

# **F<sup>2</sup>MC-8FX 家族**

## 8 位微控制器

# **MB95200H/210H 系列**

---

## 闪存操作

应用笔记

## 变更履历

日期	作者	修正记录
2008-03-20	Jacky Zhou	V1.0, 第一版

本手册共 33 页。

1. 本档记载的产品信息及规格说明如有变动，恕不预先通知。如需最新产品信息和/或规格说明，联系富士通销售代表或富士通授权经销商。
2. 基于本档记载信息或示意图的使用引起的对著作权、工业产权或第三方的其他权利的侵害，富士通不承担任何责任。
3. 未经富士通明文批准，不得对本档的记载内容进行转让、拷贝。
4. 本档所介绍的产品并不旨在以下用途：需要极高可靠性的设备，诸如航空航天装置、海底中继器、核控制系统或维系生命的医用设施。
5. 本档介绍的部分产品可能是“外汇及外贸管理法”规定的战略物资(或专门技术)，出口该产品或其中部分元件前，应根据该法获得正式批准。

版权© 2008 富士通微电子(上海)有限公司 版权所有

# 目录

变更履历 .....	ERROR! BOOKMARK NOT DEFINED.
目录 .....	3
1 概要 .....	ERROR! BOOKMARK NOT DEFINED.
2 编程演算 .....	ERROR! BOOKMARK NOT DEFINED.
2.1 概要 .....	<b>Error! Bookmark not defined.</b>
2.2 闪存的特征 .....	<b>Error! Bookmark not defined.</b>
2.3 扇区配置 .....	<b>Error! Bookmark not defined.</b>
2.4 闪存的寄存器 .....	<b>Error! Bookmark not defined.</b>
2.5 启动闪存自动演算 .....	<b>Error! Bookmark not defined.</b>
3 源代码 .....	ERROR! BOOKMARK NOT DEFINED.
3.1 如何编译 RAM 代码 .....	<b>Error! Bookmark not defined.</b>
3.1.1 概要 .....	<b>Error! Bookmark not defined.</b>
3.1.2 在 RAMCODE 扇区定义代码 .....	<b>Error! Bookmark not defined.</b>
3.1.3 RAM 和 ROM 的连接代码 .....	<b>Error! Bookmark not defined.</b>
3.1.4 将代码从 ROM 复制到 RAM .....	<b>Error! Bookmark not defined.</b>
3.1.5 跳转到 RAM 执行 RAMCODE .....	<b>Error! Bookmark not defined.</b>
3.2 Flash 操作程序 .....	<b>Error! Bookmark not defined.</b>
3.2.1 闪存擦除程序的流程图 .....	<b>Error! Bookmark not defined.</b>
3.2.2 闪存写入程序的流程图 .....	<b>Error! Bookmark not defined.</b>
3.2.3 Flash.asm .....	<b>Error! Bookmark not defined.</b>
3.3 子程序 .....	<b>Error! Bookmark not defined.</b>
3.3.1 闪存擦除 C 代码 .....	<b>Error! Bookmark not defined.</b>
3.3.2 闪存写入 C 代码 .....	<b>Error! Bookmark not defined.</b>
3.4 如何使用编程功能 .....	<b>Error! Bookmark not defined.</b>
4 闪存操作的使用注意事项 .....	ERROR! BOOKMARK NOT DEFINED.
4.1 擦除后重写 NVR .....	<b>Error! Bookmark not defined.</b>
4.2 在 RST 引脚施加高压 .....	<b>Error! Bookmark not defined.</b>
5 附加信息 .....	ERROR! BOOKMARK NOT DEFINED.
6 附录 .....	ERROR! BOOKMARK NOT DEFINED.
6.1 图一览 .....	<b>Error! Bookmark not defined.</b>

6.2	样本代码.....	<b>Error! Bookmark not defined.</b>
6.2.1	工程.....	<b>Error! Bookmark not defined.</b>

## 1 概要

本应用笔记介绍 MB95200H/210H 系列的闪存操作。

本应用笔记介绍闪存操作的编程演算并用完整工程举例说明。工程包括 `flash.asm`、闪存擦除 C 代码、闪存写入 C 代码和 `main.c`。

## 2 编程演算

### 闪存操作的编程演算

#### 2.1 概要

若用户程序在闪存上运行，就不可同时擦/写闪存。用户只可将闪存程序保存到 RAM 区，这样 CPU 可在该存储区执行指令，同时也可擦/写闪存。

#### 2.2 闪存的特征

- 扇区配置: 4KB x 8 位/ 8KB x 8 位/ 16KB x 8 位
- 自动编程演算(嵌入式演算)
- 通过数据轮询或跳转位功能检测编程/擦除完成
- 使用 CPU 中断检测编程/擦除完成
- 支持 JEDEC 标准命令
- 擦/写周期(最少): 100,000 次

#### 2.3 扇区配置

图 2-1 显示的是 32/64/128-KBIT 闪存的配置。图中具体给出了每个扇区的高位和低位地址。

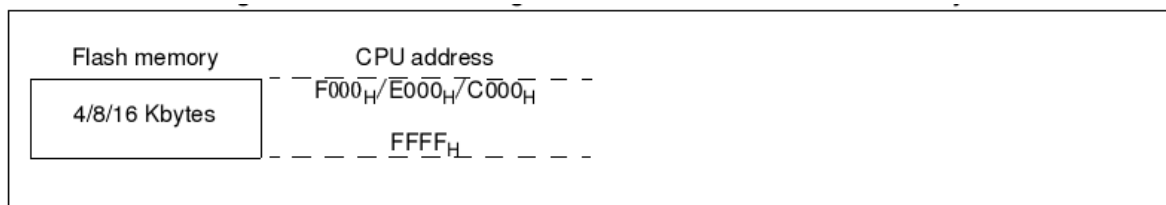


图2-1: 32/64/128-KBIT 闪存的扇区配置

## 2.4 闪存的寄存器

图 2-2 是闪存的寄存器。详情参考 MB95200H/210H 系列硬件手册的第 20 章。

Flash memory status register (FSR)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0072 <sub>H</sub>	-	-	RDYIRQ	RDY	Reserved	IRQEN	WRE	Reserved	000X0000 <sub>B</sub>
	R0/WX	R0/WX	R(RM1),W	R/WX	R/W0	R/W	R/W	R/W0	

- R/W: Readable/writable (Read value is the same as write value)
- R(RM1), W: Readable/writable (Read value is different from write value, "1" is read by read-modify-write instruction)
- R/WX: Read only (Readable, writing has no effect on operation)
- R/W0: Reserved bit (Write value is "0", read value is the same as write value)
- R0/WX: Undefined bit (Read value is "0", writing has no effect on operation)
- X: Indeterminate

