User's Guide

F29H85x 闪存 API 版本 21.00.00.00

TEXAS INSTRUMENTS

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。本指南介绍了如何使用闪存 API 库对 F29H85x 器件的片上闪存存储器执行擦除、编程和验证操作。

内容

1 简介....3
1.1 与 C28x 的区别....3
1.2 函数清单格式....6
2 F29H85x 闪存 API 概述....7
2.1 简介....7
2.2 API 概述....7
2.3 使用 API....8
3 API 函数....10
3.1 初始化函数....10
3.2 闪存状态机函数....11
3.3 读取函数....27
3.4 信息函数....31
3.5 实用功能....31
4 使用闪存 API 对 SECCFG 和 BANKMGMT 编程....34
4.1 BANKMGMT 编程....35
4.2 SECCFG 编程....37
5 所有模式允许的编程范围....38
6 推荐的 FSM 流程....39
6.1 新出厂器件....39
6.2 推荐的擦除流程....39
6.3 推荐的存储组擦除流程....40
6.4 推荐的编程流程....41
7 参考资料....41
A 闪存状态机命令....42
B typedef、定义、枚举和结构....43
B.1 类型定义....43
B.2 定义....43
B.3 枚举....43
B.4 结构....46
修订历史记录....46

插图清单

图 1-1. F29H85x 闪存架构....5
图 6-1. 推荐的擦除流程....39
图 6-2. 推荐的闪存组擦除流程....40
图 6-3. 推荐的编程流程....41

表格清单

表 2-1. 初始化函数汇总....7
表 2-2. 闪存状态机 (FSM) 函数汇总....7

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

表 2-3. 读取函数汇总.....8
表 2-4. 信息函数汇总.....8
表 2-5. 实用程序函数汇总.....8
表 3-1. 不同编程模式的使用.....15
表 3-2. Fapi_issueProgrammingCommand() 的允许编程范围.....16
表 3-3. Fapi_issueAutoEcc512ProgrammingCommand() 的允许编程范围.....20
表 3-4. Fapi_issueDataAndEcc512ProgrammingCommand() 的允许编程范围.....21
表 3-5. Fapi_issueDataOnly512ProgrammingCommand() 的允许编程范围.....23
表 3-6. 64 位 ECC 数据解释.....24
表 3-7. Fapi_issueEccOnly64ProgrammingCommand() 的允许编程范围.....24
表 3-8. STATCMD 寄存器.....27
表 3-9. STATCMD 寄存器字段说明.....27
表 4-1. BANKMGMT 寄存器.....35
表 4-2. BANKMODE 值.....35
表 4-3. SECCFG 起始地址.....37
表 5-1. 主阵列范围.....38
表 5-2. BANKMGMT 编程范围.....38
表 5-3. SECCFG 编程范围.....38
表 A-1. 闪存状态机命令.....42

商标

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
所有商标均为其各自所有者的财产。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

1 简介

本参考指南详细描述了德州仪器 (TI) F29H85x 闪存 API 库 (F29H85x_NWFlashAPI_v21.00.00.00.lib) 函数，这些函数可用于擦除、编程和验证 F29H85x 器件上的闪存。请注意，闪存 API v21.00.00.00 仅能与 F29H85x 器件一起使用。闪存 API 库位于 F29H85x SDK 的“f29h85x-sdk > source > flash_api”中

1.1 与 C28x 的区别

C28 闪存 API 函数签名	C29 闪存 API 函数签名

`Fapi_FlashStatusType Fapi_getFsmStatus(void);`	`Fapi_FlashStatusType Fapi_getFsmStatus(`
	`uint32_t u32StartAddress,`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_checkFsmForReady(void);`	`Fapi_StatusType Fapi_checkFsmForReady(`
	`uint32_t u32StartAddress,`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_setActiveFlashBank(`
`Fapi_FlashBankType oNewFlashBank`
`);`
	不再需要/已弃用

`void Fapi_flushPipeline(void);`	`void Fapi_flushPipeline(`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_setupBankSectorEnable(`
`uint32_t reg_address,`
`uint32_t value`
`);`
	`Fapi_StatusType Fapi_setupBankSectorEnable(`
	`uint32_t *pu32StartAddress,`
	`uint32_t u32UserFlashConfig,`
	`uint32_t reg_address,`
	`uint32_t value`
	`);`

`Fapi_StatusType`
`Fapi_issueAsyncCommandWithAddress(`
`Fapi_FlashStateCommandsType oCommand,`
`uint32 *pu32StartAddress`
`);`
	`Fapi_StatusType`
	`Fapi_issueAsyncCommandWithAddress(`
	`Fapi_FlashStateCommandsType oCommand,`
	`uint32_t *pu32StartAddress,`
	`uint8_t u8Iterator,`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_issueAsyncCommand(`
`Fapi_FlashStateCommandsType oCommand`
`);`
	`Fapi_StatusType Fapi_issueAsyncCommand(`
	`uint32_t u32StartAddress,`
	`uint32_t u32UserFlashConfig,`
	`Fapi_FlashStateCommandsType oCommand`
	`);`

`Fapi_StatusType Fapi_issueBankEraseCommand(`
`uint32 *pu32StartAddress`
`);`
	`Fapi_StatusType Fapi_issueBankEraseCommand(`
	`uint32_t *pu32StartAddress,`
	`uint8_t u8Iterator,`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_doBlankCheck(`
`uint32 *pu32StartAddress,`
`uint32 u32Length,`
`Fapi_FlashStatusWordType`
`*poFlashStatusWord`
`);`
	`Fapi_StatusType Fapi_doBlankCheck(`
	`uint32_t *pu32StartAddress,`
	`uint32_t u32Length,`
	`Fapi_FlashStatusWordType`
	`*poFlashStatusWord,`
	`uint8_t u8Iterator,`
	`uint32_t u32UserFlashConfig`
	`);`

`Fapi_StatusType Fapi_doVerify(`
`uint32 *pu32StartAddress,`
`uint32 u32Length,`
`uint32 *pu32CheckValueBuffer,`
`Fapi_FlashStatusWordType`
`*poFlashStatusWord`
`);`
	`Fapi_StatusType Fapi_doVerify(`
	`uint32_t *pu32StartAddress,`
	`uint32_t u32Length,`
	`uint32_t *pu32CheckValueBuffer,`
	`Fapi_FlashStatusWordType`
	`*poFlashStatusWord,`
	`uint8_t u8Iterator,`
	`uint32_t u32UserFlashConfig`
	`);`

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

C28 闪存 API 函数签名	C29 闪存 API 函数签名
Fapi_StatusType Fapi_doVerifyBy16bits(uint16 *pu16StartAddress, uint32 u16Length, uint16 *pu16CheckValueBuffer, Fapi_FlashStatusWordType *poFlashStatusWord);	已弃用。使用 Fapi_StatusType Fapi_doVerifyByByte(uint8_t *pu8StartAddress, uint32_t u32Length, uint8_t *pu8CheckValueBuffer, Fapi_FlashStatusWordType *poFlashStatusWord, uint8_t u8Iterator, uint32_t u32UserFlashConfig);
Fapi_StatusType Fapi_issueProgrammingCommand(uint32 *pu32StartAddress, uint16 *pu16DataBuffer, uint16 u16DataBufferSizeInWords, uint16 *pu16EccBuffer, uint16 u16EccBufferSizeInBytes, Fapi_FlashProgrammingCommandsType oMode);	Fapi_StatusType Fapi_issueProgrammingCommand(uint32_t *pu32StartAddress, uint8_t *pu8DataBuffer, uint8_t u8DataBufferSizeInBytes, uint8_t *pu8EccBuffer, uint8_t u8EccBufferSizeInBytes, Fapi_FlashProgrammingCommandsType oMode, uint32_t u32UserFlashConfig);
Fapi_StatusType Fapi_issueDataOnly512ProgrammingCommand(uint32 *pu32StartAddress, uint16 *pu16DataBuffer, uint16 u16DataBufferSizeInWords);	Fapi_StatusType Fapi_issueDataOnly512ProgrammingCommand(uint32_t *pu32StartAddress, uint8_t *pu8DataBuffer, uint8_t u8DataBufferSizeInBytes, uint32_t u32UserFlashConfig, uint8_t u8Iterator);
Fapi_StatusType Fapi_issueAutoEcc512ProgrammingCommand(uint32 *pu32StartAddress, uint16 *pu16DataBuffer, uint16 u16DataBufferSizeInWords);	Fapi_StatusType Fapi_issueAutoEcc512ProgrammingCommand(uint32_t *pu32StartAddress, uint8_t *pu8DataBuffer, uint8_t u8DataBufferSizeInWords, uint32_t u32UserFlashConfig, uint8_t u8Iterator);
Fapi_StatusType Fapi_issueDataAndEcc512ProgrammingCommand(uint32 *pu32StartAddress, uint16 *pu16DataBuffer, uint16 u16DataBufferSizeInWords, uint16 *pu16EccBuffer, uint16 u16EccBufferSizeInBytes);	Fapi_StatusType Fapi_issueDataAndEcc512ProgrammingCommand(uint32_t *pu32StartAddress, uint8_t *pu8DataBuffer, uint8_t u8DataBufferSizeInWords, uint8_t *pu8EccBuffer, uint8_t u8EccBufferSizeInBytes, uint32_t u32UserFlashConfig, uint8_t u8Iterator);
Fapi_StatusType Fapi_issueEccOnly64ProgrammingCommand(uint32 *pu32StartAddress, uint16 *pu16EccBuffer, uint16 u16EccBufferSizeInBytes);	Fapi_StatusType Fapi_issueEccOnly64ProgrammingCommand(uint32_t *pu32StartAddress, uint8_t *pu8EccBuffer, uint8_t u8EccBufferSizeInBytes, uint32_t u32UserFlashConfig, uint8_t u8Iterator);
uint8 Fapi_calculateEcc(uint32 u32Address, uint64 u64Data);	uint8_t Fapi_calculateEcc(uint32_t *pu32Address, uint64_t *pu64Data, uint8_t u8Iterator);

