Reference Guide

TMS320F28E12x 闪存 API 版本 5.00.00.00

摘要

闪存是一种电可擦除/可编程非易失性存储器，可进行多次编程和擦除以简化代码开发。闪存存储器主要可用作内核的程序存储器，其次可用作静态数据存储器。本用户指南介绍了如何使用闪存 API 库对 TMS320F28E12x 器件的片上闪存存储器执行擦除、编程和验证操作。

备注

此闪存 API 文档和相关库与 F28E12x 和 MCPC029x 器件兼容并适用于此类器件。两类器件实现相同的闪存 API 接口，并利用相同的底层库。使用 MCPC029x 的用户应参考此文档，了解闪存 API 实现的详细信息和使用指南。

内容

插图清单

表格清单

商标

1 简介

本参考指南详细介绍了德州仪器 (TI) 的 TMS320F28E12x 闪存 API 库（FAPI_F28E12x_v5.00.00.00 和 FAPI_ROM_F28E12x_v5.00.00.00）函数，可用于对 TMS320F28E12x 器件进行擦除、编程和验证。请注意，闪存 API V5.00.XX.XX 仅能与 TMS320F28E12x 器件一起使用。在 C2000Ware 中提供了 TMS320F28E12x 的闪存 API 库，位置为 C2000Ware_x_xx_xx_xx\libraries\flash_api\f28e12x。

1.1 参考资料

请将本指南与 TMS320F28E12x 器件的 *TMS320F28E12x 实时微控制器数据手册* 和 *TMS320F28E12x 实时微控制器技术参考手册* 结合使用。

1.2 函数清单格式

针对函数、编译器内在函数或宏的条目的通用格式。

对 **function_name()** 的作用的简短描述。

概要

为 **function_name()** 提供原型。

```
<return_type> function_name(
                  <type_1> parameter_1,
                  <type_2> parameter_2,
                  <type_n> parameter_n
                           )
```

参数

```
parameter_1 [in]            Type details of parameter_1
parameter_2 [out]           Type details of parameter_2
parameter_n [in/out]        Type details of parameter_3
```

参数传递分类如下：

- in：表示该函数使用提供的参数中的一个或多个值而不保存任何更改。
- out：表示该函数将一个或多个值保存在提供的参数中。您可以检查保存的值从而找出有关您的应用程序的有用信息。

- in/out：表示函数更改提供的参数中的一个或多个值并保存结果。您可以检查保存的值从而找出有关您的应用程序的有用信息。

说明

描述了该函数。本节还描述了可能适用的任何特殊特性或限制：

- 在某些情况下，函数可能会阻止或阻止所请求的操作
- 函数具有明显的前置条件
- 函数有限制或特殊行为

限制

指定使用该函数时的任何限制。

返回值

指定该函数返回的任何一个或多个值。

另请参见

列出了与该函数相关的其他函数或数据类型。

实现示例

提供了演示函数用法的示例（或对示例的引用）。除了闪存 API 函数之外，此类示例还可以使用 C2000Ware 中提供的 device_support 文件夹或 driverlib 文件夹中的函数来演示在应用程序环境中如何使用给定的闪存 API 函数。

2 TMS320F28E12x 闪存 API 概述

2.1 简介

闪存 API 是一个例程库，当以正确的顺序使用正确的参数调用时，可以对闪存进行擦除、编程或验证。闪存 API 也可用于对 OTP 存储器进行编程。

备注

有关 TMS320F28E12x 闪存和 OTP 存储器映射及闪存等待状态规范，请参阅 *TMS320F28E12x 实时微控制器数据手册*。请注意，本参考指南假定用户已阅读 *TMS320F28E12x 实时微控制器技术参考手册* 中的 TMS320F28E12x *闪存和 OTP 存储器* 章节。

另外，请特别注意该器件上的 Fapi_issueAsyncCommand()、Fapi_setupBankSectorEnable()、Fapi_issueBankEraseCommand() 函数。C2000Ware 中提供的闪存 API 使用示例展示了 TMS320F28E12x 的这些函数的使用方法，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

2.2 API 概述

表 2-1. 初始化函数汇总

API 功能	说明
Fapi_initializeAPI()	为供首次使用或更改频率，对 API 进行初始化

表 2-2. 闪存状态机 (FSM) 函数汇总

API 功能说明	说明
Fapi_setActiveFlashBank()	对闪存包装程序和存储体进行初始化从而执行擦除或编程命令
Fapi_setupBankSectorEnable()	配置扇区的写/擦除保护。
Fapi_issueBankEraseCommand()	针对给定的闪存组地址向闪存状态机发出闪存组擦除命令。

表 2-2. 闪存状态机 (FSM) 函数汇总 （续）

API 功能说明	说明
Fapi_issueAsyncCommandWithAddress()	针对给定地址向 FSM 发出擦除扇区命令
Fapi_issueProgrammingCommand()	设置编程所需的寄存器并向 FSM 发出命令 注意：一次最多可编程 128 位（8 个 16 位字）。
Fapi_issueProgrammingCommandForEccAddress()	将 ECC 地址重新映射到主数据空间，然后调用 Fapi_issueProgrammingCommand() 对 ECC 进行编程
Fapi_issueAutoEcc512ProgrammingCommand()	设置使用 AutoECC 生成模式进行 512 位（32 个 16 位字）编程所需的寄存器并向 FSM 发出命令
Fapi_issueDataAndEcc512ProgrammingCommand()	设置使用用户提供的闪存数据和 ECC 进行 512 位（32 个 16 位字）编程所需的寄存器，并向 FSM 发出命令
Fapi_issueDataOnly512ProgrammingCommand()	设置使用用户提供的闪存数据进行 512 位（32 个 16 位字）编程所需的寄存器，并向 FSM 发出命令
Fapi_issueEccOnly64ProgrammingCommand()	设置使用用户提供的 ECC 数据进行 64 位（4 个 16 位字）ECC 编程所需的寄存器，并向 FSM 发出命令
Fapi_issueAsyncCommand()	向 FSM 发出命令（清除状态）以进行不需要地址的操作
Fapi_checkFsmForReady()	返回闪存状态机 (FSM) 是否处于就绪或繁忙状态
Fapi_getFsmStatus()	从闪存包装程序返回 STATCMD 状态寄存器值

表 2-3. 读取函数汇总

API 功能	说明
Fapi_doBlankCheck()	根据擦除状态验证指定的闪存范围
Fapi_doVerify()	根据提供的值验证指定的闪存范围

备注

Fapi_calculatePsa() 和 Fapi_doPsaVerify() 已弃用。

表 2-4. 信息函数汇总

API 功能	说明
Fapi_getLibraryInfo()	返回特定于 API 库编译版本的信息

表 2-5. 实用程序函数汇总

API 功能	说明
Fapi_flushPipeline()	刷新闪存包装程序中的数据高速缓存
Fapi_calculateEcc()	计算所提供地址和 64 位字的 ECC
Fapi_isAddressEcc()	确定地址是否在 ECC 范围内
Fapi_remapEccAddress()	将 ECC 地址重新映射到相应的主地址
Fapi_calculateFletcherChecksum()	函数计算指定内存范围的 Fletcher 校验和。Fapi_calculateFletcherChecksum() 将在未来的器件中被弃用。

2.3 使用 API

本节介绍各种 API 函数的使用流程。

2.3.1 初始化流程

2.3.1.1 器件上电后

器件首次上电后，必须先调用 *Fapi_initializeAPI()* 函数，然后才能调用任何其他 API 函数（*Fapi_getLibraryInfo()* 函数除外）。此过程根据用户指定的操作系统频率配置闪存包装程序。

2.3.1.2 闪存封装器和闪存组设置

在首次进行闪存操作之前，必须调用 *Fapi_setActiveFlashBank()* 函数。

2.3.1.3 关于系统频率变化

如果在初始调用 *Fapi_initializeAPI()* 函数后更改了系统工作频率，则必须再次调用该函数，然后才能使用任何其他 API 函数（*Fapi_getLibraryInfo()* 函数除外）。该过程会更新 API 内部状态变量。

2.3.2 使用 API 进行构建

2.3.2.1 对象库文件

闪存 API 目标文件以 Arm® 标准 EABI elf 和 COFF 目标格式分发。

备注

编译需要启用“启用对 GCC 扩展的支持”选项。编译器版本 6.4.0 及更高版本默认启用该选项。

2.3.2.2 分布文件

以下 API 文件分布在 C2000Ware\libraries\flash_api\f280013x\ 文件夹中：

- 库文件
  - TMS320F28E12x 闪存 API 嵌入到该器件的引导 ROM 中，该 API 完全属于软件类型。提供的软件库采用 EABI elf (FAPI_F28E12x_v5.00.00.lib) 和 (FAPI_ROM_F28E12x_v5.00.00.lib) 对象格式。为了使应用程序能够擦除或编程闪存/OTP，这两个库文件之一可包含在应用程序中，具体取决于应用程序正在使用的输出对象格式。
- FAPI_F28E12x_v5.00.00.lib – 这是用于 TMS320F28E12x 器件的闪存 API EABI elf 对象格式库（为构建启用了 FPU32 标志）。
- FAPI_ROM_F28E12x_v5.00.00.lib – 这是用于 TMS320F28E12x 器件的闪存 API EABI elf 对象格式库（为构建启用了 FPU32 标志）。请注意，此库包含该器件引导 ROM 中的嵌入式闪存 API 库的符号。
- 未提供 API 库的定点版本。
- 头文件：
  - FlashTech_F28E1xx_C28x.h – TMS320F28E12x 器件的主头文件。该文件设置特定于编译的定义并包括 FlashTech.h 主头文件。
  - hw_flash_command.h – 闪存写入/擦除保护寄存器的定义
- 以下头文件未直接包含在用户代码中，但此处列出了此类文件以供用户参考：
  - FlashTech.h – 该头文件列出了所有公共 API 函数并包括所有其他头文件。
  - Registers.h – 所有寄存器实现通用的定义，包括所选器件类型的相应寄存器头文件。
  - Registers_C28x.h — 包含小端字节序和闪存控制器寄存器结构。
  - Types.h – 包含 API 使用的所有枚举和结构。
  - Constants/F28E1xx.h – F28E12x 器件的常量定义。

