Shanghai Fudan Microelectronics Group Company Limited

复旦微电子

FM33LE0xxA
车用系列 MCU

FLASH 模拟 EEPROM

V1.3.1.3

本资料是为了让用户根据用途选择合适的上海复旦微电子集团股份有限公司（以下简称复旦微电子）的产品而提供的参考资料，不转让属于复旦微电子或者第三者所有的知识产权以及其他权利的许可。

在使用本资料所记载的信息最终做出有关信息和产品是否适用的判断前，请您务必将所有信息作为一个整体系统来进行评价。

采购方对于选择与使用本文描述的复旦微电子的产品和服务全权负责，复旦微电子不承担采购方选择与使用本文描述的产品和服务的责任。除非以书面形式明确地认可，复旦微电子的产品不推荐、不授权、不担保用于包括军事、航空、航天、救生及生命维持系统在内的，由于失效或故障可能导致人身伤亡、严重的财产或环境损失的产品或系统中。

今后日常的产品更新会在适当的时候发布，恕不另行通知。 在购买本资料所记载的产品时，请预先向复旦微电子在当地的销售办事处确认最新信息，并请您通过各种方式关注复旦微电子公布的信息，包括复旦微电子的网站(http://www.fmsh.com/)。

如果您需要了解有关本资料所记载的信息或产品的详情，请与上海复旦微电子集团股份有限公司在当地的销售办事处联系。

商 标

目　　录

图 目 录

表 目 录

1 说明

1.1 FLASH 与 EEPROM 之间的主要区别

电可擦可编程只读存储器（EEPROM）是许多嵌入式应用程序的关键组成部分，这些应用程序要求非易失性存储数据在运行的时候一个字或者一个字节的更新。另一方面，微控制器在那些系统中每次使用的时候更多的是基于 FLASH。为了消除组件，节省硅的空间和减少系统花费可用 FLASH 来代替 EEPROM 去同步代码和存储数据。

不像 FLASH，外部 EEPROM 在重写数据的时候不需要一个擦除动作去挪出空间。所以在用 FLASH 模拟 EEPROM 时需要一些特殊的软件技术进行管理。很显然仿真软件方案取决于许多因素，包括 EEPROM 的可靠性，使用的 FLASH 的构架，还有产品的需求。

FLASH 和外部串口 EEPROM 的主要区别在表 1 中概括起来了。

特性	外部 EEPROM	Flash memory 模仿 EEPROM
写时间	— 几个ms — 随机字节：5 到10ms — 页：每个字100us（每页5 到10ms）	字程序时间：20us
擦时间	N/A	页/块擦除时间：20ms
写方法	— 只要开始，便不再依靠CPU — 仅仅需要适当的供应	一旦开始，便依靠CPU：一个CPU复位将会停止写动作，即使电源停留在规格内
读访问	— 串口：100us — 随机字：92us — 页：22.5us 每个字节	— 并口：100ns — 每个字：非常少的CPU 周期 — 访问时间：35ns
写/擦周期	— 从 10 千周到 1000 千周	— 从10千周到100千周（片上FLASH页的使用相当于增加了写周期的次数）

表 1表 1 –1 FLASH 和 EEPROM 之间的主要区别

1.1.1 写访问时间的区别

FLASH有一个很短的写访问时间，相比较与外部串口EEPROM，关键的参数可以更快的被存储到模拟EEPROM 中去，然后改善数据存储。

1.1.2 写方法的区别

外部EEPROM 和FLASH模拟EEPROM 在应用方面的一个主要区别就是写方法。

外部独立EEPROM：一旦被CPU 启动，一个字的写动作便不能被CPU 重启给打断。只有断电可以打断写过程，所以在独立EEPROM 内，适当大小的退偶电容可以获得完整的写动作过程。

使用FLASH模拟EEPROM：一旦被CPU启动，一个字的写动作可以被断电和CPU重启打断。系统设计者应该分析这个差异，明白他们应用程序上的可能影响并且决定一个合适的处理方法。

