TMS470M TI Flash EEPROM Emulation Driver

User's Guide

TEXAS INSTRUMENTS

Literature Number: SPNU518
May 2012

Contents

List of Figures

List of Tables

Read This First

About This Manual

This User's Manual serves as a software programmer's handbook for working with the TI FEE Driver. It provides necessary information regarding how to effectively install, build and use TI FEE Driver in user systems and applications.

It also provides details regarding the TI FEE Driver functionality, the requirements it places on the hardware and software environment where it can be deployed, how to customize/configure it, etc. It also provides supplementary information regarding steps to be followed for proper installation/un-installation of the TI FEE Driver.

Abbreviations

Table 0-1. Table of Abbreviations

Abbreviation	Description
TI FEE Driver	TI coined name for the product.
FEE	Flash EEPROM Emulation

Chapter 1
SPNU518–May 2012

TI FEE Driver Introduction

This chapter introduces the TI FEE Driver to the user by providing a brief overview of the purpose and construction of the TI FEE Driver along with hardware and software environment specifics in the context of TI FEE Driver deployment.

1.1 Overview

This section describes the functional scope of the TI FEE Driver and its feature set. It introduces the TI FEE Driver to the user along with the functional decomposition and run-time specifics regarding deployment of TI FEE Driver in user's application.

Many applications require storing small quantities of system related data (e.g., calibration values, device configuration) in a non-volatile memory, so that it can be used, modified or reused even after power cycling the system. EEPROMs are primarily used for this purpose. EEPROMs have the ability to erase and write individual bytes of memory many times over and the programmed locations retain the data over a long period even when the system is powered down.

The objective of TI FEE Driver is to provide a set of software functions intended to use a Sector of on-chip Flash memory as the emulated EEPROM. These software functions are transparently used by the application program for writing, reading and modifying the data.

A list of functions supported by the TI FEE Driver can be found in Section 1.1.1. The primary function responsible for Fee management is the TI_FeeTask function. This function shall operate asynchronously and with little or no user intervention after configuration, maintaining the Fee structures in Flash memory. This function should be called on a cyclic basis when no other pending Fee operations are pending so that it can perform internal operations.

1.1.1 Functions Supported in the TI FEE Driver

The TI FEE Driver provides the following functional services:

Initialization:

- TI_Fee_Start

Operations:

- TI_FEE_WriteAsync
- TI_FEE_WriteSync
- TI_FEE_Read
- TI_FEE_EraseBlock
- TI_FEE_InvalidateBlock
- TI_FEE_Shutdown

Information:

- TI_FEE_getStatus
- TI_FEE_getJobResult
- TI_FEE_getVersionInfo
- TI_FeeErrorCode

Internal Operations:

- TI_FeeTask
- TI_FEE_Format
- TI_FeeManager

1.1.2 Other Components

The TI FEE Driver requires the following components for complete deployment:

1. **TI Fee Configuration Files:** The user needs to generate the following two configuration files using HALCoGen to deploy and use TI FEE Driver.

   (a) fee_config.h

   (b) fee_config.c
   These two files define which Flash sectors to be used for EEPROM emulation, define Data Blocks, Block Size and other configuration parameters.
   HALCoGen also generates **device specific files** that defines the memory mapping for the Flash FEE bank.
2. **Flash API library:** The TI FEE Driver uses the Flash API library for performing program/erase operations. The appriopriate Flash API library depending on the type of Flash technology has to be included in the application to deploy and use the TI FEE Driver.

1.1.3 Development Platform

The TI FEE Driver was developed and validated on a system with the following operating system and software installed:

- Operating System : WinXP
- Codegeneration tools : TMS470 Code Generation tools 4.6.4

TI FEE Driver Design Overview

This chapter describes the implementation method followed for Flash EEPROM emulation in the TI FEE Driver.

2.1 Flash EEPROM Emulation Methodology

The EEPROM Emulation Flash bank is divided into two or more Virtual Sectors. Each Virtual Sector is further partitioned into several Data Blocks. A minimum of two Virtual Sectors are required for Flash EEPROM emulation.

The initialization routine (TI_Fee_Start) identifies which Virtual Sector to be used and marks it as Active. The data is written to the first empty location in the Active Virtual Sector. Whenever a Data Block has to be updated, it follows the link list concept wherein the previous Data block will be updated to point to the new location of the data. If there is insufficient space in the current Virtual Sector to update the data, it switches over to the next Virtual Sector and copies all the valid data from the other Data Blocks in the current Virtual Sector to the new one. After copying all the valid data, the current Virtual Sector is erased and the new one is marked as Active Virtual Sector. Any new data is now written into the new Active Virtual Sector and the erased Virtual Sector is used again once this new Virtual Sector has insufficient space.

Virtual Sectors and Data Blocks have certain space allocated to maintain the status information which is described in more detail in the following sections.

2.1.1 Virtual Sector Organization

The Virtual Sector is the basic organizational unit used to partition the EEPROM Emulation Flash Bank. This structure can contain one or more contiguous Flash Sectors contained within one Flash Bank. A minimum of 2 Virtual Sectors are required to support the TI FEE Driver.

The internal structure of the Virtual Sector contains a Virtual Sector Header, a static Data Structure and the remaining space is used for Data Blocks.

