Application Note

EEPROM 仿真 A 型设计

摘要

本应用手册介绍了 A 型 EEPROM 仿真库。本文档描述了 A 型 EEPROM 仿真的结构和行为。然后，介绍了相关 API 的使用。此外，还向用户提供了相关代码。用户可以调用应用程序中的函数来写入、读取和修改数据。

内容

商标

1 引言

许多应用程序都需要将数据存储在非易失性存储器中，这样，即使在系统再次上电后也可以重复使用或修改应用程序。EEPROM 是专为此类应用程序设计的。尽管 MSPM0 MCU 没有内部 EEPROM，但 MSPM0 内部闪存支持 EEPROM 仿真。与使用外部串行 EEPROM 相比，使用内部闪存的 EEPROM 仿真可节省引脚用量和成本。

不同应用程序所需的存储结构不尽相同。本文中描述的 A 型解决方案适用于存储大“块”数据。如果应用程序需要存储少量“变量”数据，则可以参考 B 型解决方案。

EEPROM 与片上闪存的区别

EEPROM 可以多次擦除和写入存储器的单个字节，即使系统断电，已编程的位置也能长时间保留数据。

闪存的密度高于 EEPROM，因此可以在芯片上实现更大的存储器阵列（扇区）。闪存擦除/写入周期是通过对各个存储单元施加时控电压来执行的。在擦除时，每个单元（位）读取逻辑值 1。因此，每个闪存位置在擦除后的读数为 0xFFFF。通过编程可以将存储单元更改为逻辑 0。可以重写任何字，将位从逻辑 1 更改为逻辑 0；但反过来则不行。此外，闪存有一项限制是必须按区域擦除存储器。对于 MSPM0 MCU，擦除分辨率是大小为 1k 字节的“扇区”。

EEPROM 与闪存的一个主要区别是耐写次数。闪存耐写次数通常为 10000 次，远低于实际 EEPROM。EEPROM 仿真是一种基于闪存的软件，旨在提供可满足应用程序需求的等效耐写次数。它还会仿真 EEPROM 的行为，从而简化读写数据的操作。

典型的仿真方案涉及使用闪存的一部分并将其划分成多个区域。这些区域交替用于存储数据。由于闪存需要按块进行擦除，因此必须保留完整的闪存扇区用于 EEPROM 仿真。为了实现磨损均衡，至少使用两个扇区。

2 实现

原理

在本应用手册所述的实现方案中，扇区根据虚拟 EEPROM 的大小分为“记录”区域。每条记录都包含标头和数据。标头显示记录的状态。记录的其余部分（总记录大小减去标头的 8 字节大小）用于存储用户数据。一个扇区中的记录数为（扇区大小/记录大小）。对于 128 字节记录大小，一个扇区中有 8 条记录。图 2-1 显示了 EEPROM 仿真的结构。

所有这些记录都用于存储同一虚拟 EEPROM 的数据。当尝试修改虚拟 EEPROM 的数据时，实际上会创建一条新记录，而不是在原来记录上进行修改。闪存中记录之间的差异体现在数据是新数据还是旧数据。换句话说，它们是同一数据的不同版本。它更像是 RAM 中存储器区域的备份。

共有三个用户可配置的参数，可根据应用程序要求在 eeprom_emulation_type_a.h 中进行配置。这些参数会影响空间利用和耐写次数，稍后将对此进行分析。

- 记录大小：64 字节、128 字节或 256 字节
- 使用的扇区数：至少 2 个
- 扇区地址

EEPROM 仿真的基本行为如图 2-2 所示。当执行写入操作时，用户的数据将作为新记录存储到闪存中。执行读取操作时，将读取最新记录。仅当扇区已满时才执行擦除操作。