2.3.2.3 闪存 API 使用的关键事实

以下是有关 API 使用的一些重要事实：

- 闪存 API 函数的名称以前缀 "Fapi_" 开头。
- 闪存 API 不配置 PLL。用户应用程序根据需要配置 PLL 并将配置的 CPUCLK 值传递给 Fapi_initializeAPI() 函数（该函数的详细信息在本文档后面给出）。请注意，当系统频率小于或等于 20MHz 时，闪存 API 库不支持闪存擦除/编程操作。
- 闪存 API 不会检查 PLL 配置来确认用户输入频率。这由系统集成商决定。TI 建议使用 DCC 模块来检查系统频率。有关实现示例，请参阅 C2000Ware driverlib 时钟配置函数。
- 在调用闪存 API 函数之前，请始终根据器件特定数据手册配置等待状态。如果应用程序配置的等待状态不适合应用程序的工作频率，闪存 API 会发出错误。

- 闪存 API 执行可中断。但是，不能从正在进行擦除/编程操作的闪存组进行任何读取/获取访问。因此，闪存 API 函数、调用闪存 API 函数的用户应用程序函数以及任何中断服务例程 (ISR) 必须从 RAM 执行。例如，除了闪存 API 函数之外，上述条件还适用于下面显示的整个代码片段。之所以会这样，是因为 Fapi_issueAsyncCommandWithAddress() 函数向 FSM 发出了擦除命令，但并没有等到擦除操作结束。只要 FSM 忙于进行当前操作，就不能访问正在擦除的闪存组。

```
//
// Erase a Sector
//
oReturnCheck = Fapi_issueAsyncCommandWithAddress(Fapi_EraseSector,(uint32*)0x0080000);
//
// Wait until the erase operation is over
//
while (Fapi_checkFsmForReady() != Fapi_Status_FsmReady){}
```

- 闪存 API 不配置（启用/禁用）看门狗。用户应用程序可以配置看门狗并根据需要对其进行维护。因此，不再提供 Fapi_ServiceWatchdogTimer() 函数。
- 闪存 API 根据需要在内部使用 EALLOW 和 EDIS 来允许/禁止对受保护的寄存器进行写入。
- 主阵列闪存编程必须与 64 位地址边界对齐（建议在 128 位地址边界上对齐），并且每个 64 位字在每个写/擦除周期只能编程一次。
- 允许单独对数据和 ECC 进行编程。但是，每个 64 位数据字和相应的 ECC 字在每个写入擦除周期中只能编程一次。
- TMS320F28E12x 器件中没有泵信标。
- 不要针对链路指针位置对 ECC 进行编程。当为编程操作提供的起始地址是三个链路指针地址中的任何一个时，API 将跳过对 ECC 的编程。即使用户将 Fapi_AutoEccGeneration 或 Fapi_DataAndEcc 模式作为编程模式参数传递，API 也会使用 Fapi_DataOnly 模式对此类位置进行编程。Fapi_EccOnly 模式不支持对此类位置进行编程。用户应用程序需谨慎处理这一点。需注意为应用程序中的链路指针位置维护一个单独的结构/段。请勿将此类字段与其他 DCSM OTP 设置混合。如果其他字段与链路指针混合，API 也会跳过对非链路指针位置的 ECC 编程。这会导致应用程序中出现 ECC 错误。
- 使用 128 位 DCSM OTP 编程时，DCSM OTP 地址必须与 128 位地址边界对齐，并且每个 128 位字只能编程一次。例外包括：
  - DCSM OTP 中的 DCSM Zx-LINKPOINTER1 和 Zx-LINKPOINTER2 值可一起编程，并且可以按照 DCSM 操作的要求一次编程 1 位。
  - DCSM OTP 中的 DCSM Zx-LINKPOINTER3 值可按照 DCSM 操作的要求，一次编程 1 位。
  - 如果需要，用户可以单独对 JLM_Enable 进行编程。如果用户要同时对所有 64 位进行编程，则必须读取链路指针（ZxOTP_LINKPOINTER1、ZxOTP_LINKPOINTER2、ZxOTP_LINKPOINTER3），并将该值与 JLM_Enable 值一起编程。
  - 在给定对齐的 128 位字中，当对齐的 64 位被保留时，可以对其他 64 位进行编程。

备注

除链路指针范围（0x00078000 至 0x00078008 和 0x00078200 至 0x00078208）外，在一些 512 位编程模式下允许对 DCSM OTP 进行编程。要对链路指针位置进行编程，用户可将 Fapi_issueProgrammingCommand() 与 AutoECCGeneration 模式配合使用。有关更多详细信息，请参见下表。

512 位编程功能	对 DCSM OTP 和链路指针范围进行编程
Fapi_issueAutoEcc512ProgrammingCommand()	允许 DCSM OTP，不允许链路指针
Fapi_issueDataAndEcc512ProgrammingCommand()	允许 DCSM OTP，不允许链路指针
Fapi_issueDataOnly512ProgrammingCommand()	不允许 DCSM OTP 和链路指针
Fapi_issueEccOnly64ProgrammingCommand()	允许 ECC DCSM OTP，不允许 ECC 链路指针

- 为了避免 zone1 和 zone2 之间的冲突，DCSM 寄存器中提供了信标 (FLSEM) 来配置闪存寄存器。在对闪存进行初始化和调用闪存 API 函数之前，用户应用程序可配置该信标寄存器。有关该寄存器的更多详细信息，请参阅 *TMS320F28F28E12x 实时微控制器技术参考手册* (对于 TMS320F28E12x) 。
- 请注意，闪存 API 函数不配置任何 DCSM 寄存器。请确保为此用户应用程序配置所需的 DCSM 设置。例如，如果某个区域受到保护，则可从同一区域执行闪存 API，以便能够擦除或编程该区域的闪存扇区。或者可解锁该区域。否则，闪存 API 对闪存寄存器的写入将不会成功。闪存 API 不会检查对闪存寄存器的写入是否正在进行。其按照擦除/编程序列的要求向它们写入数据，并在假设写入完成后就返回。这会导致闪存 API 返回错误的成功状态。例如，调用 Fapi_issueAsyncCommandWithAddress(Fapi_EraseSector, Address) 时，可以返回成功状态，但这并不意味着扇区擦除成功。务必使用 Fapi_getFSMStatus() 和 Fapi_doBlankCheck () 检查擦除状态。
- 请注意，不能访问任何正在进行闪存擦除/编程操作的闪存组/OTP。

3 API 功能

3.1 初始化函数

3.1.1 Fapi_initializeAPI()

对闪存 API 进行初始化。

概要

```
Fapi_StatusType Fapi_initializeAPI(
                    Fapi_FmcRegistersType *poFlashControlRegister,
            uint32 u32HclkFrequency)
                                    )
```

参数

poFlashControlRegister [in]	指向闪存包装程序寄存器基地址的指针。使用 FlashTech_CPU0_BASE_ADDRESS。
u32HclkFrequency [in]	以 MHz 为单位的系统时钟频率

说明

在执行任何其他闪存 API 操作之前，需要使用该函数来对闪存 API 进行初始化。如果更改系统频率或 RWAIT，也必须调用该函数。

备注

调用该函数前必须设置 RWAIT 寄存器值。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Error_InvalidHclkValue**（失败：系统时钟与指定的等待值不匹配）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁）

实现示例

请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2 闪存状态机函数

3.2.1 Fapi_setActiveFlashBank()

对闪存包装程序进行初始化从而进行擦除和编程操作。

概要

```
Fapi_StatusType Fapi_setActiveFlashBank(
                          Fapi_FlashBankType oNewFlashBank
                                           )
```

参数

oNewFlashBank [in]	设置为有效的闪存组编号。由于只有一个闪存组，因此该器件可使用 Fapi_FlashBank0。

说明

该函数用于设置闪存封装器，以便在闪存组上进行进一步的操作。在使用 Fapi_initializeAPI() 函数之后以及在执行任何其他闪存 API 操作之前需要调用该函数。

备注

应用程序只需要调用该函数一次，其可以与 Fapi_FlashBank0 一起使用。

返回值

- **Fapi_Status_Success**（成功）

实现示例

请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2.2 Fapi_setupBankSectorEnable()

配置扇区的写入（编程）/擦除保护。

概要

```
Fapi_StatusType Fapi_setupBankSectorEnable(
                    uint32 WEPROT_register,
                    uint32 oSectorMask
                                          )
```

参数

pu32StartAddress [in]	用于写/擦除保护配置的寄存器地址。 对前 32 个 (0-31) 扇区使用 *FLASH_WRAPPER_PROGRAM_BASE+FLASH_O_CMDWEPROTA*。 对其余的主阵列 (32-127) 扇区使用 *FLASH_WRAPPER_PROGRAM_BASE+FLASH_O_CMDWEPROTB*。 对 USER OTP 使用 *FLASH_WRAPPER_PROGRAM_BASE+FLASH_O_CMDWEPROT_UO*。
OSectorMask [in]	32 位掩码，指示擦除和编程操作中要屏蔽的扇区。

说明

在此器件上，默认情况下，所有闪存主阵列扇区和 USER OTP 均受到保护，无法对其执行擦除和编程操作。用户应用程序必须禁用想要对其执行擦除和/或编程操作的扇区的保护。该函数可用于启用/禁用保护。可在每次擦除和编程命令之前调用该函数，如 C2000Ware 中提供的闪存 API 使用示例所示。

该函数的第一个输入参数可以是以下三个寄存器中任何一个的地址：*CMDWEPROTA*、*CMDWEPROTB*、*CMDWEPROT_UO*

CMDWEPROTA 寄存器用于配置前 32 个扇区（0 至 31）的保护。此寄存器中的每个位对应每个扇区 - 例如：此寄存器的位 0 用于配置扇区 0 的保护，该寄存器的位 31 用于配置扇区 31 的保护。用户提供的 32 位扇区掩码（传递给该函数的第二个参数）指示用户想要屏蔽擦除和编程操作的扇区，即不会被擦除和编程的扇区。如果掩码中的某个位为 1，则不会擦除/编程该特定扇区。

CMDWEPROTB 寄存器用于配置对主阵列闪存存储体中 32 - 127 个扇区的保护。但请注意，此寄存器中的每个位用于一起配置对 8 个扇区的保护。这意味着，位 0 用于一起配置对所有扇区 32 至 39 的保护，位 1 用于一起配置对所有扇区 40 至 47 的保护，依此类推。用户提供的 32 位扇区掩码（传递给该函数的第二个参数）指示用户想