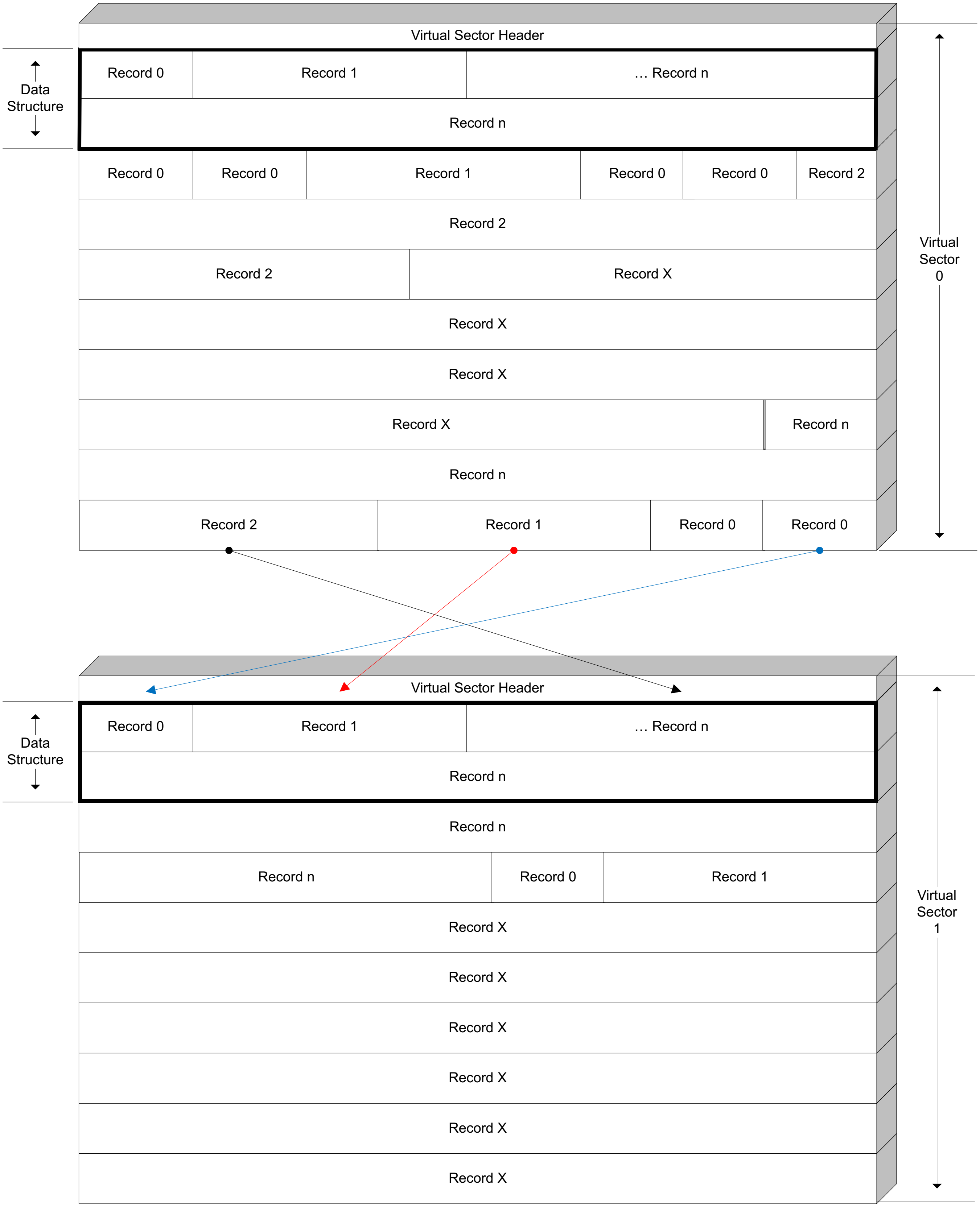

Figure 2-1. Virtual Sector Organization

2.1.1.1 Virtual Sector Header

The Virtual Sector Header consists of two 64bit words (16 bytes) that start at the first address of a Virtual Sector Structure. The state of the Virtual Sector Structure is maintained in the Virtual Sector Header.

64 bit Status Word		
40 bits Reserved	**Erase Count (20 bits)**	**Version Number (4 Bits)**

Figure 2-2. Virtual Sector Header

The Status Word is the first 64 bit word of the Virtual Sector Header and is used to indicate the current state of the Virtual Sector.

The following table indicates the various states of a Virtual Sector.

Table 2-1. Virtual Sector Header States

State	Value
Invalid Virtual Sector	0xFFFFFFFFFFFFFFFF
Empty Virtual Sector	0x0000FFFFFFFFFFFF
Copy Virtual Sector	0x00000000FFFFFFFF
Active Virtual Sector	0x000000000000FFFF
Ready for Erase	0x0000000000000000

Invalid Virtual Sector: This Virtual Sector is either in process of being erased or has not yet been initialized.

Empty Virtual Sector: This indicates the Virtual Sector has been erased and can be used to store data.

Copy Virtual Sector: This indicates that the Data Block Structure is being moved from a full Virtual Sector to this one to allow for moving of the Active Virtual Sector.

Active Virtual Sector: This Virtual Sector is the active one.

Ready for Erase: This Virtual Sector's Data Block Structure has been correctly replicated to a new Virtual Sector and is now ready to be erased and initialized for re-use.

Virtual Sector Information Record is the second 64 bit word in the Virtual Sector header. It is used to record information needed by the Virtual Sector management algorithm. Currently the first 4 bits are used to indicate the current version of the Virtual Sector and the next 20 bits are used to indicate the number of times the Virtual Sector has been erased. The erase count is incremented each time the Virtual Sector is erased. The remaining bits are reserved for future use

2.1.2 Data Block Organization

The Data Block is used to define where the data within a Virtual Sector is mapped. One or more variables can be within a Data Block based on the user definition. The smallest amount of data that can be stored within the Data Block is 64 bits. If the Data size exceeds 64 bits, the Data Packets are added in 64 bit increments. The Data Block Structure is limited to the size of the Virtual Sector it resides in.

NOTE: The size of all the Data Blocks cannot exceed the Virtual Sector length.

When a Data Packet write exceeds the available space of the current Virtual Sector, the Data Block structure is duplicated in the next Virtual Sector to be made active.

Record 0 Header	Record 1 Header	Record 1 Data Packet 0	Record 1 Data Packet 1	Record 1 Data Packet 2	Record 2 Header	Record 2 Data Packet 0	Record 2 Data Packet 1
Record 2 Data Packet 2	Record 2 Data Packet 3	Record 2 Data Packet 4	Record 2 Data Packet 5	Record 2 Data Packet 6	Record 3 Header	Record 3 Data Packet 0	Record 3 Data Packet 1
Record n-1 Data Packet 0	Record n-1 Data Packet 1	Record n-1 Data Packet 2	Record n-1 Data Packet 3	Record n-1 Data Packet 4	Record n-1 Data Packet 5	Record n-1 Data Packet 6	Record n Header
Record n Data Packet 0	Record n Data Packet 1	Record n Data Packet 2	Record n Data Packet 3	Record n Data Packet 4	Record n Data Packet 5	Record n Data Packet 6	Record n Data Packet 7

Figure 2-3. Data Block Structure

2.1.2.1 Data Block Header

The Data Block Header is 8 bytes in length and is used to indicate the location information (address) of valid data within a Virtual Sector.

Reserved	23 bit Address Offset
23 bit Address Offset	8 bit ECC Padding
32 bit Block Status	
32 bit Block Status	

Figure 2-4. Data Block Header - Logical Structure

A Standard Data Block Header has the following fields:

Table 2-2. Data Block Header Field Definitions

Bit(s)	Field	Description
63	Reserved	This bit is reserved.
62-40	23bit Address Offset	This field is used to indicate the address of the next data block that replaces this one. This is only updated after the Status of the Data Block Header that replaces this Data Block is marked as Valid.
39-32	8bit ECC Padding	This is used to allow writing of the 23bit Address Offset without creating an ECC error using ECC progressive programming techniques.
31-0	Status of the Block	These 32 bits indicate the Status of the Block. The following Table lists all the possible combinations for the Block Status.

Table 2-3. Data Block States

State	Value
Empty Block	0xFFFFFFFF
Start Program Block	0xFFFFFF00
Valid Block	0xFFFF0000
Invalid Block	0xFF000000
Corrupt Block	0x00000000

Block Status is used to ensure that data integrity is maintained even if the Block (data) update process is interrupted by an uncontrolled event such as a power supply failure or reset.

Empty Block: New Data can be written to this Block.

Start Program Block: This indicates that the Data Block is in the progress of being programmed with data.

Valid Block: This indicates that the Data Block is fully programmed and contains Valid Data.

Invalid Block: This indicates that the Data Block contains invalid or old data.

Corrupt Block: This indicates that the Data Block is corrupted and the Software should ignore this Block.

2.1.3 Supported Commands

The following list describes the supported commands:

1. **WriteAsync:** This command shall program a Flash Data block asynchronously.
2. **WriteSync:** This command shall program a Flash Data block synchronously.
3. **Read:** This command shall copy a continuous Flash Data block.
4. **Erase:** This command will erase a Flash Data block. It will update the address field in the Data Block to point to a location which is blank (all 1's).
5. **Invalidate Block:** This command shall mark the block as invalid in Data Block header.

2.1.4 Status Codes

This indicates the status of the Fee module. It can be in one of the following states:

1. **Uninitialized:** The Fee Module has not been initialized.
2. **Idle:** The Fee Module is currently idle.
3. **Busy:** The Fee Module is currently busy.
4. **Busy Internal:** The Fee Module is currently busy with internal management operations.

2.1.5 Job Result

This indicates the result of the last job. The job result can be any one of the following states:

1. **OK:** The last job has finished successfully.
2. **Pending:** The last job is waiting for execution or is currently being executed.
3. **Failed:** The last read/erase/write job failed.
4. **Inconsistent:** The requested block is inconsistent, it may contain corrupted data.
5. **Invalid:** The requested block has been invalidated. The requested read operation cannot be performed.

Installation Guide

This chapter discusses the TI FEE Driver installation, how and what software and hardware components to be availed in order to complete a successful installation of TI FEE Driver.

3.1 List of Installable Components

The installation files are summarized in the table below.

Table 3-1. Installation Setup Files

File Name	Description
TI FEE Driver 1.00.00 -Setup.exe	This file should be executed to install the TI FEE Driver files.
HALCoGen	This tool is used to configure the Fee module and also generate device specific files.

3.2 Component Folder

The files and directory structure of the installed TI FEE Driver in the system is described below. A viewgraph of the actual directory tree (collapsed image of the recursive directories) as seen in the deployed environment is depicted below.

Figure 3-1. View Graph of TI FEE Driver Directory Tree

Files created after the successful installation of TI FEE Driver are listed in the table below.

Table 3-2. TI FEE Driver File List

File Name	Destination Directory
ti_fee.h	Include
ti_fee_Types.h	Include
ti_fee_utils.c	Source
ti_fee_EraseBlock.c	Source
ti_fee_Format.c	Source
ti_fee_Info.c	Source
ti_fee_InvalidateBlock.c	Source
ti_fee_Links.c	Source
ti_fee_Manager.c	Source
ti_fee_Read.c	Source
ti_fee_Shutdown.c	Source
ti_fee_Start.c	Source
ti_fee_Task.c	Source
ti_fee_WriteAsync.c	Source
ti_fee_WriteSync.c	Source
ti_fee_CalcEcc.c	Source
M3_ECC_Enable_Disable.asm	Source

Files generated using HALCoGen are listed in the table below.

Table 3-3. TI FEE HALCoGen File List

File Name	Destination Directory
device_types.h	Include
fee_device.h	Include
fee_config.h	Include
fee_config.c	Source
fee_TMS470Mxx.h [(1)]	Include
fee_TMS470Mxx.c [(1)]	Source

(1) xx indicates device part number; e.g. if the target device chosen is TMS470MF066, then the device specific files generated are fee_TMS470MF066.h and fee_TMS470MF066.c

Getting Started Guide

This chapter describes the steps for using the TI FEE Driver. This chapter also discusses the TI FEE Driver run-time interfaces that comprise the API classification, usage scenarios and the API specification. The entire source code to implement the TI FEE Driver is included in the delivered product.

4.1 Build Procedure

The build procedure mentions how to go about building the TI FEE Driver into systems and applications.

1. The files created after installation of TI FEE Driver (listed inTable 3-2) should be included in the application.
2. The files listed in Table 3-3 (fee configuration files and device specific files) generated using HALCoGen should be included in the application. The configuration files (fee_config.h & fee_config.c) define which Flash sectors to be used for EEPROM emulation, define Data Blocks, Block Size and other configuration parameters whereas the device specific files define the memory mapping for the Flash FEE bank.
3. The appropriate Flash API library needs to be included in the application. The TI FEE Driver uses these APIs for performing program/erase operations on the Flash memory. The appropriate F035 Flash API library needs to be included if the device Flash technology is F035.

4.2 Symbolic Constants and Enumerated Data Types

This section summarizes the symbolic constants specified as either #define macros and/or enumerated C data types. Described alongside the macro or enumeration is the semantics or interpretation of the same in terms of what value it stands for and what it means.

Table 4-1. TI FEE Driver Symbolic Constants

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
TI_FeeStatusType	TI_FEE_OK	Function returned no error
	TI_FEE_ERROR	Function returned an error
VirtualSectorStatesType	VsState_Invalid =1	Virtual Sector is Invalid
	VsState_Empty =2	Virtual Sector is Empty
	VsState_Copy =3	Virtual Sector is Copy
	VsState_Active =4	Virtual Sector is Active
	VsState_ReadyForErase =5	Virtual Sector is Ready for Erase
BlockStatesType	Block_Empty=1	Block is Empty
	Block_StartProg=2	Write/Erase/Invalid operation is in progress on this Block
	Block_Valid=3	Block is Valid
	Block_Invalid=4	Block is Invalid
	Block_Corrupt=5	Block is Corrupt

Table 4-1. TI FEE Driver Symbolic Constants (continued)

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
Fee_ErrorCodeType	Error_Nil=0	
	Error_TwoActiveVS=1	
	Error_TwoCopyVS=2	
	Error_MorethanOneBank=3	
	Error_SetupStateMachine=4	
	Error_CopyButNoActiveVS=5	
	Error_NoActiveVS=6	
	Error_BlockInvalid=7	
	Error_NullDataPtr=8	
	Error_NoVSFoundforCopy=9	
	Error_InvalidVirtualSectorParameter=10	
	Error_ExceedSectorOnBank=11	
	Error_WriteVSHeader=12	
	Error_CalculateECC=13	
	Error_EraseVS=14	
	Error_BlockOffsetGtBlockSize=15	
	Error_LengthParam=16	
	Error_FeeUninit=17	
	Error_Suspend=18	
	Error_InvalidBlockIndex=19	
	Error_NoErase=20	
	Error_CurrentAddress=21	
TI_FeeStatusCodeType	Uninitialized	FEE Module is Uninitialized
	Idle	FEE Module is Idle
	Busy	FEE Module is Busy
	BusyInternal	FEE Module is performing internal operations
Fee_StatusWordType_UN	Read	If set to '1' indicates Read operation is in progress
	WriteAsync	If set to '1' indicates Async Write operation is in progress
	WriteSync	If set to '1' indicates Sync Write operation is in progress
	EraseBlock	If set to '1' indicates Erase operation is in progress
	InvalidateBlock	If set to '1' indicates Invalidate operation is in progress
	Copy	If set to '1' indicates Copy operation is in progress
TI_FEE_SW_MAJOR_VERSION	#define Macro which indicates the Major version of the FEE	
TI_FEE_SW_MINOR_VERSION	#define Macro which indicates the Minor version of the FEE	
TI_FEE_SW_PATCH_VERSION	#define Macro which indicates the Patch version of the FEE	

4.3 Data Structures

This section summarizes the entire user visible data structure elements pertaining to the TI FEE Driver run-time interfaces.

Table 4-2. TI FEE Driver Published Information Data Structure

Name	Fee_PublishedInformationType		
Description	Used to contain Published Information		
Fields	**Data type**	**Range**	**Description**
TI_FEE_BLOCK_OVERHEAD	uint8	0x8	Block OverHead in bytes
TI_FEE_VIRTUAL_PAGE_SIZE	uint8	0x8	Virtual Page Size in bytes
TI_FEE_PAGE_OVERHEAD	uint8	0x0	Page overhead in bytes
TI_FEE_VIRTUAL_SECTOR_OVERHEAD	uint8	0x10	Virtual Sector overhead in bytes

Table 4-3. TI FEE Driver General Configuration Data Structure

Name	Fee_GeneralConfigType		
Description	Used to contain General configuration information		
Fields	**Data type**	**Range**	**Description**
TI_FEE_INDEX	uint32	0	Instance ID of this module. Should always be 0
*TI_FEE_JOB_END_NOTIFICATION	Fee_CallbackType	-	Mapping to upper level job end notification
*TI_FEE_JOB_ERROR_NOTIFICATION	Fee_CallbackType		Mapping to upper level job error notification
TI_FEE_MAXIMUM_NUMBER_OF_LINKS	uint16	0-0xFFFE	Defines the maximum number of links allowed to maintain worst case access time.
TI_FEE_OPERATING_FREQUENCY	uint16	Refer Datasheet	Device Operating Frequency in MHz

4.4 TI FEE Parameter Configuration

This section describes the parameters which are used to configure the TI FEE driver.