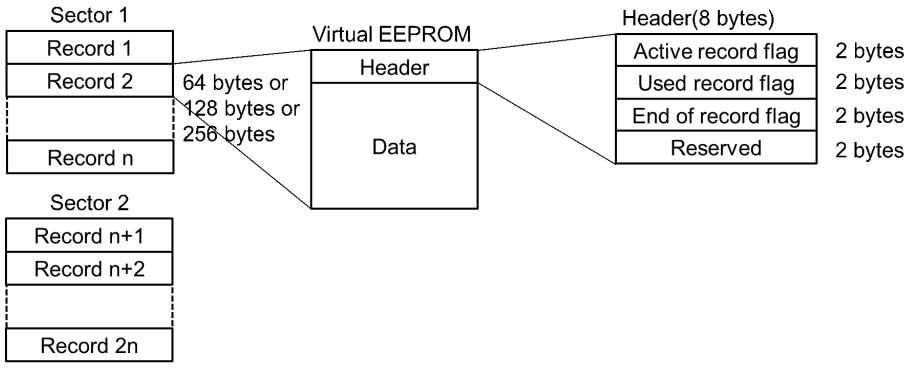

图 2-1. EEPROM 仿真的结构

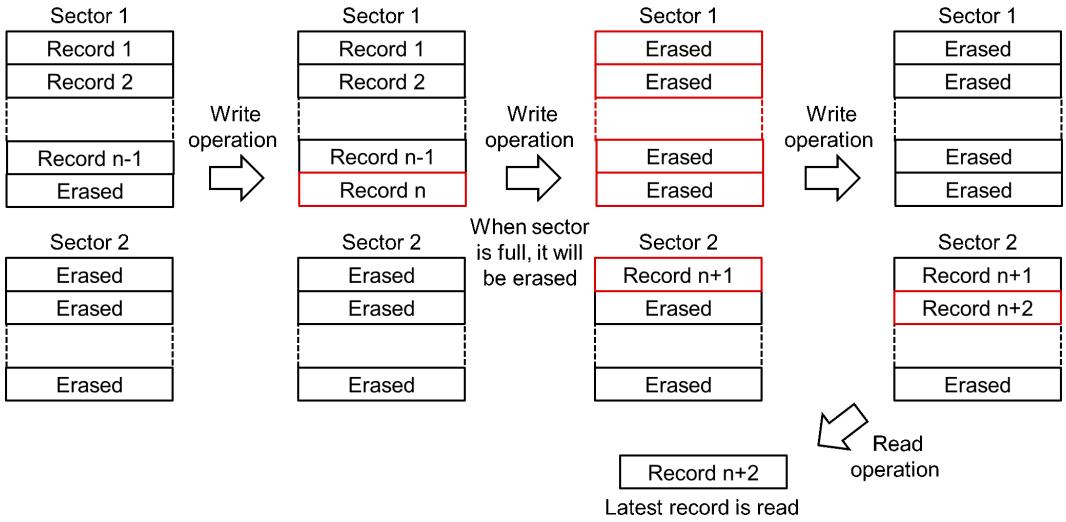

图 2-2. EEPROM 仿真的基本行为

标头

标头用于管理记录。通过检查单个记录的标头，可以确定该记录的状态。通过检查所有记录的标头，可以找到最新记录，并检查 EEPROM 仿真的格式。

每条记录都有一个可显示其状态的标头。标头设置为 8 个字节，具有 3 个标志。根据标志的不同，总共有四种记录状态。标志与记录状态之间的关系如下所示。

表 2-1. 标志与记录状态之间的关系

记录状态	活动记录标志	已用记录标志	记录结束标志
Erased	0xFFFF	0xFFFF	0xFFFF
Recording	0x0000	0xFFFF	0xFFFF
Active (latest)	0x0000	0xFFFF	0x0000
Used (Not latest)	0x0000	0x0000	0x0000

所有标志都会首先被擦除。写入新记录时，首先会设置活动记录标志并将状态更改为“Recording”。然后将数据写入记录。仅当数据完全写入时，才会设置记录结束标志。它监测写入操作的完成。如果系统在写入数据时断电，恢复时将检测到记录状态为“Recording”。

出现新的活动记录后，旧的活动记录将通过设置已用的记录标志来更改为“Used”。图 2-3 显示了执行写入操作时的记录状态变化。可以看到始终有一条活动记录，这有助于从断电状态中恢复。

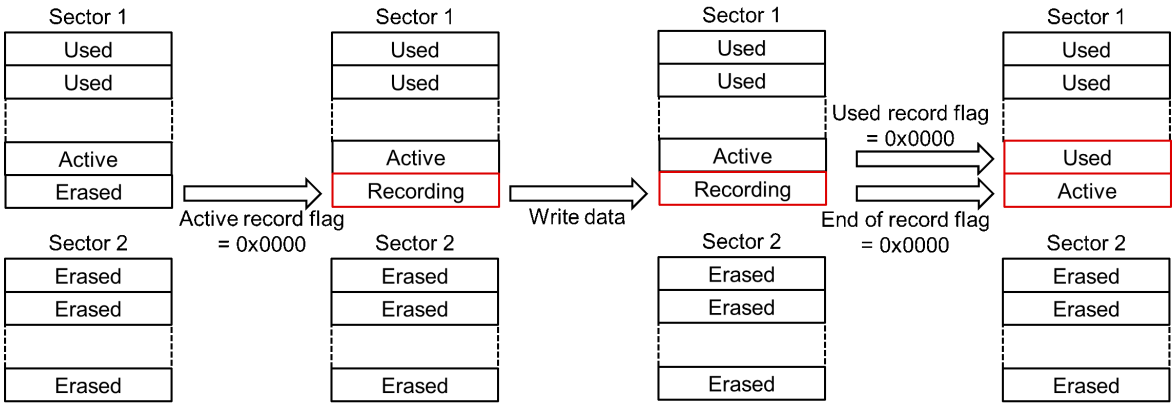

图 2-3. 执行写入操作时记录状态如何变化

3 软件说明

该软件提供基本的 EEPROM 功能。至少要使用 2 个扇区来仿真 EEPROM。如上所述，这些扇区划分为多条记录，每条记录均包含标头，用于确定数据的有效性。为了减少闪存操作的次数，RAM 中与虚拟 EEPROM 大小相同的缓冲区用于复制活动记录的数据。此外，4 个全局变量用于跟踪活动记录，3 个全局变量用于表示标志。