要屏蔽擦除和编程操作的扇区，即不会被擦除和编程的扇区。如果掩码中的某个位为 1，则不会擦除/编程该组特定扇区。如果掩码中的某个位为 0，则会擦除/编程该组特定扇区。

CMDWEPROT_UO 寄存器用于配置 USER OTP 保护。此寄存器中的位 0 用于配置 USER OTP 保护。这意味着，如果位 0 配置为 1，则 USER OTP 不会被擦除/编程。其他位可配置为 1。由于 USER OTP 不可擦除，因此 CMDWEPROT_UO 寄存器保护不适用于擦除操作。只能针对所需的编程操作进行配置。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁）
- **Fapi_Error_FeatureNotAvailable**（失败：用户请求写入非 *CMDWEPROT* 保护的寄存器）。

实现示例（请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。）

3.2.3 Fapi_issueAsyncCommandWithAddress()

向闪存状态机发出擦除命令以及用户提供的扇区地址。

概要

```
Fapi_StatusType Fapi_issueAsyncCommandWithAddress(
                            Fapi_FlashStateCommandsType oCommand,
                            uint32 *pu32StartAddress
                                                      )
```

参数

oCommand [in]	向 FSM 发出的命令。使用 Fapi_EraseSector
pu32StartAddress [in]	用于擦除操作的闪存扇区地址

说明

该函数针对用户提供的扇区地址向闪存状态机发出擦除命令。该函数不会等到擦除操作结束；它只是发出命令并返回。因此，当使用 Fapi_EraseSector 命令时，该函数始终返回成功状态。用户应用程序必须等待闪存包装程序完成擦除操作，然后才能返回到任何类型的闪存访问。Fapi_checkFsmForReady() 函数可用于监测已发出命令的状态。

备注

该函数在发出擦除命令后不检查 STATCMD。当 FSM 完成擦除操作时，用户应用程序必须检查 STATCMD 值。STATCMD 指示擦除操作期间是否有任何故障发生。用户应用程序可以使用 Fapi_getFsmStatus 函数来获取 STATCMD 值。

此外，用户应用程序可使用 Fapi_doBlankCheck() 函数来验证闪存是否被擦除。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_FeatureNotAvailable**（失败：用户请求的命令不受支持。）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁）。
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

实现示例（请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2.4 Fapi_issueBankEraseCommand()

向闪存状态机发出闪存组擦除命令以及用户提供的扇区掩码。

概要

```
Fapi_StatusType Fapi_issueBankEraseCommand(
                            uint32 *pu32StartAddress
                                                )
```

参数

pu32StartAddress [in]	用于进行闪存组擦除操作的闪存组地址

说明

该函数针对用户提供的闪存组地址，向闪存状态机发出闪存组擦除命令。如果 FSM 正忙于进行另一个操作，该函数将返回值，指示 FSM 处于繁忙状态，否则将继续进行闪存组擦除操作。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_FlashRegsNotWritable**（闪存寄存器不可写）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

实现示例（请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2.5 Fapi_issueProgrammingCommand()

设置数据并向有效的闪存或 OTP 存储器地址发出编程命令。

概要

```
Fapi_StatusType Fapi_issueProgrammingCommand(
                          uint32 *pu32StartAddress,
                          uint16 *pu16DataBuffer,
                          uint16  u16DataBufferSizeInWords,
                          uint16 *pu16EccBuffer,
                          uint16  u16EccBufferSizeInBytes,
                          Fapi_FlashProgrammingCommandType oMode
                                              )
```

参数

pu32StartAddress [in]	闪存中的起始地址，用于对数据和 ECC 进行编程。此外，起始地址可始终为偶数。
pu16DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区可以为 128 位对齐。
u16DataBufferSizeInWords [in]	数据缓冲区中 16 位字的数量
pu16EccBuffer [in]	指向 ECC 缓冲区地址的指针
u16EccBufferSizeInBytes [in]	ECC 缓冲区中的 8 位字节数
oMode [in]	指示要使用的编程模式： Fapi_DataOnly 仅对数据缓冲区进行编程 Fapi_AutoEccGeneration 对数据缓冲区进行编程，并且自动生成 ECC 并对其进行编程。 Fapi_DataAndEcc 对数据和 ECC 缓冲区进行编程 Fapi_EccOnly 仅对 ECC 缓冲区进行编程

备注

pu16EccBuffer 包含与 128 位对齐主阵列/OTP 地址处的数据对应的 ECC。pu16EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu16EccBuffer 的 MSB 与主阵列的高 64 位相对应。

说明

该函数根据提供的参数设置闪存状态机的编程寄存器，为用户提供了适用于不同场景的四种不同编程模式，如不同编程模式的使用 中所述。

有关该函数允许的编程范围，请参阅表 3-1。

表 3-1. 不同编程模式的使用

编程模式 (oMode)	使用的参数	用途
Fapi_DataOnly	pu32StartAddress、 pu16DataBuffer、 u16DataBufferSizeInWords	当任何自定义编程实用程序或用户应用（嵌入/使用闪存 API）必须单独对数据和相应 ECC 进行编程时使用。使用 Fapi_DataOnly 模式对数据进行编程，然后使用 Fapi_EccOnly 模式对 ECC 进行编程。通常大多数编程实用程序不会单独计算 ECC，而是使用 Fapi_AutoEccGeneration 模式。但是，某些安全应用程序可能需要在其闪存映像中插入有意的 ECC 错误（使用 Fapi_AutoEccGeneration 模式时无法实现），从而在运行时检查 SECDED（单错校正和双错检测）模块的运行状况。在这种情况下，ECC 是单独计算的（如果适用，使用 Fapi_calculateEcc() 函数）。应用程序可能希望根据需要在主阵列数据或 ECC 中插入错误。在这种情况下，在错误插入之后，可以使用 Fapi_DataOnly 模式和 Fapi_EccOnly 模式分别对数据和 ECC 进行编程。
Fapi_AutoEccGeneration	pu32StartAddress、 pu16DataBuffer、 u16DataBufferSizeInWords	当任何自定义编程实用程序或用户应用（其在运行时嵌入/使用闪存 API 对闪存进行编程从而存储数据或进行固件更新）必须同时对数据和 ECC 进行编程而不插入任何有意错误时使用。该模式是常用的模式。
Fapi_DataAndEcc	pu32StartAddress、 pu16DataBuffer、 u16DataBufferSizeInWords、 pu16EccBuffer、 u16EccBufferSizeInBytes	该模式的用途与同时使用 Fapi_DataOnly 和 Fapi_EccOnly 模式的用途没有什么不同。但是，当可以同时对数据和计算出的 ECC 进行编程时，该模式会很有用。
Fapi_EccOnly	pu16EccBuffer u16EccBufferSizeInBytes	请参阅 Fapi_DataOnly 模式的用途描述。

表 3-2. Fapi_issueProgrammingCommand() 的允许编程范围

闪存 API	主阵列	DCSM OTP	ECC	链路指针
Fapi_issueProgrammingCommand() 128 位， Fapi_AutoEccGeneration 模式	允许	允许	允许	允许（闪存 API 将其编程为 *Fapi_DataOnly*）
Fapi_issueProgrammingCommand() 128 位，Fapi_DataOnly 模式	允许	允许（闪存 API 将其编程为 *Fapi_AutoEccGeneration*）	不允许	允许
Fapi_issueProgrammingCommand() 128 位， Fapi_DataAndEcc 模式	允许	允许	允许	允许（闪存 API 将其编程为 *Fapi_DataOnly*）
Fapi_issueProgrammingCommand() 128 位，Fapi_EccOnly 模式	不允许	不允许	允许	不允许

备注

由于 ECC 检查在上电时启用，用户必须始终为其闪存映像对 ECC 进行编程。

编程模式：

Fapi_DataOnly — 该模式只对闪存中指定地址的数据部分进行编程。其编程范围从 1 位至 8 个 16 位字。但是，请注意该函数的限制条件中对闪存编程数据大小的限制。提供的编程起始地址加上数据缓冲区长度无法跨越 128 位对齐地址边界。使用该模式时将忽略参数 4 和 5。

Fapi_AutoEccGeneration – 该模式会对闪存中提供的数据以及自动生成的 ECC 进行编程。针对在 64 位存储器边界上对齐的每项 64 位数据计算 ECC。因此，在使用该模式时，可以针对给定的 64 位对齐存储器地址，同时对所有 64 位数据进行编程。未提供的数据全部视作 1 (0xFFFF)。针对 64 位数据计算 ECC 并对其进行编程后，即使在该 64 位数据中将位从 1 编程为 0，也无法对此类 64 位数据进行重新编程（除非扇区被擦除），因为新的 ECC 值会与先前编程的 ECC 值相冲突。使用该模式时，如果起始地址进行了 128 位对齐，则可以根据需要同时对 8 个或 4 个 16 位字进行编程。如果起始地址进行了 64 位对齐而不是 128 位对齐，则只能同时对 4 个 16 位字进行编程。该选项还存在 Fapi_DataOnly 的数据限制。忽略参数 4 和 5。

备注

Fapi_AutoEccGeneration 模式会对闪存中提供的数据部分以及自动生成的 ECC 进行编程。针对 64 位对齐地址和相应的 64 位数据计算 ECC。未提供的任何数据将视作 0xFFFF。请注意，在编写自定义编程实用程序时，该实用程序在代码项目的输出文件中流式传输，并将各段一次编程到闪存中，这会产生实际影响。如果一个 64 位的字跨越多个段（即包含一段的末尾和另一段的开头），在对第一段进行编程时，无法针对 64 位字中缺失数据假设值为 0xFFFF。当您对第二段进行编程时，您无法对第一个 64 位字的 ECC 进行编程，因为它已经（错误地）使用假定的 0xFFFF 对缺失值进行了计算和编程。避免该问题的一种方法是在代码项目的链接器命令文件中的 64 位边界上对齐链接到闪存的所有段。

下面我们举例说明：

```
SECTIONS
   {
     .text         : > FLASH, ALIGN(4)
     .cinit        : > FLASH, ALIGN(4)
     .const        : > FLASH, ALIGN(4)
     .init_array   : > FLASH, ALIGN(4)
     .switch       : > FLASH, ALIGN(4)
   }
```

如果不在闪存中对齐这些段，则需要跟踪段中不完整的 64 位字，并将这些字与其他段中的字组合在一起，从而使 64 位字变得完整。这一点很难做到。因此，建议在 64 位边界上对段进行对齐。

