文档编号：AN_062

上海东软载波微电子有限公司

应用笔记

MCU 片内非易失性存储器操作

修订历史

版本	修订日期	修改概要
V1.0	2015-05-19	初版
V1.1	2015-07-28	统一修改公司名称、logo 及网址等

地　址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目 录

内容目录

第1章 概述

1.1 非易失性存储器类型

HR7P 系列 MCU 片内非易失性存储器类型有以下三种：

- ROM Flash
- Data Flash
- EEPROM

1.2 ROM Flash

ROM Flash，即 MCU 的 ROM 程序存储区，支持应用自编程（IAP）功能，因此可以通过 IAP 实现对本身进行读写操作的功能。用户利用 IAP 功能可以实现对重要数据的非易失性存储。

在使用 IAP 功能对 ROM Flash 操作时，用户需要注意：

1. 在改写 ROM Flash 前，需要对其擦除，页擦除时间约 20ms，此过程中 MCU 内核和外设停止工作，禁止 MCU 响应任何中断，擦除结束后 MCU 内核和外设继续工作；
2. 在写入数据到 ROM Flash 时，单个 word 编程时间约 77us，此过程中 MCU 内核和外设停止工作，禁止 MCU 响应任何中断，写操作结束后 MCU 内核和外设继续工作；
3. 在读 ROM Flash 前，禁止 MCU 响应任何中断，数据读取完成后再开全局中断使能位；
4. 由于擦写时间比较长，若系统启用了看门狗，建议在擦写前用 CWDT 指令清看门狗计数，同时要设置看门狗溢出时间大于页擦除时间，以免在擦写过程中，WDT 溢出而导致复位。
5. 具体内容请查阅对应 MCU 的数据手册；

支持 ROM Flash 带 IAP 自编程的 MCU 包括 HR7P193/194/293、HR7P195、HR7P192/196 和 HR7P9x、HR7P275 等芯片。

1.3 Data Flash

Data Flash 和 ROM Flash 共用一块存储器，MCU 通过地址线将 ROM 划分为 ROM Flash 程序存储器和 Data Flash 存储器两个空间。用户可以使用规定的操作方法和指令对 Data Flash 进行操作，可以实现对重要数据的非易失性存储。

在使用 Data Flash 时，用户需要注意：

1. 在改写 Data Flash 前，需要对其擦除，页擦除时间约 2ms，此过程中 MCU 内核停止工作，外设如定时器按照原设定参数继续工作，禁止 MCU 响应任何中断，擦除结束后 MCU 内核继续工作；
2. 在写入数据到 Data Flash 时，单个 word 写操作时间约 30us，此过程中 MCU 内核停止工作，外设如定时器按照原设定参数继续工作，禁止 MCU 响应任何中断，写操作结束后 MCU 内核继续工作；
3. 读 Data Flash 操作不影响 MCU 内核和外设的工作，MCU 响应中断；

4. 由于擦写时间比较长，若系统启用了看门狗，建议在擦写前用 CWDT 指令清看门狗计数或 RCEN 关闭 WDT 时钟，以免在擦写过程中，WDT 溢出而导致复位。
5. 具体内容请查阅对应 MCU 的数据手册；

支持 Data Flash 的 MCU 包括 HR7P169/170 和 HR7P201 等芯片。

1.4 EEPROM

EEPROM 单独采用了一块 Flash 存储器，通过专利算法实现擦写次数的扩展，并具有和普通 EEPROM 相同的读写操作方式。用户可以使用规定的操作方法和指令对 EEPROM 进行操作，可以实现对重要数据的非易失性存储。

在使用 EEPROM 时，用户需要注意：

1. EEPROM 存储器读写操作以字节为单位，容量大小参照数据手册；
2. EEPROM 操作前无需擦除操作，写 EEPROM 时，MCU 的内核和外设继续工作，禁止 MCU 响应中断；

支持 EEPROM 的 MCU 包括 HR7P275 等芯片。

第2章 非易失性存储器操作可靠性问题

2.1 使用场景和故障现象

由于 ROM Flash 和 Data Flash 的特点，多数情况下，用户仅能将其用作非实时发生的数据保存，这些保存数据包括：同一产品不同个体的标定数据、系统上电的环境参数、系统掉电前的用户设定数据等。

由于对这些数据的操作多发生在系统上电、掉电或复位情况下，因此在擦写 ROM Flash 和 Data Flash 时容易受到电源波动的干扰，导致存储数据的丢失或者写入失败，造成系统工作异常。主要现象有：

1. 在电源电压变化过程中，工作电压可能超出 MCU 规定的电压范围，可能导致 MCU 工作状态出现混乱，从而使存储器中的数据发生改变。
2. MCU 未有效复位，存储器的读/写地址和数据没有被有效初始化，可能引起对存储器的误操作而使其中的数据发生改变。

2.2 提高非易失性存储器操作可靠性的解决措施

为了提高非易失性存储器操作的可靠性，用户除了在硬件上提高电源的稳定性以外，还可以利用 MCU 的各种标志和冗余算法，通过软件来加强非易失性存储器读写操作的稳定性。用户可采用的方法如下：