软件功能和流程

用户总共仅直接调用三个函数。

- EEPROM_TypeA_init
- EEPROM_TypeA_writeData
- EEPROM_TypeA_eraseLastSector

图 3-1 中显示了简要软件流程。器件应首先执行初始化代码。通过调用 EEPROM_TypeA_init，它会搜索活动记录并检查闪存的格式。如果存在活动记录，则活动的数据将复制到 RAM 中的缓冲区。如果格式不正确，将修复格式。初始化后，将有一个用于 EEPROM 仿真且正确格式化的闪存区域，几个跟踪活动记录的全局变量以及一个复制活动记录数据的 RAM 缓冲区。

在应用程序中，用户可以直接读取或编辑 RAM 中的缓冲区。仅当调用 EEPROM_TypeA_writeData 时，缓冲区才会作为新的活动记录存储到闪存中。EEPROM_TypeA_writeData 还将在扇区已满时设置擦除标志。在流程中，设置擦除标志后会立即调用 EEPROM_TypeA_eraseLastSector。根据应用程序，用户可以选择适当的时间点来擦除。

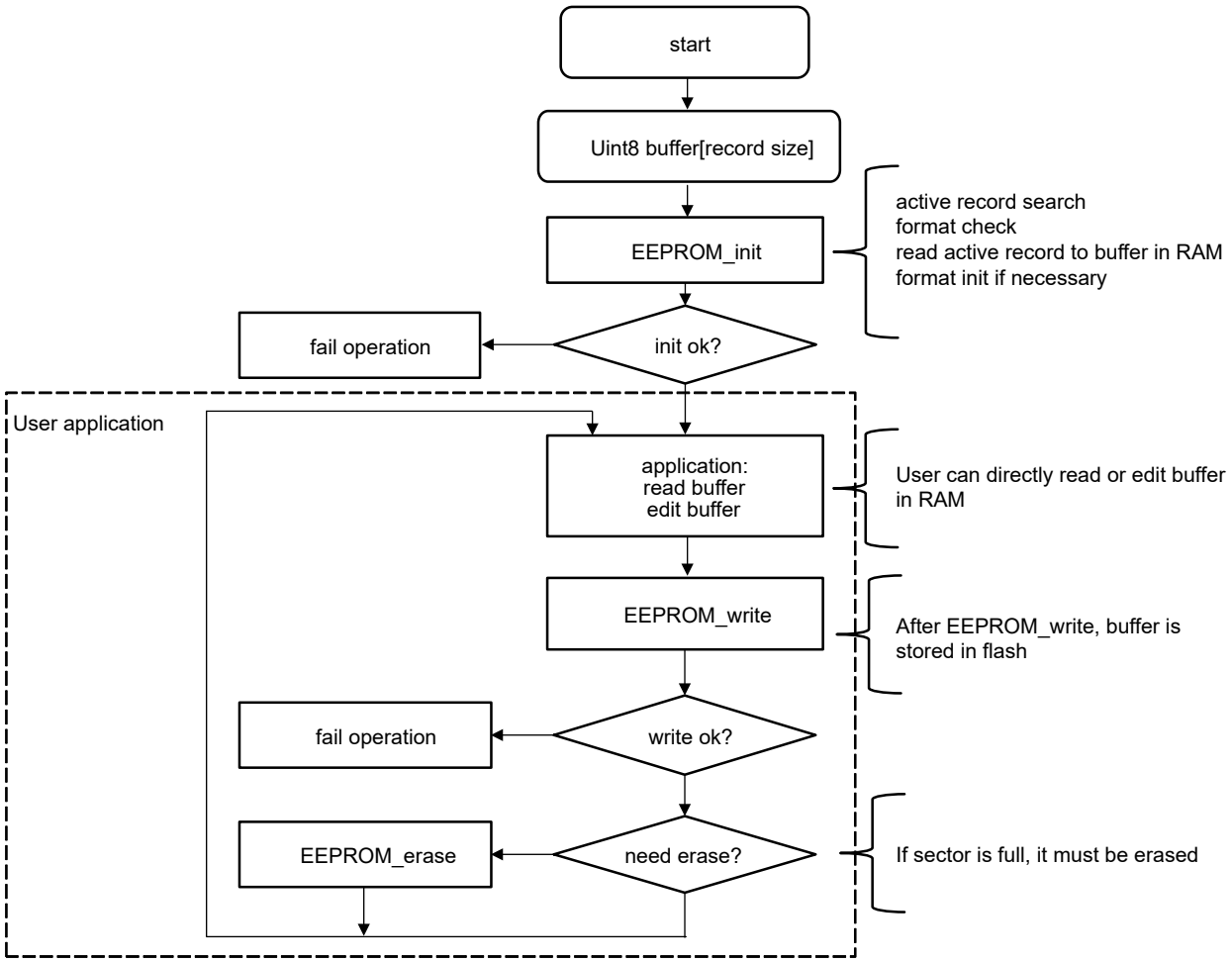

图 3-1. 简要软件流程

EEPROM 函数

实现此功能需要六个函数。除上述三个函数外，其余三个函数主要由 EEPROM_TypeA_init 调用。

- EEPROM_TypeA_init
- EEPROM_TypeA_writeData
- EEPROM_TypeA_eraseLastSector
- EEPROM_TypeA_readData
- EEPROM_TypeA_searchCheck
- EEPROM_TypeA_repairFormat