1.1.3 擦除时间的区别

擦除时间的区别是独立EEPROM 和使用FLASH模拟EEPROM 之间的另一个主要区别。不像FLASH，在写入它之前，EEPROM 不需要擦除动作去挪出空间，这就意味着一些形式的软件管理是需要存储数据在FLASH中的。另外，当FLASH进行擦除块操作时，掉电和一些其他事件可能打断擦除操作（例如重启），设计FLASH管理软件的时候这点应该考虑到。设计一个完整的FLASH管理软件，对于FLASH擦除过程的深入理解是必不可少的。

2 模拟 EEPROM 的实现

2.1 原理

考虑到FLASH的局限性和产品需求，模拟EEPROM有多种多样的执行方式。下面详细介绍的方法需要至少两页FLASH，对于被分配的相同大小的非易失数据。一页是初始化擦写，并且支持一个字节一个字节的编程；另一页是当前页需要被擦除的时候用来准备接管的。每页占用了第一个16位半字(32位一个字)的头字段表明了页状态。

头字段的信息包含了每页的基地址和这页的状态。

每页有三种可能的状态：

ERASED：页是空的。

RECEIVE_DATA：正在从其他满的页接受数据。

VALID_PAGE：这页包含有效数据并且这个状态时不会改变的，直到所有的有效数据全部被转移到擦除页。图1 展示了页与页之间的状态是如何改变的。

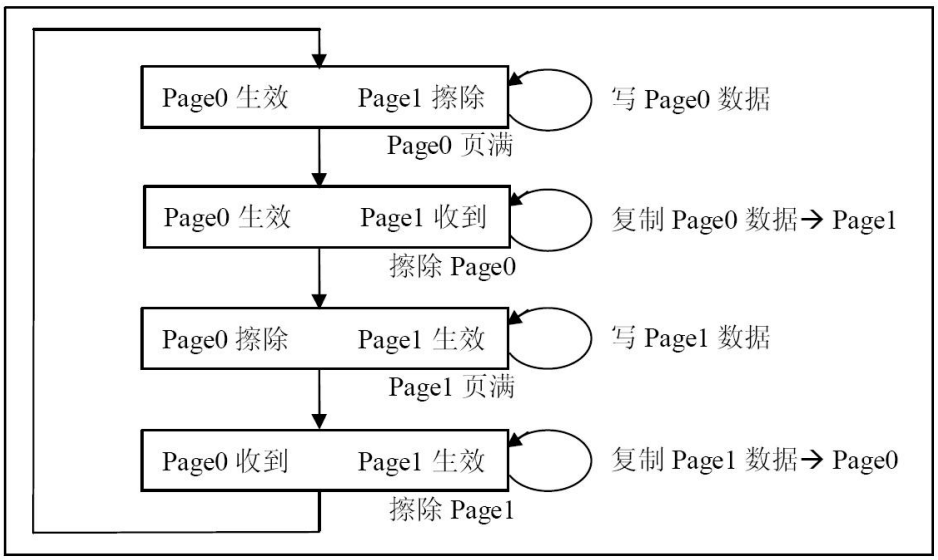


图 2-1Page0 和 Page1 之间的页状态切换

通常，在使用这个方式的时候，使用者预先不知道变量更新的频率。
这篇文章中描述软件和编译器使用两页闪存去模拟EEPROM。

为了后续检索和更新，每一个变量元素都被定义为一个虚拟地址和一个存储在闪存中的值（在编译器软件中，虚拟地址和数据都是16 位长）。当数据被修改的时候，与早先虚拟地址有关的要修改的数据被存储在一个新的闪存位置。数据检索返回修改的数据在闪存最后的位置。