某些第三方闪存编程工具或 TI 闪存编程内核示例 (*C2000Ware*) 或任何自定义闪存编程解决方案可能假定传入数据流全部为 128 位对齐，并且可能没有预想到某段从未对齐的地址开始。因此，假设提供的地址为 128 位对齐，则会尝试对最大可能的（128 位）字进行一次编程。当地址未对齐时，这可能会导致出现故障。因此，建议在 128 位边界上对齐所有段（映射到闪存）。

Fapi_DataAndEcc — 该模式会在指定地址的闪存中对提供的数据和 ECC 进行编程。提供的数据必须在 64 位存储器边界上对齐，并且数据长度必须与提供的 ECC 相关联。这意味着，如果数据缓冲区长度为 4 个 16 位字，则 ECC 缓冲区长度必须为 1 字节。如果数据缓冲区长度为 8 个 16 位字，则 ECC 缓冲区的长度必须为 2 字节。如果起始地址进行了 128 位对齐，则可根据需要同时对 8 或 4 个 16 位字进行编程。如果起始地址进行了 64 位对齐而不是 128 位对齐，则只能同时对 4 个 16 位字进行编程。

pu16EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu16EccBuffer 的 MSB 与主阵列的高 64 位相对应。

Fapi_calculateEcc() 函数可用于计算给定 64 位对齐地址和相应数据的 ECC。

Fapi_EccOnly — 该模式仅在指定的地址处（可以为该函数提供闪存主阵列地址，而不是相应的 ECC 地址）对闪存 ECC 存储空间中的 ECC 部分进行编程。它可以对 2 字节（ECC 存储器中某一位置的 LSB 和 MSB）或 1 字节（ECC 存储器中某一位置的 LSB）编程。

pu16EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu16EccBuffer 的 MSB 与主阵列的高 64 位相对应。

使用该模式时将忽略参数二和三。

备注

pu16DataBuffer 和 pu16EccBuffer 的长度分别不可超过 8 和 2。

备注

该函数在发出编程命令后不检查 STATCMD。当 FSM 完成编程操作时，用户应用程序必须检查 STATCMD 值。STATCMD 指示编程操作期间是否有任何故障发生。用户应用程序可以使用 Fapi_getFsmStatus 函数来获取 STATCMD 值。

此外，用户应用程序使用 Fapi_doVerify() 函数来验证闪存是否已正确编程。

该函数不会等到编程操作结束；它只是发出命令并返回。用户应用程序必须等待闪存包装程序完成编程操作，然后才能返回到任何类型的闪存访问。Fapi_checkFsmForReady() 函数可用于监测已发出命令的状态。

限制

- 如上所述，该函数一次最多只能对 128 位进行编程（鉴于提供的地址进行了 128 位对齐）。如果用户想对更多位进行编程，则可以循环调用该函数，从而一次对 128 位（或应用程序所需的 64 位）进行编程。
- 主阵列闪存编程必须与 64 位地址边界对齐，并且每个 64 位字在每个写入或擦除周期只能编程一次。
- 可以单独对数据和 ECC 进行编程。但是，每个 64 位数据字和相应的 ECC 字在每个写入或擦除周期中只能编程一次。
- 不能针对链接指针位置对 ECC 进行编程。即使用户选择 Fapi_AutoEccGeneration 模式或 Fapi_DataAndEcc 模式，API 也会为此类位置发出 Fapi_DataOnly 命令。链路指针位置不支持 Fapi_EccOnly 模式。
- Fapi_EccOnly 模式不能用于 Bank0 DCSM OTP 空间。如果使用，将返回错误。对于 DCSM OTP 空间，可使用 Fapi_AutoEccGeneration 或 Fapi_DataAndEcc 编程模式。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_AsyncIncorrectDataBufferLength**（失败：指定的数据缓冲区大小不正确。此外，如果在对 Bank0 DCSM OTP 空间进行编程时选择了 Fapi_EccOnly 模式，可能返回该错误）
- **Fapi_Error_AsyncIncorrectEccBufferLength**（失败：指定的 ECC 缓冲区大小不正确）
- **Fapi_Error_AsyncDataEccBufferLengthMismatch**（失败：数据缓冲区大小不是 64 位对齐的，或者数据长度跨越了 128 位对齐的存储器边界）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户需确保 API 从与闪存操作的目标地址相同的区域执行，或者用户在闪存操作之前解锁。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。有关有效地址范围的信息，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

  （对于 TMS320F28E12x）。

实现示例

请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2.6 Fapi_issueProgrammingCommandForEccAddresses()

将 ECC 地址重新映射到数据地址并调用 Fapi_issueProgrammingCommand()。

概要

```
Fapi_StatusType
Fapi_issueProgrammingCommandForEccAddresses(
          uint32 *pu32StartAddress,
          uint16 *pu16EccBuffer,
```

```
            uint16   u16EccBufferSizeInBytes
                                            )
```

参数

pu32StartAddress [in]	闪存中 ECC 的起始地址，用于对 ECC 进行编程
pu16EccBuffer [in]	指向 ECC 缓冲区地址的指针
u16EccBufferSizeInBytes [in]	ECC 缓冲区中的字节数 如果字节数为 1，则对 LSB (针对低 64 位的 ECC) 进行编程。无法使用该函数单独对 MSB 进行编程。如果字节数为 2，则将对 ECC 的 LSB 和 MSB 字节进行编程。

说明

该函数会将 ECC 存储空间中的地址重新映射到对应的数据地址空间，然后调用 Fapi_issueProgrammingCommand() 对提供的 ECC 数据进行编程。使用 Fapi_EccOnly 模式的 Fapi_issueProgrammingCommand() 的限制同样也适用于该函数。pu16EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu16EccBuffer 的 MSB 与主阵列的高 64 位相对应。

备注

pu16EccBuffer 的长度不可超过 2。

备注

该函数在发出编程命令后不检查 STATCMD。当 FSM 完成编程操作时，用户应用程序必须检查 STATCMD 值。STATCMD 指示编程操作期间是否有任何故障发生。用户应用程序可以使用 Fapi_getFsmStatus 函数来获取 STATCMD 值。

备注

Fapi_EccOnly 模式不能用于 Bank0 DCSM OTP 空间。如果使用，将返回错误。对于 DCSM OTP 空间，可使用 Fapi_AutoEccGeneration 或 Fapi_DataAndEcc 编程模式。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_AsyncIncorrectEccBufferLength**（失败：指定的数据缓冲区大小不正确）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁。

Fapi_Error_InvalidAddress（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

3.2.7 Fapi_issueAutoEcc512ProgrammingCommand()

设置数据并向有效的闪存或 OTP 存储器地址发出 512 位（32 个 16 位字）AutoEcc 生成模式编程命令。

概要

```
Fapi_StatusType
Fapi_issueAutoEcc512ProgrammingCommand (
                                uint32 *pu32StartAddress,uint16*pu16DataBuffer,
                               uint16u16DataBufferSizeInWords
                                        )
```

参数

pu32StartAddress [in]	闪存中的起始地址，用于对数据和 ECC 进行编程。
pu16DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区的地址可按 512 位对齐。
u16DataBufferSizeInWords [in]	数据缓冲区中 16 位字的数量。最大数据缓冲区大小（以字为单位）不得超过 32。

说明

此函数会自动为用户提供的 512 位数据（第二个参数）生成 8 个字节的 ECC 数据，并在用户提供的 512 位对齐闪存地址（第一个参数）处将数据和 ECC 一起编程。发出此命令时，闪存状态机会对所有 512 位以及 ECC 进行编程。因此，使用该模式时，未提供的数据全部视为 1 (0xFFFF)。针对 512 位数据计算 ECC 并对其进行编程后，即使在该 512 位数据中将位从 1 编程为 0，也无法对此类 512 位数据进行重新编程（除非扇区被擦除），因为新的 ECC 值会与先前编程的 ECC 值相冲突。

备注

Fapi_issueAutoEcc512ProgrammingCommand() 函数会对闪存中提供的数据部分以及自动生成的 ECC 进行编程。针对 512 位对齐地址和相应的 512 位数据计算 ECC。未提供的任何数据将视作 0xFFFF。请注意，在编写自定义编程实用程序时，该实用程序在代码项目的输出文件中流式传输，并将各段一次编程到闪存中，这会产生实际影响。如果一个 512 位的字跨越多个段（即包含一段的末尾和另一段的开头），在对第一段进行编程时，无法针对 64 位字中缺失数据假设值为 0xFFFF。当您对第二段进行编程时，您无法对第一个 512 位字的 ECC 进行编程，因为它已经（错误地）使用假定的 0xFFFF 对缺失值进行了计算和编程。避免该问题的一种方法是在代码项目的链接器命令文件中的 512 位边界上对齐链接到闪存的所有段。

```
SECTIONS
  {
    .text       : > FLASH, ALIGN(32)
    .cinit      : > FLASH, ALIGN(32)
    .const      : > FLASH, ALIGN(32)
    .init_array : > FLASH, ALIGN(32)
    .switch     : > FLASH, ALIGN(32)
  }
```

如果不在闪存中对齐这些段，则需要跟踪段中不完整的 512 位字，并将这些字与其他段中的字组合在一起，从而使 512 位字变得完整。这一点很难做到。因此，建议在 512 位边界上对段进行对齐。

某些第三方闪存编程工具或 TI 闪存编程内核示例 (*C2000Ware*) 或任何自定义闪存编程解决方案可能假定传入数据流全部为 512 位对齐，并且可能无法预想到某段从未对齐的地址开始。因此，假设提供的地址进行了 512 位对齐，其可能会尝试一次对最大可能的（512 位）字进行编程。当地址未对齐时，这可能会导致出现故障。因此，建议在 512 位边界上对齐所有段（映射到闪存）。

有关该函数允许的编程范围，请参阅表 3-3。

表 3-3. Fapi_issueAutoEcc512ProgrammingCommand() 的允许编程范围

闪存 API	主阵列	DCSM OTP	ECC	链路指针
Fapi_issueAutoEcc512ProgrammingCommand()	允许	允许	允许	不允许

限制

- 如上所述，该函数一次最多只能对 512 位进行编程（鉴于提供的地址进行了 512 位对齐）。如果用户需要编程更多位，可以循环调用该函数，每次编程 512 位。
- 主阵列闪存编程必须与 512 位地址边界对齐，并且 32 个 16 位字在每个写入或擦除周期只能编程一次。
- 以链路指针地址开头的 512 位地址范围应始终使用 128 位 Fapi_issueProgrammingCommand() 进行编程。