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

与 F28P65x 器件不同，F29H85x 存储器模型使用交叉闪存组系统来跟上 CPU 处理速度。图 1-1 显示了闪存架构的图表。

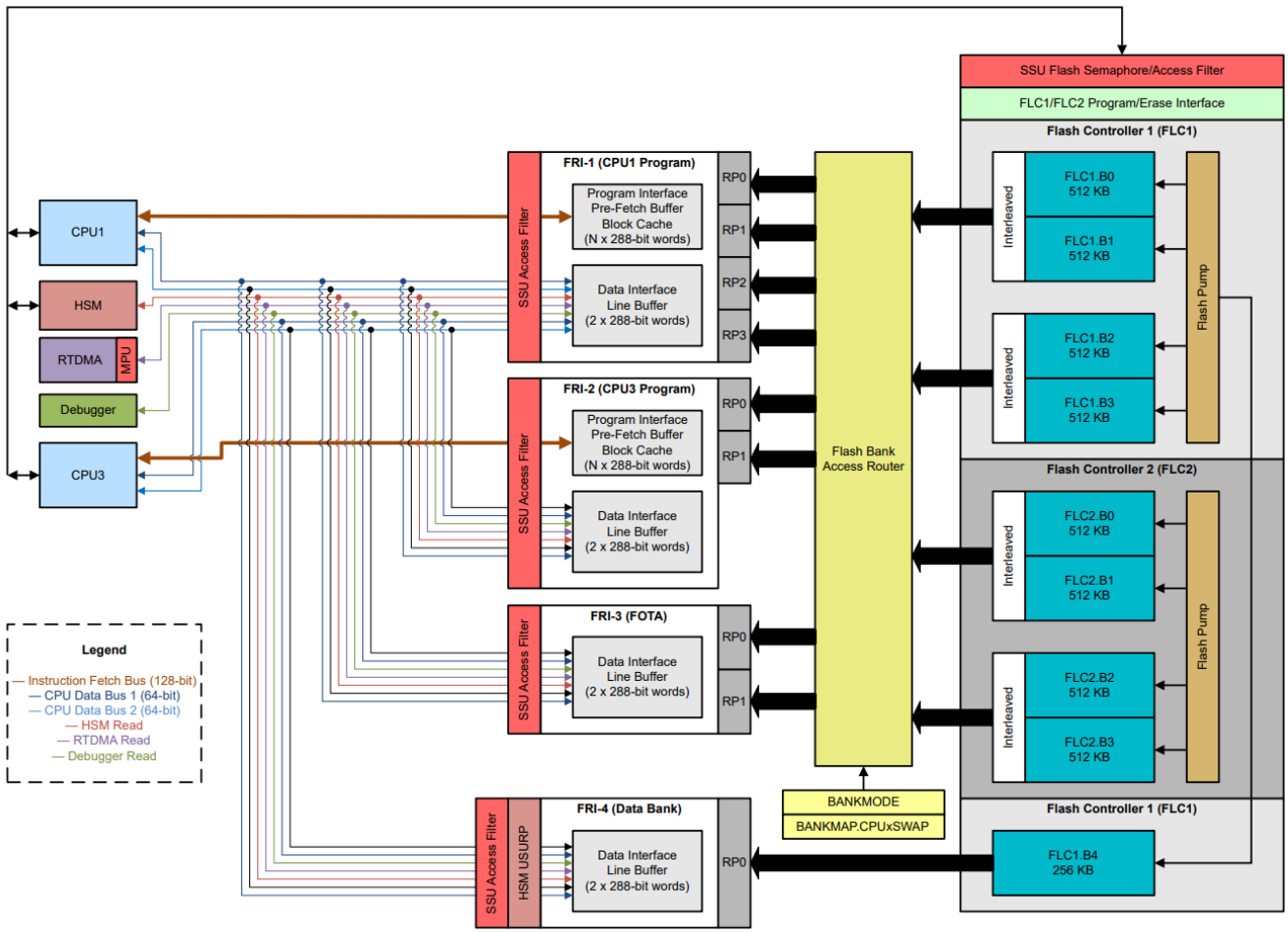

图 1-1. F29H85x 闪存架构

这意味着必须调用两次编程和擦除命令，每个交错组调用一次。具体过程由 FlashAPI 处理，现在读写通过最多四个闪存读取接口 (FRI-n) 而不是原始存储体基地址完成。有四种 BankMode 可供选择，这会影响 CPU 1 和 3 之间的存储器分配以及是否启用了交换。有关更多具体内容，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

为了满足这个新架构的要求，函数中增加了几个新参数，如上所示：

- u32StartAddress：告知闪存 API 要使用的闪存控制器 (FLCx)
- u8Iterator：供用户/API 跟踪第一个交错组和第二个交错组之间的命令迭代的附加参数
- u32UserFlashConfig：将有关存储体类型和 FOTA 的合并数据传递给闪存 API

此外、与 C28 的 16 位可寻址性不同，F29 系列器件可按字节/ 8 位寻址。

有关闪存架构的具文档，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

Copyright © 2025 Texas Instruments Incorporated

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_1> 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
- 函数有预设条件，但这些条件可能不明显
- 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）。这些示例除了使用闪存 API 函数外，还可以使用 F29H85x SDK 中 device_support 文件夹或 driverlib 文件夹提供的函数，来演示在应用程序环境中如何使用给定的闪存 API 函数。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

2 F29H85x 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 BANKMGMT 和 SECCFG 存储器编程。

备注

请阅读有关闪存存储器映射和闪存等待状态规范的数据手册。请注意，本参考指南假定用户已经阅读了《*F29H85x 和 F29P58x 实时微控制器技术参考手册*》中的“*闪存模块*”章节。另外，请特别注意该器件上的 Fapi_issueAsyncCommand()、Fapi_setupBankSectorEnable()、Fapi_issueBankEraseCommand() 函数。F29H85x SDK 中提供的闪存 API 使用示例演示了这些函数的使用方法。请参阅“f29h85x-sdk > examples > driverlib > single_core > flash”（适用于存储体模式 0）和“f29h85x-sdk > examples > driverlib > multi_core > flash”（适用于存储体模式 2）。

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
Fapi_setActiveFlashBank()	对闪存封装器和闪存组进行初始化从而执行擦除或编程命令。 已弃用。
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。
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
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() S	设置使用 AutoECC 生成模式进行 512 位（64 个字节）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（64 个字节）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（64 个字节）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（8 个字节）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

Copyright © 2025 Texas Instruments Incorporated

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
Fapi_doVerifyByByte()	根据提供的值按字节验证指定的闪存范围

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_getUserConfiguration()	根据所需的配置参数计算 u32UserFlashConfig 的值。
Fapi_setUserConfiguration()	提交 u32UserFlashConfig 值

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_getUserConfiguration()、Fapi_SetFlashCPUConfiguration() 和 Fapi_initializeAPI() 函数，然后才能使用任何其他 API 函数（Fapi_getLibraryInfo() 函数除外）。此过程设置了几个必要的用户可配置变量，并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 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 的内部状态变量

2.3.2 使用 API 进行构建

2.3.2.1 目标库文件

闪存 API 目标文件以 Arm® 标准 EABI elf 格式分发。

2.3.2.2 分布文件

以下 API 文件分布在 f29h85x-sdk > source > flash_api > flash 文件夹中：

- F29H85x_NWFlashAPI_v21.00.00.00.lib - 这是适用于 F29H85x 器件的 Flash API EABI elf 对象格式库（构建时启用 FPU32 标志）。F29H85x 闪存 API 未嵌入到该器件的引导 ROM 中，该 API 完全属于软件类型。为了使应用能够擦除或编程闪存（包括 BANKMGMT 和 SECCFG），该库文件必须链接到应用。
- 未提供 API 库的定点版本。
- 头文件：
  - FlashTech_F29H85x_C29x.h – F29H85x 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

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

- 以下包含文件不应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_C29x.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/F29H85x.h – F29H85x 器件的常量定义。

2.3.3 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。用户应用程序必须根据需要配置 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）。
- 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 F29H85x SDK driverlib 时钟配置函数。
- 闪存 API 不负责配置 SSU 寄存器或获取闪存信标。用户应用程序必须根据需要对其进行配置。有关这些寄存器的详细信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。
- 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。但无法从正在进行擦除/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所有 ISR（中断服务例程）必须从 RAM 或没有正在进行的活动擦除/编程操作的闪存组中执行。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 Sector
//
oReturnCheck = Fapi_issueProgrammingCommand(Fapi_EraseSector, u32Index,
                                    0, u32UserFlashConfig);

