HL
        MOV     E,#0                    ; 设定结果的初始值为错误
        MOV     H,A                     ; 设定 block 的高地址
        MOV     L,#DATATOP              ; 设定低地址
;
;    比较数据编号。并且如果有相等的数据编号，检查它的分隔符
;
EP_DASR_T:
        MOV     A,[HL]                  ; 读取数据编号
        CMP     A,X                     ; 和搜索编号比较
        BNZ     $EP_DASR_2              ; 如果搜索编号不相等则跳转
        MOV     A,[HL+LENG-1]           ; 读取分隔符
        CMP     A,#00H
        BNZ     $EP_DASR_1              ; 用于短程序
        PUSH    HL                      ; 如果分隔符有效,
        POP     DE                      ; 复制地址到 DE
EP_DASR_1:
        MOV     A,[HL]
;
; 无论是否可以找到空 block，搜索新地址直到 block 结束
;
EP_DASR_2:
        CMP     A,#0FFH                 ; 辨别是否数据结束
        BZ      $EP_DASR_E              ; 如果数据结束则退出
        MOV     A,L                     ; 更新地址
        ADD     A,#LENG
```

```
        MOV    L,A
        BNC    $EP_DASR_T              ; 在 block 中修复
;
;  Block search is completed
;
EP_DASR_E:
        MOV    A,E                     ; 提取数据的低地址
        CMP    A,#DATATOP              ; 如果错误则设定 CY
        POP    HL
        RET

;---------------------------------------------------------------------
; 函数名称:         SelfFlashBlockErase
; 输入:             A =擦除 block 编号
; 输出:             CY = 0 (正常完成) 或者 CY = 1 (异常终止)
; 使用的寄存器:     B
; 摘要:             擦除指定 block 并且执行空白检测。
;---------------------------------------------------------------------
SelfFlashBlockErase:
        PUSH   AX
        MOV    X,A                     ; 保存 block 编号
        MOV    B,#ERASE_RETRY          ; 设定擦除时的重试次数
        BR     $SF_BKER_ST             ; 从空白检测开始

SF_BKER_RE:
        MOV    A,X                     ; 设定擦除 block
        CALL   !FlashBlockErase        ; 擦除处理
        BNZ    $SF_BKER_01             ; 异常终止
        MOV    A,X                     ; 设定空白检测 block

SF_BKER_ST:
        CALL   !FlashBlockBlankCheck   ; 空白检测进程
        BZ     $SF_BKER_TR             ; 空白检测正常完成
SF_BKER_01:
        DBNZ   B,$SF_BKER_RE           ; 检测重试计数
SF_BKER_FL:
        SET1   CY                      ; 返回值 = 异常终止

SF_BKER_TR:
        POP    AX
        RET

;---------------------------------------------------------------------
; 函数名称:         SelfFlashModeOff
; 输入:             无
; 输出:             无
; 使用的寄存器:     A, X
; 摘要:             释放自编程模式处理
```

```
;---------------------------------------------------------------------
SelfFlashModeOff:
         MOV    FLCMD,#0
         MOV    PFS,#NO_ERR
         MOV    PFCMD,#0A5H                  ; PFCMD 寄存器控制
         MOV    FLPMC,#00H                   ; FLPMC 寄存器控制 (设定值)
         MOV    FLPMC,#0FFH                  ; FLPMC 寄存器控制 (反向设定值)
         MOV    FLPMC,#00H                   ; 设定正常模式: FLPMC 寄存器控制 (设定值)
         BT     PFS.0,$SelfFlashModeOff      ; 检测写入到指定寄存器是否完成
         EI
         MOVW   AX,d2MaskStatus
         MOV    MK1,A                        ; 使用 78K0S/KY1+ 时删除这行
         XCH    A,X
         MOV    MK0,A                        ; 恢复中断屏蔽标志
         RET