图2-2: 闪存的寄存器

## 2.5 闪存自动演算的启动

可通过三种命令启动闪存自动演算：读/复位、写(编程)和整片擦除。图 2-3 是用于写/擦闪存的命令一览。

Command sequence	Bus write cycle	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Read/reset*	1	F <sub>X</sub> XX <sub>H</sub>	F0 <sub>H</sub>	-	-	-	-	-	-	-	-	-	-
	4	UAAA <sub>H</sub>	AA <sub>H</sub>	U554 <sub>H</sub>	55 <sub>H</sub>	UAAA <sub>H</sub>	F0 <sub>H</sub>	RA	RD	-	-	-	-
Write	4	UAAA <sub>H</sub>	AA <sub>H</sub>	U554 <sub>H</sub>	55 <sub>H</sub>	UAAA <sub>H</sub>	A0 <sub>H</sub>	PA	PD	-	-	-	-
Chip erase	6	XAAA <sub>H</sub>	AA <sub>H</sub>	X554 <sub>H</sub>	55 <sub>H</sub>	XAAA <sub>H</sub>	80 <sub>H</sub>	XAAA <sub>H</sub>	AA <sub>H</sub>	X554 <sub>H</sub>	55 <sub>H</sub>	XAAA <sub>H</sub>	10 <sub>H</sub>

- RA : Read address
  - PA : Write (program) address
  - RD : Read data
  - PD : Write (program) data
  - U : Upper 4 bits same as RA, PA and SA
  - F<sub>X</sub> : FF/FE
  - X : Arbitrary address
- \*: Both commands can reset flash memory to read mode.

图2-3: 命令顺序

## 3 源代码

### 闪存操作的源代码

#### 3.1 如何编译 RAM 代码

##### 3.1.1 概要

该示例工程演示闪存操作。汇编程序 `flash.asm` 在 RAM 执行，运行时永久地位于 RAM。汇编程序 `flash.asm` 保存在闪存，但其地址需编译并连接到 RAM，这样将其从闪存复制到 RAM 时才可正确运行。要使 RAMCODE 永久运行，需采取以下四个步骤。

1. 在 RAMCODE 扇区定义代码
2. RAM 和 ROM 的连接代码
3. 在启动文件中将代码从 ROM 复制到 RAM
4. 跳转到 RAM 执行 RAMCODE

以下就四个步骤逐一详细说明。

##### 3.1.2 在 RAMCODE 扇区定义代码

RAMCODE 仅仅是扇区名，用户也可自己命名，但在 Softune Workbench 工程环境中设定扇区时需使用统一的名称。3.1.3. 中进一步说明。

可使用以下声明在 RAMCODE 扇区和汇编语言模块中定义代码：

```
.SECTION RAMCODE, CODE
```

该声明将 `flash.asm` 的全部汇编代码都定位到 RAMCODE。仅 RAMCODE 功能可另外使用别的模块(带 `'asm'` 扩展名的文档)。

### 3.1.3 RAM 和 ROM 的连接代码

Softune Workbench 中的连接器为 RAM 和 ROM 的代码提供了专门连接机制。RAMCODE section 中程序的各指令地址是 RAM 地址但所属代码数据保存到 ROM。

