User's Guide

F29H85x Flash API Version 21.00.00.00

ABSTRACT

Flash is an electrically erasable/programmable non-volatile memory that can be programmed and erased many times to ease code development. Flash memory can be used primarily as a program memory for the core, and secondarily as static data memory. This guide describes the usage of the flash API library to perform erase, program and verify operations for the on-chip Flash memory of F29H85x devices.

Table of Contents

1 Introduction 2
- 1.1 Differences From C28x 2
- 1.2 Function Listing Format 5

2 F29H85x Flash API Overview 6
- 2.1 Introduction 6
- 2.2 API Overview 6
- 2.3 Using API 7

3 API Functions 9
- 3.1 Initialization Functions 9
- 3.2 Flash State Machine Functions 10
- 3.3 Read Functions 28
- 3.4 Informational Functions 31
- 3.5 Utility Functions 31

4 SECCFG and BANKMGMT Programming Using the Flash API 34
- 4.1 BANKMGMT Programming 34
- 4.2 SECCFG Programming 36

5 Recommended FSM Flows 37
- 5.1 New Devices From Factory 37
- 5.2 Recommended Erase Flow 37
- 5.3 Recommended Bank Erase Flow 38
- 5.4 Recommended Program Flow 39

6 References 39

A Flash State Machine Commands 40

B Typedefs, Defines, Enumerations and Structure 41
- B.1 Type Definitions 41
- B.2 Defines 41
- B.3 Enumerations 41
- B.4 Structures 43

Trademarks

Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
All trademarks are the property of their respective owners.

Copyright © 2025 Texas Instruments Incorporated

1 Introduction

This reference guide provides a detailed description of Texas Instruments' F29H85x Flash API Library (F29H85x_NWFlashAPI_v21.00.00.00.lib) functions that can be used to erase, program and verify Flash on F29H85x devices. Note that Flash API v21.00.00.00 should be used only with F29H85x devices. The Flash API Library is provided in the F29H85x SDK at "f29h85x-sdk > source > flash_api"

1.1 Differences From C28x

C28 Flash API Function Signature	C29 Flash API Function Signature

`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`
`);`
	No longer required/deprecated

`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`
	`);`

Copyright © 2025 Texas Instruments Incorporated

C28 Flash API Function Signature	C29 Flash API Function Signature
Fapi_StatusType Fapi_doVerifyBy16bits(uint16 *pu16StartAddress, uint32 u16Length, uint16 *pu16CheckValueBuffer, Fapi_FlashStatusWordType *poFlashStatusWord);	Deprecated. Use 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);

Copyright © 2025 Texas Instruments Incorporated

Unlike F28P65x devices, the F29H85x memory model uses an interleaved flash bank system to keep up with the CPU processing speed. Figure 1-1 shows a diagram of the flash architecture.

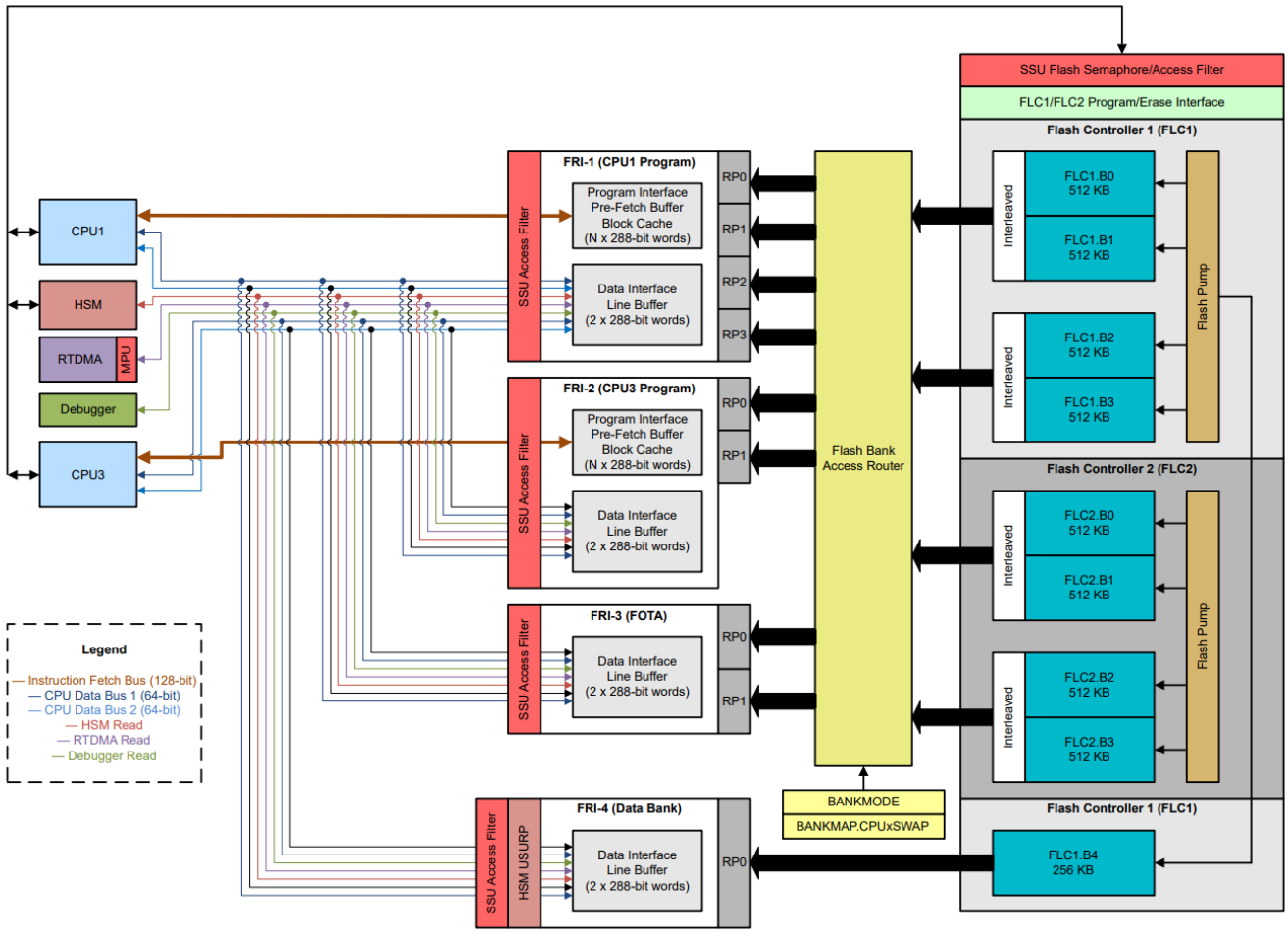

Figure 1-1. F29H85x Flash Architecture

This means that program and erase commands must be called twice, once for each interleaved bank. The exact specifics are handled within FlashAPI, with reads and writes now done through up to four Flash Read Interfaces (FRI-n) rather than a raw bank base address. There are four BankModes to choose from, affecting the memory split between CPUs 1 and 3 and whether or not swap is enabled. More specifics can be found in the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*.

Note

This reference guide is only for Bank Modes 0 and 2.

To meet the requirements of this new architecture, several new parameters have been added to the functions, as shows above:

- u32StartAddress: Tells Flash API which flash controller (FLCx) to use
- u8Iterator: additional parameter for the user/API to track the command iteration between first interleaved bank and second interleaved bank
- u32UserFlashConfig: Passes merged data about bank types and FOTA to the Flash API

Additionally, the F29 series of devices are byte/8-bit addressable, unlike the C28's 16-bit addressability.

For specific documentation on the flash architecture, see the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*.

Copyright © 2025 Texas Instruments Incorporated

1.2 Function Listing Format

This is the general format of an entry for a function, compiler intrinsic, or macro. A short description of what function_name() does.

Synopsis

Provides a prototype for function_name().

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

Parameters

```
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
```

Parameter passing is categorized as follows:

- in — Indicates the function uses one or more values in the parameter that you give it without storing any changes.
- out — Indicates the function saves one or more of the values in the parameter that you give it. You can examine the saved values to find out useful information about your application.
- in/out — Indicates the function changes one or more of the values in the parameter that you give it and saves the result. You can examine the saved values to find out useful information about your application.

Description

Describes the function. This section also describes any special characteristics or restrictions that might apply:

- Function blocks or might block the requested operation under certain conditions
- Function has pre-conditions that might not be obvious
- Function has restrictions or special behavior

Restrictions

Specifies any restrictions in using this function.

Return Value

Specifies any value or values returned by the function.

See Also

Lists other functions or data types related to the function.

Sample Implementation

Provides an example (or a reference to an example) that illustrates the use of the function. Along with the Flash API functions, these examples may use the functions from the device_support folder or driverlib folder provided in the F29H85x SDK, to demonstrate the usage of a given Flash API function in an application context.

Copyright © 2025 Texas Instruments Incorporated

2 F29H85x Flash API Overview

2.1 Introduction

The Flash API is a library of routines, that when called with the proper parameters in the proper sequence, erases, programs, or verifies Flash memory. The Flash API can be used to program BANKMGMT and SECCFG memory as well.

Note

Read the data manual for the Flash memory map and Flash waitstate specifications. Note that this reference guide assumes that the user has already read the *Flash Module* chapter in the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*. Also, pay special attention to the functions Fapi_issueAsyncCommand(), Fapi_setupBankSectorEnable(), Fapi_issueBankEraseCommand() on this device. Usage of these functions is demonstrated in the flash API usage example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash" (for bank mode 0) and "f29h85x-sdk > examples > driverlib > multi_core > flash" (for bank mode 2).