- 使能 BOR 功能
- 操作状态保护
- 校验法
- 表决法
- 软件锁

2.2.1 使能BOR功能

通过章节 2.1 的介绍，非易失性存储器误操作，多数发生在电源电压不稳定的时候，因此使能 MCU 的 BOR 配置，并设置适当的 BOR 电压点，是提高对非易失性存储器操作可靠性的关键。

1. 系统供电电压为 3V/3.6V，建议 BOR 电压选择 2.4V 或者使用外置更高电压的复位器件。
2. 系统供电电压 5V，建议 BOR 电压选择 3.8V 或 4.2V。
3. 程序在进行非易失性存储器擦、写操作前，判断 BOR 状态位，如果发生了低电压复位，则退出本次存储器操作，等待供电电压稳定后再继续操作存储器。

2.2.2 操作状态保护

- 中断使能位操作

非易失性存储器读写操作前，用户程序必须关闭全局中断使能，防止意外的中断响应而导致擦写失败。非易失性存储器擦写操作完成后，及时开启全局中断，保证其它任务的顺利执行。

- 擦写使能位操作

非易失性存储器擦写操作执行时，打开存储器擦写使能位。擦写操作执行完成后，及时关闭存储器擦写使能位，防止程序意外跑飞时误执行存储器擦写操作。

- 地址指针和数据寄存器

非易失性存储器擦写完成后，将存储器地址指针指到空闲地址空间并将存储器擦写操作数据寄存器的值赋值为 0xFF，降低因意外导致储存器数据被改写的概率。

2.2.3 校验法

采用写入-读出校验法。在对非易失性存储器写入数据后，进行读出校验，若校验正确，则判断写入成功；若校验错误，则判断写入失败，重新执行擦写操作。

2.2.4 表决法

采用数据备份和表决方法。在非易失性存储器空间足够的情况下，采用在不同页地址中双备份或多备份数据的方法。读出时采用表决方式判断写入数据的有效性。

2.2.5 软件锁

设置软件开关，对非易失性存储器擦写操作进行使能和禁止。

例如：设定 2 个全局变量 CallFlashEn 和 FlashEwEn 作为软件锁

1. CallFlashEn 用于调用 Flash 擦写函数的使能判别，在每次需要调用擦写函数前，设置为固定值，如设置为 0x55，擦写执行完毕，在擦写函数中将其清零。
2. FlashEwEn 用于擦写函数内的使能判别，擦写函数对其赋值和清零，如设置为 0xAA。
3. 标准 Flash 擦写函数中，使用与软件锁变量相关的判断语句代替 8 个 nop 语句，当程序意外走飞或者干扰导致 CallFlashEn 和 FlashEwEn 软件锁发生变化时，中断非易失性存储器擦写操作。

Flash 读函数例程：

```
#include <hic.h>
//*****************************************************************************
//                                Flash 读函数
//*****************************************************************************
void ReadFlash(void)
{
    FRAH = FlashAddr.Byte[1];        //设置读出的起始地址
    FRAL = FlashAddr.Byte[0];
    GIE = 0;                         //若开启了全局中断，则禁止中断
    __asm
    {
        TBR                          //查表读，读取数据到 ROMDH/ROMDL 寄存器
    }
```

```
    FlashData.Byte[1] = ROMDH;
    FlashData.Byte[0] = ROMDL;
    FRAH = 0xFF;                    //指到不用的地址空间
    FRAL = 0xFF;                    //指到不用的地址空间
    ROMDH = 0xFF;                   //数据初始化为 0xFF
    ROMDL = 0xFF;                   //数据初始化为 0xFF
    GIE = 1;                        //重新开启全局中断
}
```