//
// Wait until the Flash erase operation is over
//
while(Fapi_checkFsmForReady(u32Index, u32UserFlashConfig) == Fapi_Status_FsmBusy);
```

- 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。
- 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/擦除周期只能编程一次。
- 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。
- 在 SSUMODE2 或 SSUMODE3 下，默认情况下无法执行验证/空白检查操作。如果用户要在 SSUMODE2 或 SSUMODE3 下执行验证/空白检查操作，则用户必须提供必要的读取 APR 权限。有关 SSU 配置的详细信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

- 闪存状态机还会在擦除/编程脉冲后执行内部验证操作，以验证操作是否成功。根据需要，使用提供的函数完成连续编程/编程验证循环（或擦除/擦除验证循环），以便对擦除/编程进行验证。如果闪存包装程序状态机未能在最大脉冲计数设置中配置的编程/擦除脉冲数内完全编程或擦除闪存中的所有目标位，则将在 STATCMD 寄存器中设置 FAILVERIFY 位。

3 API 函数

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 FLASHCONTROLLER1_BASE。
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（waitstate 值），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）

实现示例

（请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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。该函数在 F29 器件中已弃用，但出于与旧系统的兼容性原因，该功能仍然存在。

概要

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

参数

oNewFlashBank [in]	设置为有效的闪存组编号。始终使用 Fapi_FlashBank0，无论哪个闪存组用于任何 CPU 上的擦除/编程操作。

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

无论哪一个闪存组用于擦除和编程操作，用户应用程序都只需调用该函数一次，该函数可以与 Fapi_FlashBank0 一起使用。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（失败：FSM 正忙于执行另一个命令）
- **Fapi_Error_InvalidBaseRegCntlAddress**（失败：用户提供的闪存控制寄存器基地址与预期地址不匹配）
- **Fapi_Error_InvalidBank**（失败：器件上不存在指定的闪存组）
- **Fapi_Error_InvalidHclkValue**（失败：系统时钟与指定的等待值不匹配）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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_t *pu32StartAddress,
                              uint32_t u32UserFlashConfig,
                              uint32_t reg_address,
                              uint32_t value
                                        )
```

参数

pu32StartAddress [in]	正在写入的闪存地址
u32UserFlashConfig [in]	用户闪存配置位域
reg_address [in]	用于写/擦除保护配置的寄存器地址。保护掩码适用于交错对中的两个组。 对前 32 个 (0-31) 扇区使用 FLASH_NOWRAPPER_O_CMDWEPROTA。 对其余的主阵列 (32-128) 扇区使用 FLASH_NOWRAPPER_O_CMDWEPROTB。 对 BANKMGMT 和 SECCFG 使用 FLASH_NOWRAPPER_O_CMDWEPROTNM
value [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，所有闪存主扇区和非主扇区（BANKMGMT 和 SECCFG）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。必须在每次擦除和编程命令之前调用该函数，如 F29H85x SDK 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：
FLASH_NOWRAPPER_O_CMDWEPROTA、FLASH_NOWRAPPER_O_CMDWEPROTB、FLASH_NOWRAPPER_O_CMDWEPROTNM

CMDWEPROTA 寄存器用于配置组的前 32 个扇区（0 至 31）的保护。此寄存器中的每个位对应交错的组每个扇区。因此，编程时，用户必须配置此寄存器两次（为组对 B0/2 或 B1/3 中的每个组配置一次）。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。如果掩码中的某个位为 0，则会擦除或编程特定扇区。

CMDWEPROTB 寄存器用于配置对主阵列闪存组中交错扇区 32 - 127 的保护。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。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

CMDWEPROT_NM 寄存器用于配置 BANKMGMT 和 SECCFG 区域保护。只有此寄存器中的位 0 用于配置保护。如果位 0 被配置为 1，则 BANKMGMT 和 SECCFG 不被编程。如果位 0 配置为 0，则会为区域编程。此寄存器的其他位可配置为 1。BANKMGMT 和 SECCFG 区域都可以擦除和编程。

备注

每个闪存组没有单独的专用 CMDWEPROT_x 寄存器。因此，对于任何闪存组，必须在每个闪存擦除和编程命令之前配置这些寄存器。

返回值

- Fapi_Status_Success（成功）

实现示例

（请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”）

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_t *pu32StartAddress,
                                uint8_t  u8Iterator,
                                uint32_t u32UserFlashConfig
                                                  )
```

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
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）
uint32 u32UserFlashConfig [in]	用户闪存配置位域

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。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次）以擦除两个底层存储体。在这两次调用期间，128 位对齐的起始地址保持不变。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。用户应用程序还可以使用 Fapi_doBlankCheck() 函数来验证闪存是否被擦除。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_InvalidBaseRegCntlAddress**（失败：用户提供的闪存控制寄存器基地址与预期地址不符。）
- **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。用户可确保使用正确的 CPU 执行 API。）
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。）

实现示例

（请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”）

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_t *pu32StartAddress,
                                        uint8_t u8Iterator,
                                        uint32_t  u32UserFlashConfig
                                        )
```

参数

pu32StartAddress [in]	用于进行闪存组擦除操作的闪存组地址
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器。 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）
u32UserFlashConfig [in]	用户闪存配置位域

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。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次，起始地址保持不变）。

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_InvalidBaseRegCntlAddress**（失败：用户提供的闪存控制寄存器基地址与预期地址不符。）

实现示例

（请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”）

Copyright © 2025 Texas Instruments Incorporated

3.2.5 Fapi_issueProgrammingCommand()

设置数据并向有效的闪存、BANKMGMT 和 SECCFG 内存地址发出编程命令

概要

```
Fapi_StatusType Fapi_issueProgrammingCommand(
                            uint32_t *pu32StartAddress,
                            uint8_t  *pu8DataBuffer,
                            uint8_t  u8DataBufferSizeInBytes,
                            uint8_t  *pu8EccBuffer,
                            uint8_t  u8EccBufferSizeInBytes,
                            Fapi_FlashProgrammingCommandsType oMode,
                            uint32_t  u32UserFlashConfig
                            );
```

参数

pu32StartAddress [in]	闪存中的起始地址，用于对数据和 ECC 进行编程。起始地址可以始终为偶数。
pu8DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区可以为 64 位对齐。
u8DataBufferSizeInBytes [in]	数据缓冲区中的字节数
pu8EccBuffer [in]	指向 ECC 缓冲区地址的指针
u8EccBufferSizeInBytes [in]	ECC 缓冲区中的字节数
oMode [in]	ECC 模式
u32UserFlashConfig [in]	用户闪存配置位域

备注

pu8EccBuffer 可以包含与 64 位对齐主阵列、BANKMGMT 或 SECCFG 地址的数据对应的 ECC。pu8EccBuffer 的 LSB 与主阵列的低 32 位相对应，pu8EccBuffer 的 MSB 与主阵列的高 32 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，如表 3-1 中所述。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，如表 3-1 中所述。

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、pu8DataBuffer、u8DataBufferSizeInWords	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。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

表 3-1. 不同编程模式的使用（续）

编程模式 (oMode)	使用的参数	用途
Fapi_AutoEccGeneration	pu32StartAddress、 pu8DataBuffer、 u8DataBufferSizeInBytes	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、 pu8DataBuffer、 u8DataBufferSizeInBytes、 pu8EccBuffer、 u8EccBufferSizeInBytes	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	pu8EccBuffer、 u8EccBufferSizeInBytes	请参阅 Fapi_DataOnly 模式的用途描述。

表 3-2 展示了该函数允许的编程范围。

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

闪存 API	主阵列	ECC	BANKMGMT 和 SECCFG
Fapi_issueProgrammingCommand() 128 位， Fapi_AutoEccGeneration 模式	允许	允许	允许
Fapi_issueProgrammingCommand() 128 位，Fapi_DataOnly 模式	允许	不允许	允许
Fapi_issueProgrammingCommand() 128 位，Fapi_DataAndEcc 模式	允许	允许	允许
Fapi_issueProgrammingCommand() 128 位，Fapi_EccOnly 模式	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。BANKMGMT 和 SECCFG 扇区必须仅使用 AutoEccGeneration 进行编程。此外，不支持 BANKMGMT 和 SECCFG 扇区的 512 位编程（使用任何模式）。

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 16 字节不等。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编程 16 或 8 个字节。如果起始地址进行了 64 位对齐而不是 128 位对齐，则一次只能编辑 8 个字节。该选项还存在 Fapi_DataOnly 的数据限制。忽略参数 4 和 5。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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, palign(8)
 .cinit         : > FLASH, palign(8)
 .const         : > FLASH, palign(8)
 .init_array    : > FLASH, palign(8)
 .switch        : > FLASH, palign(8)
 }
```

如果不在闪存中对齐这些段，则必须跟踪段中不完整的 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，并且无法预想到某段可能从未对齐的地址开始。因此，假设提供的地址为 128 位对齐，则可能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 8 个字节，则 ECC 缓冲区必须为 1 字节。如果数据缓冲区长度为 16 个字节，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，可以根据需要同时编程 16 个或 8 个字节。如果起始地址进行了 64 位对齐而不是 128 位对齐，则一次只能编辑 8 个字节。

pu8EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu8EccBuffer 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。pu8EccBuffer 的 LSB 与主阵列的低 64 位相对应，pu8EccBuffer 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

pu8DataBuffer 和 pu8EccBuffer 的长度分别不可超过 16 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，用户应用程序可以使用 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。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_InvalidBaseRegCntlAddress**（失败：用户提供的闪存控制寄存器基地址与预期地址不匹配）
- **Fapi_Error_AsyncIncorrectDataBufferLength**（失败：指定的数据缓冲区大小不正确。此外，如果在对 BANKMGMT 或 SECCFG 空间进行编程时选择了 Fapi_EccOnly 模式，则也会返回该错误）
- **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。用户可以确保使用正确的 CPU 执行 API）。
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。）

实现示例

如要了解更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例：“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”。

3.2.6 Fapi_issueAutoEcc512ProgrammingCommand()

设置数据并向有效的闪存、BANKMGMT 和 SECCFG 内存地址发出 512 位（64 字节）AutoEcc 生成模式程序命令。

概要

