TI FEE Driver *User Guide*

Version 1.12

Aug17, 2016

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards ought to be provided by the customer so as to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service is an unfair and deceptive business practice, and TI is neither responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

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

1-1. Table of Abbreviations

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

Revision History

Version	Date	Revision History
1.0	09/25/2012	Initial version
1.1	11/12/2012	
1.2	03/12/2013	Add more info for datasets
1.3	04/19/2013	Add section Reset Behavior
1.4	06/04/2013	Add new configuration parameters. Add software revision history.
1.5	07/05/2013	Software revision updated.
1.6	12/04/2013	ReadSync API added. Format API modified. New Configuration Tag Added.
1.7	05/23/2014	Software Revision History updated.
1.8	06/02/2014	Documentation corrections
1.9	01/21/2015	Update software revision table. New API added for supporting manual Suspend/Resume of erasing of sector
1.10	02/13/2015	Update software revision table.
1.11	02/22/2016	Update software revision table. Update documentation related to FEE errors.
1.12	06/29/2016	Update software revision table. Update documentation. Replace CRC with checksum. Added text for unconfigured blocks to copy parameter.

Software Revision History

Version	Date	Revision History
00.01.00	08/31/2012	Initial version
00.01.01	10/29/2012	Changes for implementing Error Recovery
00.01.02	11/30/2012	Misra Fixes, Memory segmentation changes
00.01.03	01/14/2013	Changes as requested by Vector. If there is an immediate erase/invalidate block request

		before writing of a block, API should return the job status as JOB_OK.
00.01.04	02/12/2013	Integration issues fix. Fixed issues regarding integration of FEE with NvM.
00.01.05	03/04/2013	Added Deleting a block feature
00.01.06	03/11/2013	Added feature: copying of unconfigured blocks.
00.01.07	03/15/2013	Added feature: Number of 8 bytes writes, fixed issue with copy blocks.
00.01.08	04/05/2013	Added feature: CRC check for unconfigured blocks, Main function modified to complete writes as fast as possible, Added Non polling mode support.
00.01.09	04/19/2013	Warning removal, Added feature comparison of data during write.
00.01.10	06/11/2013	Fixed issue with erase sector. Also fixed issue with 2 EEPROM's where if one EEPROM is locked with error condition, other EEPROM will not get locked.
00.01.11	07/05/2013	Warning removal. Fixed issue with Fee_manager API, if number of Virtual Sectors are more than 2.
01.13.00	12/04/2013	Format API modified. Read Sync API added. MISRA C fixes. WriteSync API corrected. New Configuration Tag Added.
01.13.01	05/23/2014	Data Abort issue fixed. Unexpected Job Result issue fixed.
01.14.00	05/23/2014	Unexpected Job Result issue fixed.
01.15.00	06/01/2014	Support for new devices TMS570LC4357/RM57x added.
01.16.00	07/15/2014	Misra warnings removal.
01.16.01	09/12/2014	Manual Suspend/Resume of erasing of sector added.
01.17.00	10/15/2014	RAM Optimization changes.
01.17.01	10/30/2014	Support for new devices TMS570LS07xx,TMS570LS09xx,

		TMS570LS05xx, RM44Lx added.
01.17.02	02/13/2015	FLEE errata fix for SPNZ215A. Applicable for TMS570LS04xx, RM42x, TMS570LC43xx, RM57x, TMS570LS07xx, TMS570LS09xx, TMS570LS05xx, RM44Lx devices.
1.18.00	10/12/2015	If morethan one data set is config ured, then a valid block may get invalidated if multiple valid blocks are present in FEE memory.Applicable when driver used in Autosar context.
1.18.01	11/17/2015	In TI_FeeInternal_FeeManager, do not change the state to IDLE,after completing the copy operation. Applicable when driver used in Autosar context.
1.18.02	02/05/2016	Bugfix for "If sector copy operation is interrupted, during next initialization, FEE does not read the block offset address correctly"
1.18.03	06/30/2016	CRC wording changed Checksum since driver uses checksum algorithm.
1.19.00	08/08/2016	If FEE used a partially erased sector, FEE can read from unimplemented memory location. Code changes done to not to use a partially erased sector. Option "Check EEPROM Address Range" is removed from HALCoGen GUI. Address range will always be checked during read/write.
1.19.01	08/12/2016	Synchronous Write API updated to not to copy the block which is already copied. Format API modified to erase all configured sectors, if multiple sectors are combined to form a larger virtual sector. TI_FeeInternal_FeeManager modified to update block copy status of unconfigured blocks correctly. TI_FeeInternal_UpdateBlockOffsetArray modified to update write addresses correctly, if FEE did not find a valid write address. TI_FeeInternal_WriteDataF021 API updated for LE MCU's for updating correct ECC.

Contents

Table of tables

Table of figures

Chapter 1

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 below. The primary function responsible for Fee management is the TI_FeeInternal_FeeManager function. This function shall operate asynchronously and with little or no user intervention after configuration, maintaining the Fee structures in Flash memory. This function will be called by TI_Fee_MainFunction 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_Init

Operations:

- TI_Fee_WriteAsync
- TI_Fee_WriteSync
- TI_Fee_Read
- TI_Fee_ReadSync
- TI_Fee_EraseImmediateBlock
- TI_Fee_InvalidateBlock
- TI_Fee_Shutdown
- TI_Fee_Cancel
- TI_Fee_Format

Information:

- TI_Fee_GetStatus
- TI_Fee_GetJobResult
- TI_Fee_GetVersionInfo

Internal Operations:

- TI_Fee_MainFunction
- TI_FeeInternal_FeeManager

Error Information and Recovery:

- TI_FeeErrorCode
- TI_Fee_ErrorRecovery

Suspend/Resume Erase of Sector:

- TI_Fee_SuspendResumeErase

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. ti_fee_cfg.h

   b. ti_fee_cfg.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 apprioprate 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. For TMS570 devices, F021 library version should be 02.00.00 or greater.

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 : Win7
- Codegeneration tools : TMS570 Code Generation tools 5.0.0

Chapter 2

TI FEE Driver Design Overview

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_Init) 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. 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 marked as ready for erase and the new one is marked as Active Virtual Sector. Any new data is now written into the new Active Virtual Sector and the Virtual Sector which is marked as ready for erase will be erased in background.

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.

Virtual Sector Organization

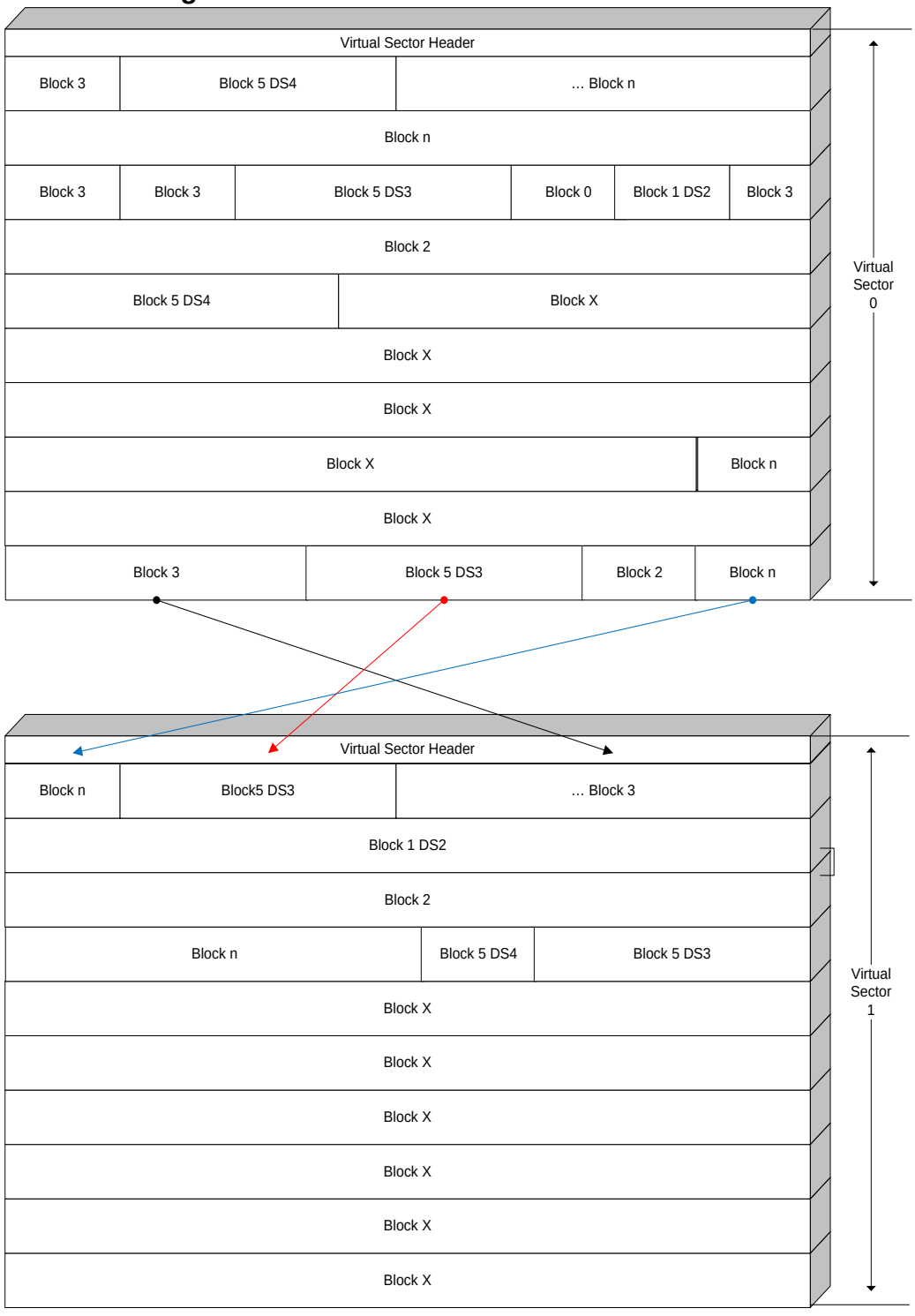


Figure 1 Virtual Sector Organization

total 8 32-bit words

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 Virtual Sector Status Word			
32 bit backup Status	**8 bits Reserved**	**Erase Count (20 bits)**	**Version Number (4 Bits)**

Figure 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 a Virtual Sector can be in.

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

2.1 Virtual Sector Header States

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.

State	Value
Copy Virtual Sector	0xFFFFFFFF
Active Virtual Sector	0x00000000

2.2 Virtual Sector Header backup States

If the normal Virtual sector header is corrupted, then the backup status will be used to know the Virtual Sector state.

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.

After Virtual Sector header, the next 8 bytes are used to know erase status of the Virtual Sector. It says, if the erase was started/completed/ready for erase. Next 8 bytes are reserved.

0x 0000FFFFFFFFFFFF – Erase of other Virtual Sector started

0x 00000000FFFFFFFF – Erase of other Virtual Sector completed

0x0000000000000FFFF – Current Virtual Sector is ready for Erase.

2.1.2 *Data Block Organization*

block header is 6 32-bit words

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.

Data Block Structure

Block5 Header	Dataset2	Block5 Header	Dataset6	Block1 Header	Dataset2	Block3 Header	Dataset1
Block4 Header	Dataset4	Block2 Header	Dataset2	Block1 Header	Dataset8	Block2 Header	Dataset3

Figure 3 Data Block Structure

2.1.2.1 *Data Block Header*

The Data Block Header is 24 bytes in length and is used to indicate the status information of valid data within a Virtual Sector.

Block Number (16 bits)	Block size(16 bits)
Block W/E Cycle count - optional (32 bits) / reserved if saving not enabled	
Checksum - optional (32 bits) / reserved if saving not enabled	
Address of previous Valid Block(32 bits)	
Block Status (64 bits)	

Figure 4 Data Block Header -> Logical Structure

A Standard Data Block Header has the following fields

Bit(s)	Field	Description
191-176	Block Number	This is used to indicate the block number.
175-160	Block size	Indicates size of block
159-128	W/E counter	Indicates write/erase counter for a block
127-96	Checksum	Indicates Checksum of block
95-64	Address	Address of the previous valid block
63-0	Status of the Block	These 64 bits indicate the Status of the Block. The following Table lists all the possible combinations for the Block Status.

2-3. Data Block Header Field Definitions

State	Value
Start Program Block	0xFFFFFFFFFFFF0000
Valid Block	0xFFFFFFFF00000000
Invalid Block	0xFFFF000000000000

2-4. Data Block States

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.

2.1.2.2 Data Set Concept

Each block can have different data sets. Data which are closely related can be clubbed as data sets of a block. Data sets of the block cannot exceed **2 power data select bits**. Use case of a dataset is : consider a car stereo which needs to display different languages based on customer input. Projects will configure Block number = 1 for language selection and use data sets for selecting different options for language Ex: 4-English, 5-German, 6-Russian,7-French etc.(Here the block has 4 data sets. Data select bits should be > = 2 =>2 power 2=4.).

Data Set concept comes from Autosar. In Autosar, the layer above FEE is NVRAM Manager. NVRAM manager defines three types of blocks, Native, Redundant and DataSet blocks.

A Native block is one which has single Non Volatile Block.

A Redundant block is one which has two Non Volatile Blocks.

A Data Set Block is the one in which the NVRAM Manager will decide the number of copies of the same block to be present in Non Volatile Block.

If projects are not using FEE in Autosar context, data sets can be ignored(Always configure TI_FEE_DATASELECT_BITS = 0).

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 asynchronously.
4. **ReadSync:** This command shall copy a continuous Flash Data block synchronously.
5. **EraseImmediate:** This command shall mark the block as invalid in Data Block header.
6. **InvalidateBlock:** 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. **UNINIT:** 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. **JOB_OK:** The last job has finished successfully
2. **JOB_PENDING:** The last job is waiting for execution or is currently being executed.
3. **JOB_CANCELLED:** The last job has been cancelled.
4. **JOB_FAILED:** The last read/erase/write job failed.
5. **JOB_INCONSISTENT:** The requested block is inconsistent, it may contain corrupted data.
6. **JOB_INVALID:** The requested block has been invalidated. The requested read operation cannot be performed.

Chapter 3

File List

This chapter provides the list of files generated from HALCoGen for TI FEE Driver.

File Name	Destination directory
ti_fee.h	Include
til_fee_types.h	Include
ti_fee_utils.c	Source
ti_fee_eraseImmediateblock.c	Source
ti_fee_format.c	Source
ti_fee_Info.c	Source
ti_fee_invalidateblock.c	Source
ti_fee_cancel.c	Source
ti_fee_read.c	Source
ti_fee_readsync.c	Source
ti_fee_shutdown.c	Source
ti_fee_ini.c	Source
ti_fee_main.c	Source
ti_fee_writeasync.c	Source
ti_fee_writesync.c	Source
fee_interface.h	Include

3-1. TI FEE Driver File List

Files generated using HALCoGen™ are listed below

File Name	Destination directory
Device_types.h	Include
Device_header.h	Include
ti_fee_cfg.h	Include
ti_fee_cfg.c	Source
Device_TMS570LSxx.h/ Device_RMxx.h	Include
Device_TMS570LSxx.c/ Device_RMxx.c	Source

3-2. TI FEE HALCoGen™ File List

Note: xx indicates device part number

E.g.: If the target device chosen is TMS570LS31, then the device specific files generated are Device_TMS570LS31.h and Device_TMS570LS31.c

Chapter 4

Integration 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 Error Recovery Implementation

Projects should implement error recovery mechanism to recover from serious errors. They should call the API **TI_FeeErrorCode( )** periodically to check if there are any severe errors(*Error_SetupStateMachine, Error_NoActiveVS, Error_CopyButNoActiveVS, Error_EraseVS*). If error is any of the above type, then API TI_Fee_ErrorRecovery( ) should be called with proper parameters.

If the error is of type *Error_CopyButNoActiveVS,* then the application has to provide info on which of the VS needs to be corrected in u8VirtualSector.For error of type *Error_CopyButNoActiveVS,* TI_Fee_u32ActCpyVS will provide info on which VS is Copy. In this case, the second argument for the TI_Fee_ErrorRecovery should be the copy Virtual Sector number. Error recovery API will mark the Virtual Sector as Active.

If the error is of type *Error_NoFreeVS,* then the application has to provide info on which of the Virtual Sector needs to be erased in u8VirtualSector. TI_Fee_u32ActCpyVS will provide info on which Virtual Sector is active.

If the error is of type *Error_SetupStateMachine,* recheck configuration. Configure RWAIT, EWAIT and operating frequency correctly.

If the error is of type *Error_EraseVS*, this means either erasing or a blank check of Virtual Sector failed. Call error recovery function to perform erase again. Check the variables TI_Fee_u8ErrEraseVS will indicate which virtual sector failed the erase

Application can access the variable “TI_Fee_u32ActCpyVS” to know details about the Virtual Sector’s.

Prototype for the API’s are:

TI_Fee_ErrorCodeType **TI_FeeErrorCode**(uint8 u8EEPIndex);

void **TI_Fee_ErrorRecovery**(TI_Fee_ErrorCodeType ErrorCode, uint8 u8VirtualSector);

If two EEPROM’s are configured, then TI_FeeErrorCode has to be called cyclically with different index.

Ex: TI_FeeErrorCode(0) and TI_FeeErrorCode(1)

If Error is of type *Error_CopyButNoActiveVS* and ***TI_Fee_u32ActCpyVS*** = 0x0001, this means VS 1 is COPY sector.

Projects has to mark the sector 1 as ACTIVE, so

Call TI_Fee_ErrorRecovery(*Error_TwoActiveVS*, 1);

Virtual sector 1 will be marked as ACTIVE.

Virtual sector numbers start from 1.

4.2 Single and Double bit Error Corrections

Hercules devices provide a mechanism to detect single and double bit errors. FEE enables the SECDED. If there are any double bit error's during read, they will be flagged as BLOCK_INCONSISTENT after read operation is completed, provided TI_FEE_FLASH_ERROR_CORRECTION_ENABLE is enabled.

4.3 Memory Mapping

Following macros can be used for reallocating code, constants and variables.

- FEE_START_SEC_CONST_UNSPECIFIED
- FEE_STOP_SEC_CONST_UNSPECIFIED
- FEE_START_SEC_CODE
- FEE_STOP_SEC_CODE
- FEE_START_SEC_VAR_INIT_UNSPECIFIED
- FEE_STOP_SEC_VAR_INIT_UNSPECIFIED

.

4.4 Build Procedure

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

1. The driver files generated from HALCoGen 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 (ti_fee_cfg.h & ti_fee_cfg.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. Flash API library : The TI FEE Driver uses the Flash API library for performing program/erase operations. Include appropriate F021 library and include files of F021. F021 version should be 02.00.00 or greater. For TMS570LCxx devices, use F021 library v02.01.01 or greater.

4.5 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.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
TI_Fee_StatusType	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_StartProg=1	Write/Erase/Invalid operation is in progress on this Block
	Block_Valid=2	Block is Valid
	Block_Invalid=3	Block is Invalid
TI_Fee_ErrorCodeType	Error_Nil=0	No Error.
	Error_TwoActiveVS=1	There are two active Virtual sectors. This error will not happen with modified design.
	Error_TwoCopyVS=2	There are two copy Virtual sectors. This error will not happen with modified design.
	Error_SetupStateMachine=3	Either HCLK or EWAIT are not configured correctly or there is OTP error.
	Error_CopyButNoActiveVS=4	There is a copy Virtual sector but no Active sector or ready for erase sector.
	Error_NoActiveVS=5	FEE was not able to find/create an active Virtual Sector.
	Error_BlockInvalid=6	Invalid Block passed as input.
	Error_NullDataPtr=7	Null Data ptr passed as input.
	Error_NoFreeVS=8	No more Free Virtual Sector present to write data. This

		error will not happen with modified design.
	Error_InvalidVirtualSectorParameter=9	This is deprecated.
	Error_ExceedSectorOnBank=10	
	Error_EraseVS=11	Blank check failed after erase.
	Error_BlockOffsetGtBlockSize=12	Block Offset is not valid.
	Error_LengthParam=13	Length Parameter is not valid.
	Error_FeeUninit=14	FEE if not initialized.
	Error_Suspend=15	This is deprecated.
	Error_InvalidBlockIndex=16	Block index is invalid.
	Error_NoErase=17	This is deprecated.
	Error_CurrentAddress=18	Address of block is not valid.
	Error_Exceed_No_Of_DataSets=19	Data sets not configured correctly.
TI_FeeModuleStatusType	UNINIT	FEE Module is Uninitialized
	IDLE	FEE Module is Idle
	BUSY	FEE Module is Busy
	BUSY_INTERNAL	FEE Module is performing internal operations
TI_Fee_StatusWordType_UN	Erase	If set to ‘1’ indicates Erase operation is in progress
	ReadSync	If set to ‘1’ indicates Synchronous Read operation is in progress
	ProgramFailed	If set to ‘1’ indicates there was an error during write operation. This is now deprecated.
	Read	If set to ‘1’ indicates Read operation is in progress
	Writesync	If set to ‘1’ indicates Sync Write operation is in progress
	WriteAsync	If set to ‘1’ indicates Async Write operation is in progress
	EraseImmediate	If set to ‘1’ indicates Erase

		immediate 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
	Initialized	If set to ‘1’ indicates FEE is initialized. This is now deprecated.
	SingleBitError	If set to ‘1’ indicates there was a single bit error during read operation. This is now deprecated.
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	

4-1. TI FEE Driver Symbolic Constants

4.6 Data Structures

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

4.7 TI FEE Parameter Configuration

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

4.7.1 Operating Frequency

Description	Device operating frequency in MHz.
Generated configuration	TI_FEE_OPERATING_FREQUENCY is set to the value assigned to FeeFrequency. FeeFrequency is equivalent to the HCLK frequency in the TMS570/RMxx clock tree. It is recommended to copy the value of HCLK obtained by configuring the TMS570/RMxx clock tree during MCU configuration to this parameter.
Default Value	160.0
Parameter Range	Device dependent parameter. Refer to the device datasheet to know the range.
Parameter Type	float
Target File	ti_fee_cfg.h

4.7.2 Number of Blocks

Description	Defines the number of Data Blocks used for EEPROM emulation. This is sum of all the blocks configured on EEP1 and EEP2.
Generated configuration	TI_FEE_NUMBER_OF_BLOCKS is set to the defined value.
Default Value	0x1
Parameter Range	0x1 to 0xFFFE
Parameter Type	uint16
Target File	ti_fee_cfg.h

Note: In HALCoGen GUI, only 16 blocks can be configured. If projects want to have more blocks, manually edit ti_fee_cfg.h and ti_fee_cfg.c files. In ti_fee_cfg.c file, add blocks under `/* USER CODE BEGIN */` and `/* USER CODE END */` inside Fee_BlockConfiguration[ ] so that they are not overwritten by HALCoGen.

4.7.3 Number of Virtual Sectors

Description	Defines the number of Virtual Sectors used for FEE.
Generated configuration	TI_FEE_NUMBER_OF_VIRTUAL_SECTORS is set to the defined value.
Default Value	0x2
Parameter Range	Min : 0x2 Max : 0x4(Depending on the device, max value can change. For TMS570LC4357, max value can be 32)
Parameter Type	uint16
Target File	ti_fee_cfg.h

4.7.4 Number of Virtual Sectors for EEP1

Description	Defines the number of Virtual Sectors used for EEP1.
Generated configuration	TI_FEE_NUMBER_OF_VIRTUAL_SECTORS_EEP1 is set to the defined value.
Default Value	0x0
Parameter Range	Min : 0x0 Max : (TI_FEE_NUMBER_OF_VIRTUAL_SECTORS-0x02)
Parameter Type	uint16
Target File	ti_fee_cfg.h

Note: TI_FEE_NUMBER_OF_VIRTUAL_SECTORS_EEP1 should be configured as zero if TI_FEE_NUMBER_OF_EEPS = 1.

4.7.5 Number of Non Configured blocks to copy

Description	Maximum number of non configured blocks to copy. If set to a value other than zero, then the non configured valid blocks in Flash will be copied to new virtual sector during virtual sector swap.
Generated configuration	TI_FEE_NUMBER_OF_UNCONFIGUREDBLOCKSTOCOPY is set to 0 if no non configured valid blocks are to be copied during Virtual Sector swap.
Default Value	0
Parameter Range	0-0xFFFE
Parameter Type	uint16
Target File	ti_fee_cfg.h

Note: This parameter is used when a project starts with, let's say 10 blocks and during development they reduce the blocks to 8. However, if they still want the remaining blocks to be present inside Flash(assume project has already written 10 blocks into Flash), they will have to configure this parameter to more than 2.

This parameter can also be used to run two different instances of FEE driver, one owned by boot loader and the other owned by application.

4.7.6 Number of Eight byte writes

Description	Defines the number of 8 byte writes to be done in main function call. If configured to 2, main function writes 16 bytes per call.
Generated configuration	TI_ FEE_NUMBER_OF_EIGHTBYTEWRITES is set required value.
Default Value	1
Parameter Range	1-255
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.7 Block OverHead

Description	Indicates the number of bytes used for Block Header.
Generated configuration	TI_FEE_BLOCK_OVERHEAD is set to the value assigned to FeeBlockOverhead.
Default Value	0x18
Parameter Range	Fixed to 0x18.
Parameter Type	uint8
Target file	ti_fee_cfg.h

4.7.8 Page OverHead

Description	Indicates the Page Overhead in bytes.
Generated configuration	TI_FEE_PAGE_OVERHEAD is set to the value assigned to FeePageOverhead. (0x0)
Default Value	0x0
Parameter Range	Fixed to 0x0.
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.9 Virtual Sector OverHead

Description	Indicates the number of bytes used for Virtual Sector Header.
Generated configuration	TI_FEE_VIRTUAL_SECTOR_OVERHEAD is set to the value assigned to FeeVirtualSectorOverhead (0x10).
Default Value	0x10
Parameter Range	Fixed to 0x10.
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.10 Virtual Sector Page Size

Description	Indicates the virtual page size in bytes.
Generated configuration	TI_FEE_VIRTUAL_PAGE_SIZE is set to the value assigned to FeeVirtualPageSize. (0x8)
Default Value	0x8
Parameter Range	Fixed to 0x8.
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.11 Driver Index

Description	Instance ID of FEE module. Should always be 0x0.
Generated configuration	TI_FEE_INDEX is set to the value assigned to FeeIndex. (0x0)
Default Value	0x0
Parameter Range	Fixed to 0x0.
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.12 Enable ECC Correction

Description	Indicates if error correction is enabled.
Generated configuration	TI_FEE_FLASH_ERROR_CORRECTION_ENABLE Is set to STD_ON if Error Correction is enabled else it is set to STD_OFF.
Default Value	STD_OFF
Parameter Range	STD_ON/STD_OFF
Parameter Type	Boolean
Target File	ti_fee_cfg.h

4.7.13 Error Correction Handling*(Not available for Configuration)*

Description	Indicates desired action to be taken on detection of bit errors.
Generated configuration	TI_FEE_FLASH_ERROR_CORRECTION_HANDLING is set to the value assigned to FeeFlashErrCorrHandlingType. Only Ti_Fee_None is supported.
Default Value	TI_Fee_None
Parameter Range	TI_Fee_None or TI_Fee_Fix
Parameter Type	# define TI_Fee_None 0U # define TI_Fee_Fix 1U
Target File	ti_fee_cfg.h

4.7.14 Block Write Counter Save

Description	Pre-processor switch to enable the block write counter. STD_ON: Block Write counter is enabled. STD_OFF:Block Write counter is disabled
Generated configuration	TI_FEE_FLASH_WRITECOUNTER_SAVE is set to STD_ON if block write counter save is enabled else it is set to STD_OFF. If enabled, the block write counter is updated for every write. With this counter, projects can know how many times a block has been written into Flash.
Default Value	STD_OFF
Parameter Range	STD_ON / STD_OFF
Parameter Type	Boolean
Target File	ti_fee_cfg.h

4.7.15 Enable Checksum

Description	Pre-processor switch to enable the Checksum for blocks. STD_ON: Checksum for blocks is enabled. STD_OFF: Checksum disabled
Generated configuration	TI_FEE_FLASH_CHECKSUM_ENABLE Is set to STD_ON if check is enabled else it is set to STD_OFF. If enabled, 16bit Checksum of the block is generated.
Default Value	STD_OFF
Parameter Range	STD_ON / STD_OFF
Parameter Type	Boolean
Target File	ti_fee_cfg.h

Note: If Checksum is enabled, during writing of a block, Checksum of the data to be written is calculated and checked against the Checksum of the same block which is already existing in Flash. If Checksum matches, data is not written. If Checksum is not enabled, then data is compared byte by byte. If data does not match, write will be initiated.

4.7.16 Number Of EEPs

Description	Number of EEP's configured. 1 - Only one EEP configured. All Virtual Sectors can be used by this EEP. 2 - Two EEP's configured. Each EEP can use two Virtual Sectors.
Generated configuration	TI_FEE_NUMBER_OF_EEPS is set to 1 if all virtual sectors are used by one EEP. If virtual sectors are shared between two EEPs, it is set to 2. If projects have data blocks which are frequently updated and also have blocks which are not frequently updated, then projects can configure 2 EPPROM’s and use each EEPROM for different set of blocks. One EEPROM can have data blocks which are frequently updated and the other can have data blocks which are not frequently updated.
Default Value	1
Parameter Range	1/2
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.17 Data Select bits

Description	Number of data sets configured for a block.
Generated configuration	TI_FEE_DATASELECT_BITS is set configured value.
Default Value	0
Parameter Range	0-8
Parameter Type	uint8
Target File	ti_fee_cfg.h

4.7.18 Check BANK7 address Range

Description	Pre processor switch to enable EEPROM address range check during read/write.
Generated configuration	TI_FEE_CHECK_BANK7_ACCESS is set configured value.
Default Value	STD_OFF
Parameter Range	STD_ON/STD_OFF
Parameter Type	Boolean
Target File	ti_fee_cfg.h

4.7.19 TI FEE Virtual Sector Configuration

Array Name	TI_FEE_VirtualSectorConfiguration	
Description	Used to define a Virtual Sector	
Array Type	TI_Fee_VirtualSectorConfigType. This is a structure having the following members.	
Members	FeeVirtualSectorNumber	Virtual Sector's Number.
	FeeFlashBank	EEPROM emulation is supported only on Bank 7 for F021 devices..
	FeeStartSector	Starting Sector in the Bank for this Virtual Sector.
	FeeEndSector	Ending Sector in the Bank for this Virtual Sector.

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

4.7.19.1 Virtual Sector Number

Description	Used to assign a number to the Virtual Sector.
Generated configuration	FeeVirtualSectorNumber is set to the value assigned to the symbolic name for the Virtual Sector.
Default Value	1
Parameter Range	Min : 0x1, Max : 0x4
Parameter Type	uint16
Target File	ti_fee_cfg.c

4.7.19.2 Flash Bank

Description	Indicates the Flash Bank used by the Virtual Sector. All the Virtual Sectors should use the same Flash Bank. EEPROM emulation is supported only on Bank 7 for F021 devices.
Generated configuration	FeeFlashBank is set to the value assigned to FeeSectorBank.
Default Value	0x7 for F021 devices.
Parameter Range	Fixed to 0x7 for F021 devices.
Parameter Type	uint16
Target File	ti_fee_cfg.c

4.7.19.3 Start Sector

Description	Indicates the Flash Sector in the Bank used by the Virtual Sector as the Start sector.
Generated configuration	FeeStartSector is set to the value assigned to FeeSectorStart.
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.
Parameter Type	uint8
Target File	ti_fee_cfg.c

4.7.19.4 End Sector

Description	Indicates the Flash Sector in the Bank used by the Virtual Sector as the End sector.
Generated configuration	FeeEndSector is set to the value assigned to FeeSectorEnd.
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.
Parameter Type	uint8
Target File	ti_fee_cfg.c

4.7.19.5 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 */
      7,    /* Bank                  */
      0,    /* Start Sector          */
      0    /* End Sector            */
   },
   /* Virtual Sector 2 */
   {
      2,    /* Virtual sector number */
      7,    /* Bank                  */
      1,    /* Start Sector          */
      1    /* End Sector            */
   },
};
```

4.7.20 TI FEE Block Configuration

Array Name	Fee_BlockConfiguration	
Description	Used to define a block	
Array Type	Fee_BlockConfigType. This is a structure with the following members.	
Members	FeeBlockNumber	Indicates Block's Number.
	FeeBlockSize	Defines Block's Size in bytes.
	FeeImmediateData	Indicates if the block is used for immediate data.
	FeeNumberOfWriteCycles	Number of write cycles required for this block .
	FeeDeviceIndex	Indicates the device index.
	FeeNumberofDatasets	Indicates the number of Datasets for this Block.
	FeeEEPNumber	Indicates the number of EEP.

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

4.7.20.1 BlockNumber

Description	Assigns a number for the Block.
Generated configuration	FeeBlockNumber is set to a numeric value. It is equal to the BlockNumber.
Default Value	1
Parameter Range	Min : 0x1 Max : 0xFFFE
Parameter Type	uint16
Target File	ti_fee_cfg.c

4.7.20.2 Block Size

Description	Indicates the size of the Block in bytes.
Generated configuration	FeeBlockSize is set to the value assigned to FeeBlockSize.
Default Value	0x008
Parameter Range	0x1 to 0xFFF
Parameter Type	uint16
Target File	ti_fee_cfg.c

4.7.20.3 Immediate Data

Description	Indicates the number of clock cycles required to write to a flash address location.
Generated configuration	FeeNumberOfWriteCycles is set to the value assigned to FeeNumberOfWriteCycles.
Default Value	0x1
Parameter Range	Device or core/flash tech dependent parameter.
Parameter Type	uint32
Target File	ti_fee_cfg.c

4.7.20.4 Number of Write Cycles

Description	Indicates if the block is used for immediate data.
Generated configuration	FeeImmediateData is set to the value assigned to FeeImmediateData.
Default Value	FALSE
Parameter Range	TRUE / FALSE
Parameter Type	Boolean
Target File	ti_fee_cfg.c

4.7.20.5 Device Index

Description	Indicates the device index. This will always be 0.
Generated configuration	FeeDeviceIndex is set to the value 0x0.
Default Value	0x0
Parameter Range	Fixed to 0x0.
Parameter Type	uint8
Target File	ti_fee_cfg.c

4.7.20.6 Data sets

Description	Indicates the number of Datasets for this particular Block .
Generated configuration	FeeNumberOfDataSets is set to the value assigned to FeeDataset. It should not be greater than 2 power TI_FEE_DATASELECT_BITS.
Default Value	0x01
Parameter Range	0x1 to 0xFF
Parameter Type	uint8
Target File	ti_fee_cfg.c

4.7.20.7 EEPNumber

Description	Number indicating into which EEP does the block go. 0 -- Block will be configured on EEP1. 1 -- Block will be configured on EEP2.
Generated configuration	FeeEEPNumber is set to the value assigned.
Default Value	0x0
Parameter Range	0x00/0x01
Parameter Type	uint8
Target File	ti_fee_cfg.c

4.7.20.8 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 */
     TRUE,       /* Block immediate data used    */
      0,             /* Device Index                    */
      1,             /* Number of DataSets              */
      0              /* EEP Number                      */
   },
   /* Block 2 */
   {
      0x02,         /* Block number                    */
      0x008,        /* Block size                       */
      0x10,          /* Block number of write cycles */
      TRUE,       /* Block immediate data used    */
      0,             /* Device Index                    */
      2,             /* Number of DataSets              */
      0              /* EEP Number                      */
   },
   /* Block 3 */
   {
      0x03,          /* Block number                    */
      0x0004,       /* Block size                      */
      0x10,          /* Block number of write cycles */
     TRUE,       /* Block immediate data used    */
      0,             /* Device Index                    */
      3,             /* Number of DataSets              */
      1              /* EEP Number                      */
   },

/* Block 4 */
   {
```

```
        0x04,       /* Block number                      */
        0x001A,     /* Block size                        */
        0x10,       /* Block number of write cycles      */
        TRUE,       /* Block immediate data used         */
        0,          /* Device Index                      */
        4,          /* Number of DataSets                */
        1           /* EEP Number                        */
    },

};
```

4.8 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 section 4.6.

4.8.1 Initialization

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

Name	Description
TI_Fee_Init	Used to initialize the FEE module

4-2. TI FEE Driver Initialization APIs

4.8.2 Data Operations

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

Name	Description
TI_Fee_WriteAsync	Used to initiate an Asynchronous Write Operation to a Data Block. TI_Fee_MainFunction function should be called at regular intervals to finish the 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_MainFunction function should be called at regular intervals to finish the operation
TI_Fee_ReadSync	Used to read Data from a Data Block Synchronously.
TI_Fee_EraseImmediateBlock	Used to initiate an Erase Operation of a Data Block. TI_Fee_MainFunction function should be called at regular intervals to finish the operation
TI_Fee_InvalidateBlock	Used to initiate an Invalidate Operation on a Data Block. TI_Fee_MainFunction function should be called at regular intervals to finish the 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.
TI_Fee_Format	Used to erase all the configured Virtual Sectors.

4-3. TI FEE Driver Data Operation APIs

4.8.3 Information

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

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.

4-4. TI FEE Driver Information APIs

4.8.4 Internal Operations

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

Name	Description
TI_Fee_MainFunction	Used to complete the Data Operations initiated by any of the Data Operation functions.
TI_FeeInternal_FeeManager	Used to perform internal operations (Copy, Erase Virtual Sector).

4-5. TI FEE Driver Internal Operation APIs

4.8.5 Error Information and Recovery Operations

The TI FEE Driver APIs that are used to provide error information and recover from severe errors.

Name	Description
TI_FeeErrorCode	Function to know the error type.
TI_Fee_ErrorRecovery	Function to recover from severe errors.

4-6. TI FEE Driver Error Info and Recovery APIs

4.8.6 Suspend/Resume Erase Sector

The TI FEE Driver APIs that are used to provide error information and recover from severe errors.

Name	Description
TI_FeeErrorCode	Function to know the error type.
TI_Fee_ErrorRecovery	Function to recover from severe errors.

4-7. TI FEE Driver Suspend/Resume Erase sector API

4.9 Fee Operation Flow

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

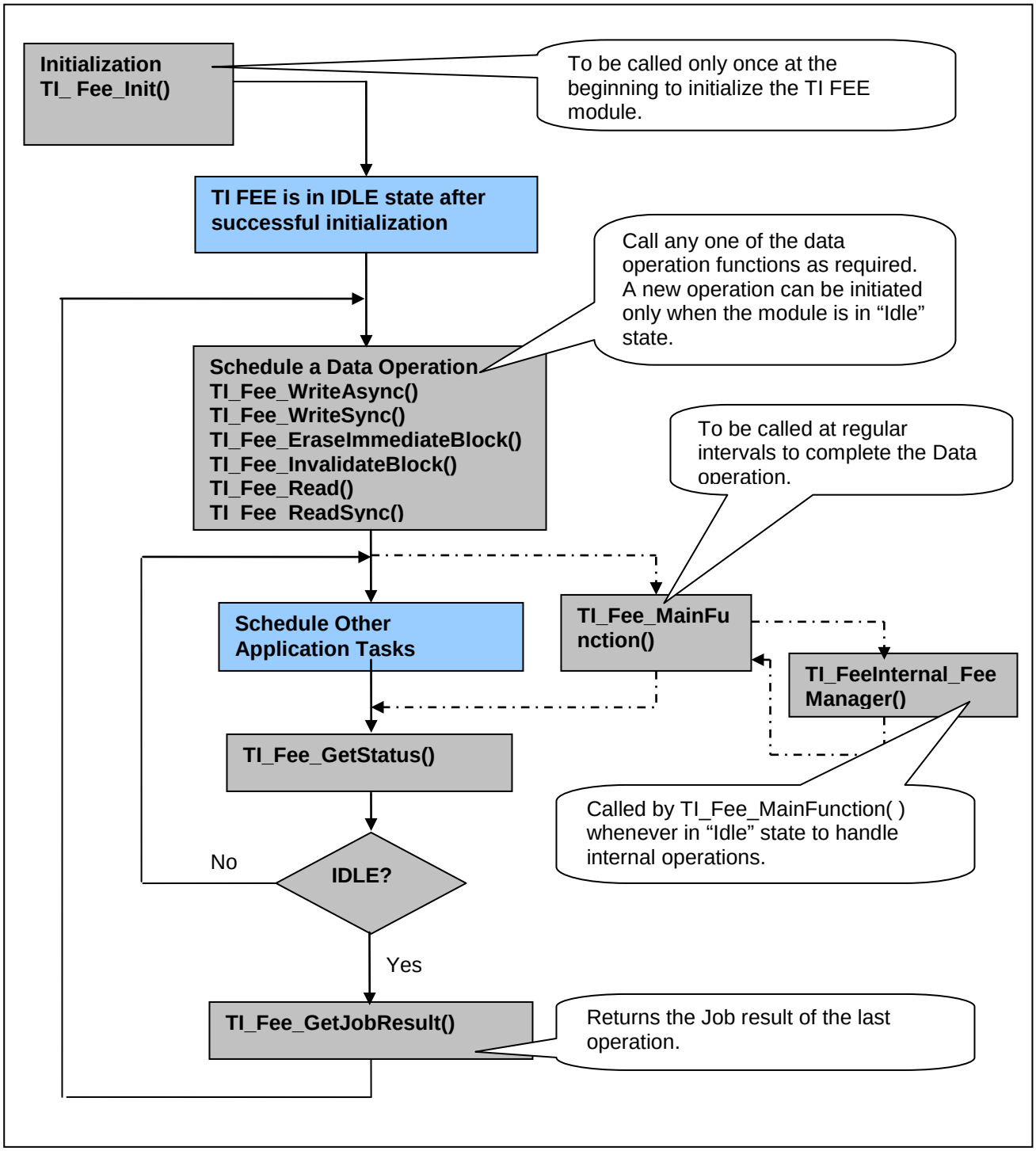


Figure 5 Flow chart of a typical FEE operation

4.10 API Specification

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

4.10.1 TI FEE Driver Functions

4.10.1.1 Initilization Function (TI_Fee_Init)

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.

Function Name:	TI_Fee_Init
Syntax:	void TI_Fee_Init (void)
Sync/Async:	Synchronous
Parameters(in):	None
Return value:	None
Description:	Function to initialize the TI Fee module.

4.10.1.2 Async Write Function (TI_Fee_WriteAsync)

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

Function Name:	TI_Fee_WriteAsync	
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:	Function to initiate an Async Write job.	

4.10.1.3 Sync Write Function (TI_Fee_WriteSync)

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

Function Name:	TI_Fee_WriteSync	
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:	Function to program Data to a Block synchronously.	

4.10.1.4 Read Function (TI_Fee_Read)

This function provides functionality for reading of data from a Block asynchronously. TI_Fee_MainFunction() function should be called at regular intervals to finish the Read operation.

Function Name:	TI_Fee_Read	
Syntax:	Std_ReturnType TI_Fee_Read(uint16 BlockNumber, uint16 BlockOffset, uint8* DataBufferPtr, uint16 Length)	
Sync/Async:	Asynchronous	
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:	Function to read data from a Block.	

4.10.1.5 Erase Function (TI_Fee_EraseImmediateBlock)

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

Function Name:	TI_Fee_EraseImmediateBlock	
Syntax:	Std_ReturnType TI_Fee_EraseImmediateBlock(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:	Function to initiate Erase operation on a Data Block	

4.10.1.6 Invalidate Function (TI_Fee_InvalidateBlock).

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

Function Name:	TI_Fee_InvalidateBlock	
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:	Function to initiate an Invalidate operation on a Data Block	

4.10.1.7 Shutdown Function (TI_Fee_Shutdown)

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.

Function Name:	TI_Fee_Shutdown	
Syntax:	Std_ReturnType TI_Fee_Shutdown()	
Sync/Async:	Synchronous	
Parameters (in):	None	
Return value:	Std_ReturnType	E_OK: The Async job was completed
		E_NOT_OK: The Async job was not completed.
Description:	Function to perform bulk Data write prior to system shutdown.	

4.10.1.8 Get Version Info Function (TI_Fee_GetVersionInfo)

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

Function Name:	TI_Fee_GetVersionInfo	
Syntax:	void TI_Fee_GetVersionInfo(Std_VersionInfoType* VersionInfoPtr)	
Sync/Async:	Synchronous	
Parameters (in):	None	
Return value:	VersionInfoPtr	Pointer to standard version information structure
Description:	Function to return the version information of the TI Fee module.	

4.10.1.9 Get Status Function (TI_Fee_GetStatus)

This function returns the status of the TI FEE module.

Function Name:	TI_Fee_GetStatus	
Syntax:	TI_FeeModuleStatusType TI_Fee_GetStatus(uint8 u8EEPIndex)	
Sync/Async:	Synchronous	
Parameters (in):	u8EEPIndex	Index for EEP. (0/1)
Return value:	TI_FeeModuleStatusType	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:	Function gets the status of the TI Fee module.	

4.10.1.10 Get Job Result Function (TI_Fee_GetJobResult)

This function returns the result of the last job synchronously.

Function Name:	TI_Fee_GetJobResult	
Syntax:	TI_FeeJobResultType TI_Fee_GetJobResult(uint8 u8EEPIndex)	
Sync/Async:	Synchronous	
Parameters in):	u8EEPIndex	Index for EEP.(0/1)
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:	Function gets the job result from the TI Fee module.	

4.10.1.11 Task Function (TI_Fee_MainFunction)

This function handles the Write/Read/Erase/Invalidate asynchronous jobs initiated by TI_Fee_WriteAsync()/TI_Fee_Read()/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_FeeInternal_FeeManager whenever there is no other job pending ("IDLE" State). TI_FeeInternal_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.

Function Name:	TI_Fee_MainFunction
Syntax:	void TI_Fee_MainFunction(void)
Sync/Async:	Asynchronous
Parameters (in):	None
Return value:	None
Description:	Function to handle the requested Async data operations

4.10.1.12 Manager Function (TI_FeeInternal_FeeManager)

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

This routine is responsible to

- Determine 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.
- Determine 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.

Function Name:	TI_FeeInternal_FeeManager	
Syntax:	TI_FeeStatusType TI_FeeInternal_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:	Function to perform background operations.	

4.10.1.13 Format Function (TI_Fee_Format)

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

Function Name:	TI_Fee_Format
Syntax:	boolean TI_Fee_Format(uint32 u32FormatKey)
Sync/Async:	Synchronous
Parameters (in):	u32FormatKey – 0xA5A5A5A5/ 0x5A5A5A5A should be passed as input for formatting the emulated Flash.
Return value:	TRUE/FALSE
Description:	Function formats all the Virtual Sectors.

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.
If u32FormatKey=0xA5A5A5A5, this API will format only configured sectors. However, TI_Fee_Init has to be called before.
If u32FormatKey=0x5A5A5A5A, then TI_Fee_Init API call is not required. However, in this case complete EEPROM bank will be formatted.

4.10.1.14 TI_FeeErrorCode

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

Function Name:	TI_FeeErrorCode( )	
Syntax:	TI_FeeErrorCodeType TI_FeeErrorCode(uint8 u8EEPIndex)	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in)	EEP Index	
Parameters (out):	None	
Return value:	TI_FeeErrorCodeType	Returns an Error Code
Description:	Returns '0' if no error has occurred else it returns an Error code.	

4.10.1.15 TI_Fee_ErrorRecovery

This function provides functionality to recover from any severe errors.

Function Name:	TI_Fee_ErrorRecovery( )	
Syntax:	void TI_Fee_ErrorRecovery(TI_Fee_ErrorCodeType ErrorCode, uint8 u8VirtualSector)	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in)	Error Code	Error_TwoActiveVS(Deprecated)
		Error_TwoCopyVS(Deprecated)
		Error_SetupStateMachine
		Error_NoActiveVS
		Error_CopyButNoActiveVS(Deprecated)
		Error_NoFreeVS
		Error_EraseVS
	Virtual Sector Number	
Parameters (out):	None	
Return value:	None	
Description:	Function recovers from any severe errors.	

4.10.1.16 Synchronous Read Function (TI_Fee_ReadSync)

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

4.10.1.17 S

Function Name:	TI_Fee_ReadSync	
Syntax:	Std_ReturnType TI_Fee_ReadSync(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 and completed by the TI Fee module
		E_NOT_OK: The Read job was not accepted by the TI Fee module.
Description:	Function to read data from a Block.	

of sector Function (TI_Fee_SuspendResumeErase)

This function provides functionality to suspend/Resume of erasing a sector.

Note: This API can be called once after TI_Fee_Init is executed with Suspend_Erase as function argument. It has to be called again after application has completed all the initialization sequence with Resume_Erase as function argument. This can be called if projects do not want TI_Fee_Init API to trigger erasing of virtual sector.

Function Name:	TI_Fee_SuspendResumeErase	
Syntax:	void TI_Fee_SuspendResumeErase(TI_Fee_EraseCommandType Command)	
Sync/Async:	Synchronous	
Parameters (in):	Command	Suspend_Erase/Resume_Erase
Return value:	None	
Description:	Function to suspend/Resume erasing of sector.	

4.11 Privilege Mode access

FEE needs following API's to be executed in Privilege mode:

- TI_Fee_Init

- TI_FeeInternal_WriteDataF021

4.12 Power Fail Behavior

FEE will be able to recover from any kind of resets.

Following are the scenarios where if power fail happens, how FEE behaves:

=>Assume an erase command is issued and F021 starts erasing the sector. Now, before the sector is completely erased, if there was a power fail, in next initialization, this sector is added to the erase queue since sector header will not match with the expected sector header states(Active, Copy, Ready for Erase, Empty).
Erasing will happen in the background.

=>Similarly, if there was a power fail during writing of Sector header, above step is performed.

=>If there was a power fail during writing of a block
Block is written in following way
1. Block status is programmed as start program block.
2. Block number and block size are written.
3. Write data of the block.
4. After completion of writing of data, Checksum and address of previous block are written
5. Block status is marked as Active.

If power fail happens after setp1, then in next initialization, writing of next block is shifted by 24 bytes. If power fail happens after step2, step3,step4 or step5, then the writing of next block will happen after current block size+block header.

=>Similarly if power fail happens during copy operation, during the next initialization, INI API detects that during previous shutdown, copy was started but did not get completed. Copy operation is initiated in the current driving cycle. Only blocks which were not copied during previous driving cycle are copied in current driving cycle.

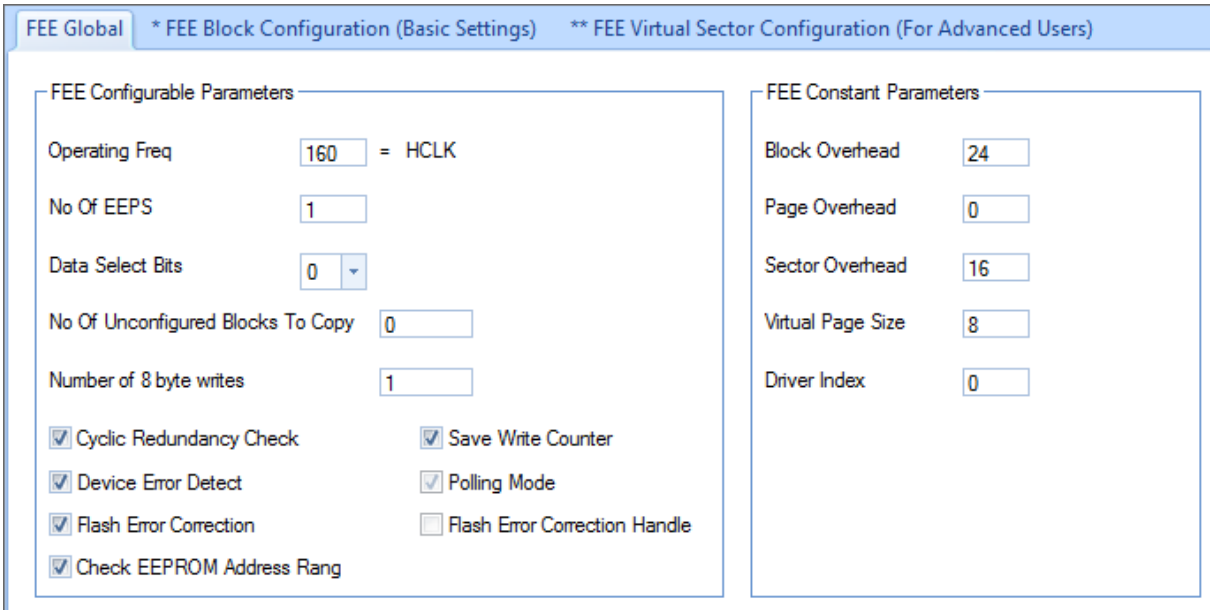
4.13 Known Issues / Not supported features

- Non Polling mode not supported.
- Immediate block writing not accepted when FEE is performing copy of blocks / erase of sectors.
- No Jobs accepted during copy of blocks /erase of sectors ongoing. (The write job which triggered the copy operation will be pending until copy of blocks and erase of sectors is completed.)
- Maximum Blocking time not supported.

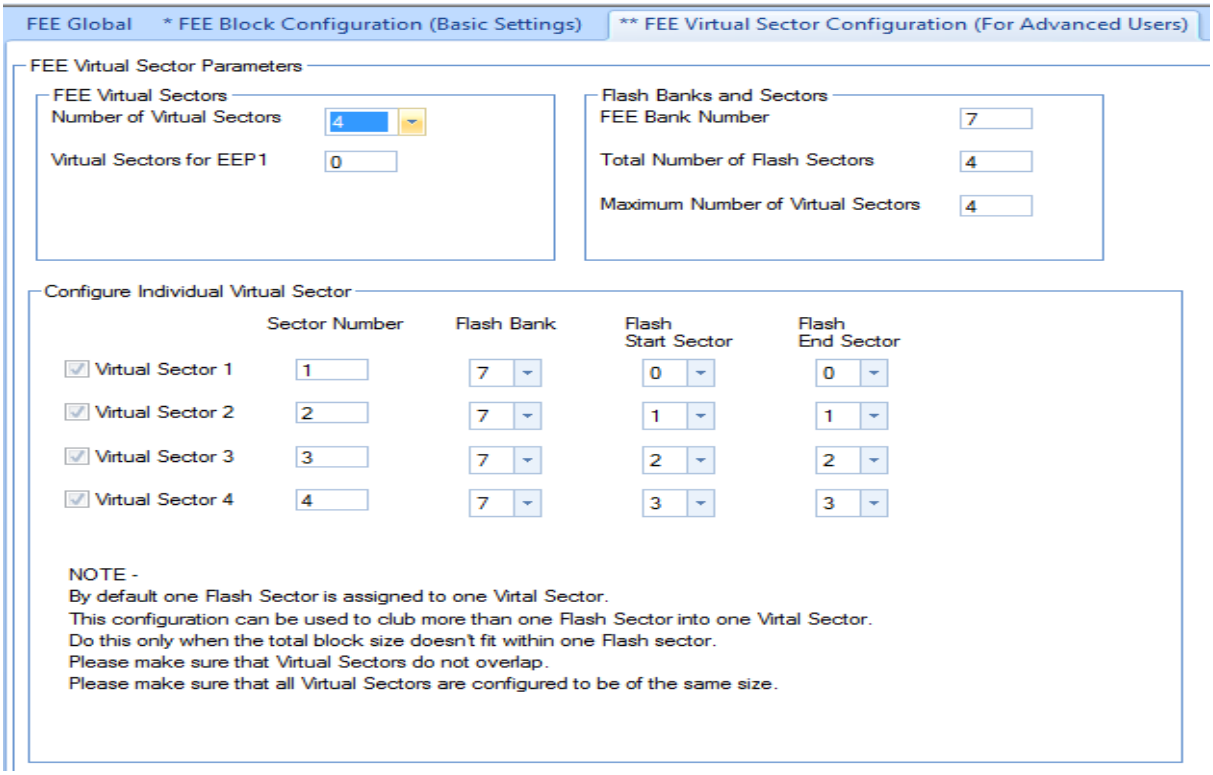
4.14 Example Configurations

4.14.1 Four Virtual Sectors on four physical sectors – Single EEPROM

Step 1: Configure FEE Global



Step 2: Configure FEE Virtual Sector Configuration

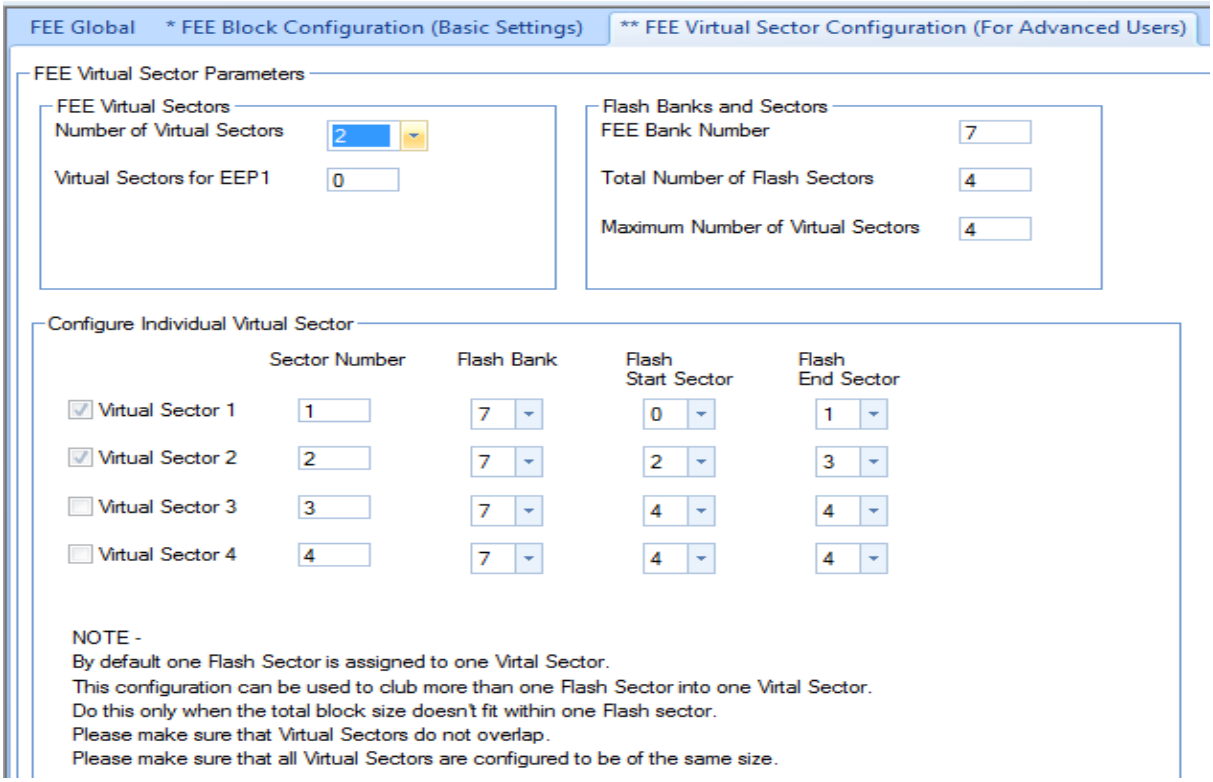


Step 3: Add blocks in FEE Block Configuration

4.14.2 Two Virtual Sectors on four physical sectors– Single EEPROM

Step 1: Same as in 4.14.1

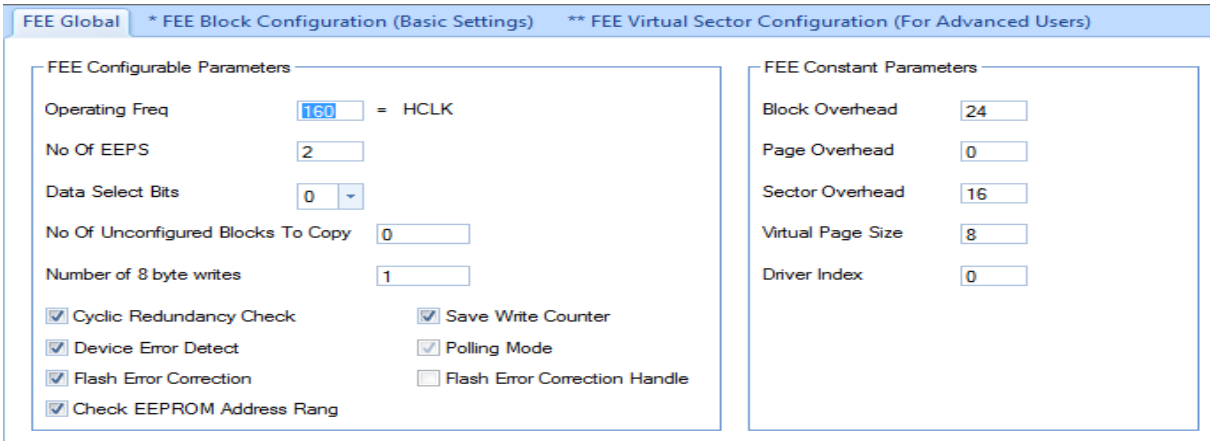
Step 2: Configure FEE Virtual Sector Configuration



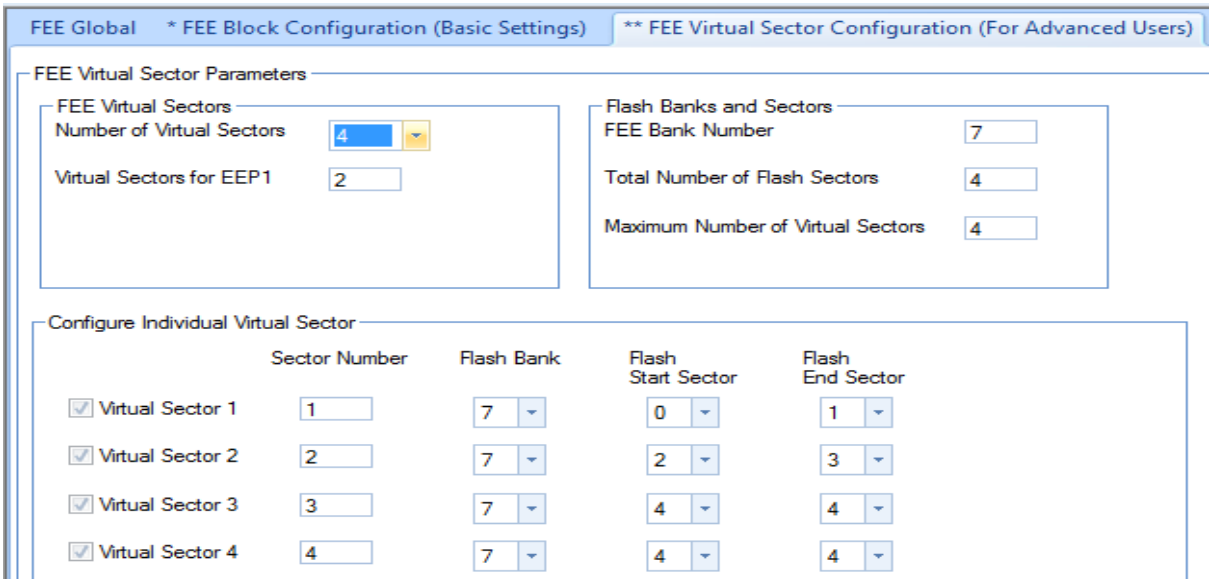
Step 3: Same as in 4.14.1

4.14.3 Two Virtual Sectors for each EEPROM on four physical sectors – Two EEPROM

Step 1: Configure FEE Global as shown below.



Step 2: Configure FEE Virtual Sector Configuration



Step 3: Add blocks in FEE Block Configuration. Configure blocks on to EEP0/EEP1