此外，7 个全局变量用于记录 EEPROM 仿真的状态。有 4 个全局变量用于跟踪活动记录。

- uint32_t gActiveRecordAddress；
- uint32_t gNextRecordAddress；
- uint16_t gActiveRecordNum；
- uint16_t gActiveSectorNum；

gActiveRecordAddress 和 gNextRecordAddress 用于存储活动记录相关地址。

gActiveRecordNum 和 gActiveSectorNum 用于跟踪活动记录的位置。

有 3 个全局变量用于表示标志。

- bool gEEPROMTypeASearchFlag；
- bool gEEPROMTypeAEraseFlag；
- bool gEEPROMTypeAFormatErrorFlag；

当存在活动记录时设置 gEEPROMTypeASearchFlag。

当扇区已满且需要擦除时设置 gEEPROMTypeAEraseFlag。

当发现格式错误时设置 gEEPROMTypeAFormatErrorFlag。

EEPROM_TypeA_init

该函数用于初始化 EEPROM 仿真。通过此函数，可恢复用户数据并跟踪闪存中的活动记录。此函数包含以下功能：

- 活动记录搜索和格式检查
- 将活动记录读取到 RAM 中的缓冲区
- 如有必要，通过调用 EEPROM_TypeA_repairFormat 进行格式修复

图 3-2 中显示了软件流程。首先，它通过调用 EEPROM_TypeA_searchCheck 搜索活动记录并检查格式。根据 EEPROM_TypeA_searchCheck 设置的标志，可以确定活动记录是否存在或格式是否正确。如果存在活动记录，则通过调用 EEPROM_TypeA_ReadData 将活动记录的数据复制到 RAM 中的缓冲区，并将指针设置为活动记录。如果不存在活动记录，则擦除所有扇区并将指针设置为第一个扇区的开头。

如果格式不正确，将通过调用 EEPROM_TypeA_repairFormat 来修复格式。格式修复后，将擦除所有扇区并恢复活动记录。图 3-3 和图 3-4 分别显示了 EEPROM_TypeA_init 前后的闪存区域比较。