4.4.1 Maximum Number of Links

Parameter Name	TI_FEE_MAXIMUM_NUMBER_OF_LINKS
Description	Defines the maximum number of links allowed for each block before switching from current Virtual Sector to a new Virtual Sector.
Default Value	0x100
Parameter Range	0x1 to 0xFFFE
Sample Configuration	#define TI_FEE_MAXIMUM_NUMBER_OF_LINKS 0x100

4.4.2 Job Error Notification

Parameter Name	TI_FEE_JOB_ERROR_NOTIFICATION
Description	Call back function to notify a Job Error.
Default Value	JobErrorNotification
Parameter Range	User defined function name.
Sample Configuration	#define TI_FEE_JOB_ERROR_NOTIFICATION JobErrorNotification

4.4.3 Job End Notification

Parameter Name	TI_FEE_JOB_END_NOTIFICATION
Description	Call back function to notify end of a Job.
Default Value	JobEndNotification
Parameter Range	User defined function name.
Sample Configuration	#define TI_FEE_JOB_END_NOTIFICATION JobEndNotification

4.4.4 Operating Frequency

Parameter Name	TI_FEE_OPERATING_FREQUENCY
Description	Device operating frequency in MHz. It is equivalent to the HCLK frequency in the TMS5470M clock tree.
Default Value	80
Parameter Range	Device dependent parameter. Refer to the device datasheet to know the range.
Sample Configuration	#define TI_FEE_OPERATING_FREQUENCY 80.0

4.4.5 Number of Blocks

Parameter Name	TI_FEE_NUMBER_OF_BLOCKS
Description	Defines the number of Data Blocks used for EEPROM emulation.
Default Value	0x1
Parameter Range	0x1 to 0xFFFE
Sample Configuration	#define TI_FEE_NUMBER_OF_BLOCKS 1

4.4.6 Number of Virtual Sectors

Parameter Name	TI_FEE_NUMBER_OF_VIRTUAL_SECTORS
Description	Defines the number of Virtual Sectors used for FEE.
Default Value	0x2
Parameter Range	Min: 0x2; Max: 0x4
Sample Configuration	#define TI_FEE_NUMBER_OF_VIRTUAL_SECTORS 2

4.4.7 TI FEE Virtual Sector Configuration

Array Name	TI_FeeVirtualSectorConfiguration	
Description	Used to define a Virtual Sector.	
Array Type	TI_FeeVirtualSectorConfigType This is a structure having the following members.	
Members	VirtualSectorNumber	Virtual Sector's Number.
	FlashBank	Flash Bank to use for Virtual Sector.
	StartSector	Starting Sector in the Bank for this Virtual Sector.
	EndSector	Ending Sector in the Bank for this Virtual Sector.

The configurations described in the following section are repeated for each Virtual Sector.

4.4.7.1 Virtual Sector Number

Parameter Name	VirtualSectorNumber
Description	Each Virtual Sector is assigned a number starting from 0x1
Default Value	0x1
Parameter Range	Min: 0x1; Max: 0x4

4.4.7.2 Flash Bank

Parameter Name	FlashBank
Description	Indicates the Flash Bank used by the Virtual Sector. All the Virtual Sectors should use the same Flash Bank.
Default Value	0x1
Parameter Range	Bank 0 is not supported for FEE. Any other Flash Bank on the device can be used. Please refer to the device datasheet “Flash Memory Map” for more details.

4.4.7.3 Start Sector

Parameter Name	StartSector
Description	Indicates the Flash Sector in the Bank used by the Virtual Sector as the Start sector.
Default Value	0x0
Parameter Range	Device specific, can use any Sector of the selected Flash Bank. Please refer to the device datasheet “Flash Memory Map” for more details.

4.4.7.4 End Sector

Parameter Name	EndSector
Description	Indicates the Flash Sector in the Bank used by the Virtual Sector as the End sector.
Default Value	0x0
Parameter Range	Device specific, can use any Flash Sector of the selected Flash Bank. It should be greater than the FEE Start Sector. Please refer to the device datasheet “Flash Memory Map” for more details.

4.4.7.5 Sample Virtual Sector Configuration

The following code snippet indicates one of the possible configurations for the Virtual Sectors from the file fee_config.c:

```
/* Virtual Sector Configuration */
const TI_FeeVirtualSectorConfigType TI_FeeVirtualSectorConfiguration[ ] =
{
    /* Virtual Sector 1 */
    {
        1,      /* Virtual sector number */
        1,      /* Bank                 */
        0,      /* Start Sector          */
        0       /* End Sector            */
    },
    /* Virtual Sector 2 */
    {
        2,      /* Virtual sector number */
        1,      /* Bank                 */
        1,      /* Start Sector          */
        1       /* End Sector            */
    },
};
```

4.4.8 TI FEE Block Configuration

Array Name	TI_Fee_BlockConfiguration	
Description	Used to define a Data Block.	
Array Type	TI_ FeeBlockConfigType This is a structure having the following members.	
Members	BlockNumber	Indicates Block's Number.
	BlockSize	Defines Block's Size in bytes.
	NumberOfWriteCycles	Number of write cycles required for this block
	DeviceIndex	Indicates the device index.

The configurations described in the following section are repeated for each Data Block.

4.4.8.1 Block Number

Parameter Name	BlockNumber
Description	Each block is assigned a unique number starting from 0x1.
Default Value	0x1
Parameter Range	Min: 0x1; Max: 0xFFFE

4.4.8.2 Block Size

Parameter Name	BlockSize
Description	Indicates the size of the Block in bytes.
Default Value	0x8
Parameter Range	0x8 to 0xFFFF (Multiples of 8)

4.4.8.3 Number of Write Cycles

Parameter Name	NumberOfWriteCycles
Description	Indicates the number of clock cycles required to write to a flash address location.
Default Value	0x10
Parameter Range	Device or core/flash tech dependent parameter.

4.4.8.4 Device Index

Parameter Name	DeviceIndex
Description	Indicates the device index. This will always be 0.
Default Value	0x0
Parameter Range	Fixed to 0x0

4.4.8.5 Sample Block Configuration

The following code snippet indicates one of the possible configurations for the Blocks from the file fee_config.c:

```
/* Block Configuration */

const TI_FeeBlockConfigType TI_Fee_BlockConfiguration[ ] =

{
    /* Block 1 */
    {
        0x01,      /* Block number                    */
        0x004,     /* Block size                       */
        0x10,      /* Block number of write cycles     */
        0          /* Device Index                     */
    },
    /* Block 2 */
    {
        0x02,       /* Block number                       */
        0x008,      /* Block size                       */
        0x10,         /* Block number of write cycles     */
        0           /* Device Index                     */
    },
    /* Block 3 */
    {
        0x03,       /* Block number                     */
        0x0004,     /* Block size                       */
        0x10,         /* Block number of write cycles     */
        0        /* Device Index                     */
    },

/* Block 4 */
    {
        0x04,       /* Block number                     */
        0x001A,     /* Block size                       */
        0x10,        /* Block number of write cycles     */
        0        /* Device Index                      */
    }

};
```

4.4.9 Block OverHead

Parameter Name	TI_FEE_BLOCK_OVERHEAD
Description	Indicates the number of bytes used for Block Header.
Default Value	0x8
Parameter Range	Fixed to 0x8
Sample Configuration	#define TI_FEE_BLOCK_OVERHEAD 8

4.4.10 Page OverHead

Parameter Name	TI_FEE_PAGE_OVERHEAD
Description	Indicates the Page Overhead in bytes.
Default Value	0x0
Parameter Range	Fixed to 0x0
Sample Configuration	#define TI_FEE_PAGE_OVERHEAD 0

4.4.11 Virtual Sector OverHead

Parameter Name	TI_FEE_VIRTUAL_SECTOR_OVERHEAD
Description	Indicates the number of bytes used for Virtual Sector Header.
Default Value	0x10
Parameter Range	Fixed to 0x10
Sample Configuration	#define TI_FEE_VIRTUAL_SECTOR_OVERHEAD 16

4.4.12 Virtual Sector Page Size

Parameter Name	TI_FEE_VIRTUAL_PAGE_SIZE
Description	Indicates the virtual page size in bytes.
Default Value	0x8
Parameter Range	Fixed to 0x8
Sample Configuration	#define TI_FEE_VIRTUAL_PAGE_SIZE 8

4.4.13 Driver Index

Parameter Name	TI_FEE_INDEX
Description	Instance ID of TI FEE module. Should always be 0x0.
Default Value	0x0
Parameter Range	Fixed to 0x0
Sample Configuration	#define TI_FEE_INDEX 0

4.4.14 Read Cycle Count

Parameter Name	TI_FEE_READ_CYCLE_COUNT
Description	Indicates the number of clock cycles required to access a flash address location.
Default Value	0xA
Parameter Range	Device or core/flash technology dependent parameter.
Sample Configuration	#define TI_FEE_READ_CYCLE_COUNT 10

4.4.15 Enable ECC Correction

Parameter Name	TI_FEE_FLASH_ERROR_CORRECTION_ENABLE
Description	Used to enable/disable Error Correction.
Default Value	0
Parameter Range	0 (FALSE) or 1 (TRUE)
Sample Configuration	#define TI_FEE_FLASH_ERROR_CORRECTION_ENABLE 0

4.5 API Classification

This section introduces the application-programming interface for the TI FEE Driver by grouping them into logical units. This is intended for the user to get a quick understanding of the TI FEE Driver APIs. For detailed descriptions, please refer to the API specification in Section 4.7.

4.5.1 Initialization

The TI FEE Driver APIs that are intended for use in *initialization* of the FEE module are listed below.

Table 4-4. TI FEE Driver Initialization APIs

Name	Description
TI_Fee_Start	Used to initialize the FEE module

4.5.2 Data Operations

The TI FEE Driver APIs that are intended for performing *Data operations* on Data Blocks are listed below.

Table 4-5. TI FEE Driver Data Operation APIs

Name	Description
TI_FEE_WriteAsync	Used to initiate an Asynchronous Write Operation to a Data Block. TI_FeeTask function should be called at regular intervals to finish the Async Write Operation.
TI_FEE_WriteSync	Used to perform a Synchronous Write Operation to a Data Block.
TI_FEE_Read	Used to read Data from a Data Block.
TI_FEE_EraseBlock	Used to initiate an Erase Operation of a Data Block. TI_FeeTask function should be called at regular intervals to finish the Write Operation.
TI_FEE_InvalidateBlock	Used to initiate an Invalidate Operation on a Data Block. TI_FeeTask function should be called at regular intervals to finish the Write Operation.
TI_FEE_Shutdown	This function completes the Async jobs which are in progress by performing a bulk Data Write while shutting down the system synchronously.

4.5.3 Information

The TI FEE Driver APIs that are intended to get information about the status of the FEE Module are listed below.

Table 4-6. TI FEE Driver Information APIs

Name	Description
TI_FEE_getVersionInfo	Used to get the Driver version.
TI_FEE_getStatus	Used to get the status of the FEE module.
TI_FEE_getJobResult	Used to get the job result of a Data Operation.
TI_FeeErrorCode	Used to determine occurrence of an error.

4.5.4 Internal Operations

The TI FEE Driver APIs that are used to perform internal operations of the FEE Module are listed below.

Table 4-7. TI FEE Driver Internal Operation APIs

Name	Description
TI_FeeTask	Used to complete the Data Operations initiated by any of the Data Operation functions.
TI_FeeManager	Used to perform internal operations (Copy, Erase Virtual Sector).
TI_FEE_Format	Used to erase all the configured Virtual Sectors.

4.6 FEE Operation Flow

This section depicts a flow chart for a typical FEE operation.

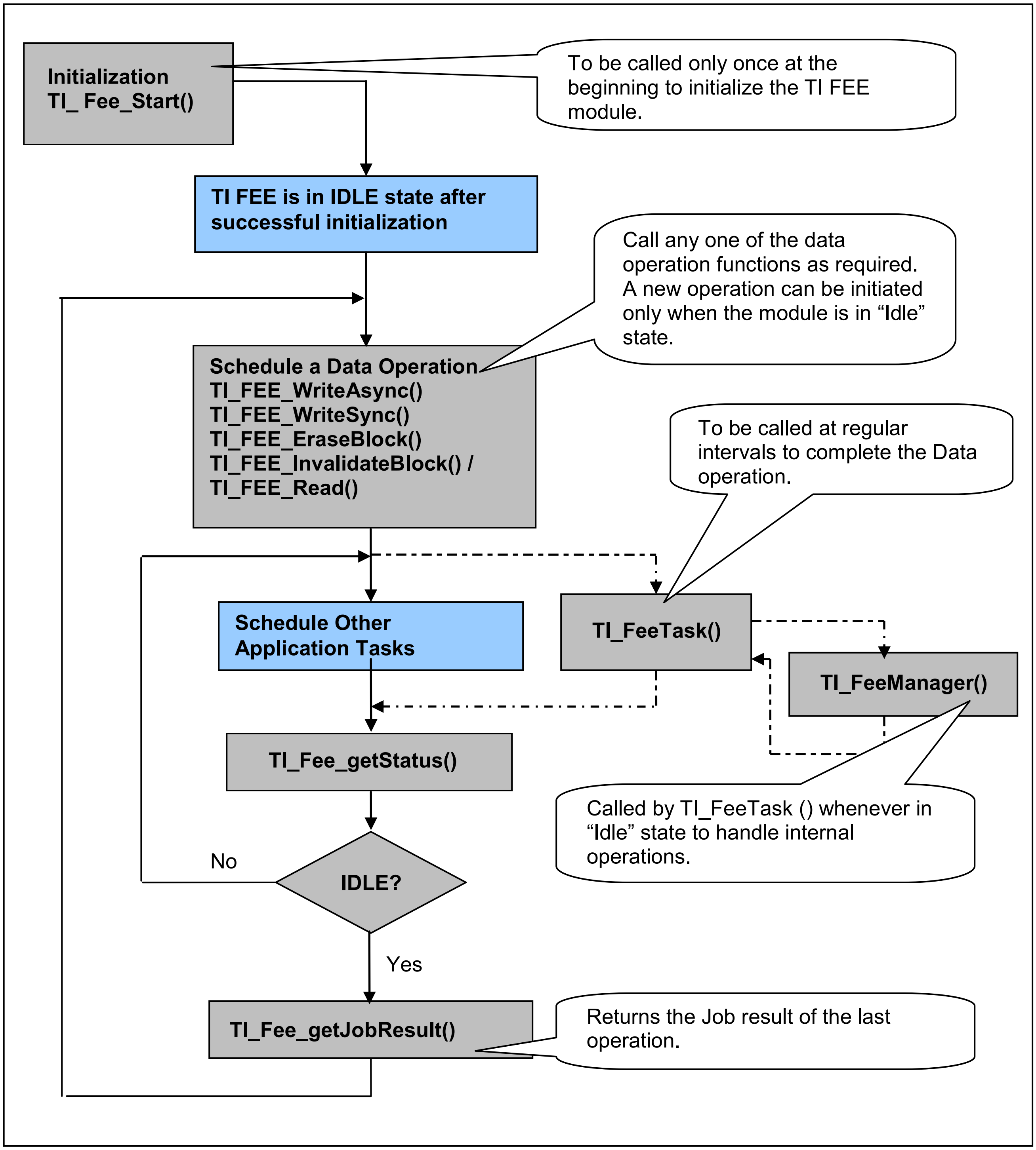

Figure 4-1. Flow Chart of a Typical FEE Operation

4.7 API Specification

This section constitutes the detailed reference for the entire API set published to users of the TI FEE Driver.

4.7.1 TI FEE Driver Functions

4.7.1.1 Initialization Function (TI_Fee_Start)

Function to initialize the TI Fee module.

Syntax

```
void TI_Fee_Start (void)
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

None

Description

This function provides functionality for initializing the TI FEE module. This routine must be called only once at the beginning before commencing any data operation.

4.7.1.2 Async Write Function (TI_FEE_WriteAsync)

Function to initiate an Async Write job.

Syntax

```
Std_ReturnType TI_FEE_WriteAsync(
        uint16 BlockNumber,
        uint8* DataBufferPtr)
```

Sync/Async

Asynchronous

Parameters (in)

BlockNumber	Number of logical block, also denoting start address of that block in Flash memory.
DataBufferPtr	Pointer to data buffer.

Return Value

Std_ReturnType

- E_OK: The write job was accepted by the TI Fee module.
- E_NOT_OK: The write job was not accepted by the TI Fee module.

Description

This function initiates an Asynchronous Write operation to a Data Block. TI_FEE_Task() function should be called at regular intervals to finish the Async Write operation.

4.7.1.3 Sync Write Function (TI_FEE_WriteSync)

Function to program Data to a Block synchronously.

Syntax

```
Std_ReturnType TI_FEE_WriteSync(
        uint16 BlockNumber,
        uint8* DataBufferPtr)
```

Sync/Async

Synchronous

Parameters (in)

BlockNumber	Number of logical block, also denoting start address of that block in Flash memory.
DataBufferPtr	Pointer to data buffer.

Return Value

Std_ReturnType

- E_OK: The write job was accepted by the TI Fee module.
- E_NOT_OK: The write job was not accepted by the TI Fee module.

Description

This function provides the functionality to program data to a Block synchronously.

4.7.1.4 Read Function (TI_FEE_Read)

Function to read data from a Block.

Syntax

```
Std_ReturnType TI_FEE_Read(
        uint16 BlockNumber,
        uint16 BlockOffset,
        uint8* DataBufferPtr,
        uint16 Length)
```

Sync/Async

Synchronous

Parameters (in)

BlockNumber	Number of logical block, also denoting start address of that block in Flash memory.
BlockOffset	Read address offset inside the block.
DataBufferPtr	Pointer to data buffer.
Length	Number of bytes to read.

Return Value

Std_ReturnType

- E_OK: The Read job was accepted by the TI Fee module.
- E_NOT_OK: The Read job was not accepted by the TI Fee module.

Description

This function provides functionality for reading of data from a Block.

4.7.1.5 Erase Function (TI_FEE_EraseBlock)

Function to initiate Erase operation on a Data Block.

Syntax

```
Std_ReturnType TI_FEE_EraseBlock(
        uint16 BlockNumber)
```

Sync/Async

Asynchronous

Parameters (in)

BlockNumber	Number of logical block, also denoting start address of that block in Flash memory.

Return Value

Std_ReturnType

- E_OK: The Erase job was accepted by the TI Fee module.
- E_NOT_OK: The Erase job was not accepted by the TI Fee module.

Description

This function provides functionality for Erasing a Data Block asynchronously. TI_FEE_Task() function should be called at regular intervals to finish the Erase operation.

4.7.1.6 Invalidate Function (TI_FEE_InvalidateBlock)

Function to initiate an Invalidate operation on a Data Block.

Syntax

```
Std_ReturnType TI_FEE_InvalidateBlock(
        uint16 BlockNumber)
```

Sync/Async

Asynchronous

Parameters (in)

BlockNumber	Number of logical block, also denoting start address of that block in Flash memory.

Return Value

Std_ReturnType

- E_OK: The Invalidate Block job was accepted by the TI Fee module.
- E_NOT_OK: The Invalidate Block job was not accepted by the TI Fee module.

Description

This function provides functionality for invalidating a D ata Block asynchronously. TI_FEE_Task() function should be called at regular intervals to finish the Invalidate Block operation.

4.7.1.7 Shutdown Function (TI_FEE_Shutdown)

Function to perform bulk Data write prior to system shutdown.

Syntax

```
Std_ReturnType TI_FEE_Shutdown()
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

Std_ReturnType

- E_OK: The Async job was accepted by the TI Fee module.
- E_NOT_OK: The Async job was not accepted by the TI Fee module.

Description

This function provides functionality for performing a bulk data write when shutting down the system synchronously. This function completes the Async jobs which are in progress by performing a bulk Data Write while shutting down the system synchronously.

4.7.1.8 Get Version Info Function (TI_FEE_getVersionInfo)

Function to return the version information of the TI Fee module.

Syntax

```
void TI_FEE_getVersionInfo(
        Std_VersionInfoType* VersionInfoPtr)
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

VersionInfoPtr

- Pointer to standard version information structure.

Description

This function returns the version information for the TI Fee module.

TI Fee specific version numbers MM.mm.rr

- MM – Major Version
- mm – Minor Version
- rr – Revision

4.7.1.9 Get Status Function (TI_FEE_getStatus)

Function gets the status of the TI Fee module.

Syntax

```
TI_FeeStatusCodeType TI_FEE_getStatus()
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

TI_FeeStatusCodeType

- UNINIT: TI Fee Module has not been initialized.
- IDLE: TI Fee Module is currently idle.
- BUSY: TI Fee Module is currently busy.
- BUSY_INTERNAL: TI Fee Module is currently busy with internal management operations.

Description

This function returns the status of the TI FEE module.

4.7.1.10 Get Job Result Function (TI_FEE_getJobResult)

Function gets the job result from the TI Fee module.

Syntax

```
TI_FeeJobResultType TI_FEE_getJobResult()
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

TI_FeeJobResultType

- JOB_OK: The last job has finished successfully.
- JOB_PENDING: The last job is waiting for execution or is currently being executed.
- JOB_CANCELLED: The last job has been cancelled.
- JOB_FAILED: The last job failed.
- BLOCK_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data.
- BLOCK_INVALID: The requested block has been invalidated. The requested read operation cannot be performed.

Description

This function returns the result of the last job synchronously.

4.7.1.11 Get Error Code (TI_FeeErrorCode)

Returns ‘0’ if no error has occurred else it returns an Error code.

Syntax

```
TI_FeeErrorCodeType TI_FeeErrorCode()
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

TI_FeeErrorCodeType

- Returns an Error Code.

Description

This function provides functionality to identify occurrence of an error. It returns ‘0’ if no error has occurred else it returns an Error code.

4.7.1.12 Task Function (TI_FeeTask)

Function to handle the requested Async data operations.

Syntax

```
void TI_FeeTask(void)
```

Sync/Async

Asynchronous

Parameters (in)

None

Return Value

None

Description

This function handles the Write/Erase/Invalidate asynchronous jobs initiated by TI_Fee_WriteAsync()/TI_Fee_EraseBlock()/TI_Fee_InvalidateBlock() functions.

This function should be called at regular intervals by a scheduler. This function internally calls another function “TI_FeeManager” whenever there is no other job pending (“IDLE” State). “TI_FeeManager” function handles all the background tasks/internal operations to manage the TI FEE module.

NOTE: The user has to schedule the tasks/data operations such that the TI FEE module is in “IDLE” state for some time so that the internal operations are handled correctly.

4.7.1.13 Manager Function (TI_FeeManager)

Function to handle the requested Async data operations.

Syntax

```
TI_FeeStatusType TI_FeeManager(void)
```

Sync/Async

Asynchronous

Parameters (in)

None

Return Value

TI_FeeStatusType

- TI_FEE_OK: The job was completed.
- TI_FEE_ERROR: The job was not completed due to an error.

Description

The function TI_FeeManager() manages the Flash EEPROM Emulation and is called when no other job is pending by the TI_FeeTask function. This function handles all the background tasks to manage the FEE.

This routine is responsible for:

- Determining whether a Virtual Sector Copy operation is in progress. If so, it should identify all the Valid Data Blocks in the old Virtual Sector and copy them to the new Virtual Sector.
- Determining if any of the Virtual Sector needs to be erased. If so, it should erase that particular Virtual Sector.
- This function is only called when the Fee module is in IDLE state. It should set the Fee module to BUSY_INTERNAL state.

4.7.1.14 Format Function (TI_FEE_Format)

Function formats all the Virtual Sectors.

Syntax

```
void TI_FEE_Format(void)
```

Sync/Async

Synchronous

Parameters (in)

None

Return Value

None

Description

This function provides functionality for erasing all the Virtual Sectors synchronously.

NOTE: Calling this function will result in loss of data. This function should be called only if you want to reconfigure the Data Blocks/Virtual Sectors or detect a serious error condition.

Revision History

Table A-1 lists the version history of this user's guide.

Table A-1. Version History

Version	Additions/Modifications/Deletions
1.0	Initial version
1.1	Added description for Configuration parameters