2.2 API Overview

Table 2-1. Summary of Initialization Functions

API Function	Description
Fapi_initializeAPI()	Initializes the API for first use or frequency change

Table 2-2. Summary of Flash State Machine (FSM) Functions

API Function	Description
Fapi_setActiveFlashBank()	Initializes Flash Wrapper and bank for an erase or program command.
Fapi_setupBankSectorEnable()	Configures the Write/Erase protection for the sectors.
Fapi_issueBankEraseCommand()	Issues bank erase command to the Flash State Machine for the given bank address.
Fapi_issueAsyncCommandWithAddress()	Issues an erase sector command to FSM for the given address
Fapi_issueProgrammingCommand()	Sets up the required registers for programming and issues the command to the FSM
Fapi_issueProgrammingCommandForEccAddress()	Remaps an ECC address to the main data space and then call Fapi_issueProgrammingCommand() to program ECC
Fapi_issueAutoEcc512ProgrammingCommand() S	Sets up the required registers for 512-bit (64 byte) programming with AutoECC generation mode and issues the command to the FSM
Fapi_issueDataAndEcc512ProgrammingCommand()	Sets up the required registers for 512-bit (64 byte) programming with user provided flash data and ECC, and issues the command to the FSM
Fapi_issueDataOnly512ProgrammingCommand()	Sets up the required registers for 512-bit (64 byte) programming with user provided flash data and issues the command to the FSM
Fapi_issueEccOnly64ProgrammingCommand()	Sets up the required registers for 64-bit (8 byte) ECC programming with user provided ECC data and issues the command to the FSM
Fapi_issueAsyncCommand()	Issues a command (Clear Status) to FSM for operations that do not require an address
Fapi_checkFsmForReady()	Returns whether or not the Flash state machine (FSM) is ready or busy
Fapi_getFsmStatus()	Returns the STATCMD status register value from the Flash Wrapper

Copyright © 2025 Texas Instruments Incorporated

Table 2-3. Summary of Read Functions

API Function	Description
Fapi_doBlankCheck()	Verifies specified Flash memory range against erased state
Fapi_doVerify()	Verifies specified Flash memory range against supplied values
Fapi_doVerifyByByte()	Verifies specified Flash memory range against supplied values by byte

Table 2-4. Summary of Information Functions

API Function	Description
Fapi_getLibraryInfo()	Returns the information specific to the compiled version of the API library

Table 2-5. Summary of Utility Functions

API Function	Description
Fapi_flushPipeline()	Flushes the data cache in Flash Wrapper
Fapi_calculateEcc()	Calculates the ECC for the supplied address and 64-bit word
Fapi_isAddressEcc()	Determines if the address falls in ECC ranges
Fapi_remapEccAddress()	Remaps an ECC address to corresponding main address
Fapi_getUserConfiguration()	Calculates the value for u32UserFlashConfig given the desired configuration parameters.
Fapi_setUserConfiguration()	Commits the u32UserFlashConfig value

2.3 Using API

This section describes the flow for using various API functions.

2.3.1 Initialization Flow

2.3.1.1 After Device Power Up

After the device is first powered up, the Fapi_getUserConfiguration(), Fapi_SetFlashCPUConfiguration(), and Fapi_initializeAPI() functions must be called before any other API function (except for the Fapi_getLibraryInfo() function) can be used. This procedure sets several required user-configurable variables and configures the Flash Wrapper based on the user specified operating system frequency.

2.3.1.2 On System Frequency Change

If the System operating frequency is changed after the initial call to the Fapi_initializeAPI() function, this function must be called again before any other API function (except the Fapi_getLibraryInfo() function) can be used. This procedure will update the API internal state variables

2.3.2 Building With the API

2.3.2.1 Object Library Files

The Flash API object file is distributed in the Arm® standard EABI elf format.

2.3.2.2 Distribution Files

The following API files are distributed in the f29h85x-sdk > source > flash_api > flash folder:

- Library Files
  - F29H85x Flash API is NOT embedded into the Boot ROM of this device, it is wholly software. The software libraries provided are in the EABI elf (F29H85x_NWFlashAPI_v21.00.00.00.lib) object format. In order for the application to be able to erase or program flash (including BANKMGMT and SECCFG), one of these two library files should be included in the application, depending on the output object format the application is using.
- F29H85x_NWFlashAPI_v21.00.00.00.lib– This is the Flash API EABI elf object format library (FPU32 flag enabled for build) for F29H85x devices.
- Fixed point version of the API library is not provided.

Copyright © 2025 Texas Instruments Incorporated

- Include Files:
  - FlashTech_F29H85x_C29x.h – The master include file for F29H85x devices. This file sets up compile specific defines and then includes the FlashTech.h master include file.
  - hw_flash_command.h – Definitions of the flash write/erase protection registers
- The following include files should not be included directly by the user's code, but are listed here for user reference:
  - FlashTech.h – This include file lists all public API functions and includes all other include files.
  - Registers.h – Definitions common to all register implementations and includes the appropriate register include file for the selected device type.
  - Registers_C29x.h – Containts Little Endian and Flash memory controller registers structure.
  - Types.h – Contains all the enumerations and structures used by the API.
  - Constants/F29H85x.h – Constant definitions for F29H85x devices.

2.3.3 Key Facts for Flash API Usage

Here are some important facts about API usage:

- Names of the Flash API functions start with a prefix "Fapi_".
- Flash API does not configure PLL. The user application should configure the PLL as needed and pass the configured CPUCLK value to Fapi_initializeAPI() function (details of this function are given later in this document).
- Flash API does not check the PLL configuration to confirm the user input frequency. This is up to the system integrator - TI suggests to use the DCC module to check the system frequency. For an example implementation, see the F29H85x SDK driverlib clock configuration function.
- Flash API does not configure the SSU registers or obtain the semaphore. User applications should configure as needed. For details of these registers, see the F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual
- Always configure waitstates as per the device-specific data manual before calling the Flash API functions. The Flash API will issue an error if the waitstate configured by the application is not appropriate for the operating frequency of the application.
- Flash API execution is interruptible. There should not be any read/fetch access from the Flash bank on which an erase/program operation is in progress. Therefore, the Flash API functions, the user application functions that call the Flash API functions, and any ISRs (Interrupt service routines) must be executed from RAM or the flash bank on which there is no any active erase/program operation in progress. For example, the above mentioned conditions apply to the entire code-snippet shown below in addition to the Flash API functions. The reason for this is because the Fapi_issueAsyncCommandWithAddress() function issues the erase command to the FSM, but it does not wait until the erase operation is over. As long as the FSM is busy with the current operation, the Flash bank being erased should not be accessed.

```
//
// Program 128 bits
//
oReturnCheck = Fapi_issueProgrammingCommand((uint32 *)u32Index,Buffer+i,
                                    16, 0, 0, Fapi_AutoEccGeneration, u32UserFlashConfig);

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

- Flash API does not configure (enable/disable) watchdog. The user application can configure watchdog and service it as needed.
- The Main Array flash programming must be aligned to 64-bit address boundaries (alignment on 128-bit address boundary is suggested) and each 64-bit word may only be programmed once per write/erase cycle.
- It is permissible to program the data and ECC separately. However, each 64-bit dataword and the corresponding ECC word may only be programmed once per write/erase cycle.
- Note that there should not be any access to the Flash bank on which the Flash erase/program operation is in progress.

Copyright © 2025 Texas Instruments Incorporated

3 API Functions

3.1 Initialization Functions

3.1.1 Fapi_initializeAPI()

Initializes the Flash API

Synopsis

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

Parameters

poFlashControlRegister [in]	Pointer to the Flash Wrapper Registers' base address. Use FLASHCONTROLLER1_BASE.
u32HclkFrequency [in]	System clock frequency in MHz

Description

This function is required to initialize the Flash API before any other Flash API operation is performed. This function must also be called if the System frequency or RWAIT is changed.

Note

RWAIT register value must be set before calling this function.

Return Value

- Fapi_Status_Success (success)
- Fapi_Error_InvalidHclkValue (failure: System clock does not match specified wait value)

Sample Implementation

(Refer to the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”)

Copyright © 2025 Texas Instruments Incorporated

3.2 Flash State Machine Functions

3.2.1 Fapi_setActiveFlashBank()

Initializes the Flash Wrapper for erase and program operations. This function is deprecated with F29 devices and has been kept for legacy purposes.

Synopsis

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

Parameters

oNewFlashBank [in]	Bank number to set as active. Always use Fapi_FlashBank0 irrespective of which flash bank is targeted for erase/program operations on any CPU.

Description

This function sets the Flash Wrapper for further operations to be performed on the bank. This function is required to be called after the Fapi_initializeAPI() function and before any other Flash API operation is performed.

Note

Irrespective of which flash bank is targeted for the erase and program operations, the user application needs to call this only once and that can be with Fapi_FlashBank0.

Return Value

- **Fapi_Status_Success** (Success)
- **Fapi_Status_FsmBusy** (failure: FSM busy with another command)
- **Fapi_Error_InvalidBaseRegCntlAddress** (failure: Flash control register base address provided by user does not match the expected address)
- **Fapi_Error_InvalidBank** (failure: Bank specified does not exist on device)
- **Fapi_Error_InvalidHclkValue** (failure: System clock does not match specified wait value)

Copyright © 2025 Texas Instruments Incorporated

3.2.2 Fapi_setupBankSectorEnable()

Configures Write(program)/Erase protection for the sectors.

Synopsis

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

Parameters

pu32StartAddress [in]	Flash memory address being written to
u32UserFlashConfig [in]	User flash configuration bitfield
reg_address [in]	Register address for Write/Erase protection configuration. Use FLASH_NOWRAPPER_O_CMDWEPROTA for the first 32 (0-31) sectors. Use FLASH_NOWRAPPER_O_CMDWEPROTB for the remaining main-array (32-512) sectors. Use FLASH_NOWRAPPER_O_CMDWEPROTNM for BANKMGMT and SECCFG
value [in]	32-bit mask indicating which sectors to mask from the erase and program operations.

Description

On this device, all of the flash main and nonmain (BANKMGMT, and SECCFG) sectors are protected from the erase and program operations by default. User applications must disable the protection for the sectors on which they want to perform erase and/or program operations. This function can be used to enable/disable the protection. This function should be called before each erase and program command as shown in the flash API usage example provided in the F29H85x SDK.

First input parameter for this function can be the address of any of these three registers: FLASH_NOWRAPPER_O_CMDWEPROTA, FLASH_NOWRAPPER_O_CMDWEPROTB, FLASH_NOWRAPPER_O_CMDWEPROTNM

CMDWEPROTA register is used to configure the protection for the first 32 sectors (0 to 31). Each bit in this register corresponds to each sector – Example: Bit 0 of this register is used to configure the protection for Sector 0 and Bit 31 of this register is used to configure the protection for Sector 31. A 32-bit user-provided sector mask (second parameter passed to this function) indicates which sectors the user wants to mask from the erase and program operations, that is, sectors that will not be erased and programmed. If a bit in the mask is 1, that particular sector will not be erased/programmed. If a bit in the mask is 0, that particular sector will be erased or programmed.

CMDWEPROTB register is used to configure the protection for the 32 – 256 sectors in the main-array flash bank. However, please note that each bit in this register is used to configure protection for 8 sectors together. This means, bit 0 is used to configure the protection for all of the sectors 32 to 39 together, bit 1 is used to configure the protection for all of the sectors 40 to 47 together, and so on. A 32-bit user-provided sector mask (second parameter passed to this function) indicates which sectors the user wants to mask from the erase and program operations, that is, sectors that will not be erased and programmed. If a bit in the mask is 1, that particular set of sectors will not be erased/programmed. If a bit in the mask is 0, that particular set of sectors will be erased/programmed.

Copyright © 2025 Texas Instruments Incorporated

CMDWEPROT_NM register is used to configure the protection for the USER OTP. Bit 0 in this register is used to configure the protection for the USER OTP. This means, if bit 0 is configured as 1, USER OTP will not be programmed. If bit 0 is configured as 0, USER OTP will be programmed. Other bits of this register can be configured as 1s. Since USER OTP is not erasable, the CMDWEPROT_NM register protection is not applicable for erase operations. This should be configured only for the program operation as needed.

Note

There are no separate dedicated CMDWEPROT_x registers for each bank. Hence, these registers should be configured before each and every flash erase and program command for any bank.

Return Value

- Fapi_Status_Success (success)

Sample Implementation

(Refer to the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”)

3.2.3 Fapi_issueAsyncCommandWithAddress()

Issues an erase command to the Flash State Machine along with a user-provided sector address.

Synopsis

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

Parameters

oCommand [in]	Command to issue to the FSM. Use Fapi_EraseSector.
pu32StartAddress [in]	Flash sector address for erase operation
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)
uint32 u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function issues an erase command to the Flash State Machine for the user-provided sector address. When operating on an interleaved bank, this function must be called twice (once with each iterator value) to erase both underlying banks. The 128-bit aligned start address stays the same during these two calls. This function does not wait until the erase operation is over; it just issues the command and returns back. Hence, this function always returns success status when the Fapi_EraseSector command is used. The user application must wait for the Flash Wrapper to complete the erase operation before returning to any kind of Flash accesses. The Fapi_checkFsmForReady() function can be used to monitor the status of an issued command.

Note

This function does not check STATCMD after issuing the erase command. The user application must check the STATCMD value when FSM has completed the erase operation. STATCMD indicates if there is any failure occurrence during the erase operation. The user application can use the Fapi_getFSMStatus function to obtain the STATCMD value. Also, the user application should use the Fapi_doBlankCheck() function to verify that the Flash is erased.

Copyright © 2025 Texas Instruments Incorporated

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_InvalidBaseRegCntlAddress** (failure: Flash control register base address provided by user does not match the expected address.)
- **Fapi_Error_FeatureNotAvailable** (failure: User requested a command that is not supported.)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register write failed. The user should make sure that the API is executing from the correct CPU.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Sample Implementation

(Refer to the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”)

3.2.4 Fapi_issueBankEraseCommand()

Issues a bank erase command to the Flash State Machine along with a user-provided sector mask.

Synopsis

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

Parameters

pu32StartAddress [in]	Flash bank address for bank erase operation
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)
u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function issues a bank erase command to the Flash state machine for the user-provided bank address. If the FSM is busy with another operation, the function returns indicating the FSM is busy, otherwise it proceeds with the bank erase operation. When operating on interleaved banks, this function must be called twice (once with each iterator value, the start address stays the same).

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_FlashRegsNotWritable** (Flash registers not writable)
- **Fapi_Error_InvalidBaseRegCntlAddress** (failure: Flash control register base address provided by user does not match the expected address.)

Sample Implementation

(Refer to the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”)

Copyright © 2025 Texas Instruments Incorporated

3.2.5 Fapi_issueProgrammingCommand()

Sets up data and issues program command to valid Flash, BANKMGMT, and SECCFG memory addresses

Synopsis