该函数的输入是 RAM 中缓冲区的地址。该函数的输出为操作状态。此外，函数中更新了全部 7 个全局变量。

- 输入：&buffer[0]
- 输出：操作状态

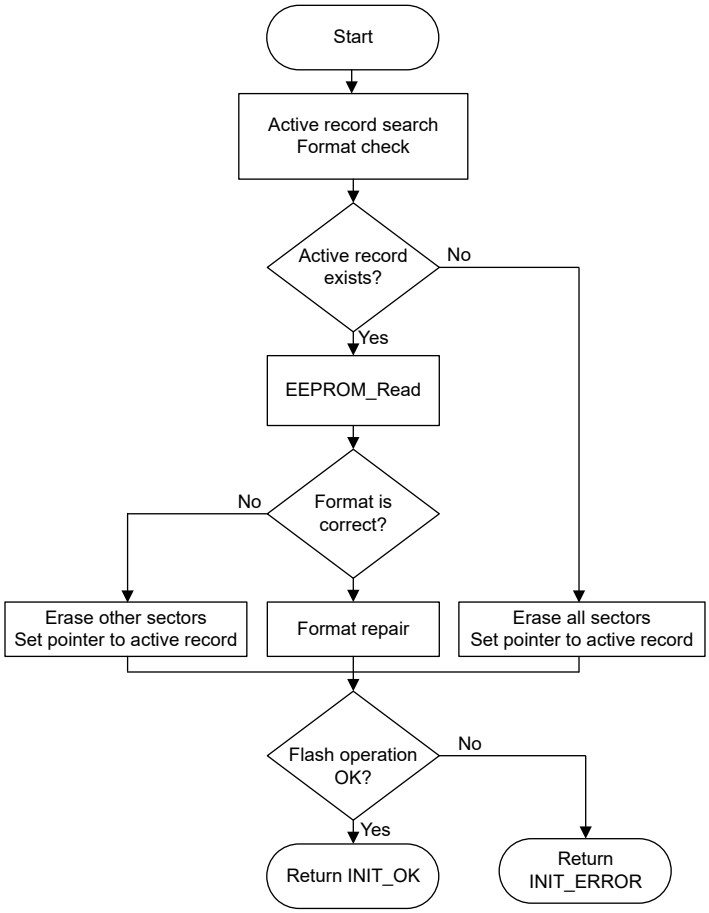

图 3-2. EEPROM_TypeA_init 的软件流程

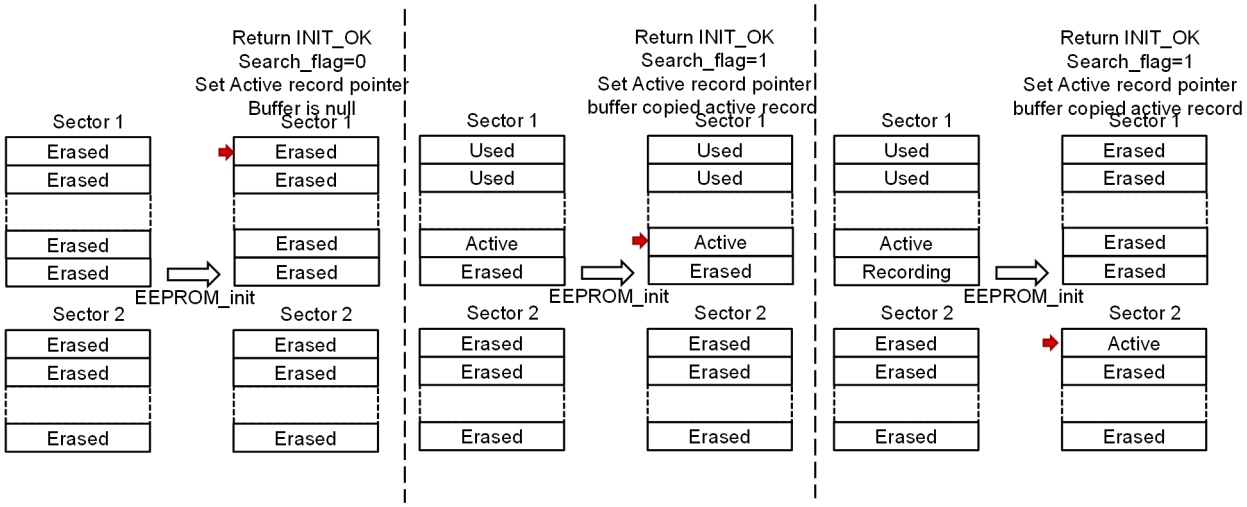

图 3-3. 用于正常场景的 EEPROM_TypeA_init

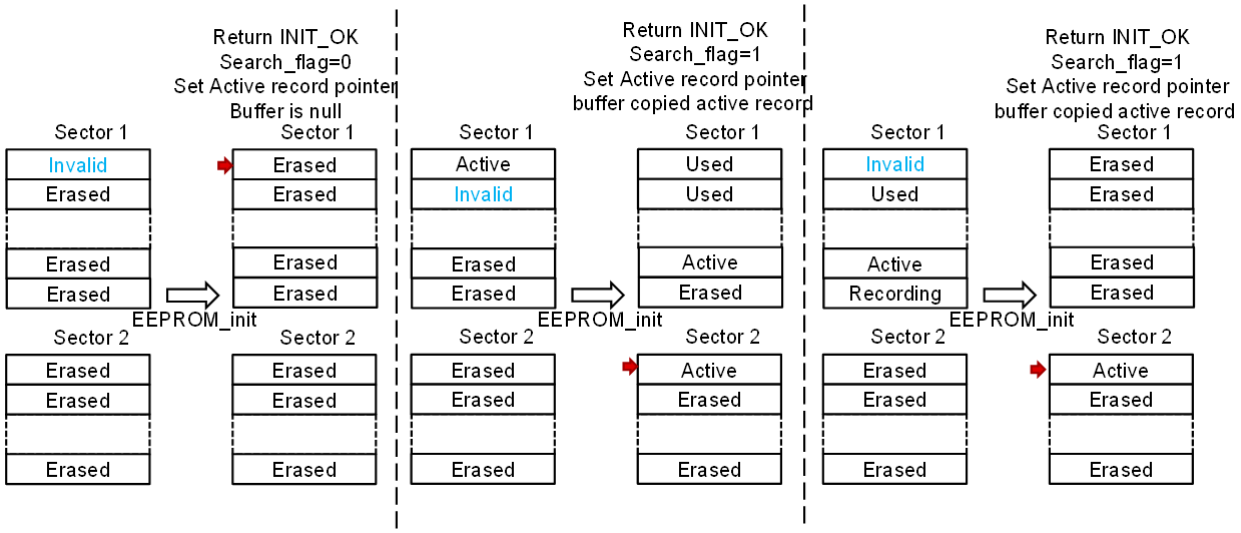

图 3-4. 用于非正常场景的 EEPROM_TypeA_init

EEPROM_TypeA_writeData

EEPROM_TypeA_writeData 用于将数据从 RAM 中的缓冲区存储到闪存。通过该函数，可以将新的活动记录添加到闪存中。

用户可以直接读取或编辑 RAM 中的缓冲区。但是，只有在调用此函数后，缓冲区的数据才会复制到新的活动记录中。换句话说，闪存区域用于记录/备份缓冲区。断电时，RAM 中的所有数据都会丢失，并且活动记录用于恢复数据。