返回值

- Fapi_Status_Success（成功）
- Fapi_Status_FsmBusy（FSM 处于繁忙状态）
- **Fapi_Error_AsyncIncorrectDataBufferLength**（失败：指定的数据缓冲区大小不正确。此外，如果在对 Bank0 DCSM OTP 空间进行编程时选择了 Fapi_EccOnly 模式，可能返回该错误）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 微控制器数据手册*。

实现示例

将在下一个 C2000ware 版本中发布此功能的示例实现。

3.2.8 Fapi_issueDataAndEcc512ProgrammingCommand()

设置使用用户提供的闪存数据和 ECC 数据进行 512 位（32 个 16 位字）编程的闪存状态机寄存器，并向有效的闪存和 OTP 存储器发出编程命令。

概要

```
Fapi_StatusType Fapi_issueDataAndEcc512ProgrammingCommand(
                                uint32 *pu32StartAddress,
                                uint16 *pu16DataBuffer,
                                uint16 u16DataBufferSizeInWords,
                                uint16 *pu16EccBuffer,
                                uint16 u16EccBufferSizeInBytes
                                                            )
```

参数

pu32StartAddress [in]	用于对提供的数据和 ECC 进行编程的 512 位对齐闪存地址
pu16DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区的地址可按 512 位对齐。
u16DataBufferSizeInWords [in]	数据缓冲区中 16 位字的数量。最大数据缓冲区大小（以字为单位）不得超过 32。
pu16EccBuffer [in]	指向 ECC 缓冲区地址的指针
u16EccBufferSizeInBytes [in]	ECC 缓冲区中的 8 位字节数。最大 Eccbuffer 大小（以字为单位）不得超过 8。

说明

此函数在用户提供的 512 位对齐闪存地址处对用户提供的 512 位数据（第二个参数）和 8 字节的 ECC 数据（第四个参数）进行编程。所提供数据的地址必须在 512 位存储器边界上对齐，并且数据长度必须与提供的 ECC 相关联。这意味着，如果数据缓冲区长度为 32 个 16 位字，则 ECC 缓冲区必须为 8 个字节（1 个 ECC 字节对应于 64 位数据）。

pu16EccBuffer 的每个字节与 *pu16DataBuffer* 中提供的主阵列数据的每个 64 位相对应。

如需了解更多详情，请参阅表 3-6。

Fapi_calculateEcc() 函数可用于计算给定 64 位对齐地址和相应数据的 ECC

有关该函数允许的编程范围，请参阅表 3-4。

表 3-4. Fapi_issueDataAndEcc512ProgrammingCommand() 的允许编程范围

闪存 API	主阵列	DCSM OTP	ECC	链路指针
Fapi_issueDataAndEcc512ProgrammingCommand()	允许	允许	允许	禁止

限制

- 如上所述，该函数一次最多只能对 512 位进行编程（鉴于提供的地址进行了 512 位对齐）。如果用户需要编程更多位，可以循环调用该函数，每次编程 512 位。
- 主阵列闪存编程必须与 512 位地址边界对齐，并且 32 个 16 位字在每个写入或擦除周期只能编程一次。
- 以链路指针地址开头的 512 位地址范围应始终使用 128 位 Fapi_issueProgrammingCommand() 进行编程。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_AsyncIncorrectDataBufferLength**（失败：指定的数据缓冲区大小不正确。此外，如果在对 Bank0 DCSM OTP 空间进行编程时选择了 Fapi_EccOnly 模式，可能返回该错误）
- **Fapi_Error_AsyncIncorrectEccBufferLength**（失败：指定的 ECC 缓冲区大小不正确）
- **Fapi_Error_AsyncDataEccBufferLengthMismatch**（失败：数据缓冲区大小不是 64 位对齐的，或者数据长度跨越了 128 位对齐的存储器边界）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）

- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 微控制器数据手册*。

实现示例

将在下一个 C2000ware 版本中发布此功能的示例实现。

3.2.9 Fapi_issueDataOnly512ProgrammingCommand()

设置使用用户提供的闪存数据进行 512 位（32 个 16 位字）编程的闪存状态机寄存器，并向有效闪存发出编程命令。

概要

```
Fapi_StatusType
Fapi_issueDataOnly512ProgrammingCommand(
                        uint32 *pu32StartAddress,
                        uint16 *pu16DataBuffer,
                        uint16 u16DataBufferSizeInWords
                                        )
```

参数

pu32StartAddress [in]	用于对提供的数据进行编程的 512 位对齐闪存地址
pu16DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区可以为 512 位对齐。
u16DataBufferSizeInWords [in]	数据缓冲区中 16 位字的数量。最大数据缓冲区大小（以字为单位）不得超过 32。

说明

此函数只对闪存中指定地址的数据部分进行编程。它可以在用户提供的 512 位对齐闪存地址处对 512 位数据（第二个参数）进行编程。

当用户应用程序（嵌入/使用闪存 API）必须单独对 512 位数据和相应的 64 位 ECC 数据进行编程时，使用此函数。使用 Fapi_issueDataOnly512ProgrammingCommand () 函数对 512 位数据进行编程，然后使用 Fapi_issueEccOnly64ProgrammingCommand() 函数对 64 位 ECC 进行编程。通常，大多数编程实用程序不会单独计算 ECC，而是使用 Fapi_issueAutoEcc512ProgrammingCommand() 函数。但是，某些安全应用程序可能需要在其闪存映像中插入有意的 ECC 错误（使用 Fapi_AutoEccGeneration 模式时无法实现），从而在运行时检查 SECDED（单错校正和双错检测）模块的运行状况。在这种情况下，ECC 是单独计算的（如果适用，使用 Fapi_calculateEcc() 函数）。应用程序可能希望根据需要在主阵列数据或 ECC 中插入错误。在这种情况下，在错误插入之后，可以使用 Fapi_issueDataOnly512ProgrammingCommand () API 和 Fapi_issueEccOnly64ProgrammingCommand() API 分别对 512 位数据和 64 位 ECC 进行编程。

有关该函数允许的编程范围，请参阅表 3-5。

表 3-5. Fapi_issueDataOnly512ProgrammingCommand() 的允许编程范围

闪存 API	主阵列	DCSM OTP	ECC	链路指针
Fapi_issueDataOnly512ProgrammingCommand()	允许	不允许	不允许	不允许

限制

- 如上所述，该函数一次最多只能对 512 位进行编程（鉴于提供的地址进行了 512 位对齐）。如果用户需要编程更多位，可以循环调用该函数，每次编程 512 位。
- 主阵列闪存编程必须与 512 位地址边界对齐，并且 32 个 16 位字在每个写入或擦除周期只能编程一次。
- 以链路指针地址开头的 512 位地址范围应始终使用 128 位 Fapi_issueProgrammingCommand() 进行编程。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_AsyncIncorrectDataBufferLength**（失败：指定的数据缓冲区大小不正确。此外，如果在对 Bank0 DCSM OTP 空间进行编程时选择了 Fapi_EccOnly 模式，可能返回该错误）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

实现示例

将在下一个 C2000ware 版本中发布此功能的示例实现。

3.2.10 Fapi_issueEccOnly64ProgrammingCommand()

设置使用用户提供的 ECC 数据进行 64 位（4 个 16 位字）编程的闪存状态机寄存器，并向有效闪存和 OTP 存储器发出编程命令。

概要

```
Fapi_StatusType
Fapi_issueEccOnly64ProgrammingCommand(
                    uint32 *pu32StartAddress,
                    uint16 *pu16EccBuffer,
                    uint16 u16EccBufferSizeInBytes
                                          )
```

参数

pu32StartAddress [in]	用于对提供的 ECC 数据进行编程的 512 位对齐闪存地址
pu16EccBuffer [in]	指向 ECC 缓冲区地址的指针
u16EccBufferSizeInBytes[in]	ECC 缓冲区中的 8 位字节数。最大 Eccbuffer 大小（以字为单位）不能超过 8。

说明

此函数仅在指定的地址处（可能为该函数提供闪存主阵列地址，而不是相应的 ECC 地址）对闪存 ECC 存储空间中的 ECC 部分进行编程。它可以在与用户提供的 512 位对齐闪存地址相对应的 ECC 地址处对 64 位 ECC 数据（第二个参数）进行编程。

64 位 ECC 数据可以拆分为与 512 位对齐数据相关的 8 字节 ECC 数据（4 * 128，每 2 字节对应每 128 个数据）。有关更多信息，请参阅 表 3-6。

表 3-6. 64 位 ECC 数据解释

512 位数据（4 * 128 位）			
第 1 个 128 位数据	**第 2 个 128 位数据**	**第 3 个 128 位数据**	**第 4 个 128 位数据**
pu16EccBuffer[0] 的 LSB	pu16EccBuffer[1] 的 LSB	pu16EccBuffer[2] 的 LSB	pu16EccBuffer[3] 的 LSB
pu16EccBuffer[0] 的 MSB	pu16EccBuffer[1] 的 MSB	pu16EccBuffer[2] 的 MSB	pu16EccBuffer[3] 的 MSB

有关该函数允许的编程范围，请参阅表 3-7。

表 3-7. Fapi_issueEccOnly64ProgrammingCommand() 的允许编程范围

闪存 API	主阵列	DCSM OTP	ECC	链路指针
Fapi_issueEccOnly64ProgrammingCommand()	禁止	禁止	允许	禁止

限制

- 如上所述，该函数一次最多只能对 64 位 ECC 进行编程。如果用户需要编程更多位，可以循环调用该函数，每次编程 64 位。
- 主阵列闪存编程必须与 512 位地址边界对齐，并且 64 位 ECC 字在每个写入或擦除周期只能编程一次。
- 不能针对链路指针位置对 ECC 进行编程。以链路指针地址开头的 512 位地址范围应始终使用 128 位 Fapi_issueProgrammingCommand() 进行编程。

返回值

- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Status_Success**（成功）
- **Fapi_Error_AsyncDataEccBufferLengthMismatch**（失败：数据缓冲区大小不是 64 位对齐的，或者数据长度跨越了 128 位对齐的存储器边界）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可确保 API 从与闪存操作的目标地址相同的区域执行，或者用户可在闪存操作之前解锁。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 微控制器数据手册*。