```
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
                            );
```

Parameters

pu32StartAddress [in]	Start address in Flash for the data and ECC to be programmed. The start address should always be even.
pu8DataBuffer [in]	Pointer to the Data buffer address. Data buffer should be 64-bit aligned.
u8DataBufferSizeInBytes [in]	Number of bytes in the Data buffer
pu8EccBuffer [in]	Pointer to the ECC buffer address
u8EccBufferSizeInBytes [in]	Number of bytes in the ECC buffer
oMode [in]	ECC mode
u32UserFlashConfig [in]	User flash configuration bitfield

Note

The pu8EccBuffer should contain ECC corresponding to the data at the 64-bit aligned main array, BANKMGMT, or SECCFG address. The LSB of the pu8EccBuffer corresponds to the lower 32 bits of the main array and the MSB of the pu8EccBuffer corresponds to the upper 32 bits of the main array.

Description

This function sets up the programming registers of the Flash State Machine based on the supplied parameters. It offers four different programming modes to the user for use in different scenarios as mentioned in Table 3-1. This function sets up the programming registers of the Flash State Machine based on the supplied parameters. It offers four different programming modes to the user for use in different scenarios as mentioned in Table 3-1.

Copyright © 2025 Texas Instruments Incorporated

Table 3-1. Uses of Different Programming Modes

Programming Mode (oMode)	Arguments Used	Usage Purpose
Fapi_DataOnly	pu32StartAddress, pu8DataBuffer, u8DataBufferSizeInWords	Used when any custom programming utility or an user application (that embed/use Flash API) has to program data and corresponding ECC separately. Data is programmed using Fapi_DataOnly mode and then the ECC is programmed using Fapi_EccOnly mode. Generally most of the programming utilities do not calculate ECC separately and instead use Fapi_AutoEccGeneration mode. However, some Safety applications may require to insert intentional ECC errors in their Flash image (which is not possible when Fapi_AutoEccGeneration mode is used) to check the health of the SECDED (Single Error Correction and Double Error Detection) module at run time. In such case, ECC is calculated separately (using the Fapi_calculateEcc() function as applicable). Application may want to insert errors in either main array data or in the ECC as needed. In such scenarios, after the error insertion, Fapi_DataOnly mode and Fapi_EccOnly modes can be used to program the data and ECC respectively.
Fapi_AutoEccGeneration	pu32StartAddress, pu8DataBuffer, u8DataBufferSizeInBytes	Used when any custom programming utility or user application (that embed/use Flash API to program Flash at run time to store data or to do a firmware update) has to program data and ECC together without inserting any intentional errors. This is the most prominently used mode.
Fapi_DataAndEcc	pu32StartAddress, pu8DataBuffer, u8DataBufferSizeInBytes, pu8EccBuffer, u8EccBufferSizeInBytes	Purpose of this mode is not different than that of using Fapi_DataOnly and Fapi_EccOnly modes together. However, this mode is beneficial when both the data and the calculated ECC can be programmed at the same time.
Fapi_EccOnly	pu8EccBuffer, u8EccBufferSizeInBytes	See the usage purpose given for Fapi_DataOnly mode.

Table 3-2 shows the allowed programming range for the function.

Table 3-2. Permitted Programming Range for Fapi_issueProgrammingCommand()

Flash API	Main Array	ECC	BANKMGMT and SECCFG
Fapi_issueProgrammingCommand() 128-bit, Fapi_AutoEccGeneration mode	Allowed	Allowed	Allowed
Fapi_issueProgrammingCommand() 128-bit, Fapi_DataOnly mode	Allowed	Not allowed	Allowed
Fapi_issueProgrammingCommand() 128-bit, Fapi_DataAndEcc mode	Allowed	Allowed	Allowed
Fapi_issueProgrammingCommand() 128-bit, Fapi_EccOnly mode	Not allowed	Allowed	Not allowed

Note

Users must always program ECC for their flash image since ECC check is enabled at power up.

Copyright © 2025 Texas Instruments Incorporated

Programming modes:

Fapi_DataOnly – This mode will only program the data portion in Flash at the address specified. It can program from 1-bit up to 16 bytes. However, review the restrictions provided for this function to know the limitations of flash programming data size. The supplied starting address to program at plus the data buffer length cannot cross the 128-bit aligned address boundary. Arguments 4 and 5 are ignored when using this mode.

Fapi_AutoEccGeneration – This mode will program the supplied data in Flash along with automatically generated ECC. The ECC is calculated for every 64-bit data aligned on a 64-bit memory boundary. Hence, when using this mode, all the 64 bits of the data should be programmed at the same time for a given 64-bit aligned memory address. Data not supplied is treated as all 1s (0xFFFF). Once ECC is calculated and programmed for a 64-bit data, those 64 bits can not be reprogrammed (unless the sector is erased) even if it is programming a bit from 1 to 0 in that 64-bit data, since the new ECC value will collide with the previously programmed ECC value. When using this mode, if the start address is 128-bit aligned, then either 16 or 8 bytes can be programmed at the same time as needed. If the start address is 64-bit aligned but not 128-bit aligned, then only 8 bytes can be programmed at the same time. The data restrictions for Fapi_DataOnly also exist for this option. Arguments 4 and 5 are ignored.

Note

Fapi_AutoEccGeneration mode will program the supplied data portion in Flash along with automatically generated ECC. The ECC is calculated for 64-bit aligned address and the corresponding 64-bit data. Any data not supplied is treated as 0xFFFF. Note that there are practical implications of this when writing a custom programming utility that streams in the output file of a code project and programs the individual sections one at a time into flash. If a 64-bit word spans more than one section (that is, contains the end of one section, and the start of another), values of 0xFFFF cannot be assumed for the missing data in the 64-bit word when programming the first section. When you go to program the second section, you will not be able to program the ECC for the first 64-bit word since it was already (incorrectly) computed and programmed using assumed 0xFFFF for the missing values. One way to avoid this problem is to align all sections linked to flash on a 64-bit boundary in the linker command file for your code project.

Here is an example:

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

If you do not align the sections in flash, you would need to track incomplete 64-bit words in a section and combine them with the words in other sections that complete the 64-bit word. This will be difficult to do. So it is recommended to align your sections on 64-bit boundaries.

Some 3rd party Flash programming tools or TI Flash programming kernel examples, or any custom Flash programming solution may assume that the incoming data stream is all 128-bit aligned and may not expect that a section might start on an unaligned address. Thus it may try to program the maximum possible (128-bits) words at a time assuming that the address provided is 128-bit aligned. This can result in a failure when the address is not aligned. So, it is suggested to align all the sections (mapped to Flash) on a 128-bit boundary.

Fapi_DataAndEcc – This mode will program both the supplied data and ECC in Flash at the address specified. The data supplied must be aligned on a 64-bit memory boundary and the length of data must correlate to the supplied ECC. That means, if the data buffer length is 8 bytes, the ECC buffer must be 1 byte. If the data buffer length is 16 bytes, the ECC buffer must be 2 bytes in length. If the start address is 128-bit aligned, then either 16 or 8 bytes should be programmed at the same time as needed. If the start address is 64-bit aligned but not 128-bit aligned, then only 8 bytes should be programmed at the same time.

The LSB of pu8EccBuffer corresponds to the lower 64-bits of the main array and the MSB of pu8EccBuffer corresponds to the upper 64-bits of the main array.

Copyright © 2025 Texas Instruments Incorporated

The Fapi_calculateEcc() function can be used to calculate ECC for a given 64-bit aligned address and the corresponding data.

Fapi_EccOnly – This mode will only program the ECC portion in Flash ECC memory space at the address (Flash main array address should be provided for this function and not the corresponding ECC address) specified. It can program either 2 bytes (both LSB and MSB at a location in ECC memory) or 1 byte (LSB at a location in ECC memory). The LSB of pu8EccBuffer corresponds to the lower 64-bits of the main array and the MSB of pu8EccBuffer corresponds to the upper 64-bits of the main array. Arguments two and three are ignored when using this mode.

Note

The length of pu8DataBuffer and pu8EccBuffer cannot exceed 16 and 2, respectively.

Note

This function does not check STATCMD after issuing the program command. The user application must check the STATCMD value when FSM has completed the program operation. STATCMD indicates if there is any failure occurrence during the program operation. The user application can use the Fapi_getFsmStatus function to obtain the STATCMD value.
Also, the user application should use the Fapi_doVerify() function to verify that the Flash is programmed correctly.

This function does not wait until the program operation is over; it just issues the command and returns back. Hence, the user application must wait for the Flash Wrapper to complete the program operation before returning to any kind of Flash accesses. The Fapi_checkFsmForReady() function should be used to monitor the status of an issued command.

Restrictions

- As described above, this function can program only a max of 128-bits (given the address provided is 128-bit aligned) at a time. If the user wants to program more than that, this function should be called in a loop to program 128-bits (or 64-bits as needed by application) at a time.
- The Main Array flash programming must be aligned to 64-bit address boundaries and each 64-bit word may only be programmed once per write or erase cycle.
- It is alright to program the data and ECC separately. However, each 64-bit dataword and the corresponding ECC word may only be programmed once per write or erase cycle.

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_InvalidBaseRegCntlAddress** (failure: Flash control register base address provided by user does not match the expected address)
- **Fapi_Error_AsyncIncorrectDataBufferLength** (failure: Data buffer size specified is incorrect. This error will also be returned if Fapi_EccOnly mode is selected when programming to the BANKMGMT or SECCFG spaces)
- **Fapi_Error_AsyncIncorrectEccBufferLength** (failure: ECC buffer size specified is incorrect)
- **Fapi_Error_AsyncDataEccBufferLengthMismatch** (failure: Data buffer size either is not 64-bit aligned or data length crosses the 128-bit aligned memory boundary)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU).
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Copyright © 2025 Texas Instruments Incorporated

Sample Implementation

For more information, see the flash programming example provided in the F29H85x SDK at: “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”.

3.2.6 Fapi_issueProgrammingCommandForEccAddresses()

Remaps an ECC address to data address and calls Fapi_issueProgrammingCommand().

Synopsis