图 3-5 中显示了软件流程。首先，它会检查下一条记录是否擦除。然后开始将数据存储到下一条记录中。该过程如图 2-3 所示：

1. 将记录的标头设置为“Recording”
2. 将数据从 RAM 中的缓冲区复制到记录中
3. 将记录的标头设置为“Active”
4. 如果存在最后一条活动记录，将其标头设置为“Used”

最后，检查该扇区是否已满。如果是，将设置 gEEPROMTypeAEraseFlag。此外，有关活动记录的全局变量也会更新。EEPROM_TypeA_writeData 前后的闪存区域比较如图 3-6 所示。

该函数的输入是 RAM 中缓冲区的地址。该函数的输出为操作状态。

- 输入：&buffer[0]
- 输出：操作状态

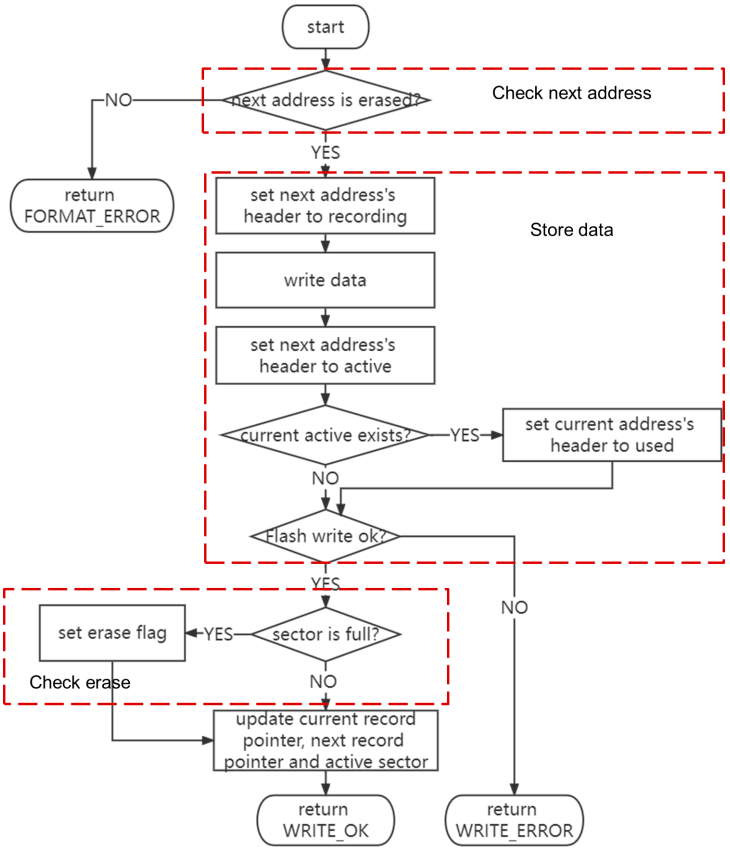

图 3-5. EEPROM_TypeA_writeData 的软件流程

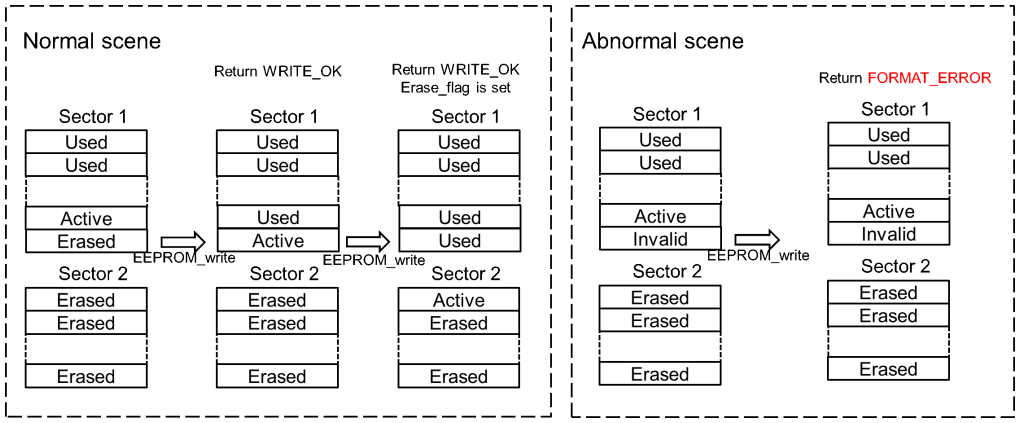

图 3-6. EEPROM_TypeA_writeData 的软件流程

EEPROM_TypeA_eraseLastSector

EEPROM_TypeA_eraseLastSector 用于在闪存扇区已满时擦除该扇区。调用 EEPROM_TypeA_writeData 时，如果扇区已满，将设置 gEEPROMTypeAEraseFlag。建议在设置 gEEPROMTypeAEraseFlag 后立即调用 EEPROM_TypeA_eraseLastSector，如图 3-1 所示。但是，用户可以通过修改简要软件流程来更改要擦除扇区的时间点。