需要采用以下 4 个步骤在工程中设置与 Workbench 相同的设置。

步骤 1: 在 Project 选择 Setup Project。

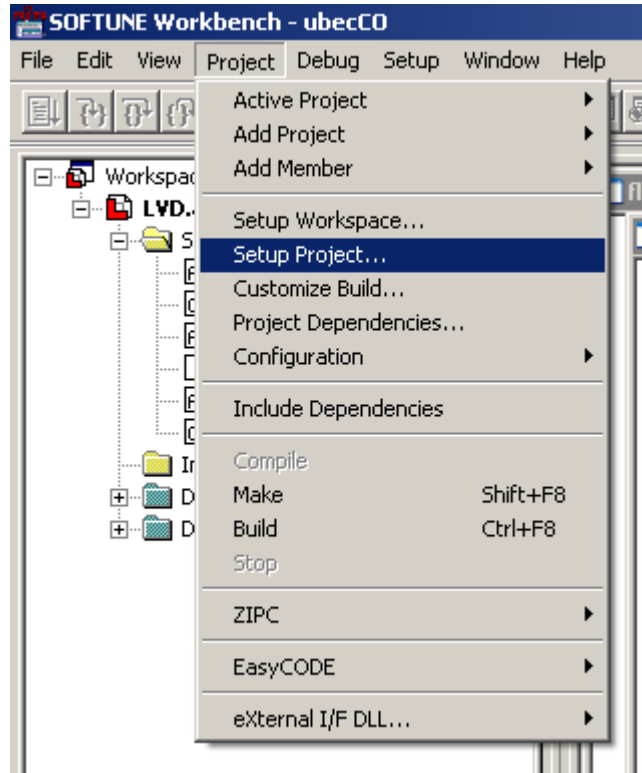


图3-1: 在 Project 选择 Setup Project

步骤 2: 在 Setup Project 选择 Disposition/Connection

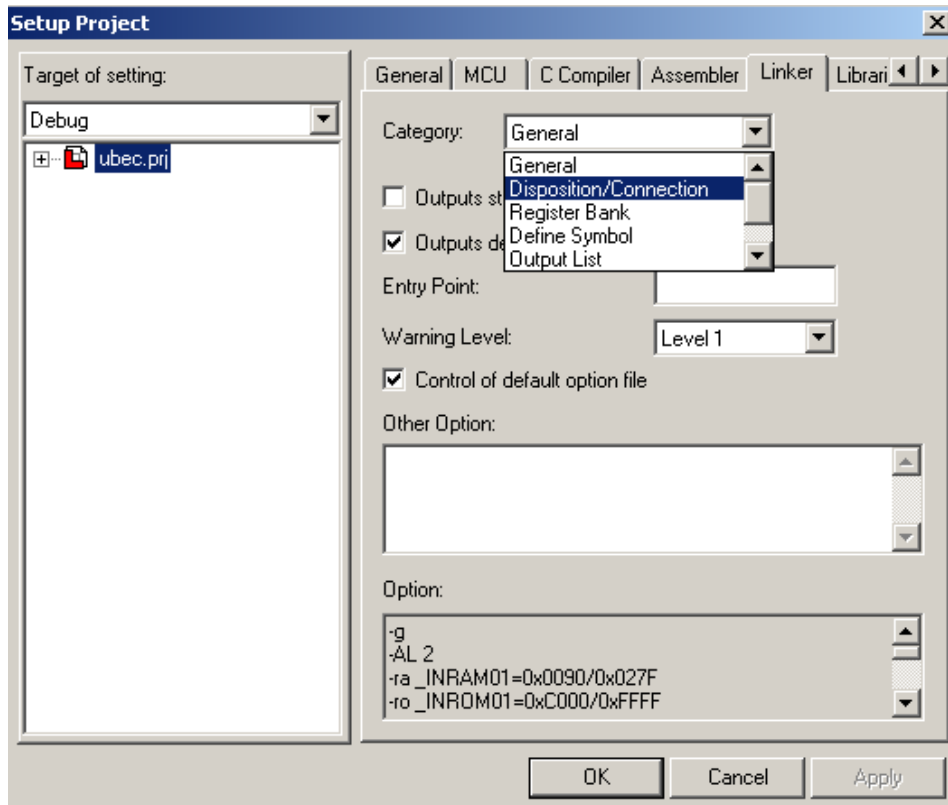


图3-2: 在 Setup Project 选择 Disposition/Connection

步骤 3: 在 Disposition/Connection 选择 Section

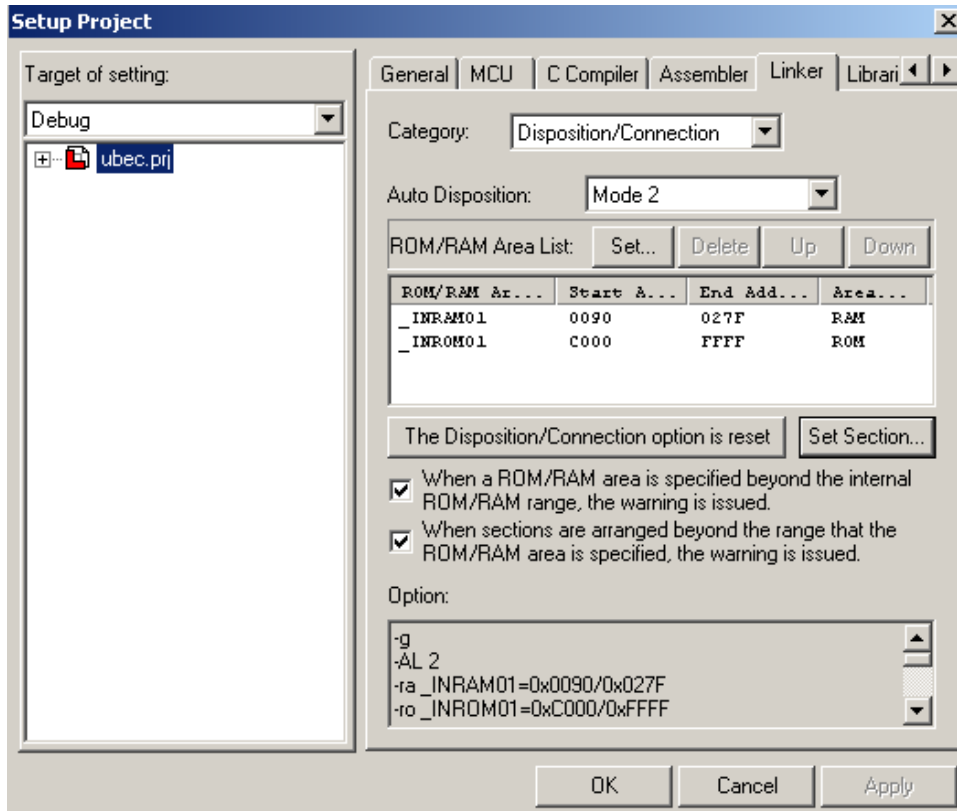


图3-3: 在 Disposition/Connection 选择 Section

步骤 4: 在 Setup Section 中设置 RAM

设置 RAMCODE/Code in \_INRAM01 后，连接器了解到一个 code section 位于 RAM。

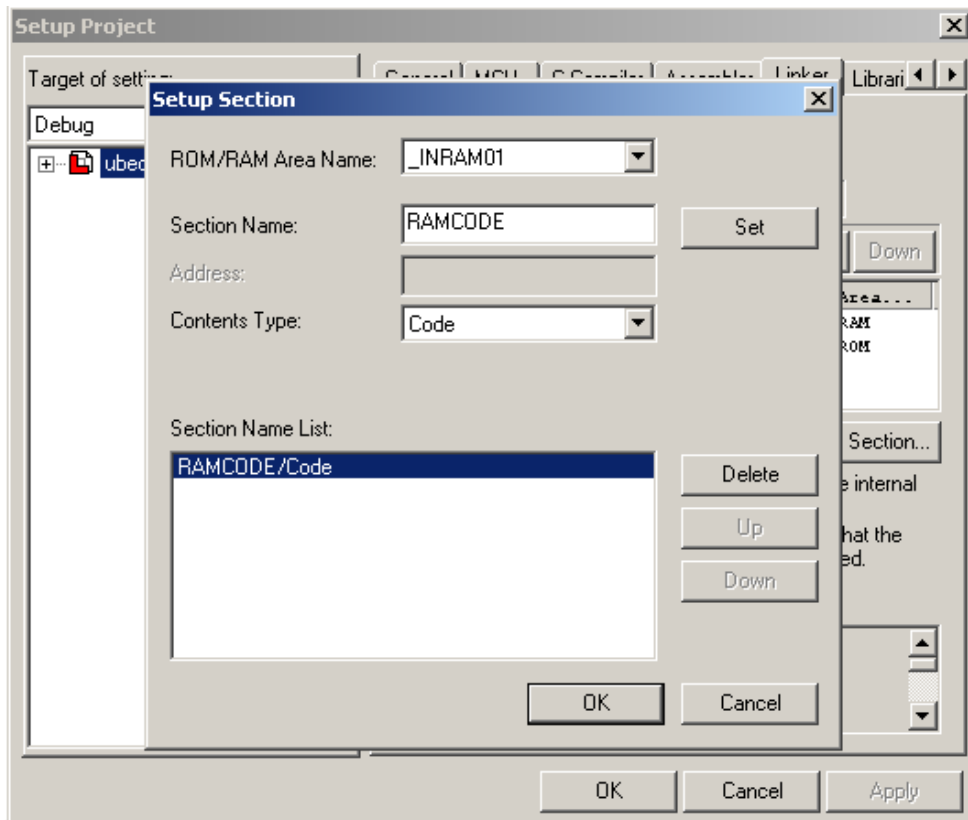


图3-4: Setup Section 中的 RAM 设置

## 步骤 5: Setup Section 中的 ROM 设置

用同样的名字" RAMCODE"设置"@ " section 并设置@RAMCODE/Const in \_INROM01 后，连接器就会识别将初始化数据保存到 ROM 的愿望。