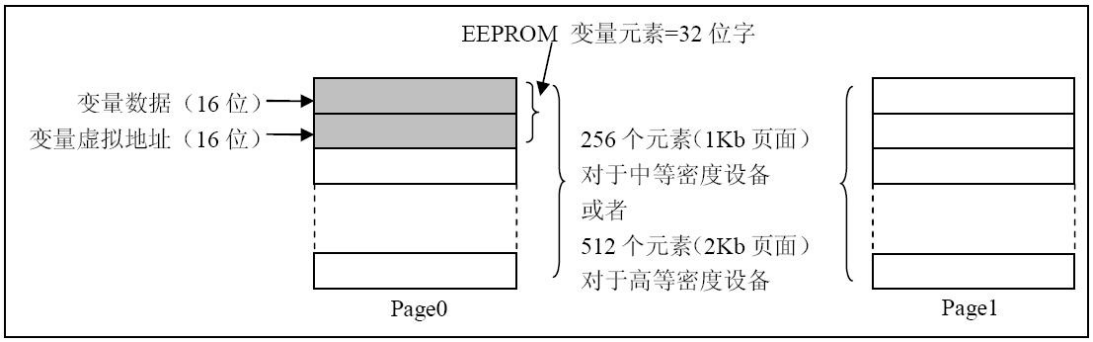


图 2-2 EEPROM 变量格式化

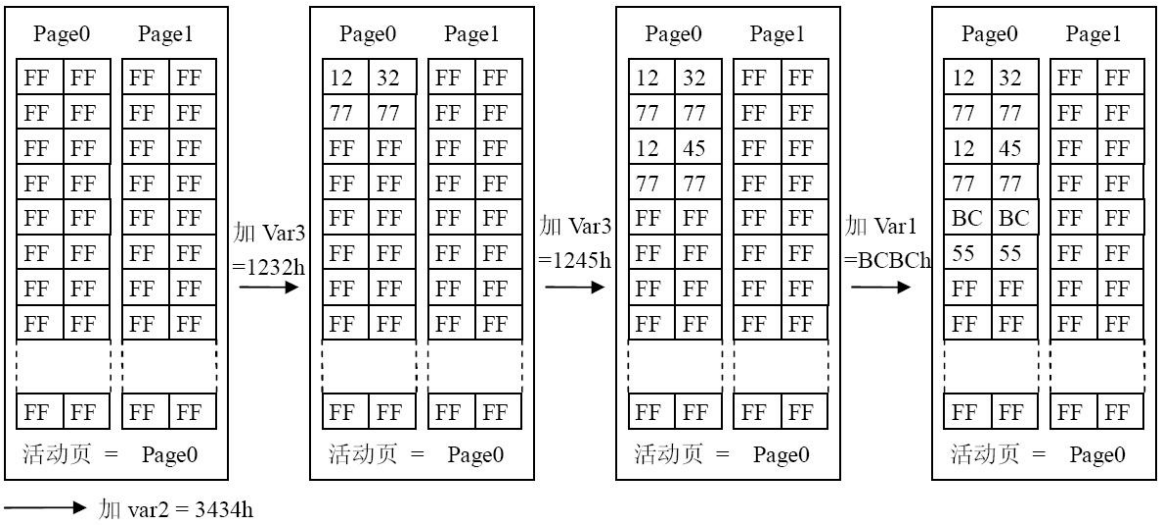
2.2 应用举例

下面的例子展示了三个EEPROM 变量（Var1，Var2 和Var3）同下面虚拟地址的软件管理：

Var1：5555h

Var2：6666h

Var3：7777h



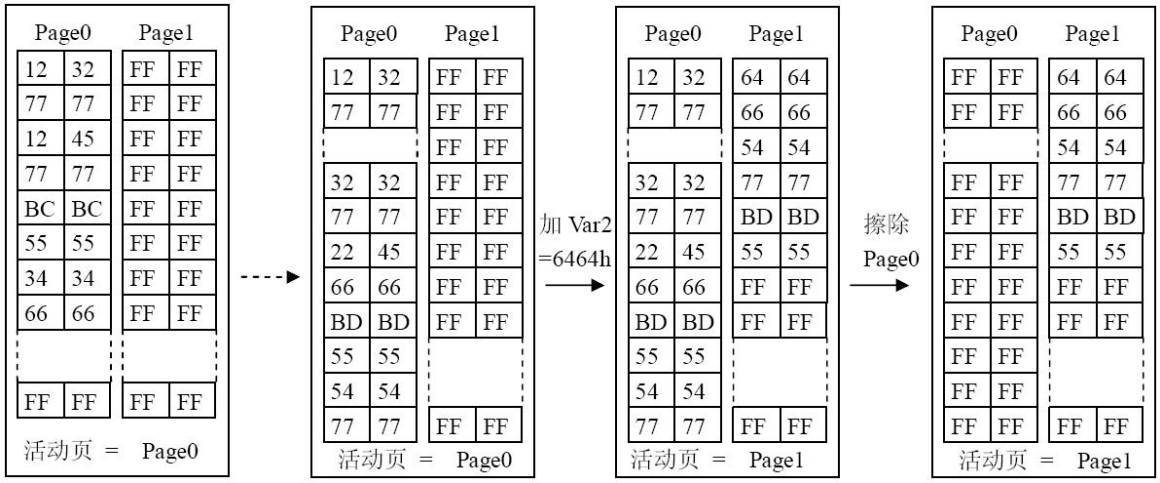


图 2-3 数据更新流程

3 软件描述

这节描述了FLASH模拟EEPROM 驱动程序的实现。一个演示程序的例子也会被提供，用来论证和测试EEPROM 模拟驱动程序，这里使用的三个变量Var1，Var2和Var3是表格VirtAddVarTab 中定义的，VirtAddVarTab 表格在软件 main.c 文件里面声明的。

除了闪存的库文件，这个工程包含了三个源文件：

- **eeprom.c**：它包含了下面工程常规用到的c 代码：

  EE_Init()
  EE_Format()
  EE_FindValidPage()
  EE_VerifyPageFullWriteVariable()
  EE_ReadVariable()
  EE_PageTransfer()
  EE_WriteVariable()

- **eeprom.h**：它包含了常规的原型和一些声明。

- **main.c**：这个应用程序是一个使用上面描述的常规的例子，为的是写入和读取EEPROM。

用户API 定义函数的设置在eeprom.c 文件内，然后用于模拟EEPROM，描述如下：

- **EE_Init()**

扇区擦除或转移和数据更新的时候发生掉电，可能会发生扇区头错误。在这种情况下，EE_Init()函数将会尝试修复数据库到一个良好的状态。每一次断电后访问数据库之前，这个函数都会被调用。它不接受任何参数。

- **EE_Format()**

这个函数擦除page0 和page1 并且写有效页头到page0。

- **EE_FindValidPage()**

这个函数读取所有的页数据头并且返回有效页页码。这个返回的参数表明了有效页是否在寻求写入或读取操作(READ_FROM_VALID_PAGE or WRITE_IN_VALID_PAGE)。

- **EE_VerifyPageFullWriteVariable()**

必须更新或者建立一个变量的第一个实例，它使这个写过程生效。它表现为从最后一位开始找到活动页的第一个空位置，并写进去已经传送过的虚拟地址和变量的数据。在活动页满的情况下，返回页满值。这段程序用到了一下参数：

— 虚拟地址：可能是任何三个已经声明的变量的虚拟地址（Var1，Var2 或者Var3）

— 数据：将要存储变量的值

这个函数成功的话返回闪存完成，如果没有足够的内存区更新变量，就返回页满，或者闪存错误代码，用来指示操作失败（擦除或编程）。

- **EE_ReadVariable()**

这个函数返回的是虚拟地址所对应的数据，虚拟地址是作为一个参数被传送的。只有最后更新的被读取。这个函数进入的时候是一个循环，在这个循环内读变量目录知道最后一个。如果变量没有出现，读状态变量将返回1，否则就重置表明变量已经被发现并且变量值被返回到Read_data变量。

- **EE_PageTransfer()**

它转移最近期的数据（上次更新的变量），这些数据从一个满的页转移到一个空的里面。起初，他决定活动页，活动页就是要转移这页的数据的那一页。新页的数据头字段将被定义和写入（新页状态是RECEIVE_DATA，它是在接收数据的过程中被给出的）。当数据传送完毕的时候，新页的数据头就是VALID_PAGE，旧页将被擦除并且它的数据头会变成ERASED。

- **EE_WriteVariable(..)**

当使用者更新变量的应用时，这个函数会被调用。它使用EE_VerifyPageFullWriteVariable() 和EE_PageTransfer() 函数，这些函数前面描述过。

图4 展示了EEPROM 更新数据变量写入的过程。

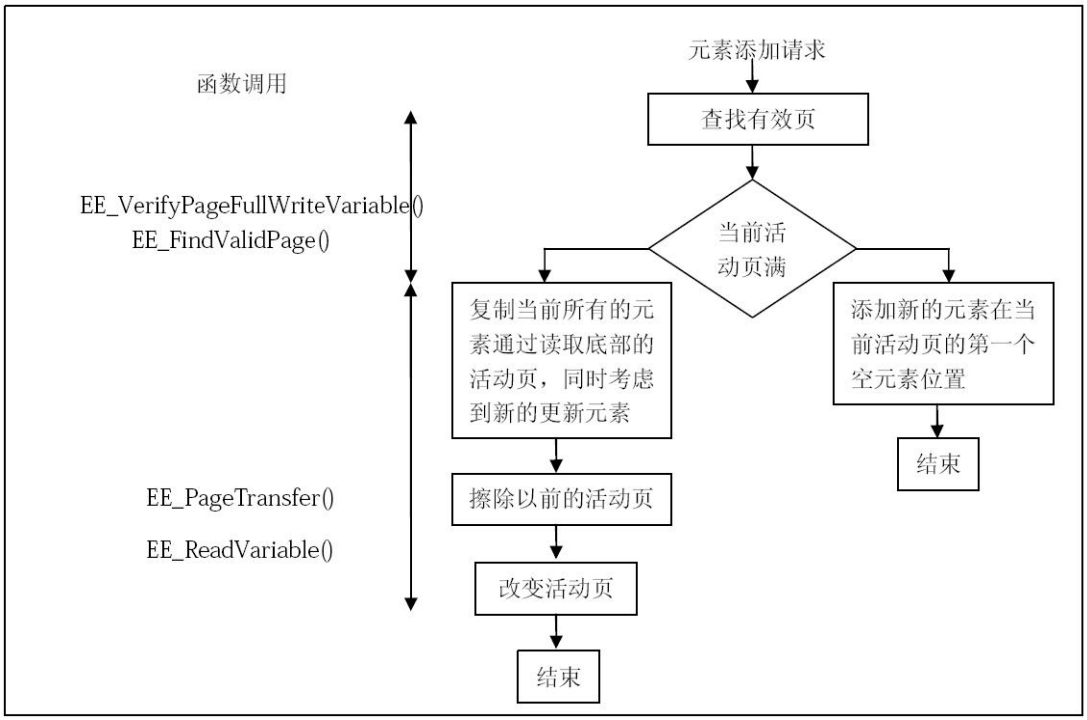


图 3-1 写变量流程图

4 示例程序说明

4.1 PAGE 基本信息

本示例程序使用 PAGE0 与 PAGE1 两页来进行数据交换。用于模拟 EEPROM 的 Flash 起始地址为 0x00010000，PAGE 大小为 2Kbyte。以上 PAGE0、PAGE1 的基本信息均存储于 eeprom.h 文件中。

```
#define PAGE_SIZE  (uint16_t)0x800  /* Page size = 2KByte */

/* EEPROM start address in Flash */
#define EEPROM_START_ADDRESS  ((uint32_t)0x00010000) /* EEPROM emulation start address:
                                                        after 64KByte of used Flash memory */

/* Pages 0 and 1 base and end addresses */
#define PAGE0_BASE_ADDRESS  ((uint32_t)(EEPROM_START_ADDRESS + 0x000))
#define PAGE0_END_ADDRESS  ((uint32_t)(EEPROM_START_ADDRESS + (PAGE_SIZE - 1)))

#define PAGE1_BASE_ADDRESS  ((uint32_t)(EEPROM_START_ADDRESS + PAGE_SIZE))
#define PAGE1_END_ADDRESS  ((uint32_t)(EEPROM_START_ADDRESS + (2 * PAGE_SIZE - 1)))
```

图 4-1 PAGE 配置信息定义

PAGE0 与 PAGE1 的页状态可通过 Page status 来判断，页状态存储于每个 PAGE 前 32 个位。程序在执行过程中会读取每页状态。

```
/* Page status definitions */
#define ERASED  ((uint16_t)0xFFFF)  /* PAGE is empty */
#define RECEIVE_DATA  ((uint16_t)0xEEEE)  /* PAGE is marked to receive data */
#define VALID_PAGE  ((uint16_t)0x0000)  /* PAGE containing valid data */
```

图 4-2 PAGE 状态信息定义

4.2 有效变量的写入过程

变量的虚拟地址以及数据均采用 32 位数据写入的编程方式。本例程中的有效变量个数及虚拟地址分别定义于文件 eeprom.h 及 main.c 文件中。其中有效变量个数 NumbOfVar 为 10 个，虚拟地址分别为：0x1111，0x2222，0x3333，0x4444，0x5555，0x6666，0x7777，0x8888，0x9999，0xAAAA。

```
/* Variables' number */
#define NumbOfVar                    ((uint8_t)0x0A)


/* Virtual address defined by the user: 0xFFFF value is prohibited */
uint16_t VirtAddVarTab[NumbOfVar] = {0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666, 0x7777, 0x8888, 0x9999, 0xAAAA};
```

图 4-3 有效数据个数及地址

例程中，PAGE0 起始地址为 0x00010000，PAGE1 起始地址为 0x00010800。先向 PAGE0 中写入 255 个变量，将 PAGE0 填满。如图 4-4 所示，此时以第 255 个变量为例：0x34 为虚拟地址，0x12 为数据。由于 0x34 没有被定义为有效变量地址，在进行页转换时第 255 个变量会被擦除，而最后虚拟地址为 0x1111、0x2222…x0AAAA 的变量会被转移至 PAGE1。

```
/* --- Store successively many values of the three variables in the EEPROM ---*/
  /* Store 254 values of Variable1 in EEPROM */
  for (VarValue = 0; VarValue < 254; VarValue++)
  {
    if ((VarValue % 254) == 0)
    {
      LED0_TOG();
    }
    EE_WriteVariable(VirtAddVarTab[VarValue % NumbOfVar], VarValue);
  }

  /* Store the 255th value of Variable1 in EEPROM */

    EE_WriteVariable(0x34,0X12);

  /* Store the 256th value of Variable1 in EEPROM */
```

Memory 1

Address: 0x00010000

```
0x00010700: 000000DF 00004444 000000E0 00005555 000000E1 00006666 000000E2 00007777
0x00010720: 000000E3 00008888 000000E4 00009999 000000E5 0000AAAA 000000E6 00001111
0x00010740: 000000E7 00002222 000000E8 00003333 000000E9 00004444 000000EA 00005555
0x00010760: 000000EB 00006666 000000EC 00007777 000000ED 00008888 000000EE 00009999
0x00010780: 000000EF 0000AAAA 000000F0 00001111 000000F1 00002222 000000F2 00003333
0x000107A0: 000000F3 00004444 000000F4 00005555 000000F5 00006666 000000F6 00007777
0x000107C0: 000000F7 00008888 000000F8 00009999 000000F9 0000AAAA 000000FA 00001111
0x000107E0: 000000FB 00002222 000000FC 00003333 000000FD 00004444 00000012 00000034
0x00010800: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010820: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010840: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010860: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010880: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

图 4-4 PAGE0 写满

当写入第 256 个变量时，由于 PAGE0 已满，第 256 个变量被写入 PAGE1 中，PAGE0 中定义的有效变量将被转移至 PAGE1 中（从页尾根据有效地址逐一寻找），随后 PAGE0 被擦除。整个数据的变化过程如图 4-5 所示。

```
/* --- Store successively many values of the three variables in the EEPROM ---*/
  /* Store 254 values of Variable1 in EEPROM */
  for (VarValue = 0; VarValue < 254; VarValue++)
  {
    if ((VarValue % 254) == 0)
    {
      LED0_TOG();
    }
    EE_WriteVariable(VirtAddVarTab[VarValue % NumbOfVar], VarValue);
  }

  /* Store the 255th value of Variable1 in EEPROM */

    EE_WriteVariable(0x34,0X12);