```
Fapi_StatusType Fapi_issueProgrammingCommandForEccAddresses(
                                    uint32_t *pu32StartAddress,
                                    uint8_t *pu8EccBuffer,
                                    uint8_t u8EccBufferSizeInBytes
                                    );
```

Parameters

pu32StartAddress [in]	ECC start address in Flash for the ECC to be programmed
pu8EccBuffer [in]	Pointer to the ECC buffer address
u8EccBufferSizeInBytes [in]	number of bytes in the ECC buffer If the number of bytes is 1, LSB (ECC for lower 64 bits) gets programmed. MSB alone cannot be programmed using this function. If the number of bytes is 2, both LSB and MSB bytes of ECC get programmed.

Description

This function remaps an address in the ECC memory space to the corresponding data address space and then call Fapi_issueProgrammingCommand() to program the supplied ECC data. The same limitations for Fapi_issueProgrammingCommand() using Fapi_EccOnly mode applies to this function. The LSB of pu8EccBuffer corresponds to the lower 64 bits of the main array and the MSB of pu8EccBuffer corresponds to the upper 64 bits of the main array.

Note

The length of the pu8EccBuffer cannot exceed 2.

Note

This function does not check STATCMD after issuing the program command. The user application must check the STATCMD value when FSM has completed the program operation. STATCMD indicates if there is any failure occurrence during the program operation. The user application can use the Fapi_getFSMStatus function to obtain the STATCMD value.

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_InvalidBaseRegCntlAddress** (failure: Flash control register base address provided by user does not match the expected address.)
- **Fapi_Error_AsyncIncorrectEccBufferLength** (failure: Data buffer size specified is incorrect.)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Copyright © 2025 Texas Instruments Incorporated

3.2.7 Fapi_issueAutoEcc512ProgrammingCommand()

Sets up data and issues 512-bit (64 bytes) AutoEcc generation mode program command to valid Flash, BANKMGMT, and SECCFG memory addresses.

Synopsis

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

Parameters

pu32StartAddress [in]	1024-bit aligned flash address to program the provided data and ECC.
pu8DataBuffer [in]	Pointer to the Data buffer address. Address of the Data buffer should be 1024-bit aligned.
u8DataBufferSizeInWords [in]	Number of bytes in the Data buffer. Max Databuffer size in words should not exceed 64.
u32UserFlashConfig [in]	User flash configuration bitfield
uint8 u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)

Description

This function automatically generates 8 bytes of ECC data for the user provided 512-bit data (second parameter) and programs the data and ECC together at the user provided 512-bit aligned flash address (first parameter). When this command is issued, the flash state machine will program all of the 512 bits along with ECC. Hence, when using this mode, data not supplied is treated as all 1s (0xFFFF). Once ECC is calculated and programmed for a 512-bit data, those 512 bits cannot be reprogrammed (unless the sector is erased) even if it is programming a bit from 1 to 0 in that 512-bit data, since the new ECC value will collide with the previously programmed ECC value.

Note

512-bit programming will program to one interleaved bank at a time. As a result, it must be called twice, once with each iterator (for a total of 1024 bits of data), to have fully contiguous data. The pu32StartAddress for each of these calls must remain the same. If both interleaved banks are not programmed, data will appear similar to the following:

Address	Data
0x10000000	Data bytes 0-15
0x10000010	0xFFFF
0x10000020	Data bytes 16-31
0x10000030	0xFFFF

And so on. If the ordering of the data buffer is important, an example of how to properly format the buffer so that it maintains its order in flash can be found in this function's sample implementation. ("f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program").

Copyright © 2025 Texas Instruments Incorporated

Note

Fapi_issueAutoEcc512ProgrammingCommand() function will program the supplied data portion in Flash along with automatically generated ECC. The ECC is calculated for 512-bit aligned address and the corresponding 512-bit data. Any data not supplied is treated as 0xFFFF. Note that there are practical implications of this when writing a custom programming utility that streams in the output file of a code project and programs the individual sections one at a time into flash. If a 512-bit word spans more than one section (that is, contains the end of one section, and the start of another), values of 0xFFFF cannot be assumed for the missing data in the 64-bit word when programming the first section. When you go to program the second section, you will not be able to program the ECC for the first 512-bit word since it was already (incorrectly) computed and programmed using assumed 0xFFFF for the missing values. One way to avoid this problem is to align all sections linked to flash on a 512-bit boundary in the linker command file for your code project.

Here is an example:

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

If you do not align the sections in flash, you would need to track incomplete 512-bit words in a section and combine them with the words in other sections that complete the 512-bit word. This is difficult to do. Hence, it is recommended to align your sections on 512-bit boundaries.

Some 3rd party Flash programming tools or TI Flash programming kernel examples, or any custom Flash programming solution may assume that the incoming data stream is all 512-bit aligned and may not expect that a section might start on an unaligned address. Thus, it may try to program the maximum possible (512-bits) words at a time assuming that the address provided is 512-bit aligned. This can result in a failure when the address is not aligned. So, it is suggested to align all the sections (mapped to Flash) on a 512-bit boundary.

For the allowed programming range for the function, see Table 3-3.

Table 3-3. Permitted Programming Range for Fapi_issueAutoEcc512ProgrammingCommand()

Flash API	Main Array	ECC	BANKMGMT and SECCFG
Fapi_issueAutoEcc512ProgrammingCommand()	Allowed	Allowed	Not allowed

Note

The length of pu8DataBuffer cannot exceed 64.

Note

This function does not check STATCMD after issuing the program command. The user application must check the STATCMD value when FSM has completed the program operation. STATCMD indicates if there is any failure occurrence during the program operation. The user application can use the Fapi_getFsmStatus function to obtain the STATCMD value.

Also, the user application should use the Fapi_doVerify() function to verify that the Flash is programmed correctly.

This function does not wait until the program operation is over; it just issues the command and returns back. Hence, the user application must wait for the Flash Wrapper to complete the program operation before returning to any kind of Flash accesses. The Fapi_checkFsmForReady() function should be used to monitor the status of an issued command.

Copyright © 2025 Texas Instruments Incorporated

Restrictions

- As described above, this function can program only a max of 512-bits (given the address provided is 512-bit aligned) at a time. If the user wants to program more than that, this function should be called in a loop to program 512-bits at a time.
- The Main Array flash programming must be aligned to 512-bit address boundaries and 64 bytes may only be programmed once per write or erase cycle.
- 512-bit address range starting with a BANKMGMT or SECCFG address shall always be programmed using 128-bit Fapi_issueProgrammingCommand().

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_AsyncIncorrectDataBufferLength** (failure: Data buffer size specified is incorrect. Also, this error will be returned if Fapi_EccOnly mode is selected when programming to the BANKMGMT or SECCFG spaces,)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU).
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address.) For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Sample Implementation

(For more information, see the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”)

3.2.8 Fapi_issueDataAndEcc512ProgrammingCommand()

Sets up the flash state machine registers for the 512-bit (64 bytes) programming with user provided flash data and ECC data and issues the programing command to valid Flash memory.

Synopsis