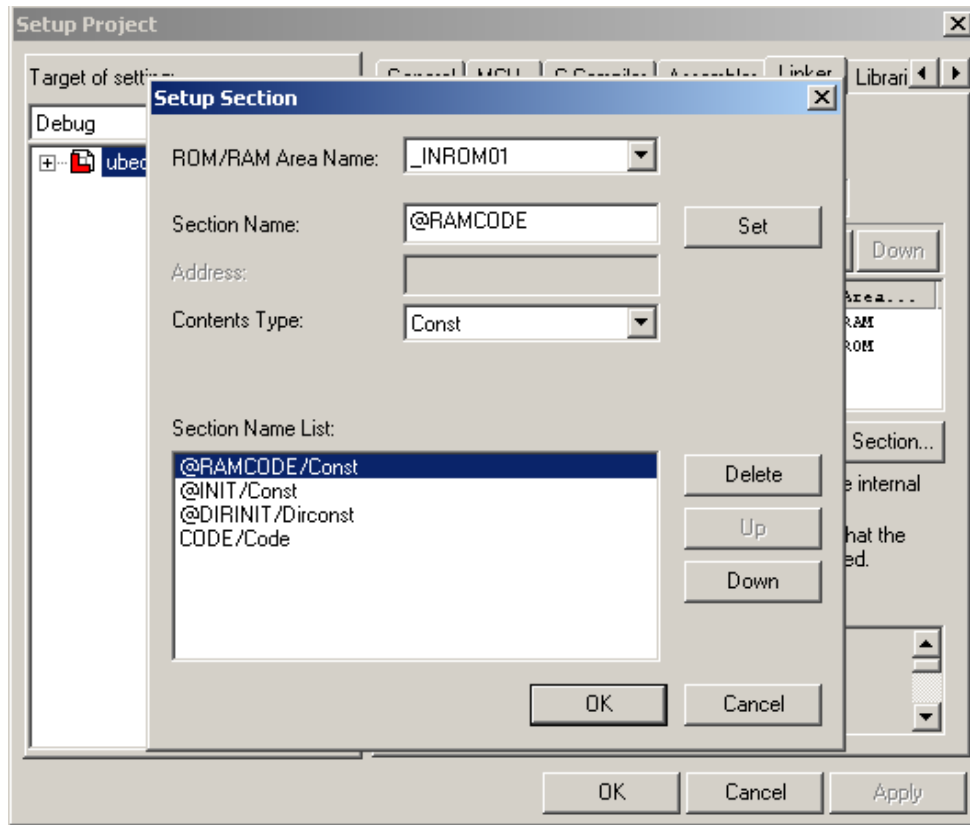


图3-5: Setup Section 中的 ROM 设置

### 3.1.4 将代码从 ROM 复制到 RAM

为了处理这些 section，连接器为 RAM 和 ROM 中 section 的开始地址提供通用标签。

`_RAM_ + section name`

`_ROM_ + section name`

因在本示例中 section 命名为"RAMCODE"，便生成带 RAMCODE 标签的后缀：

`_RAM_RAMCODE`

`_ROM_RAMCODE`

将代码从 ROM 复制到 RAM 前，须将这些标签输入到 startup 文件。

要将代码从 ROM 复制到 RAM，在当前 startup 文件中须增加 4 个 points。

```

;-----
; external declaration of symbols
;-----
    .IMPORT      _RAM_RAMCODE      ; point 1 added here
    .IMPORT      _ROM_RAMCODE     ; point 2 added here
    
```

如下所示添加 point 1 和 point 2 之后，startup 文件输入这些标签。

startup 文件添加 RAM 和 ROM 中 section 的描述及其大小。当添加了 point 3 时，startup 文件为复制以"RAMCODE"命名的 section 做好准备。

```

;-----
; definition to start address of data, const and code section
;-----
    .SECTION     RAMCODE, CODE, ALIGN=1 ; point 3 added here
    
```

如下所示执行指令时，RAMCODE section 的代码从 ROM 复制到 RAM。

```

;-----
; copy initial value *CONST(ROM) section to *INIT(RAM) section
;-----
    ICOPY        _ROM_RAMCODE, _RAM_RAMCODE, RAMCODE; point 4 added here
    
```

### 3.1.5 跳转到 RAM 执行 RAMCODE

用指令调用从 ROM 复制到 RAM 的代码。

例如，若将闪存擦除和闪存写入程序复制到 RAM，且 `_EraseStart` 为闪存擦除的开始地址、`_WriteStart` 为闪存写入程序的开始地址，如下所示可用指令调用：

```
CALL _EraseStart
```

或

```
CALL _WriteStart
```

然后程序跳转到 RAMCODE 所在的 RAM 且在 RAM 执行 RAMCODE。

## 3.2 Flash 操作程序

以下汇编源代码包括闪存擦/写程序。下图显示了程序演算流程。

### 3.2.1 闪存擦除程序的流程图

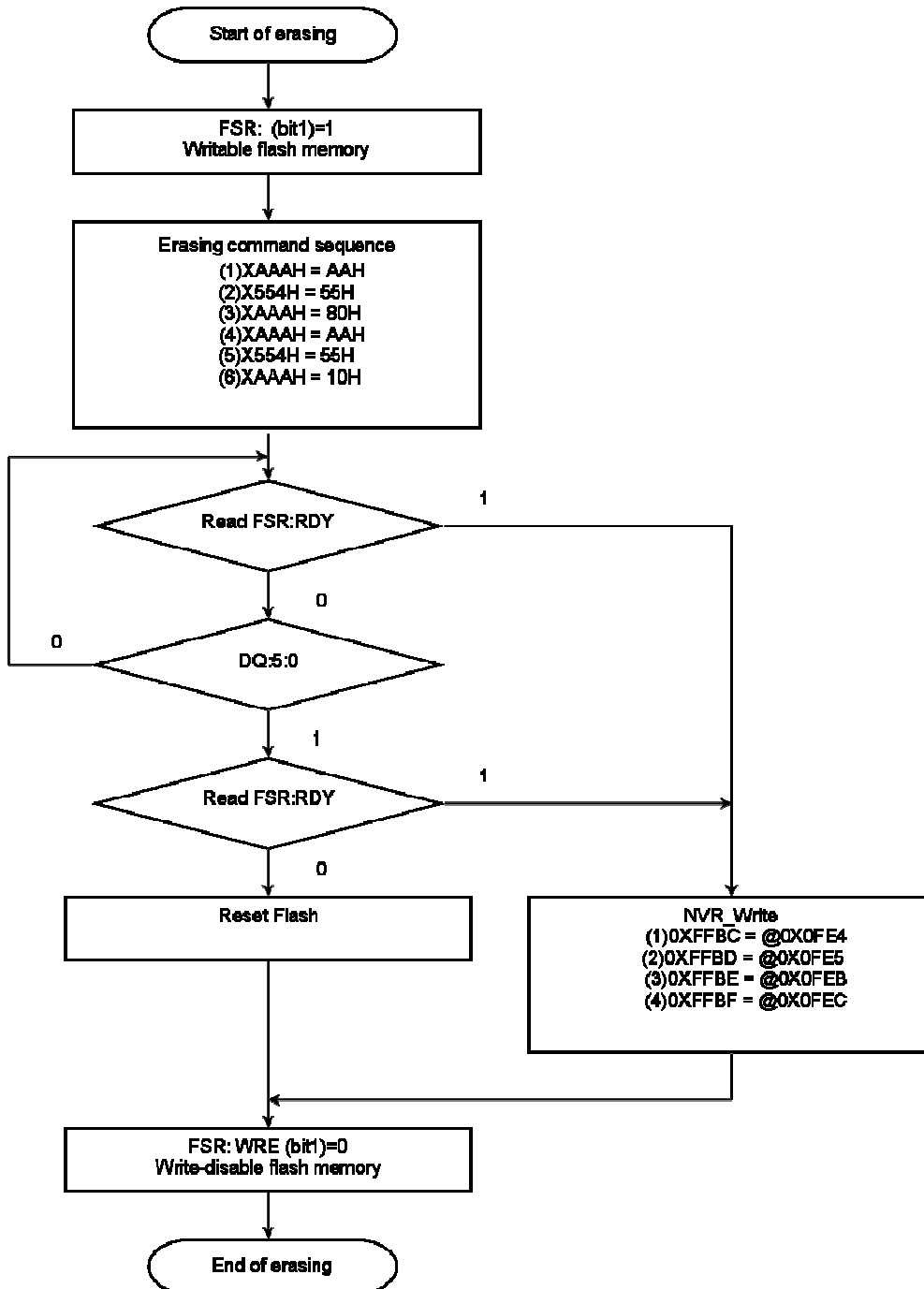


图3-6: 闪存擦除程序的流程图

3.2.2 闪存写入程序的流程图

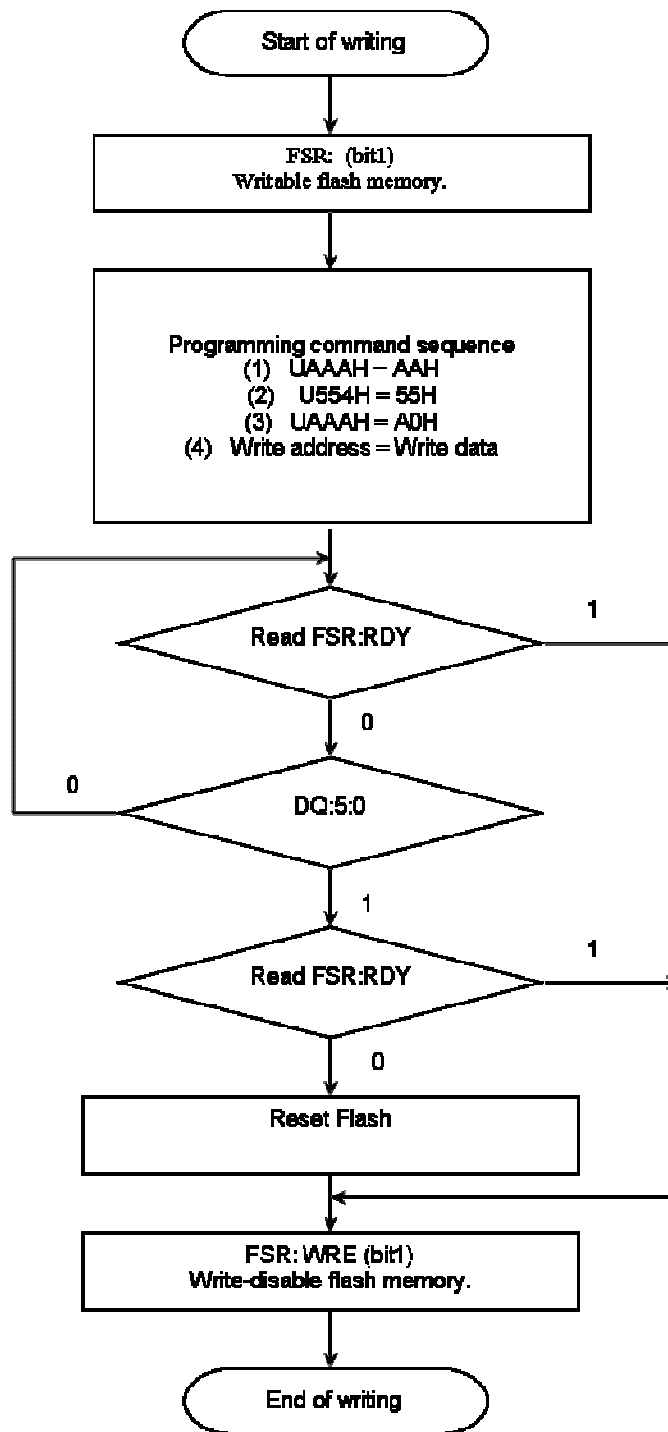


图3-7: 闪存写入程序的流程图

### 3.2.3 Flash.asm

Flash.asm 是执行闪存写/擦的汇编代码。

以下说明几个要点。关于全部代码，参照第 6 节附录中的 6.2 样本代码。

1. `_EraseStart` 是 Flash.asm 中的闪存擦除程序的开始地址。`_WriteStart` 是 Flash.asm 中的闪存写入程序的开始地址。输出这两个地址，这样 `main ()` 中的指令可调用它们。

```
.EXPORT _EraseStart
.EXPORT _WriteStart
```

2. 由 `main ()` 指定的擦除地址保存在 `EP` 中，闪存擦除程序从 `EP` 取出该值，并将其保存在

```
_EraseStart:
    MOVW  A,EP
    PUSHW A           ; Save erase address
```

堆栈中。

由 `main ()` 指定的写地址保存在 `EP` 中，闪存写入程序从 `EP` 取出该值，并将其保存在堆栈中。由 `main ()` 指定的写数据保存在 `IX` 中，闪存写入程序从 `IX` 取出该值，并将其保存在堆

```
_WriteStart:
    PUSHW IX           ; Save write data
    MOVW  A,EP
    PUSHW A           ; Save write address
```

栈中。

3. 根据擦/写闪存自动演算，`UAAAH` 和 `U5554H` 的 `U` 是高 4 位，与写地址的高 4 位相同。以下代码计算 `U` 并获得正确的 `UAAAH` 和 `U5554H`。

```
MOVW  A,#0xF000
ANDW  A
MOVW  A,#0x0AAA
ORW   A
MOVW  EP,A           ; 0x0AAA | (address & 0xf000)
XCHW  A,T
MOVW  A,#0x0554
ORW   A
MOVW  IX,A           ; 0x0554 | (address & 0xf000)
```