;---------------------------------------------------------------------
; 函数名称:         SelfFlashModeOn
; 输入:             无
; 输出:             无
; 使用的寄存器:     A, X
; 摘要:             设定自编程模式的处理e
;---------------------------------------------------------------------
SelfFlashModeOn:
         MOV    A,MK0
         XCH    A,X
         MOV    A,MK1                 ; 使用 78K0S/KY1+ 时删除这行
         MOVW   d2MaskStatus,AX       ; 保存中断屏蔽标志
         MOV    MK0,#11111111B
         MOV    MK1,#11111110B        ; 屏蔽除 INTFLC 之外的终端
                                      ; 使用 78K0S/KY1+ 时删除这行
         DI
SF_MDON_LP:
         MOV    PFS,#NO_ERR
         MOV    PFCMD,#0A5H           ; PFCMD 寄存器控制
         MOV    FLPMC,#01H            ; FLPMC 寄存器控制 (设定值)
         MOV    FLPMC,#0FEH           ; FLPMC 寄存器控制 (反向设定值)
        MOV     FLPMC,#01H            ; 设定自变成模式: FLPMC 寄存器控制 (设定值)

         NOP
         HALT
         BT     PFS.0,$SF_MDON_LP     ; 检测写入到指定寄存器是否完成
         RET

;---------------------------------------------------------------------
; 函数名称:         FlashBlockErase
; 输入:             A = block 编号
; 输出:             Z (零标志) = 正常完成 (1) 或者异常终止 (0)
```

```
; 使用的寄存器:     A
; 摘要:             擦除指定的 block。
;--------------------------------------------------------------------
FlashBlockErase:
        MOV    FLCMD,#CMD_BLOCK_ERASE ; 设定 flash 控制命令 (block 擦除)
        MOV    FLAPH,A                ; 设定 blank 检测 block 编号
        MOV    FLAPL,#00H
        MOV    FLAPHC,A
        MOV    FLAPLC,#00H
;
;   继续执行目前的 flash 自编程
;
;--------------------------------------------------------------------
; 函数名称:         SubFlashSelfPrg
; 输入:             A = Block 编号
; 输出:             Z (零标志) = 正常完成 (1) 或者异常终止 (0)
; 使用的寄存器:     A
; 摘要:             调用 flash 自编程函数。
;--------------------------------------------------------------------
SubFlashSelfPrg:
        MOV    PFS,#NO_ERR     ; 清除 flash 状态寄存器
        MOV    WDTE,#0ACH      ; 清楚和重启 WDT
        HALT                   ; 开始自编程
        MOV    A,PFS
        CMP    A,#NO_ERR
        RET
;--------------------------------------------------------------------
; 函数名称:         FlashBlockBlankCheck
; 输入:             A = Block 编号
; 输出:             Z(零标志) = 正常完成 (1) 或者异常终止 (0)
; 使用的寄存器:     A
; 摘要:             执行指定 block 的空白检测。
;--------------------------------------------------------------------
FlashBlockBlankCheck:
        MOV    FLCMD,#CMD_BLANK_CHECK ; 设定 flash 控制命令 (block 空白检测)
        MOV    FLAPH,A                ; 设定空白检测 block 编号
        MOV    FLAPL,#00H
        MOV    FLAPHC,A
        MOV    FLAPLC,#0FFH
        BR     $SubFlashSelfPrg

;--------------------------------------------------------------------
; 函数名称:         Setvalid
; 输入:             A = 目标 block 编号
; 输出:             Z (零标志) =正常完成(1) or 或者异常终止 (0)
; 使用的寄存器:     A, X, C, D, E
; 摘要:             设定使用的 block 有效。
;--------------------------------------------------------------------
```

```
Setvalid:
        MOV   X,#0                  ; AX 指示有效标志的地址
Setinvalid:
        MOVW  DE,#Setinvalid-1      ; 指示 0 作为要写入的数据
        MOV   C,#1                  ; 写入数据是 1 字节
;
;   下面是写入函数处理
;

;---------------------------------------------------------------------
; 函数名称:         FlashEEPROMWrite
; 输入:             DE = 写入起始地址
;                   C = 要写入的数据 block 的数量
;                   AX = 写入数据存储地址
; 输出:             Z(零标志) =正常完成(1) 或者异常终止 (0)
; 使用的寄存器:     D, E
; 摘要:             写入数据到 EEPROM 并且内部校验数据。
;---------------------------------------------------------------------
FlashEEPROMWrite:
        PUSH  HL
        PUSH  AX
        PUSH  BC
        MOVW  HL,AX
        MOV   FLCMD,#CMD_BYTE_WRITE         ; 设定 flash 控制命令 (字节写入)
FL_WR_W1:
        MOVW  AX,HL
        MOV   FLAPH,A
        XCH   A,X
        MOV   FLAPL,A                       ; 设定写入/校验地址
        MOV   A,[DE]
        MOV   FLW,A                         ; 设定写入数据
        CALL  !SubFlashSelfPrg
        BZ    $FL_WR_W2                     ; 如果正常完成则退出
        POP   BC
        BR    $FL_WR_E
FL_WR_W2:
        INCW  DE                            ; 读地址 + 1
        INCW  HL                            ; 写地址 + 1
        DBNZ  C,$FL_WR_W1                   ; 循环指定字节的数量

        POP   BC                            ; 写入完成
        MOV   FLCMD,#CMD_INTERNAL_VERIFY2   ; 设定 flash 控制命令 (内部校验)
        MOV   FLAPH,A
        XCH   A,X
        MOV   FLAPL,A                       ; 设定校验开始地址
        DEC   C
        ADD   A,C
        XCH   A,X
```

```
        ADDC   A,#0
        MOV    FLAPHC,A
        XCH    A,X
        MOV    FLAPLC,A                    ; 设定校验结束地址
        CALL   !SubFlashSelfPrg
FL_WR_E:
        POP    AX
        POP    HL
        RET

      END
```

B.2 测试例子程序(定长多数据方法，汇编语言)

```
;*******************************************************************************
;
;       78K0S/Kx1+ EEPROM 模拟例子主程序
;
;*******************************************************************************

;===============================================================================
;
;       EEPROM 外部引用声明
;
;===============================================================================
EXTRN __eeprom_read    ; EEPROM 读函数
EXTRN __eeprom_write   ; EEPROM 写函数

;===============================================================================
;
;       RAM 定义
;
;===============================================================================
XRAM    DSEG    SADDR
EEPROM_NO:      DS      1       ; 数据编号
EEPROM_DATA:    DS      2       ; EEPROM 数据
DELIMITER:      DS      1       ; 分隔符

;===============================================================================
;
;       可靠的堆栈区域
;
;===============================================================================
XSTK    DSEG    AT      0FED0H
STACKEND:
        DS      30H             ; 保护堆栈区域的 32 字节
STACKTOP:                       ; 堆栈区域起始地址 = FF00H

;===============================================================================
;
;       向量表设定
;
;===============================================================================
XVCT    CSEG    AT      0000H
        DW      RESET_START     ;(00)   复位

;*******************************************************************************
;
;       复位释放后的初始化处理
;
```

```
;*******************************************************************************
XMAIN   CSEG    UNIT
RESET_START:

;-------------------------------------------------------------------------------
;       堆栈指针设定
;-------------------------------------------------------------------------------
        MOVW    AX,     #STACKTOP
        MOVW    SP,     AX              ; 设定堆栈指针

;-------------------------------------------------------------------------------
;       看门狗定时器设定
;-------------------------------------------------------------------------------
        MOV     WDTM,   #01110111B      ; 停止看门狗定时器操作

;-------------------------------------------------------------------------------
;       时钟频率设定
;-------------------------------------------------------------------------------
        MOV     PPCC,   #00000000B      ; 外围硬件提供时钟 fxp = fx (= 8MHz)
        MOV     PCC,    #00000000B      ; CPU 时钟 fcpu = fxp (= 8MHz)
        MOV     LSRCM,  #00000001B      ; 停止低速内部振荡器