该函数的输出为操作状态。

- 输入：void
- 输出：操作状态

EEPROM_TypeA_readData

EEPROM_TypeA_readData 用于将数据从闪存中的活动记录复制到 RAM 中的缓冲区。该函数在 EEPROM_TypeA_init 中调用。虽然用户可以直接读取或编辑 RAM 中的缓冲区，但用户通常不会直接使用 EEPROM_TypeA_readData。

该函数的输入是 RAM 中缓冲区的地址。

- 输入：&buffer[0]
- 输出：void

EEPROM_TypeA_searchCheck

EEPROM_TypeA_searchCheck 用于搜索活动记录并检查格式。该函数遍历所有记录的标头。如果找到活动记录，将设置搜索标志并更新活动记录相关的全局变量。如果存在“Recording”标头或无效标头，或存在其他格式错误情况，将设置错误标志。该函数在 EEPROM_TypeA_repairFormat 中调用。图 3-7 中显示了软件流程。

该函数通过设置全局变量来输出结果，因此函数的输入和输出都是 void。

- 输入：void
- 输出：void

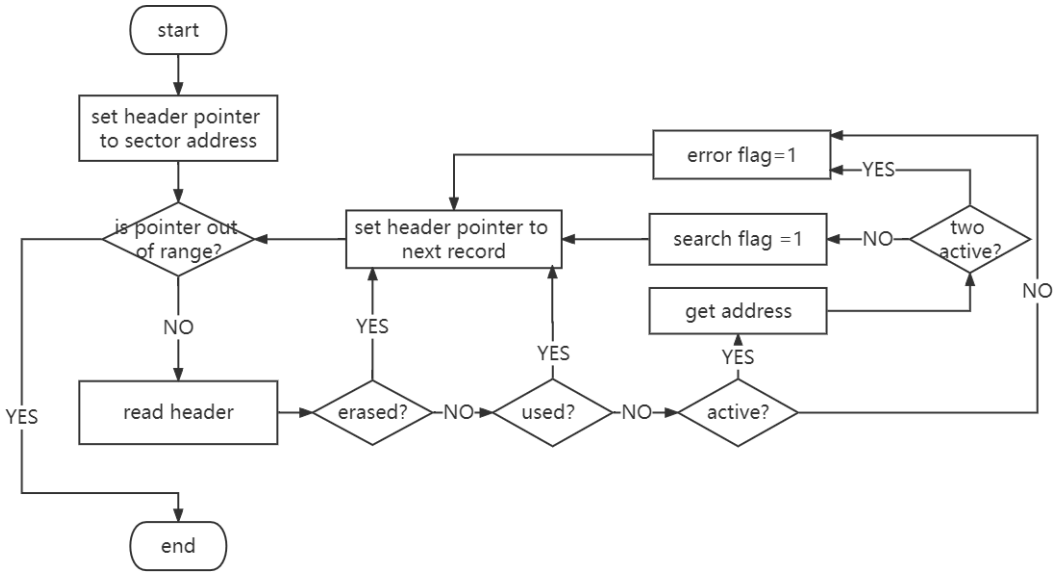

图 3-7. EEPROM_TypeA_searchCheck 的软件流程

EEPROM_TypeA_repairFormat

EEPROM_TypeA_repairFormat 用于修复格式。在调用函数之前，活动记录应已读取到 RAM 中的缓冲区。通过此函数，可擦除所有扇区并将数据从 RAM 中的缓冲区复制到闪存中的新记录中。该函数在 EEPROM_TypeA_repairFormat 中调用。图 3-8 中显示了软件流程。

该函数的输入是 RAM 中缓冲区的地址。该函数的输出为操作状态。

- 输入：&buffer[0]
- 输出：操作状态

图 3-8. EEPROM_TypeA_repairFormat 的软件流程

应用程序集成

需要此功能的应用程序必须包含为 MSPM0 MCU 提供的 eeprom_emulation_type_a.c 和 eeprom_emulation_type_a.h 文件。另外，还需包含面向特定器件的闪存 API。例如，对于 MSPM0G3507/MSPM0L1306，需要包含以下文件：

- eeprom_emulation_type_a.c
- eeprom_emulation_type_a.h
- ti_msp_dl_config.c
- ti_msp_dl_config.h

支持 MSPM0 产品的 SDK 中已包含 EEPROM 仿真库。

该 SDK 中还包含所有闪存 API 文件。

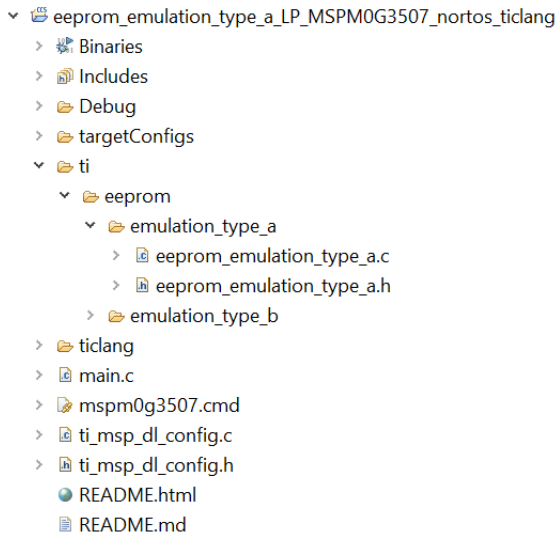

图 3-9. 软件所需的文件

EEPROM 仿真存储器占用空间

表 3-2 详细说明了 EEPROM 仿真驱动程序在闪存大小和 RAM 大小方面的占用空间。下表和下图是在使用 Code Composer Studio (版本：11.2.0.00007) 且优化级别为 2 的条件下确定的。