4. 设置 `FSR` 寄存器的 `WRE` 位使能闪存写入。

```
SETB  FSR:WRE       ; Write Enable
```

## 5. 擦除闪存自动演算。

```

MOV    A, #0xAA          ; 0x*AAAH <= 0xAA
MOV    @EP, A

MOV    A, #0x55          ; 0x*554 <= 0x55
MOV    @IX, A
MOV    A, #0x80          ; 0x*AAAH <= 0x80
MOV    @EP, A

MOV    A, #0xAA          ; 0x*AAAH <= 0xAA
MOV    @EP, A

MOV    A, #0x55          ; 0x*554 <= 0x55
MOV    @IX, A

POPW   A                  ; Restore Erase address
MOVW   EP, A
MOV    A, #10H           ; The last data.
MOV    @EP, A            ; Start Erase
    
```

## 写入闪存自动演算。

```

MOV    A, #0xAA          ; 0xUAAAH <= 0xAA
MOV    @EP, A

MOV    A, #0x55          ; 0xU554 <= 0x55
MOV    @IX, A

MOV    A, #0xA0          ; 0xUAAAH <= 0xA0
MOV    @EP, A
POPW   A
MOVW   EP, A             ; write address
POPW   IX
MOVW   A, IX             ; write data
MOV    @EP, A            ; to write flash
    
```

## 6. 擦除循环

```

EraseLoop:
    BBS    FSR:RDY,EraseEnd ; Erase Flash successes?

    MOV    A,@EP
    AND    A,#0x20          ; Check Time Out?
    BZ     EraseLoop

    BBS    FSR:RDY,EraseEnd ; Erase Flash successes?
    NOP
    BBS    FSR:RDY,EraseEnd ; Erase Flash successes?
    
```

## 写入循环

```

WriteLoop:
    BBS    FSR:RDY,WriteEnd ; write Flash successes?

    MOV    A,@EP
    AND    A,#0x20          ; to check time out?
    BZ     WriteLoop

    BBS    FSR:RDY,WriteEnd ; write Flash successes?
    NOP
    BBS    FSR:RDY,WriteEnd ; write Flash successes?
    
```

## 7. 若闪存擦除失败，复位闪存并在 A 设置错误标志。

```

EraseError:
    MOV    A,#0xF0
    MOV    0xFF00,A        ; Reset Flash
    MOV    A,#01H         ; Set error Flag
    
```

## 若闪存写入失败，复位闪存并在 A 设置错误标志。

```

WriteError:
    MOV    A,#0xF0
    MOV    0xFF00,A        ; Reset Flash
    MOV    A,#01H         ; Set error Flag
    
```

## 8. 清零寄存器 FSR 的位 WRE 以禁止写入闪存。

```

CLRB    FSR:WRE          ; write disable
    
```

9. 若擦除闪存成功，在 A 设置成功标志。

```
EraseEnd:  
    MOV    A, #00H                ; normal ack
```

若写入闪存成功，在 A 设置成功标志。

```
WriteEnd:  
    MOV    A, #00H                ; normal ack
```

若成功擦除闪存，闪存为空白状态且 NVR 数据应写入闪存。详情参照第 4 节闪存操作注意事项中的 4.1 擦除后重写 NVR。

```
NVR_Write:  
    MOVW   EP, #0xFFBC  
    MOVW   IX, #0xFE4  
    CALL   _WriteStart  
  
    MOVW   EP, #0xFFBD  
    MOVW   IX, #0xFE5  
    CALL   _WriteStart  
  
    MOVW   EP, #0xFFBE  
    MOVW   IX, #0xFEB  
    CALL   _WriteStart  
    MOVW   EP, #0xFFBF  
    MOVW   IX, #0xFEC  
    CALL   _WriteStart
```

### 3.3 子程序

以下是闪存擦除和闪存写入子程序，该程序可将闪存操作地址和数据传递给 RAM 区内的闪存汇编程序，然后调用该闪存汇编程序。关于全部代码，参照第 6 节附录中的 6.2 示例代码。

#### 3.3.1 闪存擦除 C 代码

定义全局变量。

```
unsigned char result, Flag;  
unsigned short address;  
unsigned char data;
```

根据汇编和 c 语言编译规则，汇编语言中的变量“\_address”与 c 语言中的变数“address”相同。以下代码将 main ()指定的擦除地址传递到 EP 中，以便 flash.asm 使用。

```
MOVW A, _address  
MOVW EP,A
```

用以下代码调用 RAM 区内的闪存擦除汇编程序。

```
CALL _EraseStart
```

\_EraseStart 是 Flash.asm 中的闪存擦除程序的开始地址。然后程序跳转到 RAM 并在 RAM 运行。

#### 3.3.2 闪存写入 C 代码

以下代码将 main ()指定的写地址传递到 EP，将写数据传递到 IX。这些将在 flash.asm 中使用。

```
MOV A,_data ;write data  
MOVW IX,A  
MOVW A, _address ;write address  
MOVW EP,A
```

用以下代码调用 RAM 区内的闪存写入汇编程序。

```
CALL _EraseStart
```

\_WriteStart 是 Flash.asm 中的闪存写入程序的开始地址。然后程序跳转到 RAM 并在 RAM 运行。

### 3.4 如何使用编程功能

以下 C main 代码演示如何使用闪存编程功能。关于全部代码，参考第 6 节附录中 6.2 的示例代码。

EraseStart 是 Flash.asm 中的闪存擦除程序的开始地址。WriteStart 是 Flash.asm 中的闪存写入程序的开始地址。输入 EraseStart 和 WriteStart，这样 main ()中的指令可以调用它们。根据汇编和 c 语言编译规则，汇编语言中的“\_EraseStart”与 c 语言中的“EraseStart”相同；“\_WriteStart”与“WriteStart”相同。

```
extern EraseStart;  
extern WriteStart;
```

用户可根据闪存操作结果在此添加程序。

```
void error(void)  
{  
    // do something here  
}  
void success(void)  
{  
    // do something here  
}
```

全局变量地址和数据将写地址“0xF800”和数据“0xA0”传递到 Flash.asm。用户可指定写地址“0xF800”和数据“0xA0”。

```
address = 0xF800; //write address  
data = 0xA0;     //write data
```

调用 flash\_write ()，写闪存的结果返回到 Flag。

```
Flag = flash_write();
```

若 Flag == 1，则写闪存失败，在 error ()做相应处理。若 Flag == 0，写闪存成功，在 success ()中做相应处理。

```
if (Flag == 1)  
    error();  
else  
    success();
```

## 4 闪存操作的使用注意事项

---

本节介绍闪存操作的使用注意事项。

---

### 4.1 擦除后重写 NVR

1. 闪存擦除操作可擦除全部 NVR 数据。

闪存编程器需进行以下操作以保持原有系统设置。

(1) 备份 CRTH:CRTH4-CRTH0 和 CRTL:CRTL4-CRTL0 的数据。

(2) 擦除闪存。

(3) 将 CRTH:CRTH4-CRTH0 和 CRTL:CRTL4-CRTL0 中的数据保存到 NVR 闪存区。

若 CRTH:CRTH4-CRTH0 和 CRTL:CRTL4-CRTL0 中有新数据，闪存编程器将新数据写入 NVR 闪存区。

2. 写入 NVR 数据同对闪存写入字节相同。详情参考 MB95200H/210H 系列硬件手册的第 20 章和第 23 章。

### 4.2 在 RST 引脚施加高压

对闪存写入数据或整片擦除数据时，在 RST 引脚施加高 DC 电压(+10V)。施加高压后，等等待 10ms，方可写数据到闪存或整片擦除数据。保持 RST 引脚的电压，直到数据写入或擦除完成。

## 5 附加信息

关于富士通微电子更多的产品信息，请访问以下网站：

## 6 附录

### 6.1 图一览

图 2-1: 32/64/128-KBIT 闪存的扇区配置.....	6
图 2-2: 闪存的寄存器.....	7
图 2-3: 命令顺序.....	7
图 3-1: 在 Project 选择 Setup Project.....	9
图 3-2: 在 Setup Project 选择 Disposition/Connection .....	10
图 3-3: 在 Disposition/Connection 选择 Section.....	10
图 3-4: Setup Section 中的 RAM 设置 .....	11
图 3-5: Setup Section 中的 ROM 设置 .....	12
图 3-6: 闪存擦除程序的流程图.....	15
图 3-7: 闪存写入程序的流程图.....	16

## 6.2 示例代码

### 6.2.1 工程

名称: flashoperation

功能: 演示闪存操作

main.c

```

/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS */
/* OR ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics (Shanghai) Co., LTD. */
/*****
Main.C: flash.asm located in RAM.
See README.TXT for project description and disclaimer.
*****/
#include "mb95200.h"

unsigned char result, Flag;
unsigned short address;
unsigned char data;

extern EraseStart;
extern WriteStart;

/*****
Name: error ();
Function: flash operation error handle process
*****/
void error(void)
{
    // do something here
}

/*****
Name: success ();
Function: flash operation success handle process
*****/
void success(void)
{
    // do something here
    
```

```
}
/*****
Name:      flash_erase ();
Function:  flash erase operation entrance
*****/
unsigned char flash_erase(void)
{
    #pragma asm
        MOVW A, _address    // set erase address
        MOVW EP,A
        CALL _EraseStart    // Start erase subroutin
        MOV  _result,A      // Reture the result for flash erase
    #pragma endasm

    return result;
}

/*****
Name:      flash_write ();
Function:  flash write operation entrance
*****/
unsigned char flash_write(void)
{
    #pragma asm
        MOV  A,_data        // set write data
        MOVW IX,A
        MOVW A, _address    // set write address
        MOVW EP,A
        CALL _WriteStart    // Start Flash write subroutine
        MOV  _result,A      // Reture the result for flash write
    #pragma endasm

    return result;
}
```

```
/******  
Name:      main ();  
Function:   main loop, a sample for flash write function usage  
*****/  
  
void main (void)  
{  
  // Please follow procedure below to write flash  
  // During writing data or erasing all data in flash memory,  
  // a typical +10VDC voltage should be applied at the RST pin.  
  // After applying the high voltage,  
  // wait for 10ms before writing data or erasing all data in flash memory.  
  // And this applied voltage should be kept at the RST pin until data writing or  
  // erasing has completed.  
      address = 0xF800;          // set write address  
      data = 0xA0;              // set write data  
      Flag = flash_write();     // flash write routine  
  
      if (Flag == 1)  
          error();  
      else  
          success();  
  
  // Please follow procedure below to erase flash  
  // (1) Copy out all NVR contents from register: 0x0FE4, 0x0FE5, 0X0FEB, 0x0FEC.  
  // (2) Erase flash  
  // (3) Restore all NVR contents to flash area: 0xFFBC, 0xFFBD, 0XFFBE, 0xFFBF.  
  // (1) Copy out,add your routine  
  // (2) Erase flash  
  //      address = 0xF800;  
  //      Flag = flash_erase();  
  //      if (Flag == 1)  
  //          error();  
  //      else  
  //          success();  
  // (3) Restore, add your routine  
  // In this demonstration, restore is implemented in _EraseStart, if flash erase  
  // successes.And because flash has been erased, the routine can't return to  
  // main routine which is located in Flash Memory. So, all NVR contents have to be  
  // restored to flash area in _EraseStart.  
  // NOTE: when flash_erase() routine is used, pay more attention please!
```

}

flash.asm

; THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU  
; MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS  
; OR ELIGIBILITY FOR ANY PURPOSES.

; (C) Fujitsu Microelectronics (Shanghai) Co., LTD.

-----  
; flash.asm  
; - description and disclaimer see readme.txt  
; flash.asm is a code module to demonstrate how to use Flash memory write/erase.  
; flash.asm is located in RAM area and all functions within flash.asm are also  
; located in RAM.  
; flash.asm is called by main(), which is located in Flash Memory.

; A is used to indicate the status of the programming:

; A : Return Flag  
; 0 : Succeeded  
; 1 : Fail

-----  
.SECTION RAMCODE, CODE  
.EXPORT \_EraseStart  
.EXPORT \_WriteStart

; FlashROM control register  
WRE .equ 1 ; Flash enable flag  
RDY .equ 4 ; Flash finish flag  
FSR .equ 0x0072 ; Flash control register  
; EP : Operation Address  
; IX : Operation Data

;+++++

; Command\_Process  
; Flash Erase  
; IN : EP : Erase Sector Address  
; OUT : A : Return Flag  
; 0 : Succeeded  
; 1 : Fail

;+++++

\_EraseStart:  
MOVW A,EP  
PUSHW A ; Save write address  
  
MOVW A,#0xF000  
ANDW A  
MOVW A,#0x0AAA  
ORW A  
MOVW EP,A ; 0x0AAA | (address & 0xf000)  
XCHW A,T  
MOVW A,#0x0554  
ORW A  
MOVW IX,A ; 0x0554 | (address & 0xf000)

EraseGo:

SETB FSR:WRE ; Write Enable  
  
MOV A,#0xAA ; 0x\*AAAH <= 0xAA  
MOV @EP,A  
  
MOV A,#0x55 ; 0x\*554 <= 0x55

```

MOV      @IX,A

MOV      A,#0x80          ; 0x*AAAH <= 0x80
MOV      @EP,A

MOV      A,#0xAA         ; 0x*AAAH <= 0xAA
MOV      @EP,A

MOV      A,#0x55         ; 0x*554 <= 0x55
MOV      @IX,A

POPW     A                ; Restore Erase address
MOVW    EP,A
MOV      A,#10H          ; The last data.
mov     @EP,A            ; Start Erase

NOP
NOP

EraseLoop:
BBS     FSR:RDY,EraseEnd

MOV     A,@EP
AND     A,#0x20          ; Check Time Out?
BZ      EraseLoop

BBS     FSR:RDY,EraseEnd
NOP
BBS     FSR:RDY,EraseEnd

EraseError:
MOV     A,#0xF0
MOV     0xFF00,A        ; Reset Flash
MOV     A,#01H          ; Set error Flag
CLRB   FSR:WRE          ; write disable
RET

EraseEnd:
MOV     A,#00H          ; normal ack
CLRB   FSR:WRE          ; write disable
;In this demonstration, restore is implemented in _EraseStart, if flash erase
;successes. Because flash has been erased, the routine can't return to main
;routine which is located in Flash Memory. So, all NVR contents have to be
;restored to flash area here.
NVR_Write:
MOVW   EP,#0xFFBC
MOVW   IX,#0xFE4
CALL   _WriteStart

MOVW   EP,#0xFFBD
MOVW   IX,#0xFE5
CALL   _WriteStart

MOVW   EP,#0xFFBE
MOVW   IX,#0xFEB
CALL   _WriteStart

MOVW   EP,#0xFFBF
MOVW   IX,#0xFEC
CALL   _WriteStart
RET
    
```

```

;=====
;+++++
; Command_Process
; Flash Write
;   IN      :      EP : Write Address
;             IX : Write Data
;   OUT     :      A  : Return Flag
;             0  : Succeeded
;             1  : Fail
;+++++
_WriteStart:
    PUSHW    IX                ; Save write data
    MOVW     A,EP
    PUSHW    A                 ; Save write address

    SETB     FSR:WRE           ; Write Enable
    MOVW     A,#0xF000
    ANDW     A
    MOVW     A,#0x0AAA
    ORW      A
    MOVW     EP,A              ; 0x0AAA | (address & 0xf000)
    XCHW     A,T
    MOVW     A,#0x0554
    ORW      A
    MOVW     IX,A             ; 0x0554 | (address & 0xf000)

    MOV      A,#0xAA          ; 0xUAAAH <= 0xAA
    MOV      @EP,A
    MOV      A,#0x55          ; 0xU554 <= 0x55
    MOV      @IX,A
    MOV      A,#0xA0          ; 0xUAAAH <= 0xA0
    MOV      @EP,A

    POPW     A
    MOVW     EP,A            ; write address
    POPW     IX
    MOVW     A,IX           ; write data
    MOV      @EP,A         ; to write flash
    NOP
    NOP

WriteLoop:
    BBS      FSR:RDY,WriteEnd

    MOV      A,@EP
    AND      A,#0x20        ; to check time out?
    BZ       WriteLoop

    BBS      FSR:RDY,WriteEnd
    NOP
    BBS      FSR:RDY,WriteEnd

WriteError:
    MOV      A,#0xF0
    MOV      0xFF00,A       ; Reset Flash
    MOV      A,#01H        ; Set error Flag
    CLRB     FSR:WRE       ; write disable
    RET

WriteEnd:
    MOV      A,#00H        ; Set normal Flag
    CLRB     FSR:WRE       ; write disable

```

RET

---

Startup.asm

---

```

;=====
; F2MC-8FX SOFTUNE C Compiler sample startup routine,
; ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 2008
; LICENSED MATERIAL - PROGRAM PROPERTY OF FUJITSU LIMITED
;=====
; Sample program for initialization
;-----
                .PROGRAM    start
                .TITLE      start

;-----
; variable define declaration
;-----
#define HWD_DISABLE                ; if define this, Hard Watchdog will disable.

;-----
; external declaration of symbols
;-----
                .EXPORT     __start
                .IMPORT     _main
                .IMPORT     LMEMTOMEM
                .IMPORT     LMEMCLEAR
                .IMPORT     _RAM_INIT
                .IMPORT     _ROM_INIT
                .IMPORT     _RAM_DIRINIT
                .IMPORT     _ROM_DIRINIT
                .IMPORT     _RAM_RAMCODE
                .IMPORT     _ROM_RAMCODE

;-----
; definition to stack area
;-----
                .SECTION    STACK, STACK, ALIGN=1
                .RES.B      128-2
STACK_TOP:
                .RES.B      2

;-----
; definition to start address of data, const and code section
;-----
                .SECTION    RAMCODE, CODE, ALIGN=1
                .SECTION    DIRDATA, DIR, ALIGN=1
                .SECTION    DIRINIT, DIR, ALIGN=1
                .SECTION    DATA, DATA, ALIGN=1
                .SECTION    INIT, DATA, ALIGN=1

;-----
; code area
;-----
                .SECTION    CODE, CODE, ALIGN=1
__start:
;-----
; set stack pointer
;-----
                MOVW A, #STACK_TOP
                MOVW SP, A

```

```

;-----
; set register bank is 0
;-----
                MOVW A, PS
                MOVW A, #0x07FF
                ANDW A
                MOVW PS, A

;-----
; set ILM to the lowest level(3)
;-----
                MOVW A, PS
                MOVW A, #0x0030
                ORW  A
                MOVW PS, A

;-----
; copy initial value *CONST(ROM) section to *INIT(RAM) section
;-----
#macro          ICOPY src_addr, dest_addr, src_section
                MOVW EP, #\src_addr
                MOVW A, #\dest_addr
                MOVW A, #SIZEOF (\src_section)
                CALL  LMENTOMEM

#endm

                ICOPY _ROM_INIT,      _RAM_INIT,      INIT
                ICOPY _ROM_DIRINIT,   _RAM_DIRINIT,   DIRINIT
                ICOPY _ROM_RAMCODE,   _RAM_RAMCODE,   RAMCODE

;-----
; zero clear of *VAR section
;-----
#macro          FILL0 src_section
                MOVW A, #\src_section
                MOVW A, #SIZEOF (\src_section)
                CALL  LMEMCLEAR

#endm

                FILL0 DIRDATA
                FILL0 DATA

;-----
; call main routine
;-----
                CALL  _main
end:           JMP   end

;-----
; Hard Watchdog
;-----
#ifdef  HWD_DISABLE
                .SECTION WDT, CONST, LOCATE=H'FFBE
                .DATA.W 0xA596
#endif

```

```
-----  
; reset vector  
-----  
        .SECTION    RESET, CONST, LOCATE=0xFFFC  
        .DATA.B     0xFF  
        .DATA.B     0  
        .DATA.H     __start  
  
        .END    __start
```