实现示例

将在下一个 C2000ware 版本中发布此功能的示例实现。

3.2.11 Fapi_issueAsyncCommand()

向闪存状态机发出命令。有关该函数可发出的命令列表，请参阅说明。

概要

```
Fapi_StatusType Fapi_issueAsyncCommand(
                Fapi_FlashStateCommandsType oCommand
                                       )
```

参数

oCommand [in]	向 FSM 发出的命令。使用 Fapi_ClearStatus 命令。

说明

对于不需要任何附加信息（如地址）的命令，该函数向闪存状态机发出命令。在此器件上，可以使用此函数向闪存状态机发出 Fapi_ClearStatus 命令。请注意，可在每个编程和擦除命令之前发出 Fapi_ClearStatus 命令，如 C2000Ware 中提供的闪存编程示例所示。只有当 STATCMD 为 0 时（通过发出 Fapi_ClearStatus 命令来实现），才可以发出新的编程或擦除命令。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的命令）

实现示例

请参阅 C2000Ware 中提供的闪存编程示例，对于 TMS320F28E12x，该示例位于“C2000Ware_.....\driverlib\f28e12x\examples\flash\flashapi_ex1_programming.c”。

3.2.12 Fapi_checkFsmForReady()

返回闪存状态机的状态。

概要

```
Fapi_StatusType Fapi_checkFsmForReady(void)
```

参数

无

说明

该函数返回闪存状态机的状态，指示其是否准备好接受新命令。主要用途是检查擦除或编程操作是否已完成。

返回值

- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态，除暂停命令外，无法接受新命令）
- **Fapi_Status_FsmReady**（FSM 已准备好接受新命令）

3.2.13 Fapi_getFsmStatus()

返回 STATCMD 寄存器的值。

概要

```
Fapi_FlashStatusType Fapi_getFsmStatus(void)
```

参数

无

说明

该函数返回 STATCMD 寄存器的值。该寄存器允许用户应用程序确定擦除或编程操作是成功完成、正在进行、暂停还是失败。用户应用程序可检查该寄存器的值从而确定每次擦除和编程操作后是否有任何故障。

表 3-8. STATCMD 寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			FAILMISC				FAILINVDATA		FAILILLADDR	FAILVERIFY	FAILWEPROT		CMDINPROGRESS	CMDPASS	CMDDONE
			RO(1) - 0x0				RO - 0x0		RO - 0x0	RO - 0x0	RO - 0x0		RO - 0x0	RO - 0x0	RO - 0x0

(1) RO - 只读

表 3-9. STATCMD 寄存器字段说明

位	名称	说明	复位值
12	FAILMISC	由于出现除写/擦除保护违例或验证错误以外的其他错误，命令失败。 0：未出现失败 1：失败	0x0
8	FAILINVDATA	因为尝试将存储的 0 值编程为 1，编程命令失败。 0：未出现失败 1：失败	0x0
6	FAILILLADDR	由于使用了非法地址，命令失败。 0：未出现失败 1：失败	0x0
5	FAILVERIFY	由于验证错误，命令失败。 0：未出现失败 1：失败	0x0
4	FAILWEPROT	由于写/擦除保护扇区违例，命令失败。 0：未出现失败 1：失败	0x0
2	CMDINPROGRESS	命令进行中 0：命令完成 1：命令进行中	0x0
1	CMDPASS	命令通过 - 当 CMD_DONE 字段为 1 时有效 0：失败 1：通过	0x0
0	CMDDONE	命令完成 0：命令未完成 1：命令完成	0x0

3.3 读取函数

3.3.1 Fapi_doBlankCheck()

验证指定区域是否为擦除值。

概要

```
Fapi_StatusType Fapi_doBlankCheck(
              uint32 *pu32StartAddress,
              uint32  u32Length,
              Fapi_FlashStatusWordType *poFlashStatusWord
                                )
```

参数

pu32StartAddress [in]	要进行空白检查的区域的起始地址
u32Length [in]	要进行空白检查的区域长度（以 32 位字为单位）
poFlashStatusWord [out]	如果结果不是 Fapi_Status_Success，则返回操作状态
->au32StatusWord[0]	第一个非空白位置的地址
->au32StatusWord[1]	在第一个非空白位置读取的数据
->au32StatusWord[2	比较数据的值（始终为 0xFFFFFFFF）
->au32StatusWord[3]	不适用

说明

该函数在从指定地址开始的指定长度（以 32 位字为单位）的区域内，检查闪存是否为空白（擦除状态）。如果发现非空白位置，则在 poFlashStatusWord 参数中返回相应的地址和数据。

限制

无

返回值

- **Fapi_Status_Success（成功）**：发现指定的闪存位置处于已擦除状态
- **Fapi_Error_Fail**（失败：指定区域非空白）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

3.3.2 Fapi_doVerify()

根据提供的数据验证指定区域。

概要

```
Fapi_StatusType Fapi_doVerify(
         uint32 *pu32StartAddress,
         uint32  u32Length,
         uint32 *pu32CheckValueBuffer,
         Fapi_FlashStatusWordType *poFlashStatusWord
                           )
```

参数

pu32StartAddress [in]	要验证区域的起始地址
u32Length [in]	要验证的区域长度（以 32 位字为单位）
u32Length [in]	用于验证区域的缓冲区地址。数据缓冲区可以为 128 位对齐。
poFlashStatusWord [out]	如果结果不是 Fapi_Status_Success，则返回操作状态
->au32StatusWord[0]	首次验证失败位置的地址
->au32StatusWord[1]	在首次验证失败的位置读取的数据
->au32StatusWord[2]	比较数据的值
->au32StatusWord[3]	不适用

说明

该函数在从指定地址开始的指定长度（以 32 位字为单位）的区域内，根据提供的数据验证器件。如果位置比较失败，这些结果会在 poFlashStatusWord 参数中返回。

限制

无

返回值

- **Fapi_Status_Success**（成功：指定的区域与提供的数据匹配）
- **Fapi_Error_Fail**（失败：指定区域与提供的数据不匹配）
- **Fapi_Error_InvalidAddress**（失败：用户提供的地址无效。对于 TMS320F28E12x，有关有效地址范围，请参阅 *TMS320F28E12x 实时微控制器数据手册*。

3.4 信息函数

3.4.1 Fapi_getLibraryInfo()

返回有关该闪存 API 编译的信息。

概要

```
Fapi_LibraryInfoType Fapi_getLibraryInfo(void)
```

参数

无

说明

该函数返回特定于闪存 API 库编译的信息。该信息在 Fapi_LibraryInfoType 结构中返回。该结构的成员如下：

- u8ApiMajorVersion – 该 API 编译的主要版本号。该值为 2。
- u8ApiMinorVersion – 该 API 编译的次要版本号。F28E12x 器件的次要版本为 0。
- u8ApiRevision – 该 API 编译的修订版本号。

该版本的修订版本号为 10。

- oApiProductionStatus – 该编译的生产状态（Alpha_Internal、Alpha、Beta_Internal、Beta、生产）

该版本的生产状态为“生产”。

- u32ApiBuildNumber – 该编译的构建版本号。
- u8ApiTechnologyType – 表示 API 支持的闪存技术。该字段返回值为 0x5。
- u8ApitechnologyRevision – 表示 API 支持的技术的修订版
- u8ApiEndianness – 对于 F28E12x 器件，此字段始终返回 1（小端字节序）。
- u32ApiCompilerVersion – 用于编译 API 的 Code Composer Studio 代码生成工具的版本号

返回值

- **Fapi_LibraryInfoType**（提供有关该 API 编译的检索信息）

3.5 实用功能

3.5.1 Fapi_flushPipeline()

刷新闪存包装程序流水线缓冲区。

概要

```
void Fapi_flushPipeline(void)
```

参数

无

说明

该函数可刷新闪存包装程序数据高速缓存。在进行擦除或编程操作后，必须在读取第一个非 API 闪存之前刷新数据高速缓存。

返回值

无

3.5.2 Fapi_calculateEcc()

计算所提供地址和 64 位值的 ECC。

概要

```
uint8 Fapi_calculateEcc(
            uint32 u32Address,
            uint64 u64Data
                    )
```

参数

u32Address [in]	用于计算 ECC 的 64 位值的地址
u64Data [in]	用于计算 ECC 的 64 位值（可采用小端字节序）

说明

该函数会计算包括地址在内的 64 位对齐字的 ECC。不再需要为该函数提供左移地址。闪存 API 负责处理。

返回值

- 计算出的 8 位 ECC（可以忽略 16 位返回值的高 8 位）
- 如果发生错误，16 位返回值为 0xDEAD

3.5.3 Fapi_isAddressEcc()

表示地址位于闪存包装程序 ECC 空间。

概要

```
boolean Fapi_isAddressEcc(
            uint32 u32Address
    )
```

参数

u32Address [in]	用于确定是否位于 ECC 地址空间的地址

说明

如果地址位于 ECC 地址空间，则该函数返回 TRUE，否则返回 FALSE。

返回值

- **FALSE**（地址不在 ECC 地址空间内）
- **TRUE**（地址位于 ECC 地址空间）

3.5.4 Fapi_remapEccAddress()

获取 ECC 地址并将其重新映射到主地址空间。

概要

```
uint32 Fapi_remapEccAddress(
          uint32 u32EccAddress
     )
```

参数

u32EccAddress [in] ECC address to remap

说明

该函数返回给定闪存 ECC 地址的主阵列闪存地址。当用户想要在已知 ECC 地址处对 ECC 数据进行编程时，可以使用该函数获取对应的主阵列地址。请注意，Fapi_issueProgrammingCommand() 函数需要一个主阵列地址而不是 ECC 地址（即使是在 Fapi_EccOnly 模式下）。

返回值

- **32 位主闪存地址。**
- **0xFFFFFFFF（如果提供的闪存 ECC 地址无效）**

3.5.5 Fapi_calculateFletcherChecksum()

根据给定的地址和长度计算 Fletcher 校验和。

概要

```
uint32 Fapi_calculateFletcherChecksum(
               uint16 *pu16Data,
               uint16 u16Length
                                  )
```

参数

pu16Data [in]	计算校验和的起始地址
u16Length [in]	计算中要使用的 16 位字的数量

说明

该函数为指定数量的 16 位字从提供的地址开始生成 32 位 Fletcher 校验和。请注意，只有闪存主阵列地址范围可用于此函数。无法提供 DCSM OTP 地址范围。