;*******************************************************************************
;
;       主处理
;
;*******************************************************************************
        MOV     DELIMITER,      #00H            ; 初始化分隔符

        MOVW AX,                #EEPROM_NO      ; 传送 EEPROM 数据地址到 HL
        MOVW HL, AX

        ; 设定写入数据到 EEPROM 数据 0
        PUSH AX                                 ; 保存 EEPROM 数据地址

        MOV     A, #00H                         ; 设定 EEPROM 数据 0 的数据编号
        MOV [HL], A

        MOV     A, #01H                         ; 设定 EEPROM 数据 0 的存储数据
        MOV [HL+1], A
        MOV     A, #10H                         ; 设定 EEPROM 数据 0 的存储数据
        MOV [HL+2], A

        POP AX                                  ; 恢复 EEPROM 数据地址

        ; Tests writing
        CALL !__eeprom_write                    ; 调用 EEPROM 数据 0 写函数
        BC      $LOOP                           ; 如果错误则跳转到 LOOP
```

```
        ; Sets EEPROM data 1 write data
        PUSH AX                             ; 保存 EEPROM 数据地址

        MOV     A, #01H                     ; 设定 EEPROM 数据 1 的数据编号
        MOV [HL], A

        MOV     A, #80H                     ; 设定 EEPROM 数据 1 的存储数据
        MOV [HL+1], A
        MOV     A, #5AH                     ; 设定 EEPROM 数据 1 的存储数据
        MOV [HL+2], A

        POP AX                              ; 恢复 EEPROM 数据地址

        ; Tests writing
        CALL !__eeprom_write                ; 调用 EEPROM 数据 1 写函数
        BC      $LOOP                       ; 如果错误则跳转到 LOOP

        ; Tests reading
        PUSH AX                             ; 保存 EEPROM 数据地址

        MOV     A, #00H                     ; 设定 EEPROM 数据 0 的数据编号
        MOV [HL], A

        POP AX                              ; 恢复 EEPROM 数据地址

        CALL !__eeprom_read                 ; 调用 EEPROM 读函数
        BC      $LOOP                       ; 如果错误则跳转到 LOOP

        PUSH AX                             ; 保存 EEPROM 数据地址

        MOV     A, #01H                     ; 设定 EEPROM 数据 1 的数据编号
        MOV [HL], A

        POP AX                              ; 恢复 EEPROM 数据地址

        CALL !__eeprom_read                 ; 调用 EEPROM 读函数
        BC      $LOOP                       ; 如果错误则跳转到 LOOP

        NOP                                 ; 如果正常完成则执行 NOP

LOOP:
        BR      $LOOP                       ; 完成 EEPROM 处理


end
```

B.3 测试例子程序(定长多数据方法, C 语言)

```
/*****************************************************************
              78K0S/Kx1+ EEPROM 模拟例子主程序
*****************************************************************/

#define TRUE   0x01;
#define FALSE  0x00;

/*************************************
        函数说明
 *************************************/
// 汇编函数
extern bit _eeprom_write(unsigned char *);   // 写入到 EEPROM
extern bit _eeprom_read(unsigned char *);    // 从 EEPROM 读取

/*************************************
         EEPROM 数据结构体说明
 *************************************/
typedef struct eeprom_data{
        unsigned char  uc_data_no;           // 数据编号
        unsigned char  uc_eeprom_data[2];    // 数据
        unsigned char  uc_delimiter;         // 分隔符
} struct_eeprom_data;

/*****************************************************************
              主函数
*****************************************************************/
void  main(void)
{

        /*      变量说明 */
        struct_eeprom_data     s_eeprom_data;
        unsigned char          uc_eeprom_func_result;

        /*      使用 hwinit 函数来设置外围设定    */

        /*              初始化 EEPROM 数据  0    */
        s_eeprom_data.uc_data_no = 0;
        s_eeprom_data.uc_eeprom_data[0] = 0x01;
        s_eeprom_data.uc_eeprom_data[1] = 0x10;
        s_eeprom_data.uc_delimiter = 0;

        /*      写入到 EEPROM 数据 0            */
        if(_eeprom_write(&(s_eeprom_data.uc_data_no)))
        {
                /*      写入失败 */
                uc_eeprom_func_result = FALSE;
```

```
}
else
{
        /*      写入成功 */
        uc_eeprom_func_result = TRUE;
}

/*      初始化 EEPROM 数据 1    */
s_eeprom_data.uc_data_no = 1;
s_eeprom_data.uc_eeprom_data[0] = 0x80;
s_eeprom_data.uc_eeprom_data[1] = 0x5A;
s_eeprom_data.uc_delimiter = 0;

/*      写入到 EEPROM 数据 1     */
if(_eeprom_write(&(s_eeprom_data.uc_data_no)))
{
        /*      写入失败 */
        uc_eeprom_func_result = FALSE;
}
else
{
        /*      写入成功 */
        uc_eeprom_func_result = TRUE;
}

/*              指定读取 EEPROM 数据 0          */
s_eeprom_data.uc_data_no = 0;

/*      读取 EEPROM 数据 0      */
if(_eeprom_read(&(s_eeprom_data.uc_data_no)))
{
        /*      读取失败 */
        uc_eeprom_func_result = FALSE;
}
else
{
        /*      读取成功 */
        uc_eeprom_func_result = TRUE;
}

/*      指定读取 EEPROM 数据 1   */
s_eeprom_data.uc_data_no = 1;

/*      读取 EEPROM 数据 1      */
if(_eeprom_read(&(s_eeprom_data.uc_data_no)))
{
        /*      读取失败 */
        uc_eeprom_func_result = FALSE;
```

```
        }
        else
        {
                /*      读取成功 */
                uc_eeprom_func_result = TRUE;
        }

        while(1);
}
```

<R>

附录 C 修订历史

C.1 本版本的主要修订

页数	描述
第二章 EEPROM 模拟功能 (定长单数据方法)	
p. 9	整章修改
第三章 EEPROM 模拟程序 (定长单数据方法，汇编语言)	
p. 18	整章修改
第四章 EEPROM 模拟功能(定长多数据方法)	
p. 37	增加该章
第五章 EEPROM 模拟功能(定长多数据方法，汇编语言)	
p. 46	增加该章
附录 A 例子程序列表 (定长单数据方法)	
p. 66	整章修改
附录 B 例子程序列表 (定长多数据方法)	
p. 84	增加该章
附录 C 修订历史	
p. 104	增加该章

详细信息请联系：

中国区

网址：
http://www.cn.necel.com/
http://www.necel.com/

[北京]

日电电子（中国）有限公司
中国北京市海淀区知春路 27 号
量子芯座 7，8，9，15 层
电话：（+86）10-8235-1155
传真：（+86）10-8235-7679

[上海]

日电电子（中国）有限公司上海分公司
中国上海市浦东新区银城中路 200 号
中银大厦 2409-2412 和 2509-2510 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

上海恩益禧电子国际贸易有限公司
中国上海市浦东新区银城中路 200 号
中银大厦 2511-2512 室
电话：（+86）21-5888-5400
传真：（+86）21-5888-5230

[深圳]

日电电子（中国）有限公司深圳分公司
深圳市福田区益田路卓越时代广场大厦 39 楼
3901，3902，3909 室
电话：（+86）755-8282-9800
传真：（+86）755-8282-9899

[香港]

香港日电电子有限公司
香港九龙旺角太子道西 193 号新世纪广场
第 2 座 16 楼 1601-1613 室
电话：（+852）2886-9318
传真：（+852）2886-9022
2886-9044