```
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
                                    )
```

Parameters

pu32StartAddress [in]	1024-bit aligned flash address to program the provided data and ECC.
pu8DataBuffer [in]	Pointer to the Data buffer address. Address of the Data buffer should be 1024-bit aligned.
u8DataBufferSizeInWords [in]	Number of bytes in the Data buffer. Max Databuffer size in words should not exceed 64.
pu8EccBuffer [in]	Pointer to the ECC buffer address
u8EccBufferSizeInBytes [in]	Number of bytes in the ECC buffer. Max Eccbuffer size in words should not exceed 16.
u32UserFlashConfig [in]	User flash configuration bitfield
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)

Copyright © 2025 Texas Instruments Incorporated

Description

This function programs both the user provided 512-bit data (second parameter) and 8 bytes of ECC data (fourth parameter) together at the user provided 512-bit aligned flash address. The address of data provided must be aligned on a 512-bit memory boundary and the length of data must correlate to the supplied ECC. That means, if the data buffer length is 64 bytes, the ECC buffer must be 8 bytes (1 ECC bytes corresponding to 64-bit data).

Each byte of pu8EccBuffer corresponds to each 64-bit of the main array data provided in the pu8DataBuffer. For more details, see Table 3-6.

The Fapi_calculateEcc() function can be used to calculate ECC for a given 64-bit aligned address and the corresponding data.

For allowed programming range for the function, see Table 3-4.

Table 3-4. Permitted programming range for Fapi_issueDataAndEcc512ProgrammingCommand()

Flash API	Main Array	ECC	BANKMGMT and SECCFG
Fapi_issueDataAndEcc512ProgrammingCommand()	Allowed	Allowed	Not allowed

Restrictions

- As described above, this function can program only a max of 512-bits (given the address provided is 512-bit aligned) at a time. If the user wants to program more than that, this function should be called in a loop to program 512-bits at a time.
- The Main Array flash programming must be aligned to 512-bit address boundaries and 64 bytes may only be programmed once per write or erase cycle.
- 512-bit address range starting with a BANKMGMT or SECCFG address shall always be programmed using 128-bit Fapi_issueProgrammingCommand().

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_AsyncIncorrectDataBufferLength** (failure: Data buffer size specified is incorrect. Also, this error will be returned if Fapi_EccOnly mode is selected when programming to the BANKMGMT or SECCFG spaces.)
- **Fapi_Error_AsyncIncorrectEccBufferLength** (failure: ECC buffer size specified is incorrect.)
- **Fapi_Error_AsyncDataEccBufferLengthMismatch** (failure: Data buffer size either is not 64-bit aligned or data length crosses the 128-bit aligned memory boundary.)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU.)
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Sample Implementation

(For more information, see the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”).

Copyright © 2025 Texas Instruments Incorporated

3.2.9 Fapi_issueDataOnly512ProgrammingCommand()

Sets up the flash state machine registers for the 512-bit (64 bytes) programming with user provided flash data and issues the programing command to valid Flash.

Synopsis

```
Fapi_StatusType Fapi_issueDataOnly512ProgrammingCommand(
                                    uint32 *pu32StartAddress,
                                    uint8 *pu8DataBuffer,
                                    uint8 u8DataBufferSizeInBytes,
                                    uint32 u32UserFlashConfig,
                                    uint8 u8Iterator
                                                        )
```

Parameters

pu32StartAddress [in]	1024-bit aligned flash address to program the provided data.
pu8DataBuffer [in]	Pointer to the Data buffer address. Data buffer should be 1024-bit aligned.
u8DataBufferSizeInBytes [in]	Number of 8-bit words in the Data buffer. Max Databuffer size in bytes should not exceed 64.
u32UserFlashConfig [in]	User flash configuration bitfield
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)

Description

This function only programs the data portion in Flash at the address specified. It can program 512-bit data (Second parameter) at the user provided 512-bit aligned flash address. This function is used when a user application (that embed/use Flash API) has to program 512- bit of data and corresponding 64-bit of ECC data separately. 512-bit Data is programmed using Fapi_issueDataOnly512ProgrammingCommand () function and then the 64-bit ECC is programmed using Fapi_issueEccOnly64ProgrammingCommand() function. Generally, most of the programming utilities do not calculate ECC separately and instead use function Fapi_issueAutoEcc512ProgrammingCommand(). However, some Safety applications may require to insert intentional ECC errors in their Flash image (which is not possible when Fapi_AutoEccGeneration mode is used) to check the health of the SECDED (Single Error Correction and Double Error Detection) module at run time. In such case, ECC is calculated separately (using the Fapi_calculateEcc() function as applicable). Application may want to insert errors in either main array data or in the ECC as needed. In such scenarios, after the error insertion, Fapi_issueDataOnly512ProgrammingCommand() API and then the 64-bit ECC is programmed using Fapi_issueEccOnly64ProgrammingCommand() API can be used to program the data and ECC respectively.

for allowed programming range for the function, see Table 3-5.

Table 3-5. Permitted Programming Range for Fapi_issueDataOnly512ProgrammingCommand()

Flash API	Main Array	ECC	BANKMGMT and SECCFG
Fapi_issueDataOnly512ProgrammingCommand()	Allowed	Not allowed	Not allowed

Restrictions

- As described above, this function can program only a max of 512-bits (given the address provided is 512-bit aligned) at a time. If the user wants to program more than that, this function should be called in a loop to program 512-bits at a time.
- The Main Array flash programming must be aligned to 512-bit address boundaries and 64 bytes may only be programmed once per write or erase cycle.
- 512-bit address range starting with a BANKMGMT or SECCFG address shall always be programmed using 128-bit Fapi_issueProgrammingCommand().

Copyright © 2025 Texas Instruments Incorporated

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_AsyncIncorrectDataBufferLength** (failure: Data buffer size specified is incorrect. Also, this error is returned if the Fapi_EccOnly mode is selected when programming to the BANKMGMT or SECCFG spaces.)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. Make sure that the API is executing from the correct CPU).
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*).

Sample Implementation

(For more information, see the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”)

3.2.10 Fapi_issueEccOnly64ProgrammingCommand()

Sets up the flash state machine registers for the 64-bit (8 bytes) programming with user provided ECC data and issues the programing command to valid Flash, BANKMGMT, and SECCFG memory.

Synopsis

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

Parameters

pu32StartAddress [in]	512-bit aligned flash address to program the provided ECC data.
pu8EccBuffer [in]	Pointer to the ECC buffer address
u8EccBufferSizeInBytes [in]	Number of bytes in the ECC buffer. Max Eccbuffer size in bytes should not exceed 16.
u32UserFlashConfig	User Flash configuration.
u8Iterator	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)

Description

This function will only program the ECC portion in Flash ECC memory space at the address (Flash main array address should be provided for this function and not the corresponding ECC address) specified. It can program 64-bit of ECC data (second parameter) at the ECC address corresponding to the user provided 512-bit aligned flash address. 64-bit ECC data can be split as 8 bytes ECC data correlated to 512-bit aligned data (4 * 128, each 2 bytes corresponding to each 128 data).

Copyright © 2025 Texas Instruments Incorporated

For more information, see Table 3-6.

Table 3-6. 64-Bit ECC Data Interpretation

512 Bits Data (4 * 128bits)			
1st 128-Bit Data	**2nd 128-Bit Data**	**3rd 128-Bit Data**	**4th 128-Bit Data**
LSB of pu8EccBuffer[0]	LSB of pu8EccBuffer[1]	LSB of pu8EccBuffer[2]	LSB of pu8EccBuffer[3]
MSB of pu8EccBuffer[0]	MSB of pu8EccBuffer[1]	MSB of pu8EccBuffer[2]	MSB of pu8EccBuffer[3]

For allowed programming range for the function, see Table 3-7.

Table 3-7. Permitted Programming Range for Fapi_issueEccOnly64ProgrammingCommand()

Flash API	Main Array	ECC	BANKMGMT and SECCFG
Fapi_issueEccOnly64ProgrammingCommand()	Not allowed	Allowed	Not allowed

Restrictions

- As described above, this function can program only a max of 64-bits ECC at a time. If the user wants to program more than that, this function should be called in a loop to program 64-bits at a time.
- The Main Array flash programming must be aligned to 512-bit address boundaries and 64-bit ECC word may only be programmed once per write or erase cycle.
- ECC should not be programmed for BANKMGMT or SECCFG locations. 512-bit address range starting with a BANKMGMT or SECCFG address shall always be programmed using 128-bit Fapi_issueProgrammingCommand().

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_AsyncDataEccBufferLengthMismatch** (failure: Data buffer size either is not 64-bit aligned or data length crosses the 128-bit aligned memory boundary.)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU.)
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported.)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.)

Sample Implementation

(For more information, see the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_512_program”)

3.2.11 Fapi_issueAsyncCommand()

Issues a command to the Flash State Machine. See the description for the list of commands that can be issued by this function.

Synopsis

```
Fapi_StatusType Fapi_issueAsyncCommand(
                          uint32 u32StartAddress,
                          uint32 u32UserFlashConfig,
                          Fapi_FlashStateCommandsType oCommand
                                    )
```


Copyright © 2025 Texas Instruments Incorporated

Parameters

u32StartAddress [in]	32-bit start address in Flash to program/erase/verify.
u32UserFlashConfig [in]	User flash configuration bitfield
oCommand [in]	Command to issue to the FSM. Use Fapi_ClearStatus command.

Description

This function issues a command to the Flash State Machine for commands not requiring any additional information (such as address). On this device, Fapi_ClearStatus command should be issued to the Flash State Machine using this function. Note that Fapi_ClearStatus command should be issued (only if STATCMD is not zero) before each program and erase command as shown in the flash programming example provided in the F29H85x SDK. A new program or erase command can be given only when the STATCMD is zero (achieved by issuing the Fapi_ClearStatus command).

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a command that is not supported.)

Sample Implementation

(For more information, see the flash programming example provided in the F29H85x SDK at “f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program”)

3.2.12 Fapi_checkFsmForReady()

Returns the status of the Flash State Machine

Synopsis

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

Parameters

u32StartAddress [in]	32-bit start address in Flash to program/erase/verify
u32UserFlashConfig	User flash configuration

Description

This function returns the status of the Flash State Machine indicating if it is ready to accept a new command or not. The primary use is to check if an Erase or Program operation has finished.

Return Value

- **Fapi_Status_FsmBusy** (FSM is busy and cannot accept new command except for suspend commands)
- **Fapi_Status_FsmReady** (FSM is ready to accept new command)

Copyright © 2025 Texas Instruments Incorporated

3.2.13 Fapi_getFsmStatus()

Returns the value of the STATCMD register

Synopsis

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

Parameters

u32StartAddress [in]	32-bit start address in Flash to program/erase/verify
u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function returns the value of the STATCMD register. This register allows the user application to determine whether an erase or program operation is successfully completed or in progress or suspended or failed. Each flash controller (FLC1 and FLC2) has its own STATCMD register. The user application should check the value of the appropriate register to determine if there is any failure after each erase and program operation.

Return Value

Table 3-8. STATCMD Register

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 – Read Only

Table 3-9. STATCMD Register Field Descriptions

Bit	Name	Description	Reset value
12	FAILMISC	Command failed due to error other than write/erase protect violation or verify error. 0: No Fail 1: Fail	0x0
8	FAILINVDATA	Program command failed because an attempt was made to program a stored 0 value to a 1. 0: No Fail 1: Fail	0x0
6	FAILILLADDR	Command failed due to the use of an illegal address. 0: No Fail 1: Fail	0x0
5	FAILVERIFY	Command failed due to verify error. 0: No Fail 1: Fail	0x0
4	FAILWEPROT	Command failed due to Write/Erase Protect Sector violation. 0: No Fail 1: Fail	0x0
2	CMDINPROGRESS	Command in Progress 0: Command complete 1: Command is in progress	0x0

Copyright © 2025 Texas Instruments Incorporated

Table 3-9. STATCMD Register Field Descriptions (continued)

Bit	Name	Description	Reset value
1	CMDPASS	Command Pass - valid when CMD_DONE field is 1 0: Fail 1: Pass	0x0
0	CMDDONE	Command Done 0: Command not Done 1: Command Done	0x0

3.3 Read Functions

3.3.1 Fapi_doBlankCheck()

Verifies region specified is erased value

Synopsis