Flash 擦除函数例程：

```
#include <hic.h>
//*****************************************************************************
//                                Flash 擦除函数
//*****************************************************************************
void EraseFlash(void)
{
    FRAH = FlashAddr.Byte[1];
    FRAL = FlashAddr.Byte[0];
    PA1 = 1;
    FlashEwEn = 0xAA;               //设置软件锁变量
    __asm
    {
        JBS PWRC,0                  //判断 BOR 是否发生，若未发生，则执行擦写
        RST                         //若发生 BOR，则执行软件复位指令
        BCC INTG,GIE                //若开启了全局中断，则禁止中断
        BSS ROMCL,FPEE              //选择 Flash 擦除
        BSS ROMCL,WREN              //使能 Flash 擦除
        MOVI 0x55
        MOVA ROMCH                  //0x55 写入虚拟存储器

        MOVI 0xFF                   // 8 个 NOP 代替语句开始
        SECSEL &FlashEwEn& % 0x80       //选择变量所在的 section
        XOR &FlashEwEn& % 0x80, 0       //0xFF xor 0xAA = 0x55
        SECSEL &CallFlashEn& % 0x80     //选择变量所在的 section
        XOR &CallFlashEn& % 0x80, 0     //若 CallFlashEN 在调用函数前为 0x55，0x55 XOR 0x55 = 0x00
        JBS PSW, Z                      //如果运算结果为 0x00，则执行正确，跳至下个 NOP
        GOTO $+3                        //如果运算结果不为 0x00，则执行错误，跳出
        NOP                         //8 个 NOP 代替语句结束

        MOVI 0xAA
        MOVA ROMCH                  //0xAA 写入虚拟存储器
        MOVI 0XFF                   //8 个 NOP 代替语句开始
        SECSEL &FlashEwEn& % 0x80       //选择变量所在的 section
```

```
        XOR &FlashEwEn& % 0x80, 0          //0xFF xor 0xAA = 0x55
        SECSEL &CallFlashEn& % 0x80        //选择变量所在的 section
        XOR &CallFlashEn& % 0x80, 0        //若 CallFlashEN 在调用函数前为 0x55，0x55 XOR 0x55 = 0x00
        JBS PSW, Z                         //如果运算结果为 0x00，则执行正确，跳至下个 NOP
        GOTO $+3                           //如果运算结果不为 0x00，则执行错误，跳出。
        NOP                        //8 个 NOP 代替语句结束
        BSS ROMCL,WR               //启动擦除
        NOP
        NOP
        JBC ROMCL,WR               //判断是否擦除结束标志
        GOTO $-1                   //等待擦除结束
        BCC ROMCL,WREN             //禁止 Flash 擦除
    }
    CallFlashEn = 0;               //清零软件锁变量
    FlashEwEn = 0;
    GIE=1;                         //重新开启全局中断
}
```

Flash 写函数例程：

```
//*****************************************************************************
//                              Flash 写函数
//*****************************************************************************
void WriteFlash(void)
{
    FlashEwEn = 0xAA;              //设置软件锁变量
    FRAH=FlashAddr.Byte[1];        //设置写入的起始地址
    FRAL=FlashAddr.Byte[0];
    ROMDH=FlashData.Byte[1];       //设置写入的数据
    ROMDL=FlashData.Byte[0];
    __asm
    {
        JBS PWRC, 0                //判断 BOR 是否发生，若未发生，则执行擦写
        RST                        //若发生 BOR，则执行软件复位指令
        BCC   ROMCL,FPEE           //选择写功能
        BSS   ROMCL,WREN           //使能写
        BCC   INTG, GIE            //关闭全局中断
        MOVI 0X55
        MOVA ROMCH                 //0x55 写入虚拟存储器

        MOVI 0xFF                  // 8 个 NOP 代替语句开始
        SECSEL &FlashEwEn& % 0x80      //选择变量所在的 section
        XOR &FlashEwEn& % 0x80, 0      //0xFF xor 0xAA = 0x55
        SECSEL &CallFlashEn& % 0x80    //选择变量所在的 section
```

```
            XOR &CallFlashEn& % 0x80, 0        //若 CallFlashEN 在调用函数前为 0x55，0x55 XOR 0x55 = 0x00
            JBS PSW, Z                         //如果运算结果为 0x00，则执行正确，跳至下个 NOP
            GOTO $+3                           //如果运算结果不为 0x00，则执行错误，跳出。
            NOP                                //8 个 NOP 代替语句结束

            MOVI 0XAA
            MOVA ROMCH                         //0xAA 写入虚拟存储器
            MOVI 0xFF                          // 8 个 NOP 代替语句开始
            SECSEL &FlashEwEn& % 0x80          //选择变量所在的 section
            XOR &FlashEwEn& % 0x80, 0          //0xFF xor 0xAA = 0x55
            SECSEL &CallFlashEn& % 0x80        //选择变量所在的 section
            XOR &CallFlashEn& % 0x80, 0        //若 CallFlashEN 在调用函数前为 0x55，0x55 XOR 0x55 = 0x00
            JBS PSW, Z                         //如果运算结果为 0x00，则执行正确，跳至下个 NOP
            GOTO $+3                           //如果运算结果不为 0x00，则执行错误，跳出。
            NOP                                //8 个 NOP 代替语句结束
            BSS ROMCL,WR                       //启动写
            NOP
            NOP
            JBC ROMCL,WR                       //判断是否写结束标志
            GOTO $-1                           //等待写结束
            BCC ROMCL,WREN                     //禁止 Flash 写
        }
        CallFlashEn = 0;                       //清零软件锁变量
        FlashEwEn = 0;
        GIE=1;                                 //重新开启全局中断
}
```

2.3 小结

可靠性设计是一个系统问题，它需要从硬件和软件上协同考虑，采用硬件冗余设计和软件冗余算法，共同提高系统抵抗异常操作时的可靠性。对于非易失性存储器的操作同样如此，章节 2 的内容仅是我们针对客户反馈的情况，针对公司产品的特点而采取的部分方法。这些方法已经在客户产品上进行了验证并取得了较好的效果。

用户也可以根据具体系统的实际需求，采用更加合适的软硬件设计，提高整个系统的可靠性。