  /* Store the 256th value of Variable1 in EEPROM */

    EE_WriteVariable(0x78,0X56);
```

Memory 1

Address: 0x00010000

```
0x000107C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000107E0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010800: 00000000 FFFFFFFF 00000056 00000078 000000FA 00001111 000000FB 00002222
0x00010820: 000000FC 00003333 000000FD 00004444 000000F4 00005555 000000F5 00006666
0x00010840: 000000F6 00007777 000000F7 00008888 000000F8 00009999 000000F9 0000AAAA
0x00010860: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x00010880: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000108A0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
0x000108C0: FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
```

图 4-5 写入第 256 个元素

4. 3 可擦写性能

当使用 EEPROM 的时候，每一个字节可以在一个有限的次数内被编程或擦除，典型范围是 10000 到 100000，FLASH 中最小的擦除尺寸是一页。模拟 EEPROM 的寿命被最经常写的参数的更新速率所限制。循环的能力与用户想要处理的数据的数量和大小有关。在这个例子中，FLASH 用 32 位的数据编程。每一个变量都对应一个 32 位虚拟地址。也就是说，每一个变量占用 2 个字的存储空间。2Kb（对于高密度器件）乘以闪

存可以忍耐的周期数 10000，模拟闪存一个页面的生命周期内 20000Kb 的数据的能力。因此，在仿真过程中，模拟 EEPROM 提供的两个页面可以存储 40000Kb 的数据。如果超过两个页面被使用，这个数量也会相应的增加。当知道所要储存变量数据的宽度的时候，就可以计算出在模拟 EEPROM 的整个生命周期中能够存储的变量的数量。

通过变量的虚拟地址和数据大小，表 3 给出了一个关于模拟 EEPROM 可以存储变量的数量。

变量大小	2 x 1Kb	2 x 2Kb
8 位变量（8 位虚拟地址）	10000 x（2^{10} - 2）	10 000 × (2^{11} - 2)
16 位变量（16 位虚拟地址）	5000 x（2^{10} - 4）	5000 × (2^{11} - 4)
32 位变量（32 位虚拟地址）	2500 x（2^{10} - 8）	2500 × (2^{11} - 8)

表 4-1 模拟 EEPROM 中储存变量的最大值

4.4 注意事项

FLASH 操作过程中 VDD 禁止出现低于工作电压或发生复位，否则无法保证 FLASH 中的数据。在进行项目开发时，写操作前请检测当前 VDD 电压，防止低电压下操作。

```
uint16_t EE_WriteVariable(uint16_t VirtAddress, uint16_t Data)
{

  uint16_t Status = 0;

  /* 检测到低压后禁止写操作 */
  if (true == SVD_Result_Confirmed(SVD_BELOW_THRESHOLD, 1U))
  {
    return Status;
  }

  /* Write the variable virtual address and value in the EEPROM */
  Status = EE_VerifyPageFullWriteVariable(VirtAddress, Data);

  /* In case the EEPROM active page is full */
  if (Status == PAGE_FULL)
  {
    /* Perform Page transfer */
    Status = EE_PageTransfer(VirtAddress, Data);
  }

  /* Return last operation status */
  return Status;
}
```

图 4-6 SVD 电压监测示例

版本信息

版本号	发布日期	更改说明
1.2.1.2	2022.11	首次发布
1.3.1.3	2023.03	更新内容

上海复旦微电子集团股份有限公司销售及服务网点

上海复旦微电子集团股份有限公司
地址：上海市国泰路 127 号 4 号楼
邮编：200433
电话：(86-021) 6565 5050
传真：(86-021) 6565 9115

上海复旦微电子（香港）股份有限公司
地址：香港九龙尖沙咀东嘉连威老道 98 号东海商业中心 5 楼 506 室
电话：(852) 2116 3288　2116 3338
传真：(852) 2116 0882

北京办事处
地址：北京市东城区东直门北小街青龙胡同 1 号歌华大厦 B 座 423 室
邮编：100007
电话：（86-10）8418 6608
传真：（86-10）8418 6211

深圳办事处
地址：深圳市华强北路 4002 号圣廷苑酒店世纪楼 1301 室
邮编：518028
电话：(86-0755) 8335 0911　8335 1011　8335 2011 8335 0611
传真：(86-0755) 8335 9011

台湾办事处
地址：台北市 114 内湖区内湖路一段 252 号 12 楼 1225 室
电话：(886-2) 7721 1889
传真：(886-2) 7722 3888

新加坡办事处
地址：237, Alexandra Road, #07-01, The Alexcier, Singapore 159929
电话：(65) 6472 3688
传真：(65) 6472 3669

北美办事处
地址：2490 W. Ray Road Suite#2 Chandler, AZ 85224 USA
电话：(480) 857-6500 ext 18
公司网址：http://www.fmsh.com/