```
Fapi_StatusType Fapi_doBlankCheck(
                        uint32 *pu32StartAddress,
                        uint32 u32Length,
                        Fapi_FlashStatusWordType *poFlashStatusWord,
                        uint8 u8Iterator,
                        uint32 u32UserFlashConfig
                                )
```

Parameters

pu32StartAddress [in]	Start address for region to blank check
u32Length [in]	Length of region in 32-bit words to blank check
poFlashStatusWord [in/out]	Returns the status of the operation if result is not Fapi_Status_Success ->au32StatusWord[0] Address of first non-blank location ->au32StatusWord[1] Data read at first non-blank location ->au32StatusWord[2] Value of compare data (always 0xFFFFFFFF) ->au32StatusWord[3] N/A
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)
u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function checks if the flash is blank (erased state) starting at the specified address for the length of 32-bit words specified. If a non-blank location is found, corresponding address and data will be returned in the poFlashStatusWord parameter. When operating on interleaved banks, this function must be called twice (once with each iterator value, the start address stays the same).

Restrictions

None

Return Value

- Fapi_Status_Success (success) - specified Flash locations are found to be in erased state
- Fapi_Error_Fail (failure: region specified is not blank)
- Fapi_Error_InvalidAddress (failure: User provided an invalid address. For the valid address range), see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.

Copyright © 2025 Texas Instruments Incorporated

3.3.2 Fapi_doVerify()

Verifies region specified against supplied data

Synopsis

```
Fapi_StatusType Fapi_doVerify(
                     uint32 *pu32StartAddress,
                     uint32 u32Length,
                     uint32 *pu32CheckValueBuffer,
                     Fapi_FlashStatusWordType *poFlashStatusWord,
                     uint8 u8Iterator,
                     uint32 u32UserFlashConfig
                             )
```

Parameters

pu32StartAddress [in]	start address for region to verify
u32Length [in]	length of region in 32-bit words to verify
pu32CheckValueBuffer [in]	address of buffer to verify region against. Data buffer should be 128-bit aligned.
poFlashStatusWord [in/out]	returns the status of the operation if result is not Fapi_Status_Success ->au32StatusWord[0] address of first verify failure location ->au32StatusWord[1] data read at first verify failure location ->au32StatusWord[2] value of compare data ->au32StatusWord[3] N/A
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)
u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function verifies the device against the supplied data starting at the specified address for the length of 32-bit words specified. If a location fails to compare, these results will be returned in the poFlashStatusWord parameter. When operating on interleaved banks, this function must be called twice (once with each iterator value, the start address stays the same).

Restrictions

None

Return Value

- **Fapi_Status_Success** (success: region specified matches supplied data)
- **Fapi_Error_Fail** (failure: region specified does not match supplied data)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.

Copyright © 2025 Texas Instruments Incorporated

3.3.3 Fapi_doVerifyByByte()

Verifies region specified against supplied data by byte

Synopsis

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

Parameters

pu32StartAddress [in]	start address for region to verify
u32Length [in]	length of region in bytes to verify
pu8CheckValueBuffer [in]	address of buffer to verify region against. Data buffer should be 128-bit aligned.
poFlashStatusWord [in/out]	returns the status of the operation if result is not Fapi_Status_Success ->au32StatusWord[0] address of first verify failure location ->au32StatusWord[1] data read at first verify failure location ->au32StatusWord[2] value of compare data ->au32StatusWord[3] N/A
u8Iterator [in]	Iterator for program and erase operations on interleaved banks. 0: Data Flash/non-interleaved 1: B0 or B2 (dependent on provided address) 2: B1 or B3 (dependent on provided address)
u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function verifies the device against the supplied data starting at the specified address for the length of bytes specified. If a location fails to compare, these results will be returned in the poFlashStatusWord parameter. When operating on interleaved banks, this function must be called twice (once with each iterator value, the start address stays the same).

Restrictions

None

Return Value

- **Fapi_Status_Success** (success: region specified matches supplied data)
- **Fapi_Error_Fail** (failure: region specified does not match supplied data)
- **Fapi_Error_InvalidAddress** (failure: User provided an invalid address. For the valid address range, see the *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*.

Copyright © 2025 Texas Instruments Incorporated

3.4 Informational Functions

3.4.1 Fapi_getLibraryInfo()

Returns information about this compile of the Flash API

Synopsis

```
Fapi_LibraryInfoType Fapi_getLibraryInfo(void)
```

Parameters

None

Description

This function returns information specific to the compile of the Flash API library. The information is returned in a struct Fapi_LibraryInfoType. The members are as follows:

- u8ApiMajorVersion – Major version number of this compile of the API. This value is 21.
- u8ApiMinorVersion – Minor version number of this compile of the API. Minor version is 00 for F29H85x devices.
- u8ApiRevision – Revision version number of this compile of the API. This value is 00 for this release.

Revision number is 00 for this release.

- oApiProductionStatus – Production status of this compile (Alpha_Internal, Alpha, Beta_Internal, Beta, Production).

Production status is Beta for this release.

- u32ApiBuildNumber – Build number of this compile.
- u8ApiTechnologyType – Indicates the Flash technology supported by the API. This field returns a value of 0x5.
- u8ApiTechnologyRevision – Indicates the revision of the technology supported by the API
- u8ApiEndianness – This field always returns as 1 (Little Endian) for F29H85x devices.
- u32ApiCompilerVersion – Version number of the Code Composer Studio code generation tools used to compile the API

Return Value

- **Fapi_LibraryInfoType** (gives the information retrieved about this compile of the API)

3.5 Utility Functions

3.5.1 Fapi_flushPipeline()

Flushes the Flash Wrapper pipeline buffers

Synopsis

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

Parameters

u32UserFlashConfig [in]	User flash configuration bitfield

Copyright © 2025 Texas Instruments Incorporated

Description

This function flushes the Flash Wrapper data cache. The data cache must be flushed before the first non-API Flash read after an erase or program operation.

Return Value

None

3.5.2 Fapi_calculateEcc()

Calculates the ECC for the supplied address and 64-bit value

Synopsis

```
uint8 Fapi_calculateEcc(
                uint32 *pu32Address
                uint64 *pu64Data,
                uint8 u8Iterator
                       )
```

Parameters

pu32Address [in]	Pointer to the address of the 64-bit value to calculate the ECC
pu64Data [in]	Pointer to the address of the 64-bit value to calculate ECC on (should be in little endian order)
u8Iterator [in]	Iterator for interleaved banks to program/read/erase 1: For 128-bit program and read 2: For 512-bit program, read, and erase

Description

This function calculates the ECC for a 64-bit aligned word including address. There is no need to provide a left-shifted address to this function anymore. TMS320F28P65x Flash API takes care of it. When operating on interleaved banks, this function must be called twice (once with each iterator value, the start address stays the same).

Return Value

- 8-bit calculated ECC (upper 8 bits of the 16-bit return value should be ignored)
- If an error occurs, the 16-bit return value is 0xDEAD

3.5.3 Fapi_isAddressEcc()

Indicates is an address is in the Flash Wrapper ECC space

Synopsis

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

Parameters

u32Address [in]	Address to determine if it lies in ECC address space

Description

This function returns True if address is in ECC address space or False if it is not.

Copyright © 2025 Texas Instruments Incorporated

Return Value

- **FALSE** (Address is not in ECC address space)
- **TRUE** (Address is in ECC address space)

3.5.4 Fapi_remapEccAddress()

Takes ECC address and remaps it to main address space

Synopsis

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

Parameters

u32EccAddress [in]	ECC address to remap

Description

This function returns the main array Flash address for the given Flash ECC address. When the user wants to program ECC data at a known ECC address, this function can be used to obtain the corresponding main array address. Note that the Fapi_issueProgrammingCommand() function needs a main array address and not the ECC address (even for the Fapi_EccOnly mode).

Return Value

32-bit Main Flash Address

3.5.5 Fapi_getUserConfiguration()

Calculates a Flash API configuration bitfield based on the user-defined configuration parameters.

Synopsis

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

Parameters

BankType [in]	The type of bank the Flash API will be writing to. This should always be C29Bank.
FOTAStatus [in]	Whether or not FOTA is enabled: FOTA_Image: FOTA is enabled Active_Bank: FOTA is disabled

Description

This function calculates a Flash API configuration bitfield based on the user-defined configuration parameters.

Return Value

- 32-bit bitfield representing the user settings

Copyright © 2025 Texas Instruments Incorporated

3.5.6 Fapi_setFlashCPUConfiguration()

Commits the user flash configuration settings

Synopsis