表 3-1. EEPROM 仿真的结构

机制	所需的最小代码大小 (字节)	
	闪存	SRAM
具有 64 字节记录大小的 EEPROM 仿真 A 型	2616	71
具有 128 字节记录大小的 EEPROM 仿真 A 型	2616	135
具有 256 字节记录大小的 EEPROM 仿真 A 型	2616	273

EEPROM 仿真时序

本部分介绍了与基于两个 1KB 闪存扇区的 EEPROM 仿真驱动程序相关的时序参数。

所有时序测量均在以下条件下执行：

- MSPM0G3507
- 系统时钟为 32MHz
- 从闪存执行
- 室温

这些函数通过以下参数进行测试：

- 记录大小：128 字节
- 使用的扇区数：2
- 扇区地址：0x00001000

表 3-2. EEPROM 仿真操作时序

操作	典型值 (µs)
格式正确的 EEPROM_TypeA_init	80
格式修复后的 EEPROM_TypeA_init	4467
EEPROM_TypeA_writeData	848
EEPROM_TypeA_eraseLastSector	3612

4 应用程序方面

本节介绍了 EEPROM 仿真解决方案在应用程序方面的特性，以及如何对其进行配置以满足应用程序需求。

可配置参数的选择

eeprom_emulation_type_a.h. 中有三个用户可配置的参数。可根据应用程序的要求相应地配置这些参数。

- 记录大小：64、128 或 256 字节
- 使用的扇区数：至少 2 个
- 扇区地址

存储器空间利用率

虽然标头在结构中占用 8 个字节的空间，但存储器空间的利用率取决于用户的应用程序。

$$\text{存储器空间利用率} = \frac{\text{用户数据大小}}{\text{记录大小}}$$

$$\text{最大存储器空间利用率} = \frac{\text{记录大小} - \text{标头大小}}{\text{记录大小}}$$

建议在选择适当的记录大小之前评估应用程序所需的虚拟 EEPROM 大小。例如，如果要存储 40 字节的数据，建议选择 64 字节的记录大小。

耐写次数

闪存耐写次数通常为 10000 次，远低于实际 EEPROM。EEPROM 仿真的一个特性是它的耐写次数高于闪存。通过将闪存扇区划分为多条记录并逐一写入数据，闪存扇区无需在每次写入时擦除，只需在写满后擦除。此外，通过使用多个闪存扇区，等效耐写次数将进一步提高。

$$\text{记录数} = \frac{\text{扇区大小}}{\text{记录大小}} \times \text{扇区数}$$

$$\text{等效耐写次数} = \text{记录数} \times \text{闪存耐写次数}$$

例如，如果使用了 2 个 1KB 扇区且记录大小为 128 字节，则等效耐写次数为 160000 次。如果使用了 3 个 1KB 扇区且记录大小为 128 字节，则等效耐写次数为 240000 次。如果使用了 2 个 1KB 扇区且记录大小为 64 字节，则等效耐写次数为 320000 次。

建议用户在选择要使用的适当扇区数之前计算应用程序所需的耐写次数。

对于三个可由用户配置的参数，建议的设计流程如下：

1. 评估应用程序所需的数据大小并选择适当的记录大小。
2. 计算应用程序所需的耐写次数，并选择要使用的适当扇区数。
3. 选择合适的闪存地址

例如，有一个应用程序需要每 10 分钟更新一次 EEPROM 中的数据，数据大小为 40 字节，可确保提供 10 年连续服务。首先，记录大小可以是 64 字节。其次，应用程序所需的耐写次数为 525600 次（10 年 x 365 天 x 24 小时 x 6 次/小时）。需要 4 个扇区，等效耐写次数为 640000 次。

断电恢复

如果在 EEPROM_TypeA_writeData 或 EEPROM_TypeA_eraseLastSector 期间断电，则可能会损坏数据或标头。

为了检测损坏并从中恢复，实现了 EEPROM_TypeA_init。应在上电后立即调用该函数。EEPROM_TypeA_init 会检查所有记录的标头以确认 EEPROM 仿真的数据存储是否正确，并在必要时执行格式修复。

在 EEPROM 仿真的结构中，标头可显示相应记录的状态。一共有 4 种状态。上一节详细介绍了这四种状态之间的变化。

5 参考文献

1. *使用低内存 MSP430™ FRAM MCU 的 EEPROM 仿真*

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265