返回值

- 32 位 Fletcher 校验和值

4 推荐的 FSM 流程

4.1 新出厂器件

器件出厂时已擦除。建议对收到的器件进行空白检查从而验证其是否已被擦除，但这不是必需的。

4.2 推荐的擦除流程

图 4-1 描述了器件上扇区的擦除流程。有关更多信息，请参阅节 3.2.11、节 3.2.2、节 3.2.3。

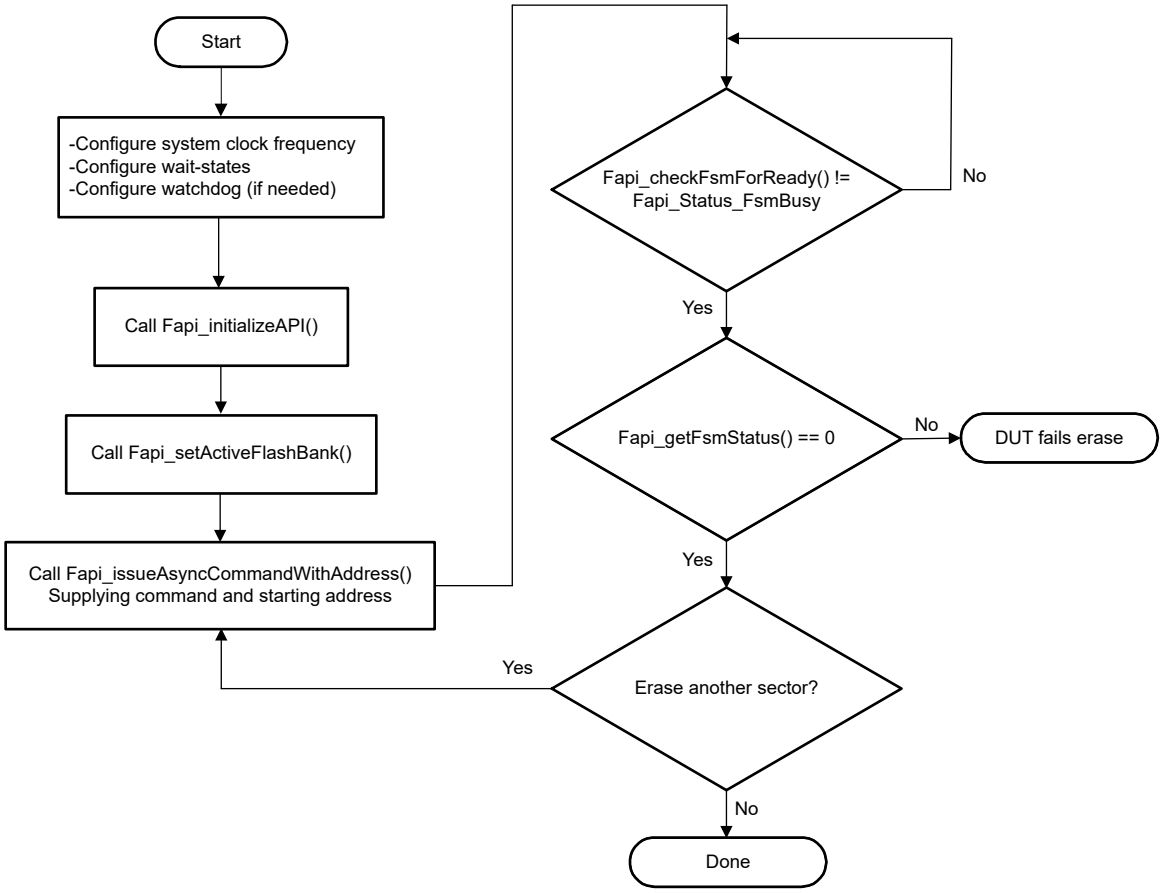

图 4-1. 推荐的擦除流程

备注

在发出扇区擦除命令之前，可发出 Fapi_ClearStatus 命令，并且可根据应用程序的需要为扇区配置写/擦除保护。

4.3 推荐的闪存组擦除流程

图 4-2 描述了闪存存储体的擦除流程。有关更多信息，请参阅节 3.2.11、节 3.2.2、节 3.2.3。

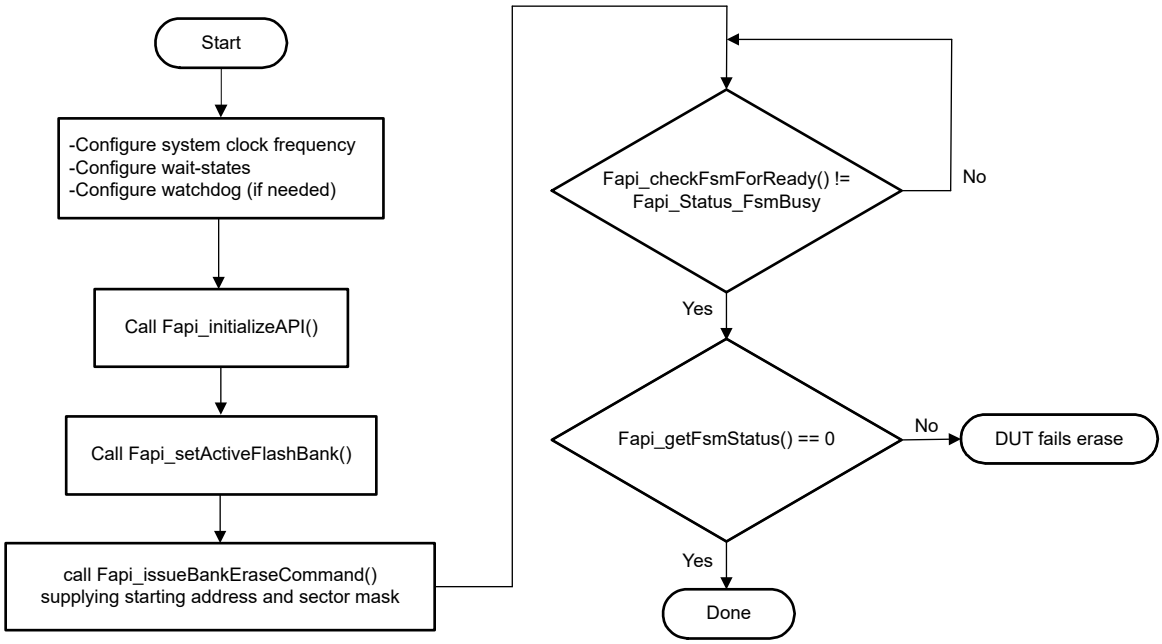

图 4-2. 推荐的存储体擦除程序流

备注

在发出编程命令之前，可发出 Fapi_ClearStatus 命令，并且可根据应用程序的需要为扇区配置写/擦除保护。

4.4 推荐的编程流程

图 4-3 描述了对器件进行编程的流程。该流程假定用户已经按照推荐的擦除流程擦除了所有受影响的扇区或闪存组。如需更多信息，请参阅节 3.2.11、节 3.2.2 和节 3.2.3。

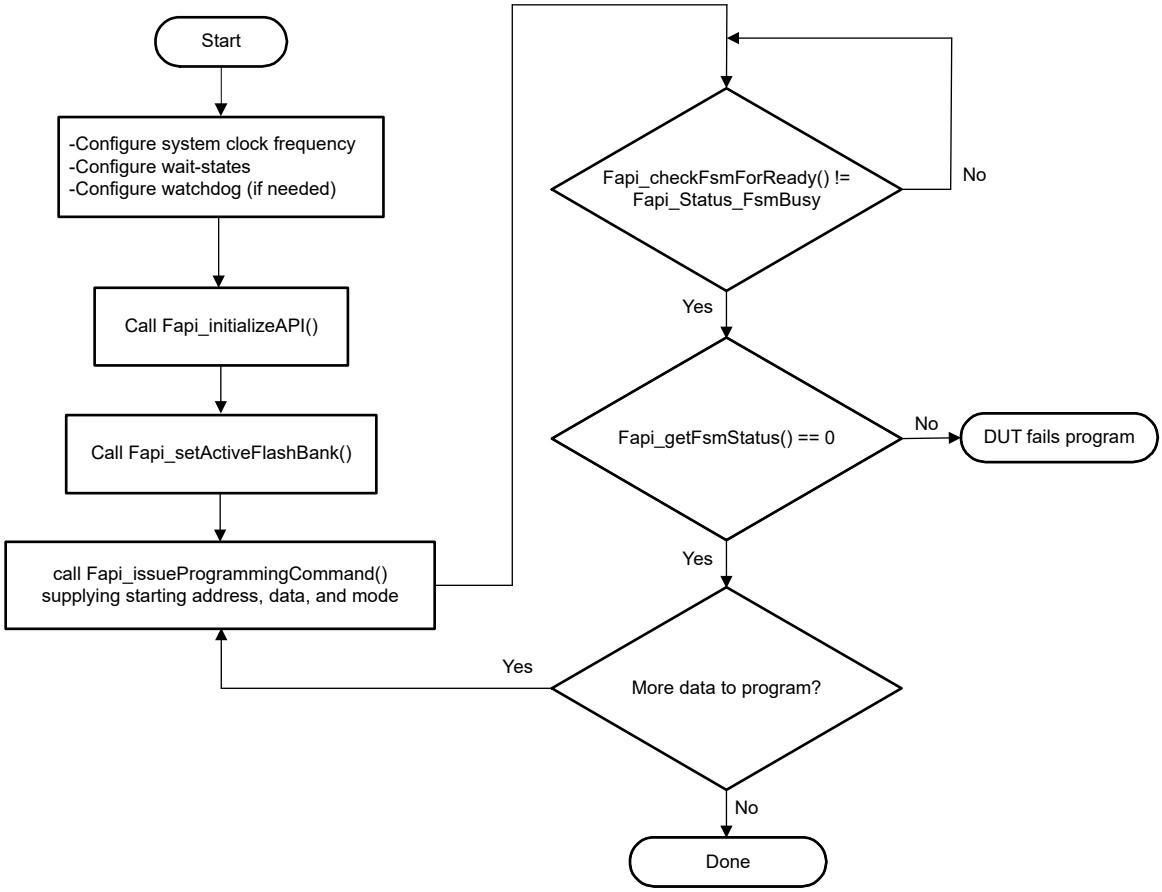

图 4-3. 推荐的编程流程

备注

在发出编程命令之前，应发出 Fapi_ClearStatus 命令，并且应根据应用程序的需要为扇区配置写/擦除保护。

5 与安全相关的软件应用程序使用假设

1. 在使用任何其他闪存 API 函数之前，必须执行 Fapi_initializeAPI() 函数一次。无需多次调用该函数。但是，只要用户应用程序在运行时更新系统频率或闪存等待状态配置（这种情况很少见），就必须调用该函数。
2. 在执行 Fapi_initializeAPI() 之前，用户应用程序必须确保为所需的输出频率正确配置 PLL。请注意，driverlib 中提供的 SysCtl_setClock() 通过使用 DCC 模块实现了这一点。SysCtl_setClock() 调用的 SysCtl_isPLLValid() 使用 DCC 模块来确保 PLL 输出处于预期范围内。有关 DCC 模块的更多详细信息，请参阅器件特定技术参考手册。
3. 如果用户应用程序需要保护闪存寄存器空间免受因代码失控等原因造成的任何损坏，则可以通过使用 ERAD 模块来实现。有关 ERAD 使用的详细信息，请参阅器件特定技术参考手册。
4. 当将 Fapi_BlankCheck() 用于闪存主阵列地址范围时，建议同时检查相应的 ECC 阵列地址范围。每个扇区的 ECC 地址范围在器件专用数据表的存储器映射 表中给出。
5. 用户应用程序软件必须根据需要使用 ECC_ENABLE 寄存器为闪存读取/获取路径启用 ECC 评估功能。闪存 API 不启用/禁用该寄存器。请注意，ECC_ENABLE 在上电时默认启用。因此，用户应用程序软件定义 NMI ISR 和闪存 single-bit 错误 ISR，以便能够响应单个和不可纠正的闪存错误。有关 FLASH_ECC_REGS 寄存器的更多详细信息，请参阅器件特定技术参考手册。

6. 如果用户应用程序的安全标准要求其定期读取整个闪存范围，从而确保数据的完整性，用户应用程序可以这样做，并计算整个闪存范围的校验和/CRC。VCU CRC 模块可用于实现这一点。有关 VCU CRC 模块的更多详细信息，请参阅器件特定技术参考手册。
7. 如果用户应用程序的安全标准要求其定期读取闪存寄存器空间，从而确保寄存器配置的完整性，应用程序可以这样做，并计算闪存寄存器存储范围的校验和/CRC。
8. 如果用户应用程序的安全标准要求其在进行编程和擦除操作后验证（使用 CPU 读取）闪存内容，它可以分别使用 Fapi_doVerify() 和 Fapi_doBlankCheck() 来验证闪存内容。
9. 闪存 API 不对看门狗进行配置/维护。用户应用程序可以配置看门狗并根据需要对其进行维护（在闪存 API 函数调用之间进行串行或使用 CPU 计时器 ISR）。
10. 如果 RAM 或闪存中的闪存 API 代码由于应用程序设计不当（例如，堆栈溢出问题）而被覆盖，则当 CPU 尝试执行闪存 API 函数时，可能会出现非法指令陷阱 (ITRAP)。因此，用户应用程序定义 ITRAP ISR 来响应此类事件。

A 闪存状态机命令

表 A-1. 闪存状态机命令

命令	说明	枚举类型	API 调用
Program Data	用于将数据编程到任何有效的闪存地址	Fapi_ProgramData	Fapi_issueProgrammingCommand() Fapi_issueProgrammingCommandForEccAddresses() Fapi_issueAutoEcc512ProgrammingCommand() Fapi_issueDataOnly512ProgrammingCommand() Fapi_issueDataAndEcc512ProgrammingCommand() Fapi_issueEccOnly64ProgrammingCommand()
Erase Sector	用于擦除指定地址处的闪存扇区	Fapi_EraseSector	Fapi_issueAsyncCommandWithAddress()
Erase Bank	用于擦除闪存组	Fapi_EraseBank	Fapi_issueBankEraseCommand()
Clear Status	清除状态寄存器	Fapi_ClearStatus	Fapi_issueAsyncCommand()

B typedef、定义、枚举和结构

B.1 类型定义

```
#if defined(__TMS320C28XX__)
typedef unsigned char boolean;
typedef unsigned int uint8; /*This is 16 bits in C28x*/
typedef unsigned int uint16;
typedef unsigned long int uint32;
typedef unsigned long long int uint64;
#endif
```

B.2 定义

```
#if (defined(__TMS320C28xx__) && __TI_COMPILER_VERSION__ < 6004000)
#if !defined(__GNUC__)
#error “F021 Flash API requires GCC language extensions. Use the -gcc option.”
#endif
#endif
#ifndef TRUE
   #define TRUE             1
#endif
#ifndef FALSE
   #define FALSE            0
#endif
```

B.3 枚举

B.3.1 Fapi_FlashProgrammingCommandsType

其中包含 Fapi_IssueProgrammingCommand() 中使用的所有可能模式。

```
typedef enum
{
Fapi_AutoEccGeneration,   /* This is the default mode for the command and
will
                             auto generate the ecc for the provided data buffer */
   Fapi_DataOnly,         /* Command will only process the data buffer */
   Fapi_EccOnly,          /* Command will only process the ecc buffer */
   Fapi_DataAndEcc        /* Command will process data and ecc buffers */
}  ATTRIBUTE_PACKED Fapi_FlashProgrammingCommandsType;
```

B.3.2 Fapi_FlashBankType

用于指示正在使用哪个闪存组。

```
typedef enum
{
   Fapi_FlashBank0
}  ATTRIBUTE_PACKED Fapi_FlashBankType;
```

B.3.3 Fapi_FlashStateCommandsType

其中包含所有可能的闪存状态机命令。

```
typedef enum
{
   Fapi_ProgramData    = 0x0002,
   Fapi_EraseSector    = 0x0006,
   Fapi_EraseBank      = 0x0008,
   Fapi_ClearStatus    = 0x0010,
}  ATTRIBUTE_PACKED Fapi_FlashStateCommandsType;
```

B.3.4 Fapi_StatusType

其为包含所有可能返回的状态代码的主类型。

```
typedef enum
{
   Fapi_Status_Success=0,           /* Function completed successfully */
   Fapi_Status_FsmBusy,             /* FSM is Busy */
   Fapi_Status_FsmReady,            /* FSM is Ready */
   Fapi_Status_AsyncBusy,           /* Async function operation is Busy */
   Fapi_Status_AsyncComplete,       /* Async function operation is Complete */
   Fapi_Error_Fail=500,             /* Generic Function Fail code */
   Fapi_Error_StateMachineTimeout,  /* State machine polling never returned ready and timed out */
   Fapi_Error_OtpChecksumMismatch,  /* Returned if OTP checksum does not match expected value */
   Fapi_Error_InvalidDelayValue,    /* Returned if the Calculated RWAIT value exceeds 15  - Legacy
Error */
   Fapi_Error_InvalidHclkValue,     /* Returned if FClk is above max FClk value - FClk is a
calculated from HClk and RWAIT/EWAIT */
   Fapi_Error_InvalidCpu,           /* Returned if the specified Cpu does not exist */
   Fapi_Error_InvalidBank,          /* Returned if the specified bank does not exist */
   Fapi_Error_InvalidAddress,       /* Returned if the specified Address does not exist in Flash or
OTP */
   Fapi_Error_InvalidReadMode,      /* Returned if the specified read mode does not exist */
   Fapi_Error_AsyncIncorrectDataBufferLength,
   Fapi_Error_AsyncIncorrectEccBufferLength,
   Fapi_Error_AsyncDataEccBufferLengthMismatch,
   Fapi_Error_FeatureNotAvailable,  /* FMC feature is not available on this device */
   Fapi_Error_FlashRegsNotWritable, /* Returned if Flash registers are not writable due to security
*/
   Fapi_Error_InvalidCPUID          /* Returned if OTP has an invalid CPUID */
}  ATTRIBUTE_PACKED Fapi_StatusType;
```

B.3.5 Fapi_ApiProductionStatusType

该结构用于在需要更大灵活性的函数中返回状态值。

```
typedef enum
{
   Alpha_Internal,          /* For internal TI use only.  Not intended to be used by customers */
   Alpha,                   /* Early Engineering release.  May not be functionally complete */
   Beta_Internal,           /* For internal TI use only.  Not intended to be used by customers */
   Beta,                    /* Functionally complete, to be used for testing and validation */
   Production               /* Fully validated, functionally complete, ready for production use */
}  ATTRIBUTE_PACKED Fapi_ApiProductionStatusType;
```

B.4 结构

B.4.1 Fapi_FlashStatusWordType

该结构用于在需要更大灵活性的函数中返回状态值。

```
typedef struct
{
   uint32 au32StatusWord[4];
}  ATTRIBUTE_PACKED Fapi_FlashStatusWordType;
```

B.4.2 Fapi_LibraryInfoType

用于返回 API 信息的结构如下。

```
typedef struct
{
   uint8  u8ApiMajorVersion;
   uint8  u8ApiMinorVersion;
   uint8  u8ApiRevision;
   Fapi_ApiProductionStatusType oApiProductionStatus;
   uint32 u32ApiBuildNumber;
   uint8  u8ApiTechnologyType;
   uint8  u8ApiTechnologyRevision;
   uint8  u8ApiEndianness;
   uint32 u32ApiCompilerVersion;
} Fapi_LibraryInfoType;
```

C FAPI 库 v5.00.00 的变更摘要

- 添加了 512 位编程的函数
  - 第 3.2.7 节 Fapi_issueAutoEcc512ProgrammingCommand()
  - 第 3.2.8 节 Fapi_issueDataAndEcc512ProgrammingCommand()
  - 第 3.2.9 节 Fapi_issueDataOnly512ProgrammingCommand()
  - 第 3.2.10 节 Fapi_issueEccOnly64ProgrammingCommand()

备注

不需要 512 位编程的用户可以继续使用 C2000ware 版本 C2000Ware_6_00_00_00 中提供的适用于 TMS320F28E12x 器件的闪存 API 库 FAPI_F28E12x_v5.00.00.lib。

- 更新了库版本号 v5.00.00
- 更新了第 2.3.2.3 节“闪存 API 使用的关键事实”，以添加 512 位允许的链路指针范围
- 使用 u8ApiRevision 和 oApiProductionStatus 更新了第 3.4.1 节“Fapi_getLibraryInfo()”函数

9 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from SEPTEMBER 25, 2025 to AUGUST 18, 2026 (from Revision * (September 2025) to Revision A (August 2026))	Page
• 向摘要添加了注释	1

重要通知和免责声明

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、与某特定用途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安全、安保法规或其他要求。

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 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月