```
uint32_t Fapi_SetFlashCPUConfiguration(
                                  uint32_t    u32UserFlashConfig,
                                  );
```

Parameters

u32UserFlashConfig [in]	User flash configuration bitfield

Description

This function commits a Flash API configuration bitfield based on the user-defined configuration parameters.

Return Value

- **Fapi_Status_Success** (success)
- **Fapi_Status_FsmBusy** (FSM busy)
- **Fapi_Error_FlashRegsNotWritable** (failure: Flash register writes failed. The user should make sure that the API is executing from the correct CPU).
- **Fapi_Error_FeatureNotAvailable** (failure: User passed a mode that is not supported)

4 SECCFG and BANKMGMT Programming Using the Flash API

Each Flash bank is made up of 2KB physical sectors. The nominal size (for example, 512KB) denotes the size of the MAIN region. Each Flash bank also includes two special regions:

- SECCFG: for storing SSU configuration settings
- BANKMGMT: for storing bank mode settings and firmware update metadata

The C29 Flash API supports programming of both the SECCFG and BANKMGMT sectors. For more information, see the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*.

4.1 BANKMGMT Programming

1. Issue an erase sector command. Users should always provide an active FLC1 address (see the table below) when erasing the BANKMGMT sector from Flash API.

BANKMGMT Sector Address	Device BANKMODE
0x10D80000	BANKMODE 0
0x10D80000	BANKMODE 1
0x10D80000	BANKMODE 2
0x10D80000	BANKMODE 3

Note

Users should always give an erase sector command using Fapi_issueAsyncCommandWithAddress() before programming the BANKMGMT sector. For more information, see the below code snippet or the flash programming example located at "f29h85x-sdk > examples > driverlib > single_core > flash > flash_mode0_128_program" for a demonstration of correct usage.

Copyright © 2025 Texas Instruments Incorporated

2. Issue a 128-bit programming command to the BANKMGMT sector, use the below tables to determine the correct values to program. A valid FLC1 address must be provided when programming the BANKMGMT sector using Flash API.

Table 4-1. BANKMGMT Registers

Register	Value	Notes
BANK_STATUS[63:0]	0x55555555_55555555	Offset 0 in the data buffer (see the below code snippet).
BANK_UPDATE_CTR[63:0]	0x00000000_00000000	Offset 8 in the data buffer (see the below code snippet). When programming the BANKMGMT sector, should always be set to 0. Flash API will internally read the counter from the active BANKMGMT sector, decrement it by 1, and program it to the inactive sector
BANKMODE[63:0]	Refer to BANKMODE Values	Offset 16 in the data buffer (see the below code snippet). Contains the current BANKMODE value

Table 4-2. BANKMODE Values

BANKMODE	BANKMODE[63:0] Value	Buffer With Offset
Mode 0	0x03	Buffer[16] = 0x03
Mode 1	0x06	Buffer[16] = 0x06
Mode 2	0x09	Buffer[16] = 0x09
Mode 3	0x0C	Buffer[16] = 0x0C

An example programming flow:

```
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. After programming the BANKMGMT sector, an external reset (XSRn) should be issued in order for boot ROM to read the new value and write it to the SSU register.

For more information, see the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*.

Copyright © 2025 Texas Instruments Incorporated

4.2 SECCFG Programming

SECCFG sectors are a portion of Flash memory designated for storing the SSU user configuration, or User Protection Policy (UPP). Each CPU has an active SECCFG start address, and an alternate (reserve) SECCFG start address. The reserve (alternate) SECCFG sectors' start address should be used when erasing/ programming SECCFG sector when using flash API. For SECCFG configuration details, see the *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*.

Note

The user must always provide the entire sector length (0x1000) when programming SECCFG using Flash API. Unused sector locations should be written as 0xFF.

Table 4-3. SECCFG Start Addresses

Bank Mode	CPUxSWAP	Region	CPU1/CPU2 Bank	CPU1 Address	CPU2 Address	CPU3 Bank	CPU3 Address
Mode 0		Active	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D8 9000
		Alternate	FLC1.B2/B3	0x10D8 5000	0x10D8 5800	FLC2.B2/B3	0x10D8 D000
Mode 1	SWAP = 0	Active	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D8 5000
		Alternate	FLC1.B2/B3	0x10D9 9000	0x10D9 9800	FLC2.B2/B3	0x10D9 D000
	SWAP = 1	Active	FLC1.B2/B3	0x10D8 1000	0x10D8 1800	FLC2.B2/B3	0x10D8 5000
		Alternate	FLC1.B0/B1	0x10D9 9000	0x10D9 9800	FLC2.B0/B1	0x10D9 D000
Mode 2		Active	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D9 1000
		Alternate	FLC1.B2/B3	0x10D8 5000	0x10D8 5800	FLC2.B2/B3	0x10D9 5000
Mode 3	SWAP = 0	Active	FLC1.B0/B1	0x10D8 1000	0x10D8 1800	FLC2.B0/B1	0x10D9 1000
		Alternate	FLC1.B2/B3	0x10D9 9000	0x10D9 9800	FLC2.B2/B3	0x10D9 D000
	SWAP = 1	Active	FLC1.B2/B3	0x10D8 1000	0x10D8 1800	FLC2.B2/B3	0x10D9 1000
		Alternate	FLC1.B0/B1	0x10D9 9000	0x10D9 9800	FLC2.B0/B1	0x10D9 D000

Steps for programming SECCFG are as follows:

1. Issue an erase sector command. Users should always provide an alternate (reserve) FLC address for the start address when erasing SECCFG using the Flash API. For more information, see the SECCFG Start Addresses.
2. Issue a 128-bit programming command to program the SECCFG sector.
   a. Users should always use the AutoEccGeneration programming mode
   b. The start address should be an alternate (reserve) sector address. For more information, see the SECCFG Start Addresses.
   c. The entire sector length must be programmed, unused locations should be programmed as 0xFF.
3. After programming the SECCFG sector, an external reset (XSRn) should be issued in order for boot ROM to read the active SECCFG sector value and write it to the necessary SSU register.

Copyright © 2025 Texas Instruments Incorporated

5 Recommended FSM Flows

5.1 New Devices From Factory

Devices are shipped erased from the factory. It is recommended, but not required, to do a blank check on devices received to verify that they are erased.

5.2 Recommended Erase Flow

Figure 5-1 describes the flow for erasing a sector(s) on a device. For further information, see 3.2.11, 3.2.2, 3.2.3.

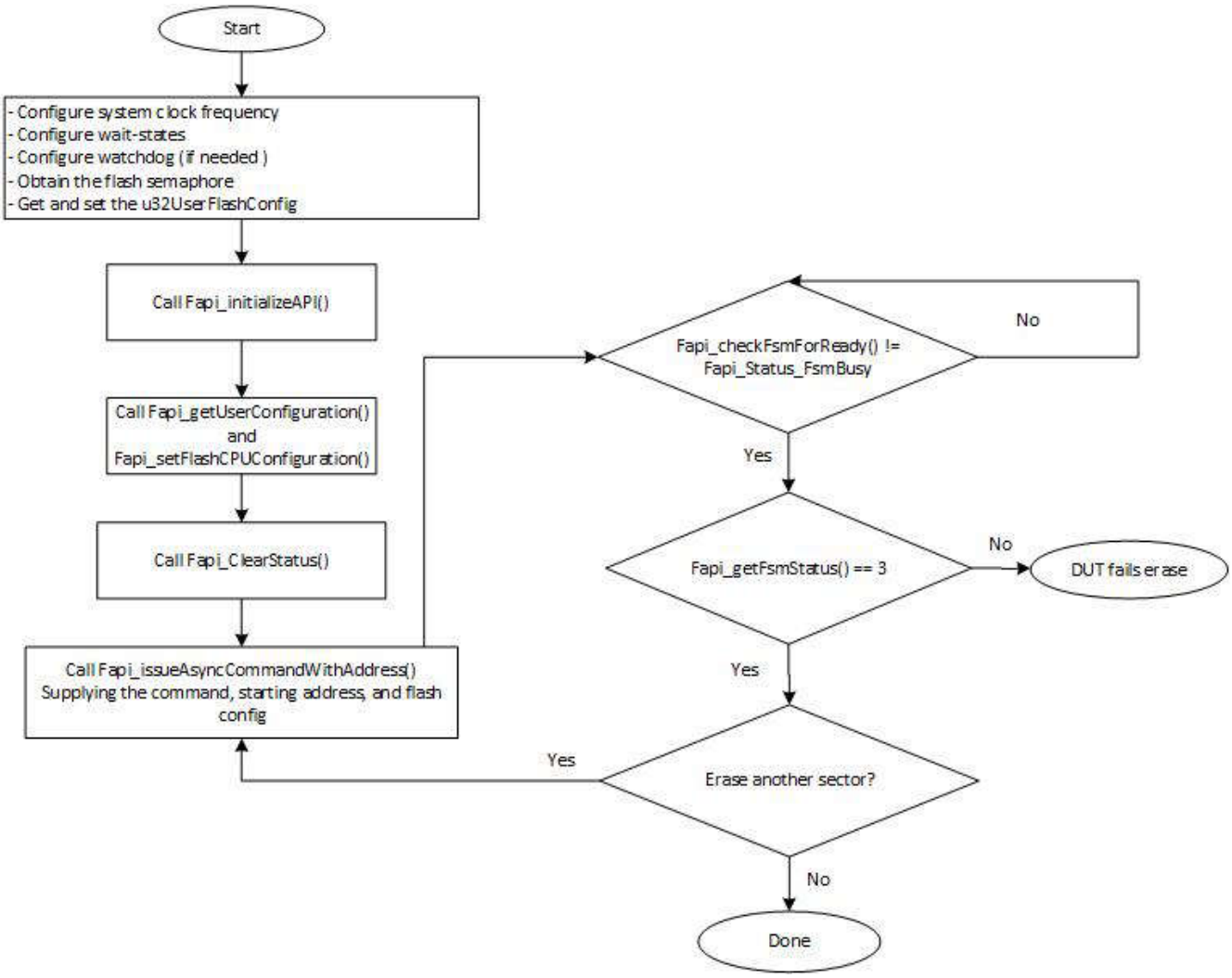

Figure 5-1. Recommended Erase Flow

Copyright © 2025 Texas Instruments Incorporated

5.3 Recommended Bank Erase Flow

Figure 5-2 describes the flow for erasing a Flash bank. For further information, see 3.2.11, 3.2.2, 3.2.4.