```
Fapi_StatusType Fapi_issueAutoEcc512ProgrammingCommand( uint32_t
                *pu32StartAddress, uint8_t *pu8DataBuffer, uint8_t u8DataBufferSizeInWords, uint32_t
                u32UserFlashConfig, uint8_t u8Iterator );
```

参数

pu32StartAddress [in]	用于对提供的数据和 ECC 进行编程的 1024 位对齐闪存地址
pu8DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区的地址可按 1024 位对齐。
u8DataBufferSizeInWords [in]	数据缓冲区中的字节数。最大 Databuffer 大小（以字为单位）不能超过 64。
u32UserFlashConfig [in]	用户闪存配置位域
uint8u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）

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

Copyright © 2025 Texas Instruments Incorporated

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

512 位编程一次对一个交替组进行编程。因此，必须调用两次，使用每个迭代器调用一次（共 1024 位数据），才能获得完全连续的数据。每个调用的 pu32StartAddress 必须保持不变。如果两个交替组都未编程，则数据与以下内容类似：

地址	数据
0x10000000	数据字节 0-15
0x10000010	0xFFFF
0x10000020	数据字节 16-31
0x10000030	0xFFFF

以此类推。如果数据缓冲区的顺序很重要，则可以在该函数的示例实现中找到如何正确格式化缓冲区以使其在闪存中保持其顺序的示例。（“f29h85x-sdk > examples > driverlib > single_core > FLASH > flash_mode0_512_program”）。

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。

下面我们举例说明：

```
SECTIONS { .text : > FLASH, palign(64) .cinit : > FLASH, palign(64)
                .const : > FLASH, palign(64) .init_array : > FLASH, palign(64) .switch : >
                FLASH, palign(64) }
```

如果不在闪存中对齐这些段，则必须跟踪段中不完整的 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，并且无法预想到某段可能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）。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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	ECC	BANKMGMT 和 SECCFG
Fapi_issueAutoEcc512ProgrammingCommand()	允许	允许	不允许

备注

pu8DataBuffer 的长度不可超过 64。

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，用户应用程序可以使用 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 512 位进行编程（鉴于提供的地址进行了 512 位对齐）。如果用户想对更多位进行编程，则应循环调用该函数，从而一次对 512 位进行编程。
- 主阵列闪存编程必须与 512 位地址边界对齐，并且在每个写入或擦除周期，只能对 64 个字节执行一次编程。
- 以 BANKMGMT 或 SECCFG 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 BANKMGMT 或 SECCFG 空间进行编程时选择了 Fapi_EccOnly 模式，则会返回该错误。）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可以确保使用正确的 CPU 执行 API）。
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。）

实现示例

（有关更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

3.2.7 Fapi_issueDataAndEcc512ProgrammingCommand()

设置使用用户提供的闪存数据和 ECC 数据进行 512 位（64 个字节）编程的闪存状态机寄存器，并向有效的闪存存储器发出编程命令。

概要

```
Fapi_StatusType Fapi_issueDataAndEcc512ProgrammingCommand(
                                    uint32_t *pu32StartAddress,
                                    uint8_t  *pu8DataBuffer,
                                    uint8_t  u8DataBufferSizeInWords,
                                    uint8_t  *pu8EccBuffer,
                                    uint8_t  u8EccBufferSizeInBytes,
                                    uint32_t u32UserFlashConfig,
                                    uint8_t  u8Iterator
                                    )
```

参数

pu32StartAddress [in]	用于对提供的数据和 ECC 进行编程的 1024 位对齐闪存地址
pu8DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区的地址可按 1024 位对齐。
u8DataBufferSizeInWords [in]	数据缓冲区中的字节数。最大 Databuffer 大小（以字为单位）不能超过 64。
pu8EccBuffer [in]	指向 ECC 缓冲区地址的指针
u8EccBufferSizeInBytes [in]	ECC 缓冲区中的字节数。最大 Eccbuffer 大小（以字为单位）不能超过 16。
u32UserFlashConfig [in]	用户闪存配置位域
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）

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 64 个字节，则 ECC 缓冲区必须为 8 个字节（1 个 ECC 字节对应于 64 位数据）。

pu8EccBuffer 的每个字节与 pu8DataBuffer 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	ECC	BANKMGMT 和 SECCFG
Fapi_issueDataAndEcc512ProgrammingCommand()	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，且每次写入或擦除周期中仅能编程 64 字节。
- 以 BANKMGMT 或 SECCFG 地址开头的 512 位地址范围将始终使用 128 位 Fapi_issueProgrammingCommand() 进行编程。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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 BANKMGMT 或 SECCFG 空间进行编程时选择了 Fapi_EccOnly 模式，则会返回该错误。）
- **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。用户可确保使用正确的 CPU 执行 API。）
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。）

实现示例

（有关更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”）。

3.2.8 Fapi_issueDataOnly512ProgrammingCommand()

设置使用用户提供的闪存数据进行 512 位（64 个字符）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,
                                    uint8 *pu8DataBuffer,
                                    uint8 u8DataBufferSizeInBytes,
                                    uint32 u32UserFlashConfig,
                                    uint8 u8Iterator
                                                        )
```

参数

pu32StartAddress [in]	用于对提供的数据进行编程的 1024 位对齐闪存地址。
pu8DataBuffer [in]	指向数据缓冲区地址的指针。数据缓冲区可以为 1024 位对齐。
u8DataBufferSizeInBytes [in]	数据缓冲区中 8 位字的数量。最大 Databuffer 大小（以字节为单位）不应超过 64。
u32UserFlashConfig [in]	用户闪存配置位域
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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，有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），从而在运行时检查单错校正和双错检测 (SECDED) 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 对数据进行编程，然后使用 Fapi_issueEccOnly64ProgrammingCommand() API 对 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	ECC	BANKMGMT 和 SECCFG
Fapi_issueDataOnly512ProgrammingCommand()	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，并且在每个写入或擦除周期，只能对 64 个字节执行一次编程。
- 以 BANKMGMT 或 SECCFG 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 BANKMGMT 或 SECCFG 空间进行编程时选择了 Fapi_EccOnly 模式，则会返回该错误。）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。确保使用正确的 CPU 执行 API）。
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。

实现示例

（有关更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”）

3.2.9 Fapi_issueEccOnly64ProgrammingCommand()

设置使用用户提供的 ECC 数据进行 64 位（8 个字节）编程的闪存状态机寄存器，并向有效闪存、BANKMGMT 和 SECCFG 内存发出编程命令。

概要

```
Fapi_StatusType Fapi_issueEccOnly64ProgrammingCommand(
                                    uint32_t *pu32StartAddress,
                                    uint8_t *pu8EccBuffer,
                                    uint8_t  u8EccBufferSizeInBytes,
                                    uint32_t u32UserFlashConfig,
                                    uint8_t  u8Iterator
                                    );
```


English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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。
pu8EccBuffer [in]	指向 ECC 缓冲区地址的指针
u8EccBufferSizeInBytes [in]	ECC 缓冲区中的字节数。最大 Eccbuffer 大小（以字节为单位）不应超过 16。
u32UserFlashConfig	用户闪存配置。
u8Iterator	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）

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）			
1st（128 位数据）	**2nd（128 位数据）**	**3rd（128 位数据）**	**4th（128 位数据）**
pu8EccBuffer[0] 的 LSB	pu8EccBuffer[1] 的 LSB	pu8EccBuffer[2] 的 LSB	pu8EccBuffer[3] 的 LSB
pu8EccBuffer[0] 的 MSB	pu8EccBuffer[1] 的 MSB	pu8EccBuffer[2] 的 MSB	pu8EccBuffer[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	ECC	BANKMGMT 和 SECCFG
Fapi_issueEccOnly64ProgrammingCommand()	不允许	允许	不允许

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 BANKMGMT 或 SECCFG 位置对 ECC 进行编程。以 BANKMGMT 或 SECCFG 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_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。用户可确保使用正确的 CPU 执行 API。）
- **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 *F29H85x 和 F29P58x 实时微控制器数据表*。）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

实现示例

（有关更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”）

3.2.10 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(
                            uint32 u32StartAddress,
                            uint32 u32UserFlashConfig,
                            Fapi_FlashStateCommandsType oCommand
                                        )
```

参数

u32StartAddress [in]	闪存中用于编程/擦除/验证的 32 位起始地址。
u32UserFlashConfig [in]	用户闪存配置位域
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，可以在每个编程和擦除命令之前发出 Fapi_ClearStatus 命令（仅在 STATCMD 为非零时），如 F29H85x SDK 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

（有关更多信息，请参阅 F29H85x SDK 中提供的闪存编程示例“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

3.2.11 Fapi_checkFsmForReady()

返回闪存状态机的状态

概要

```
Fapi_StatusType Fapi_checkFsmForReady(
                      uint32 u32StartAddress,
                      uint32 u32UserFlashConfig
                                    )
```

参数

u32StartAddress [in]	闪存中用于编程/擦除/验证的 32 位起始地址
u32UserFlashConfig	用户闪存配置

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）

3.2.12 Fapi_getFsmStatus()

根据提供的地址返回相应 FLC（FLC1 或 FLC2）的 STATCMD 寄存器值

概要

```
Fapi_FlashStatusType Fapi_getFsmStatus(
                           uint32 u32StartAddress,
                           uint32 u32UserFlashConfig
                                         )
```

参数

u32StartAddress [in]	闪存中用于编程/擦除/验证的 32 位起始地址
u32UserFlashConfig [in]	用户闪存配置位域

说明

此函数根据提供的地址返回相应 FLC（FLC1 或 FLC2）的 STATCMD 寄存器值。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。每个闪存控制器（FLC1 和 FLC2）都有自己的 STATCMD 寄存器。用户应用程序可以检查相应寄存器的值，以确定每次擦除和编程操作后是否存在任何故障。

Copyright © 2025 Texas Instruments Incorporated

返回值

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	命令成功 - 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,
                        uint8 u8Iterator,
                        uint32 u32UserFlashConfig
                                )
```


English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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 [in/out]	如果结果不是 Fapi_Status_Success，则返回操作的状态 -> 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
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）
u32UserFlashConfig [in]	用户闪存配置位域

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。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次，起始地址保持不变）。

在 SSUMODE2 和 SSUMODE3 中，用户无法执行空白检查操作。如果用户希望在 SSUMODE2 或 SSUMODE3 中执行空白检查操作，则用户可以提供必要的读取 APR 权限。有关 SSU 配置的详细信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

请注意，闪存状态机还会在擦除/编程脉冲后执行内部验证操作，以验证操作是否成功。根据需要，使用提供的函数完成连续编程/编程验证循环（或擦除/擦除验证循环），以便对擦除/编程进行验证。如果闪存包装程序状态机未能在最大脉冲计数设置中配置的编程/擦除脉冲数内完全编程或擦除闪存中的所有目标位，则将在 STATCMD 寄存器中设置 FAILVERIFY 位。

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。有关有效地址范围的信息，请参阅 *F29H85x 和 F29P58x 实时微控制器数据表*。

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,
                    uint8 u8Iterator,
                    uint32 u32UserFlashConfig
                           )
```


English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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）
pu32CheckValueBuffer [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 [in/out]	如果结果不是 Fapi_Status_Success，则返回操作的状态 -> au32StatusWord[0] 第一个验证失败位置的地址 -> au32StatusWord[1] 在首次验证失败位置处读取的数据 -> au32StatusWord[2] 比较数据的值 -> au32StatusWord[3] 不适用
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）
u32UserFlashConfig [in]	用户闪存配置位域

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。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次，起始地址保持不变）。

用户无法在 SSUMODE2 和 SSUMODE3 下执行验证操作。如果用户希望在 SSUMODE2 或 SSUMODE3 下执行验证操作，则用户可以提供必要的读取 APR 权限。有关 SSU 配置的详细信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

请注意，闪存状态机还会在擦除/编程脉冲后执行内部验证操作，以验证操作是否成功。根据需要，使用提供的函数完成连续编程/编程验证循环（或擦除/擦除验证循环），以便对擦除/编程进行验证。如果闪存包装程序状态机未能在最大脉冲计数设置中配置的编程/擦除脉冲数内完全编程或擦除闪存中的所有目标位，则将在 STATCMD 寄存器中设置 FAILVERIFY 位。

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。有关有效地址范围的信息，请参阅 *F29H85x 和 F29P58x 实时微控制器数据表*。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

3.3.3 Fapi_doVerifyByByte()

根据提供的数据按字节验证指定的区域

概要

```
Fapi_StatusType Fapi_doVerifyByByte(
                              uint8_t *pu8StartAddress,
                              uint32_t u32Length,
                              uint8_t *pu8CheckValueBuffer,
                              Fapi_FlashStatusWordType *poFlashStatusWord,
                              uint8_t u8Iterator,
                              uint32_t  u32UserFlashConfig
                              );
```

参数

pu32StartAddress [in]	要验证区域的起始地址
u32Length [in]	要验证的区域长度
pu8CheckValueBuffer [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 [in/out]	如果结果不是 Fapi_Status_Success，则返回操作的状态 -> au32StatusWord[0] 第一个验证失败位置的地址 -> au32StatusWord[1]在首次验证失败位置处读取的数据 -> au32StatusWord[2] 比较数据的值 -> au32StatusWord[3] 不适用
u8Iterator [in]	用于对交错组执行编程和擦除操作的迭代器 0：数据闪存/非交错 1：B0 或 B2（取决于提供的地址） 2：B1 或 B3（取决于提供的地址）
u32UserFlashConfig [in]	用户闪存配置位域

说明

该函数在从指定地址开始的指定长度的区域内，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。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次，起始地址保持不变）。

用户无法在 SSUMODE2 和 SSUMODE3 下执行验证操作。如果用户希望在 SSUMODE2 或 SSUMODE3 下执行验证操作，则用户可以提供必要的读取 APR 权限。有关 SSU 配置的详细信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

另请注意，闪存状态机还会在擦除/编程脉冲后执行内部验证操作，以验证操作是否成功。根据需要，使用提供的函数完成连续编程/编程验证循环（或擦除/擦除验证循环），以便对擦除/编程进行验证。如果闪存包装程序状态机未能在最大脉冲计数设置中配置的编程/擦除脉冲数内完全编程或擦除闪存中的所有目标位，则将在 STATCMD 寄存器中设置 FAILVERIFY 位。

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。有关有效地址范围的信息，请参阅 *F29H85x 和 F29P58x 实时微控制器数据表*。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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 21。
- u8ApiMinorVersion – 该 API 编译的次要版本号。F29H85x 器件的次要版本为 00。
- u8ApiRevision – 该 API 编译的修订版本号。此版本的这个值为 00。

该版本的修订版本号为 00。

- 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 F29H85x 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()

刷新闪存封装器流水线缓冲区

概要

```
void Fapi_flushPipeline(
                        uint32_t u32UserFlashConfig
                       )
```

参数

u32UserFlashConfig [in]	用户闪存配置位域

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。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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 *pu32Address
                uint64 *pu64Data,
                uint8 u8Iterator
                       )
```

参数

pu32Address [in]	指向计算 ECC 的 64 位值的地址的指针
pu64Data [in]	指向要计算 ECC 的 64 位值的地址的指针（可以按小端顺序排列）
u8Iterator [in]	用于对交错组执行编程/读取/擦除的迭代器 1：对于 128 位，用于执行编程和读取 2：对于 512 位，用于执行编程、读取和擦除

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。TMS320F28P65x 闪存 API 负责处理。在交错组上操作时，必须调用此函数两次（每个迭代器值调用一次，起始地址保持不变）。

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()

表示地址位于闪存封装器 ECC 空间内

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。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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_getUserConfiguration()

根据用户定义的配置参数计算闪存 API 配置位域。

概要

```
uint32_t Fapi_getUserConfiguration(
                                   Fapi_FlashBankType    BankType,
                                   Fapi_FOTAStatus   FOTAStatus
                                   );
```

参数

BankType [in]	闪存 API 将要写入的存储体类型。可以始终是 C29Bank。
FOTAStatus [in]	是否启用 FOTA： FOTA_Image：已启用 FOTA Active_Bank：已禁用 FOTA

说明

该函数根据用户定义的配置参数计算闪存 API 配置位域。

返回值

- 表示用户设置的 32 位位域

3.5.5 Fapi_setFlashCPUConfiguration()

提交用户闪存配置设置

概要

```
uint32_t Fapi_SetFlashCPUConfiguration(
                                   Fapi_BankMode    u32BankModeValue,
                                   );
```

参数

u32BankModeValue [in]	用户闪存配置位域

说明

该函数会根据用户定义的组模式参数提交闪存 API 组模式配置。这决定了传递到闪存 API 的地址范围，与器件的 BANKMODE 寄存器无关。用户必须始终在更新器件的 BANKMODE 之后或作为初始化的一部分调用该函数。

例如，如果 BANKMODE 寄存器 = 0x3（模式 0）并且组模式 1 被传递给该函数，则在使用 Fapi_* 函数时，用户可以使用组模式 1 的地址范围。直接从闪存读取时，用户可以使用组模式 0 的地址范围。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可以确保使用正确的 CPU 执行 API）。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

3.5.6 Fapi_issueProgBankMode()

擦除指定 FLC 的 BANKMGMT 扇区，然后对其进行编程。

概要

```
Fapi_StatusType Fapi_issueProgBankMode(
                        Fapi_BankMgmtAddress    u32StartAddress,
                        Fapi_BankMode  u32BankMode,
                        Fapi_FlashStatusWordType   *poFlashStatusWord,
                        uint32_t    u32UserFlashConfig
);
```

参数

Fapi_BankMgmtAddress [in]	接收所发出编程命令的 FLC。
Fapi_BankMode [in]	器件编程的组模式
Fapi_FlashStatusWordType [in/out]	如果结果不是 Fapi_Status_Success，则返回操作的状态 -> 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
u32UserFlashConfig [in]	用户闪存配置位域

说明

此函数使用相应 FLC 中的给定组模式擦除非活动 BANKMGMT 扇区并对其进行编程。对 BANKMGMT 扇区进行编程后，可以发出外部重置 (XSRn)，以便启动 ROM 读取新值并将其写入 SSU 寄存器，完成组模式切换。

返回值

- **Fapi_Status_Success**（成功）
- **Fapi_Status_FsmBusy**（FSM 处于繁忙状态）
- **Fapi_Error_FlashRegsNotWritable**（失败：闪存寄存器写入失败。用户可以确保使用正确的 CPU 执行 API）。
- **Fapi_Error_FeatureNotAvailable**（失败：用户传递了不受支持的模式）

4 使用闪存 API 对 SECCFG 和 BANKMGMT 编程

每个闪存存储体由 2KB 的物理扇区组成。标称大小（例如 512KB）表示 MAIN 区域的大小。每个闪存组还包括两个特殊区域：

- SECCFG，用于存储 SSU 配置设置
- BANKMGMT，用于存储存储体模式设置和固件更新元数据

C29 闪存 API 支持对 SECCFG 和 BANKMGMT 扇区的编程。有关更多信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

备注

BANKMGMT 和 SECCFG 扇区必须始终在启用 AutoEccGeneration 的情况下进行编程。任何模式下（包括使用 AutoEccGeneration）都不支持 512 位编程。因此，BANKMGMT 和 SECCFG 区域必须始终使用 Fapi_issueProgrammingCommand() 与 AutoEccGeneration 一起编程。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

4.1 BANKMGMT 编程

用户可以使用 Fapi_issueProgBankMode() 函数对 BANKMGMT 进行编程。此内存范围也可以按照以下流程进行手动编程：

1. 发出擦除扇区命令。使用闪存 API 擦除 BANKMGMT 扇区时，用户可以始终提供有效的 FLC1 或 FLCS2 地址（参见下表）。

存储体模式	FLC1 BANKMGMT 扇区地址	FLC2 BANKMGMT 扇区地址
模式 0	0x10D8 0000	不适用
模式 1	0x10D8 0000	不适用
模式 2	0x10D8 0000	不适用
模式 3	0x10D8 0000	0x10D9 0000

备注

在对 BANKMGMT 扇区进行编程之前，用户可以始终使用 Fapi_issueAsyncCommandWithAddress() 发出擦除扇区命令。有关如何擦除扇区的示例，请参阅 F29 SDK 中于“f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”提供的闪存编程示例。

2. 向 BANKMGMT 扇区发出 128 位编程命令，使用下表确定要编程的正确值。使用闪存 API 对 BANKMGMT 扇区进行编程时，必须提供有效的 FLC1 地址。

表 4-1. BANKMGMT 寄存器

寄存器	值	注释
BANK_STATUS[63:0]	0x55555555_55555555	数据缓冲器中的偏移量 0（参见下面的代码片段）。
BANK_UPDATE_CTR[63:0]	0x00000000_00000000	数据缓冲区中的偏移为 8（见下方代码片段）。 对 BANKMGMT 扇区编程时，该值可始终设置为 0。闪存 API 会从活动 BANKMGMT 扇区内部读取计数器，将其减 1，然后将其编程到非活动扇区
BANKMODE[63:0]	请参阅 BANKMODE 值	数据缓冲器中的偏移量 16（参见下面的代码片段）。 包含当前 BANKMODE 值

表 4-2. BANKMODE 值

BANKMODE	BANKMODE[63:0] 值	带偏移的缓冲器
模式 0	0x03	Buffer[16] = 0x03
模式 1	0x06	Buffer[16] = 0x06
模式 2	0x09	Buffer[16] = 0x09
模式 3	0x0C	Buffer[16] = 0x0C

Copyright © 2025 Texas Instruments Incorporated

这里提供一个假设扇区已被擦除的编程流程示例：

```
uint8 Buffer[32] = {
    0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, // BANK_STATUS
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // BANK_UPDATE_CTR
    0x09, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // BANKMODE
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF  // Unused
};

ClearFSMStatus(u32Index, u32UserFlashConfig);

Fapi_setupBankSectorEnable((uint32 *)u32Index, u32UserFlashConfig,
FLASH_NOWRAPPER_O_CMDWEPROTNM, 0x00000000);

oReturnCheck = Fapi_issueProgrammingCommand((uint32 *)u32Index,Buffer +i,
        16, 0, 0, Fapi_AutoEccGeneration, u32UserFlashConfig);

while(Fapi_checkFsmForReady(u32Index, u32UserFlashConfig) == Fapi_Status_FsmBusy);

if(oReturnCheck != Fapi_Status_Success)
{
    //
    // Check Flash API documentation for possible errors
    //
    Example_Error(oReturnCheck);
}

oFlashStatus = Fapi_getFsmStatus(u32Index, u32UserFlashConfig);
if(oFlashStatus != 3)
{
    //
    //Check FMSTAT and debug accordingly
    //
    FMSTAT_Fail();
}
```

3. 对 BANKMGMT 扇区进行编程后，必须发出外部重置 (XSRn)，以便启动 ROM 读取新值并将其写入 SSU 寄存器。

有关更多信息，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

4.2 SECCFG 编程

SECCFG 扇区是闪存存储器的一部分，用于存储 SSU 用户配置或用户保护策略 (UPP)。每个 CPU 都有一个活动的 SECCFG 起始地址和一个备用（后备）SECCFG 起始地址。当使用闪存 API 擦除/编程 SECCFG 扇区时，可以使用备用（后备）SECCFG 扇区的起始地址。有关 SECCFG 配置详情，请参阅 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

备注

在使用闪存 API 对 SECCFG 进行编程时，用户必须始终提供整个扇区长度 (0x1000)。未使用的扇区位置可以写为 0xFF。

表 4-3. SECCFG 起始地址

存储体模式	CPUxSWAP	区域	CPU1/CPU2 存储体	CPU1 地址	CPU2 地址	CPU3 存储体	CPU3 地址
模式 0		有效	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D8 9000
		备选	FLC1.B2/B3	0x10D8 5000	0x10D8 5800	FLC2.B2/B3	0x10D8 D000
模式 1	SWAP = 0	有效	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D8 5000
		备选	FLC1.B2/B3	0x10D9 9000	0x10D9 9800	FLC2.B2/B3	0x10D9 D000
	SWAP = 1	有效	FLC1.B2/B3	0x10D8 1000	0x10D8 1800	FLC2.B2/B3	0x10D8 5000
		备选	FLC1.B0/B1	0x10D9 9000	0x10D9 9800	FLC2.B0/B1	0x10D9 D000
模式 2		有效	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D9 1000
		备选	FLC1.B2/B3	0x10D8 5000	0x10D8 5800	FLC2.B2/B3	0x10D9 5000
模式 3	SWAP = 0	有效	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D9 1000
		备选	FLC1.B2/B3	0x10D9 9000	0x10D9 9800	FLC2.B2/B3	0x10D9 D000
	SWAP = 1	有效	FLC1.B2/B3	0x10D8 1000	0x10D8 1800	FLC2.B2/B3	0x10D9 1000
		备选	FLC1.B0/B1	0x10D9 9000	0x10D9 9800	FLC2.B0/B1	0x10D9 D000

编程 SECCFG 的步骤如下：

1. 发出擦除扇区命令。当使用闪存 API 擦除 SECCFG 时，用户可以始终为起始地址提供一个后备（备用）FLC 地址。有关详细信息、请参阅 SECCFG 起始地址。
2. 发出 128 位编程命令来对 SECCFG 扇区进行编程。
   a. 用户可以始终使用 AutoEccGeneration 编程模式
   b. 起始地址可以为后备（备用）扇区地址。有关详细信息、请参阅 SECCFG 起始地址。
   c. 必须对整个扇区长度进行编程，未使用的位置可以编程为 0xFF。
3. 对 SECCFG 扇区进行编程后，可以发出外部重置 (XSRn)，以便启动 ROM 读取活动的 SECCFG 扇区值并将其写入必要的 SSU 寄存器。

Copyright © 2025 Texas Instruments Incorporated

5 所有模式允许的编程范围

备注

FOTA 范围为只读。要对它们进行编程，请使用 FOTA_Image 参数配置闪存 API，并传入活动区域地址。闪存 API 会在内部将地址转换为正确的目标。有关详细信息,请参阅 *F29H85x 和 F29P58x 实时微控制器数据表* 和 *F29H85x 和 F29P58x 实时微控制器技术参考手册*。

表 5-1. 主阵列范围

存储体模式	区域	CPU1 闪存地址范围	CPU3 闪存地址范围
模式 0	有效	0x1000 0000 - 0x1040 0000	不适用
	FOTA	不适用	不适用
模式 1	有效	0x1000 0000 - 0x1020 0000	不适用
	FOTA	0x1060 0000 - 0x1080 0000	不适用
模式 2	有效	0x1000 0000 - 0x1020 0000	0x1040 0000 - 0x1060 0000
	FOTA	不适用	不适用
模式 3	有效	0x1000 0000 - 0x1010 0000	0x1040 0000 - 0x1050 0000
	FOTA	0x1000 0000 - 0x1010 0000	0x1040 0000 - 0x1050 0000

表 5-2. BANKMGMT 编程范围

存储体模式	FLC1 BANKMGMT 范围	FLC2 BANKMGMT 范围
模式 0	0x10D8 0000 - 0x10D80 0FFF	不适用
模式 1	0x10D8 0000 - 0x10D80 0FFF	不适用
模式 2	0x10D8 0000 - 0x10D80 0FFF	不适用
模式 3	0x10D8 0000 - 0x10D80 0FFF	0x10D9 0000-0x10D9 0FFF

表 5-3. SECCFG 编程范围

存储体模式	CPUxSWAP	区域	CPU1/CPU2 存储体	FLC1 地址范围	CPU3 存储体	FLC2 地址范围
模式 0		备选	FLC1.B2/B3	0x10D8 5000 - 0x10D8 5FFF	FLC2.B2/B3	0x10D8 C000 - 0x10D8 CFFF
模式 1	SWAP = 0	备选	FLC1.B2/B3	0x10D9 9000-0x10D9 9FFF	FLC2.B2/B3	0x10D8
	SWAP = 1	备选	FLC1.B0/B1	0x10D9 9000-0x10D9 9FFF	FLC2.B0/B1	不适用
模式 2		备选	FLC1.B2/B3	0x10D8 5000 - 0x10D8 5FFF	FLC2.B2/B3	0x10D9 5000-0x10D9 5FFF
模式 3	SWAP = 0	备选	FLC1.B2/B3	0x10D9 9000 - 0x10D9 9FFF	FLC2.B2/B3	0x10D9 D000 - 0x10D9 DFFF
	SWAP = 1	备选	FLC1.B0/B1	0x10D9 9000-0x10D9 9FFF	FLC2.B0/B1	0x10D9 D000 - 0x10D9 DFFF

Copyright © 2025 Texas Instruments Incorporated

6 推荐的 FSM 流程

6.1 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。

6.2 推荐的擦除流程

图 6-1 描述了器件上扇区的擦除流程。有关详细信息，请参阅 3.2.11、3.2.2、3.2.3。

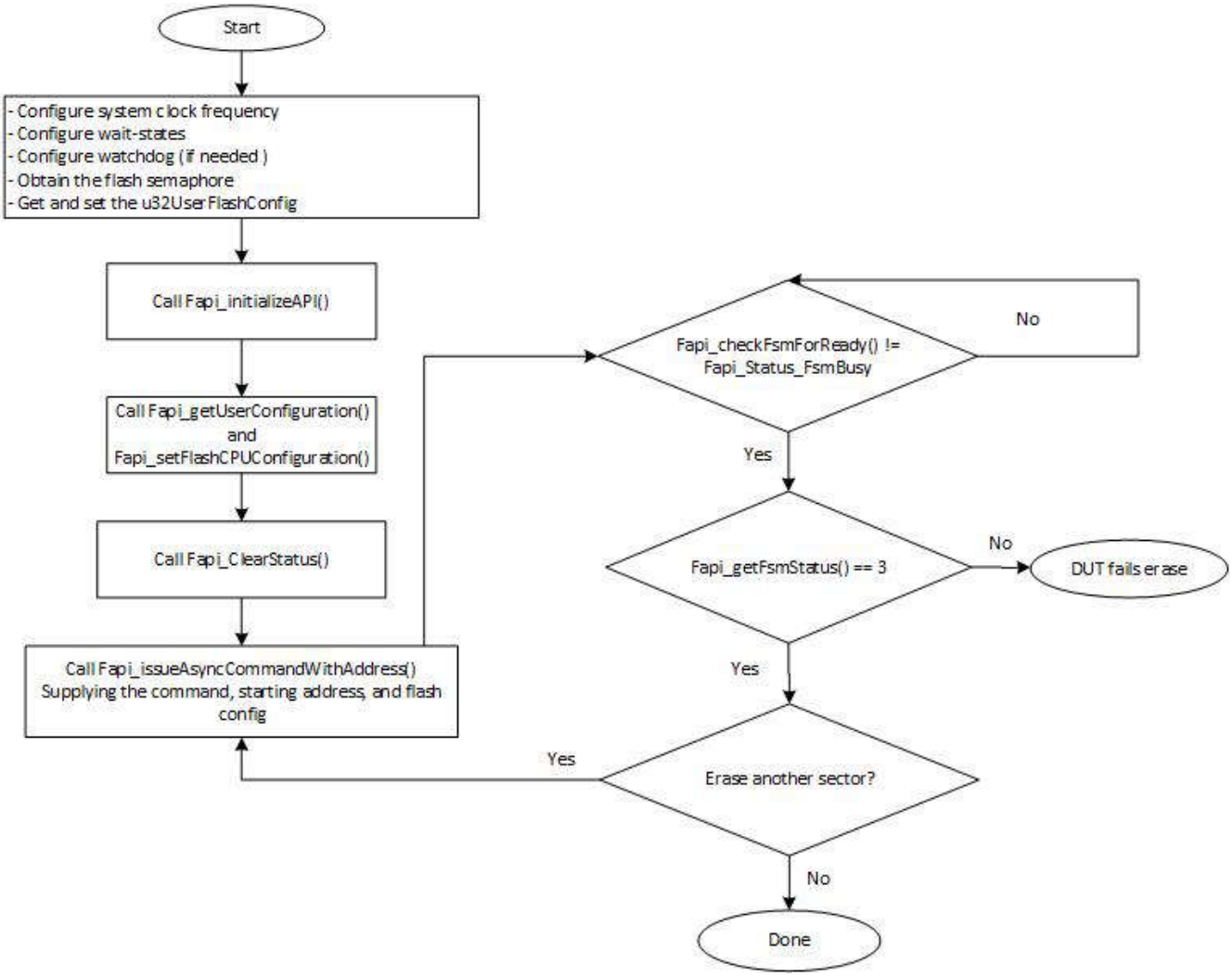

图 6-1. 推荐的擦除流程

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

6.3 推荐的存储组擦除流程

图 6-2 介绍了闪存组的擦除流程。有关详细信息，请参阅 3.2.11、3.2.2、3.2.4。

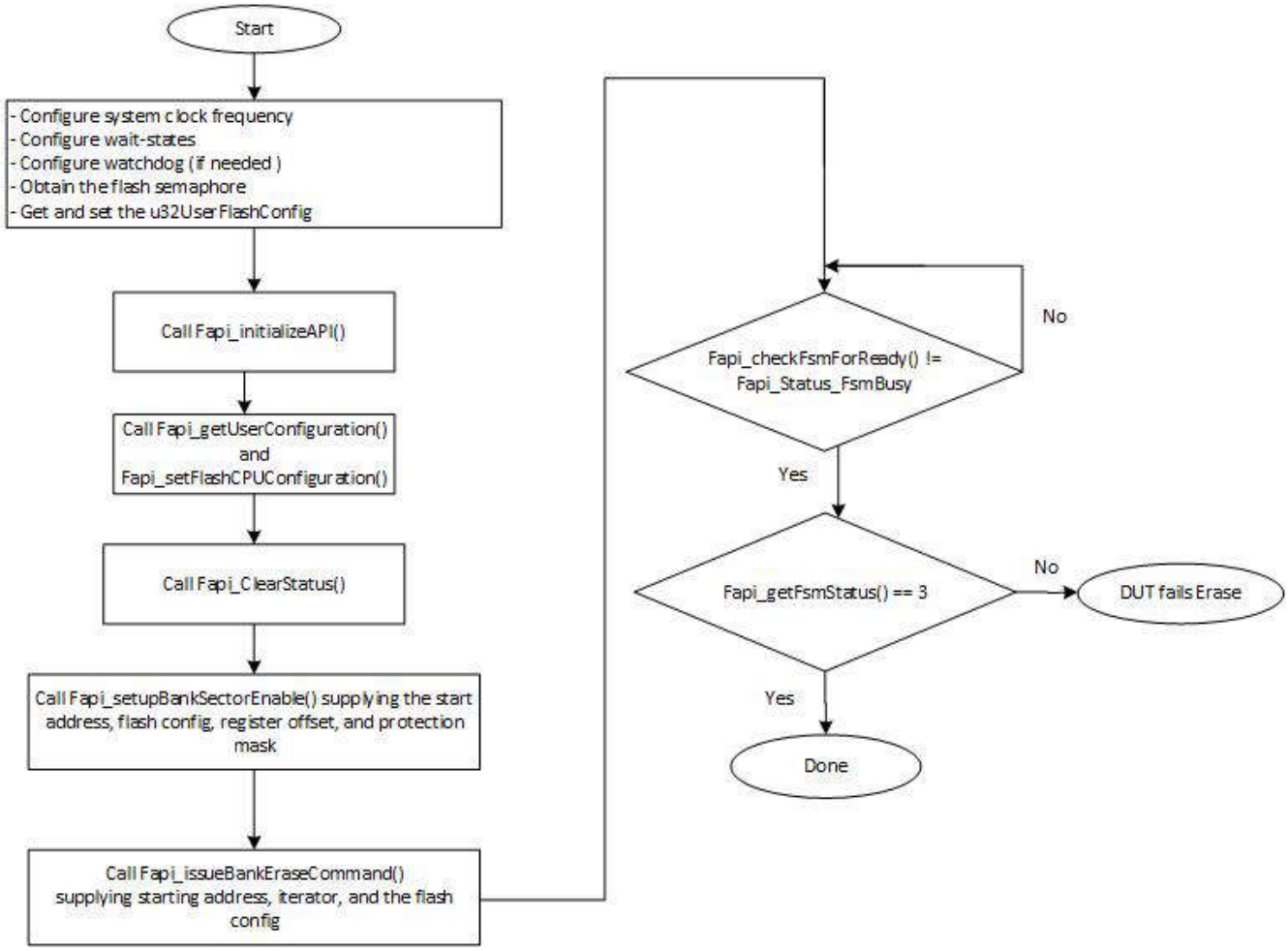

图 6-2. 推荐的闪存组擦除流程

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

6.4 推荐的编程流程

图 6-3 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。有关详细信息，请参阅 3.2.11、3.2.2、3.2.5。

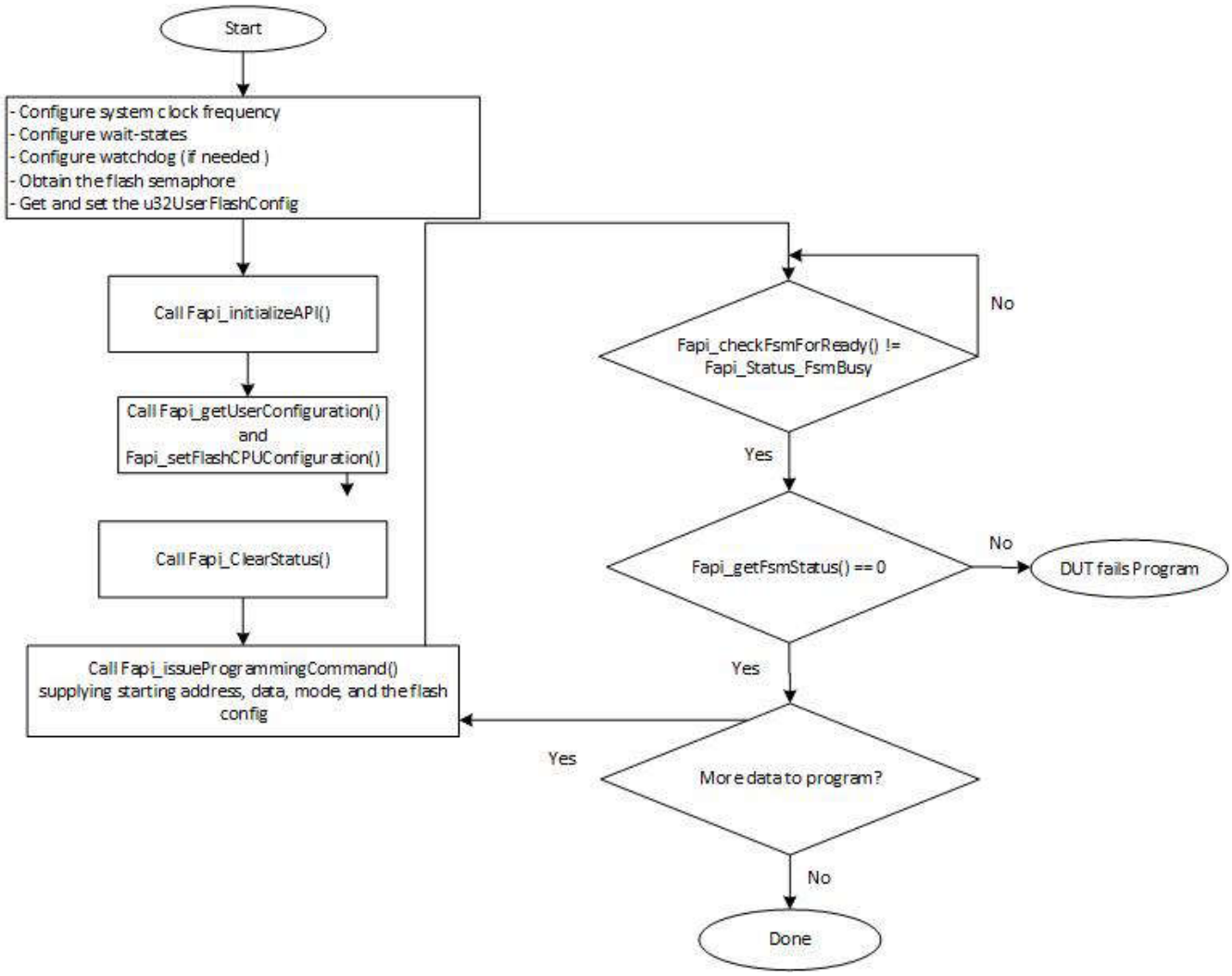

图 6-3. 推荐的编程流程

7 参考资料

- 德州仪器 (TI)：*F29H85x 和 F29P58x 实时微控制器数据表*
- 德州仪器 (TI)：*F29H85x 和 F29P58x 实时微控制器技术参考手册*

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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_issueDataAndEcc512ProgrammingCommand() Fapi_issueAutoEcc512ProgrammingCommand() Fapi_issueDataOnly512ProgrammingCommands() Fapi_issueDataAndEcc512ProgrammingCommand() Fapi_issueEccOnly65ProgrammingCommand()
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()

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

B typedef、定义、枚举和结构

B.1 类型定义

```
typedef unsigned char boolean;
typedef uint32_t Fapi_FlashStatusType;
```

B.2 定义

```
#define ATTRIBUTE_PACKED    __attribute__((packed))

#define HIGH_BYTE_FIRST     0
#define LOW_BYTE_FIRST      1

#ifndef TRUE
   #define TRUE              1
#endif

#ifndef FALSE
   #define FALSE             0
#endif

#if defined(_LITTLE_ENDIAN)
   #define CPU_BYTE_ORDER    LOW_BYTE_FIRST
#else
   #define CPU_BYTE_ORDER    HIGH_BYTE_FIRST
#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

这用于指示正在使用的闪存库类型。

```
typedef enum
{
    C29Bank             /* C29 CPU 1 */
}  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;
```


Copyright © 2025 Texas Instruments Incorporated

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

此处列出了 API 可能的不同生产状态值。

```
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.3.6 Fapi_BankID

其中包含所有可能的闪存组编号。

```
typedef enum
{
   Bank0,
   Bank1,
   Bank2,
   Bank3,
   Bank4
} ATTRIBUTE_PACKED Fapi_BankID;
```

B.3.7 Fapi_FLCID

包含 NW 控制器地址。

```
typedef enum
{
    FAPI_FLASHNW_FC1_BASE = (uint32_t)0x30100000U,
    FAPI_FLASHNW_FC2_BASE = (uint32_t)0x30110000U
} ATTRIBUTE_PACKED Fapi_FLCID;
```


English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

B.3.8 Fapi_BankMode

其中包含所有可行的存储体模式。

```
typedef enum
{
   Mode0 = 0x3,     /* CPU1 4MB, No FOTA, CPU1SWAP X CPU3SWAP X */
   Mode1 = 0x6,     /* CPU1 4MB, FOTA Enabled, CPU1SWAP 0/1 CPU3SWAP X*/
   Mode2 = 0x9,     /* CPU1 2MB CPU3 2MB, No FOTA, CPU1SWAP X CPU3SWAP X*/
   Mode3 = 0xC      /* CPU1 2MB CPU3 2MB, FOTA Enabled, (CPU1SWAP 1 CPU3SWAP 1) or (CPU1SWAP 0
CPU3SWAP 0)*/
} ATTRIBUTE_PACKED Fapi_BankMode;
```

B.3.9 Fapi_CPU1BankSwap

包含可行的 CPU1 闪存库映射配置。

```
typedef enum
{
   CPU1Swap0 = 0xC9,        //default mapping of CPU1 Banks
   CPU1Swap1 = 0x36,        //Alternate Mapping of CPU1 Banks
} ATTRIBUTE_PACKED Fapi_CPU1BankSwap;
```

B.3.10 Fapi_CPU3BankSwap

包含可行的 CPU3 闪存库映射配置。

```
typedef enum
{
   CPU3Swap0 = 0xC9,        //default mapping of CPU3 Banks
   CPU3Swap1 = 0x36,        //Alternate Mapping of CPU3 Banks
} ATTRIBUTE_PACKED Fapi_CPU3BankSwap;
```

B.3.11 Fapi_FOTAStatus

```
typedef enum
{
   FOTA_Image,    /* FOTA is enabled */
   Active_Bank    /* FOTA is disabled */
} ATTRIBUTE_PACKED Fapi_FOTAStatus;
```

B.3.12 Fapi_SECVALID

包含可行的 SECCFG 映射配置。

```
typedef enum
{
   Base = 0xC9U,    /* BASE addresses are valid */
   Alt = 0x36       /* ALT addresses are valid */
} ATTRIBUTE_PACKED Fapi_SECVALID;
```

B.3.13 Fapi_BankMgmtAddress

包含 BANKMGMT 编程的起始地址。

```
typedef enum
{
    Fapi_BankMgmtFLC1Address = (uint32_t)0x10d80000U,
    Fapi_BankMgmtFLC2Address = (uint32_t)0x10d90000U

} ATTRIBUTE_PACKED Fapi_BankMgmtAddress;
```


English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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_t 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_t  u8ApiMajorVersion;
   uint8_t  u8ApiMinorVersion;
   uint8_t  u8ApiRevision;
   Fapi_ApiProductionStatusType oApiProductionStatus;
   uint32_t u32ApiBuildNumber;
   uint8_t  u8ApiTechnologyType;
   uint8_t  u8ApiTechnologyRevision;
   uint8_t  u8ApiEndianness;
   uint32_t u32ApiCompilerVersion;
} Fapi_LibraryInfoType;
```

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 Revision * (January 2025) to Revision A (July 2025)	Page
• 节 3.5.5 中添加了有关组模式 1 和 3 的编程信息。	33
• 节 4 中添加了有关组模式 1 和 3 的信息。	34
• 节 5 中添加了所有模式下允许的编程范围的表。	38

English Document: SPRUJE7
Copyright © 2025 Texas Instruments Incorporated

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、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。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
版权所有 © 2025，德州仪器 (TI) 公司