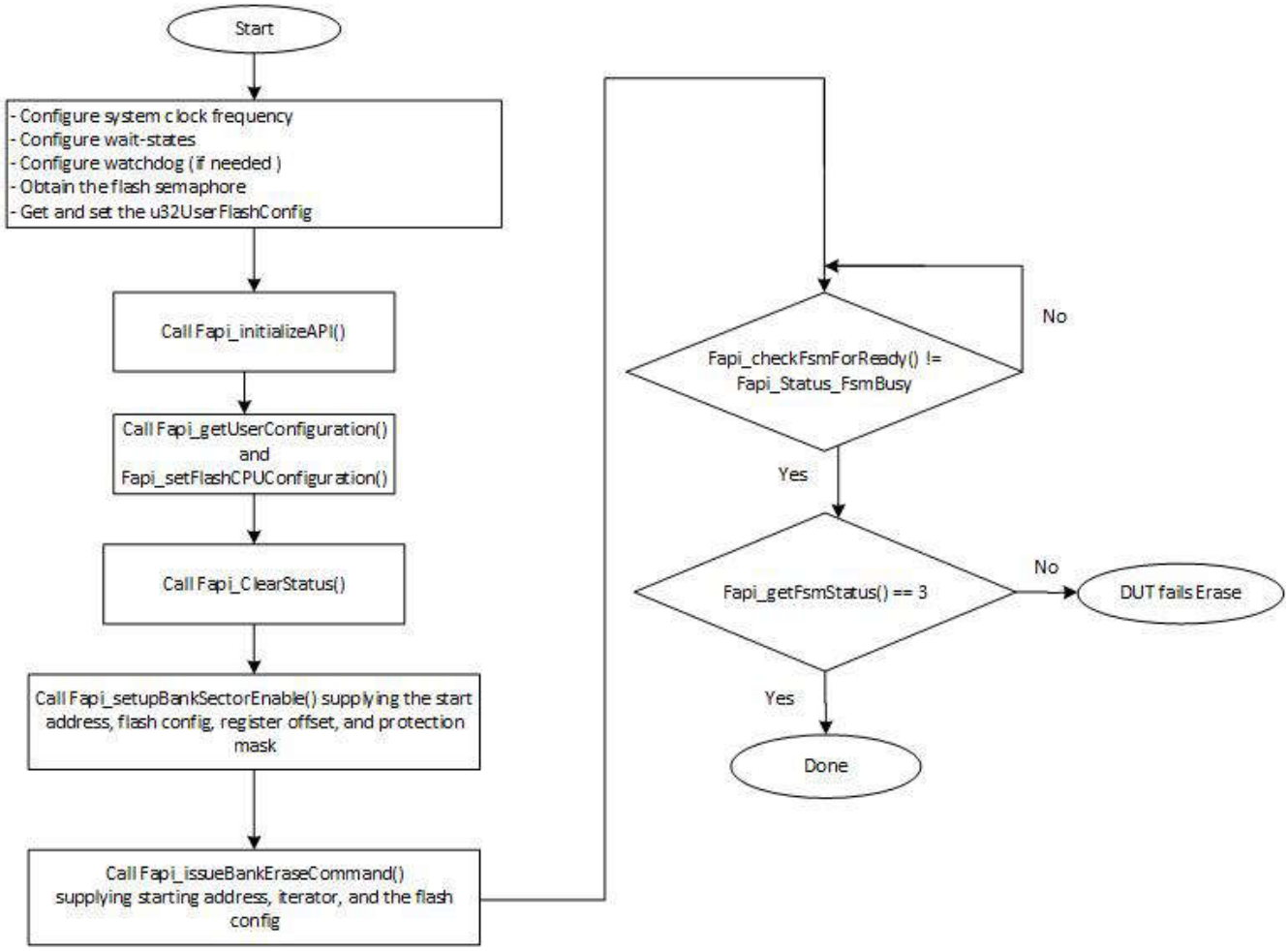

Figure 5-2. Recommended Bank Erase Flow

Copyright © 2025 Texas Instruments Incorporated

5.4 Recommended Program Flow

Figure 5-3 describes the flow for programming a device. This flow assumes the user has already erased all affected sectors or bank following the Recommended Erase Flow. For further information, see 3.2.11, 3.2.2, 3.2.5.

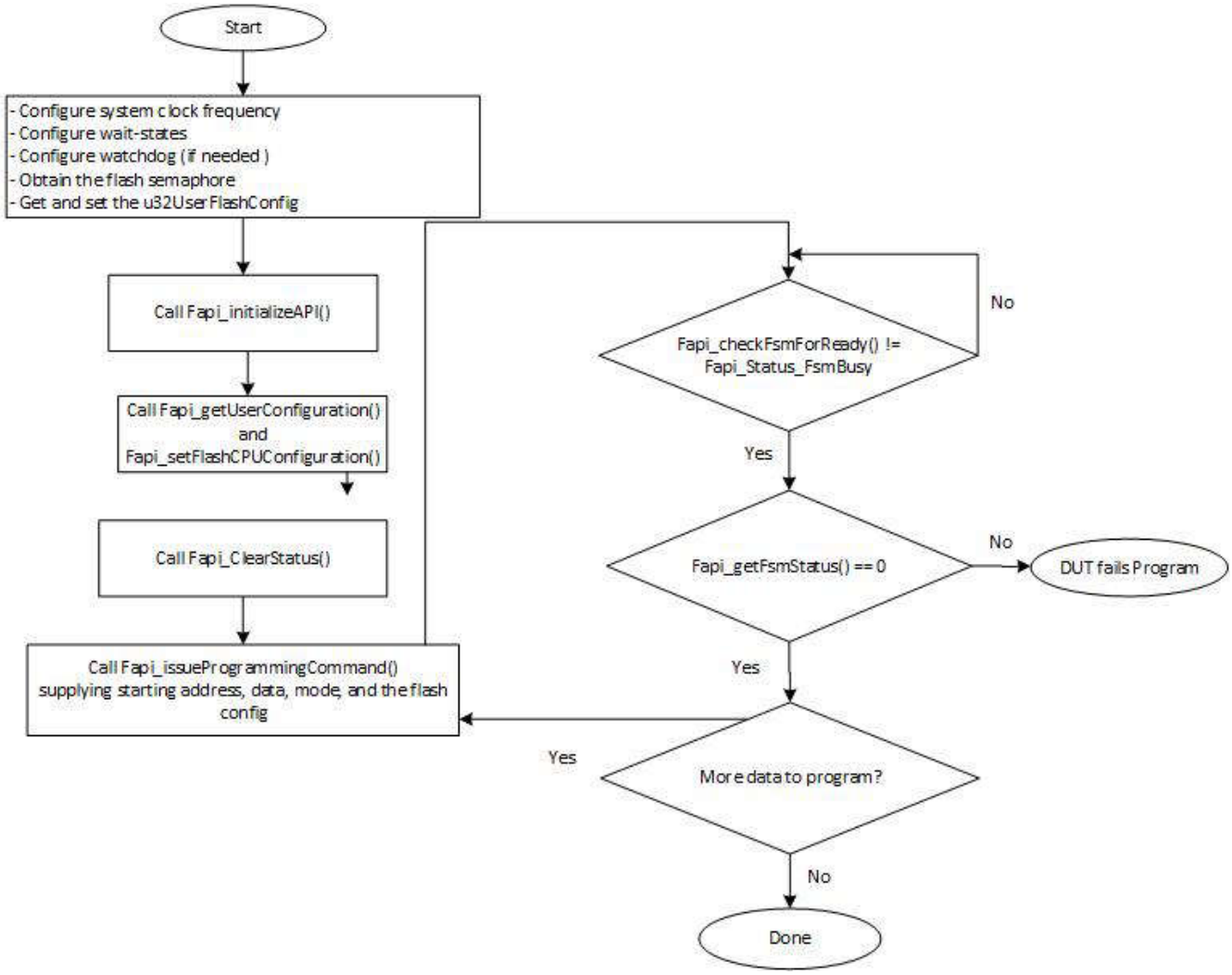

Figure 5-3. Recommended Program Flow

6 References

- Texas Instruments: *F29H85x and F29P58x Real-Time Microcontrollers Data Sheet*
- Texas Instruments: *F29H85x and F29P58x Real-Time Microcontrollers Technical Reference Manual*

Copyright © 2025 Texas Instruments Incorporated

A Flash State Machine Commands

Table A-1. Flash State Machine Commands

Command	Description	Enumeration Type	API Call(s)
Program Data	Used to program data to any valid Flash address	Fapi_ProgramData	Fapi_issueProgrammingCommand() Fapi_issueDataAndEcc512ProgrammingCommand() Fapi_issueAutoEcc512ProgrammingCommand() Fapi_issueDataOnly512ProgrammingCommands() Fapi_issueDataAndEcc512ProgrammingCommand() Fapi_issueEccOnly65ProgrammingCommand()
Erase Sector	Used to erase a Flash sector located by the specified address	Fapi_EraseSector	Fapi_issueAsyncCommandWithAddress()
Erase Bank	Used to erase a Flash bank	Fapi_EraseBank	Fapi_issueBankEraseCommand()
Clear Status	Clears the status register	Fapi_ClearStatus	Fapi_issueAsyncCommand()

Copyright © 2025 Texas Instruments Incorporated

B Typedefs, Defines, Enumerations and Structure

B.1 Type Definitions

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

B.2 Defines

```
#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 Enumerations

B.3.1 Fapi_FlashProgrammingCommandsType

This contains all the possible modes used in the Fapi_IssueProgrammingCommand().

```
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

This is used to indicate which type of Flash bank is being used.

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

B.3.3 Fapi_FlashStateCommandsType

This contains all the possible Flash State Machine commands.

```
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

This is the master type containing all possible returned status codes.

```
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

This lists the different production status values possible for the API.

```
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

This contains all the possible Flash Bank IDs.

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

B.3.7 Fapi_FLCID

Contains the NW controller addresses.

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


Copyright © 2025 Texas Instruments Incorporated

B.3.8 Fapi_BankMode

This contains all the possible bank modes.

```
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

Contains the possible CPU1 flash bank mapping configurations.

```
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

Contains the possible CPU3 flash bank mapping configurations.

```
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

Contains the possible SECCFG mapping configurations.

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

B.4 Structures

B.4.1 Fapi_FlashStatusWordType

This structure is used to return status values in functions that need more flexibility:

```
typedef struct
{
   uint32_t au32StatusWord[4];
}  ATTRIBUTE_PACKED Fapi_FlashStatusWordType;
```

B.4.2 Fapi_LibraryInfoType

This is the structure used to return API information:

```
typedef struct
{
   uint8_t  u8ApiMajorVersion;
```


Copyright © 2025 Texas Instruments Incorporated

```
    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;
```


Copyright © 2025 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated