Texas Instruments

AutoSAR FEE Driver

User Manual

Version 1.14

Aug05, 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 manual serves as a software programmer's handbook for working with the AutoSAR FEE Driver**.** It provides necessary information regarding how to build and use AutoSAR FEE Driver in user systems and applications.

It also provides details regarding the AutoSAR 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 AutoSAR FEE Driver.

Important Notes

Customers have to include F021 API library v2.01.01 or greater.

Abbreviations

1-1. Table of Abbreviations

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

Document Revision History

Version	Date	Revision History
1.0	09/25/2012	Initial version
1.1	11/08/2012	Changes for EA2
1.2	11/21/2012	Additional changes for BETA
1.3	12/12/2012	Add error recovery prototypes
1.4	06/11/2013	Add software revision history. Added new configuration tags information.
1.5	10/23/2013	Updated software revision history.
1.6	01/03/2014	Add new configuration parameter for address range check for Read/Write. Check for Multi bit error during Read. MISRA fixes.
1.7	09/11/2014	Manual Suspend/Resume feature added.
1.8	10/15/2014	RAM Optimization changes.
1.9	10/31/2014	Support TMS570LS05xx, TMS570LS07xx, TMS570LS09xx. Range updated for FEE_VirtualSectorNumber, Virtual Sectors
1.10	01/21/2015	Changes related to unification of Archer and Champion.
1.11	10/14/2015	Bugfix for block lost issue.
1.12	11/20/2015	Enhancement for “Do not change FEE state to IDLE after copying of the blocks is completed”
1.13	03/15/2016	Bugfix for “Block offset address does not get updated correctly, if copy operation was interrupted.” Added section “Important Notes”
1.14	08/05/2016	Updated software revision history.

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.
01.10.00	10/23/2013	Updated software to support more than two VS.
01.20.00	01/03/2014	Add new configuration parameter for address range check for Read/Write. Check for Multi bit error during Read. MISRA fixes.
01.20.01	09/11/2014	Manual Suspend/Resume feature added.
01.21.00	10/15/2014	RAM Optimization changes. New configuration parameter FEE_TOTAL_BLOCKS_DATASETS added.
01.22.00	01/21/2015	Add new Configuration parameters. FEE_VIRTUALSECTOR_SIZE, FEE_PHYSICALSECTOR_SIZE, FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC
01.23.00	10/14/2015	Bugfix for block lost issue.
01.23.01	11/20/2015	Enhancement for "Do not change FEE state to IDLE after copying of the blocks is completed"
01.23.02	03/15/2016	Bugfix for "Block offset address does not get updated correctly, if copy operation was interrupted."

01.23.03	08/05/2016	Fee_Getstatus API should return MEMIF_BUSY, if FEE is doing internal operations.

Table of tables

Table of figures

Chapter 1

AutoSAR FEE Driver Introduction

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

1.1 Overview

This section describes the functional scope of the AutoSAR FEE Driver and its feature set. It introduces the AutoSAR FEE Driver to the user along with the functional decomposition and run-time specifics regarding deployment of AutoSAR 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 AutoSAR 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. The AutoSAR FEE Driver contains AutoSAR interface functions and any additional management functions needed to operate properly.

A list of functions supported by the AutoSAR FEE Driver can be found in Section 1.1.1. The primary function responsible for Fee management is the function Fee_Manager. This function shall operate asynchronously and with little or no user intervention after configuration, maintaining the Fee structures in Flash memory. When using the AutoSAR interface, Fee_MainFunction function will be called on a cyclic basis which in turn calls Fee_Manager when no other pending Fee operations are pending. When not using the AutoSAR interface, the user shall be responsible for calling Fee_MainFunction function on a cyclic basis so that it can perform internal operations.

1.1.1 Functions supported in the AutoSAR FEE Driver

The Autosar FEE Driver provides the following functional services:

Initialization:

- Fee_Init

Operations:

- Fee_Write
- Fee_Read
- Fee_EraseImmediateBlock
- Fee_InvalidateBlock
- Fee_Cancel

Information:

- Fee_GetStatus
- Fee_GetJobResult
- Fee_GetVersionInfo

Internal Operations:

- Fee_MainFunction

Error Information and Recovery:

- TI_FeeErrorCode
- TI_Fee_ErrorRecovery

Suspend/Resume Erase of Sector:

- TI_Fee_SuspendResumeErase

1.1.2 System Requirements

The AutoSAR FEE Driver is supported on platforms characterized by the following Software and Hardware requirements.

1.1.2.1 Software

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

- Operating System : Win7
- Codegeneration tools : TM570 Code Generation tools 4.9.5
- Fee Configuration Files : The user needs to generate two configuration files using a configuration tool to successfully deploy and use AutoSAR FEE Driver. These two files (Fee_Cfg.h & Fee_Cfg.c) define which Flash sectors to be used for EEPROM emulation, define Data Blocks ,Block Size and other configuration parameters.
- Flash API library : The AutoSAR FEE Driver uses the Flash API library for performing program/erase operations. Customers have to use F021 v2.01.01 or greater.

Chapter 2

AutoSAR FEE Driver Design Overview

Overview

The Flash EEPROM Emulation (Fee) module contains AutoSAR interface functions and additional management functions needed to operate it properly. A list of functions supported by the AutoSAR FEE Driver can be found in Section 1.1.1.

This chapter describes the implementation method followed for Flash EEPROM emulation in the AutoSAR 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. Data Blocks can be further partitioned into Datasets. A minimum of two Virtual Sectors are required for Flash EEPROM emulation.

The initialization routine (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 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 Structure 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 Flash EEPROM Emulation (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

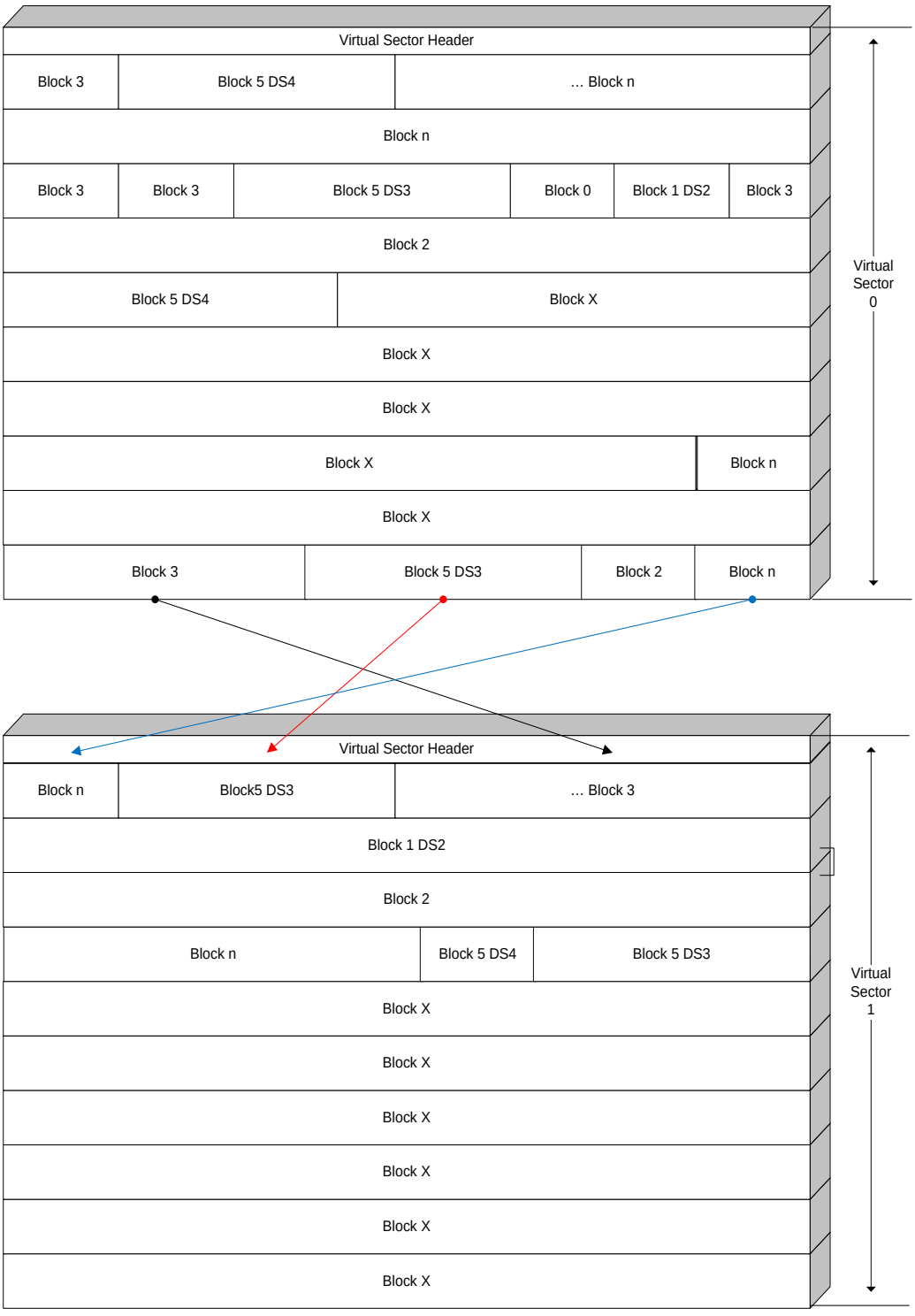


Figure 1 Virtual Sector Organization

2.1.1.1 Virtual Sector Header

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

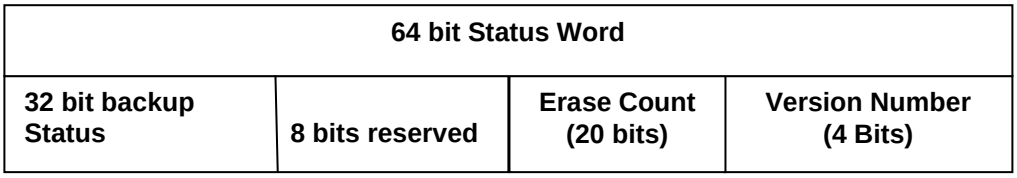


Figure 2 Virtual Sector Header

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 initialized 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.

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 VS state.

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

0x 0000FFFFFFFFFFFF – Erase of other VS started

0x 00000000FFFFFFFF – Erase of other VS completed

0x000000000000FFFF – This VS is ready for Erase.

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. 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 location information (address) of valid data within a Virtual Sector.

A Standard Data Block Header has the following fields

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

4. Data Block Header Field Definitions

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	CRC	Indicates CRC 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.

2-2. Data Block Header Field Definitions

State	Value
Empty Block	0xFFFFFFFFFFFFFFFF
Start Program Block	0xFFFFFFFFFFFF0000
Valid Block	0xFFFFFFFF00000000
Invalid Block	0xFFFF000000000000
Corrupt Block	0x0000000000000000

5. 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.

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 Available Commands

The following list describes the available commands.

1. Write: This command shall program a Flash memory block.
2. Read: This command shall copy a continuous Flash memory block.
3. Erase Immediate: This command shall change the status of the block to Invalid in the Data Block header to Erase it.
4. Invalidate Block: This command shall change the status of the block to Invalid in the Data Block header to invalidate it.

2.1.4 Status Codes

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

1. MEMIF_UNINIT: The Fee Module has not been initialized.
2. MEMIF_IDLE: The Fee Module is currently idle.
3. MEMIF_BUSY: The Fee Module is currently busy.
4. MEMIF_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. MEMIF_JOB_OK: The last job has finished successfully
2. MEMIF_JOB_PENDING: The last job is waiting for execution or is currently being executed.
3. MEMIF_JOB_CANCELLED: The last job has been cancelled.
4. MEMIF_JOB_FAILED: The last read/erase/write job failed.
5. MEMIF_JOB_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data.
6. MEMIF_JOB_INVALID: The requested block has been invalidated. The requested read operation cannot be performed

Chapter 3

Integration Guide

This chapter discusses the AutoSAR FEE Driver run-time interfaces that comprise the API classification & usage scenarios and the API specification itself in association with its data types and structure definitions. Users will have to integrate TI FEE along with the Flash F021 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.(**F021 API V2.01.01** or greater). Users also need to integrate the generated configuration files.

3.1 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_TwoActiveVS, Error_TwoCopyVS, Error_SetupStateMachine, Error_NoActiveVS, Error_CopyButNoActiveVS, Error_NoFreeVS, 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_TwoActiveVS or Error_TwoCopyVS or Error_CopyButNoActiveVS,* then the application has to provide info on which of the VS needs to be corrected in u8VirtualSector. TI_Fee_u16ActCpyVS will provide info on which of the VS's are Active/Copy. For error of type *Error_CopyButNoActiveVS,* TI_Fee_u16ActCpyVS will provide info on which VS is Copy. In this case, the second argument for the TI_Fee_ErrorRecovery should be the copy VS number. Error recovery API will mark the VS as Active.

If the error is of type *Error_NoFreeVS,* then the application has to provide info on which of the VS needs to be erased in u8VirtualSector. TI_Fee_u16ActCpyVS will provide info on which VS 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 VS failed. Call error recovery function to perform erase again. Check the variables TI_Fee_GlobalVariables[u8EEPIndex].Fee_u16ActiveVirtualSector / TI_Fee_GlobalVariables[u8EEPIndex].Fee_u16CopyVirtualSector to know which of the VS's are active/copy. Erase other sectors.

Application can access the variable "TI_Fee_u16ActCpyVS" to know details about the VS's.

Prototype for the API's are:

TI_Fee_ErrorCodeType **TI_FeeErrorCode**(uint8 u8EEPIndex);

void **TI_Fee_ErrorRecovery**(TI_Fee_ErrorCodeType Error Code, 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_TwoActiveVS* and ***TI_Fee_u16ActCpyVS*** = 0x0003, this means VS 1 and 2 are Active.

If projects want to make VS 1 as Active, then

Call TI_Fee_ErrorRecovery(*Error_TwoActiveVS*, 2);

Virtual sector 2 will be marked as Ready for Erase.

Virtual sector numbers start from 1.

3.2 Single and Double bit Error Corrections

FEE software provides a mechanism to detect single and double bit errors. In order to use this feature, application has to make sure that "**EE_EDACMODE[3:0]: Error Correction Mode"** in "**EE_CTRL1"** should be set to a value other than 0101, "**EE_ONE_EN: Error on One Fail Enable"** should be enabled, "**EE_ZERO_EN: Error on Zero Fail Enable"** should be enabled, "**EE_EDACEN[3:0]: Error Detection and Correction Enable"** should be set to a value other than 0101.

Projects have to then call error hook functions **TI_Fee_ErrorHookSingleBitError ( )** and **TI_Fee_ErrorHookDoubleBitError ( )** in ESM. For single bit error, an event is generated on channel 35 of ESM and for double bit error on channel 36 of ESM.

3.3 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

3.4 Symbolic Constants and Enumerated Data types

This section summarizes all 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
Fee_StatusType	FEE_OK	Function returned no error
	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
Fee_ErrorCodeType	Error_Nil=0	
	Error_TwoActiveVS=1	
	Error_TwoCopyVS=2	
	Error_SetupStateMachine=3	
	Error_CopyButNoActiveVS=4	
	Error_NoActiveVS=5	
	Error_BlockInvalid=6	
	Error_NullDataPtr=7	
	Error_NoFreeVS=8	
	Error_InvalidVirtualSectorParameter=9	

	Error_ExceedSectorOnBank=10	
	Error_EraseVS=11	
	Error_BlockOffsetGtBlockSize=12	
	Error_LengthParam=13	
	Error_FeeUninit=14	
	Error_Suspend=15	
	Error_InvalidBlockIndex=16	
	Error_NoErase=17	
	Error_CurrentAddress=18	
	Error_Exceed_No_Of_DataSets=19	
Fee_FlashErrorCorrectionActionType	Fee_None	Take no action on single bit errors
	Fee_Fix	Correct single bit errors
Fee_StatusCodeType	MEMIF_UNINIT	FEE Module is Uninitialized
	MEMIF_IDLE	FEE Module is Idle
	MEMIF_BUSY	FEE Module is Busy
	MEMIF_BUSY_INTERNAL	FEE Module is performing internal operations
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.
FEE_SW_MAJOR_VERSION	#define Macro which indicates the Major version of the FEE	
FEE_SW_MINOR_VERSION	#define Macro which indicates the Minor version of the FEE	
FEE_SW_PATCH_VERSION	#define Macro which indicate the Patch version of the FEE	

4-1. AutoSAR FEE Driver Symbolic Constants

3.6 AutoSAR FEE Driver Configuration Parameters

The AutoSAR FEE Driver needs two configuration files. These two files (Fee_Cfg.h & Fee_cfg.c) define which Flash sectors to be used for EEPROM emulation, define Data Blocks, Block Size and other configuration parameters. This section describes the configuration parameters in the AutoSAR FEE Driver.

3.6.1 Block Overhead

Parameter Name	FEE_BLOCK_OVERHEAD
Description	Indicates the number of bytes used for Block Header.
Default Value	0x18
Parameter Range	Fixed to 0x18
Target File	Fee_cfg.h
Sample Configuration	#define FEE_BLOCK_OVERHEAD 0x18

3.6.2 Maximum Blocking Time

Parameter Name	FEE_MAXIMUM_BLOCKING_TIME
Description	Indicates the maximum allowed blocking time for any Fee call.
Default Value	600.00
Parameter Range	Fixed to 600.00 µs.
Target File	Fee_cfg.h
Sample Configuration	#define FEE_MAXIMUM_BLOCKING_TIME 600.00

3.6.3 Page Overhead

Parameter Name	FEE_PAGE_OVERHEAD
Description	Indicates the Page Overhead in bytes.
Default Value	0x0
Parameter Range	Fixed to 0x0
Target File	Fee_cfg.h
Sample Configuration	#define FEE_PAGE_OVERHEAD 0x0

3.6.4 Sector Overhead

Parameter Name	FEE_VIRTUAL_SECTOR_OVERHEAD
Description	Indicates the number of bytes used for Virtual Sector Header.
Default Value	0x10
Parameter Range	Fixed to 0x10
Target File	Fee_cfg.h
Sample Configuration	#define FEE_VIRTUAL_SECTOR_OVERHEAD 0x10

3.6.5 Virtual Page Size

Parameter Name	FEE_VIRTUAL_PAGE_SIZE
Description	Indicates the virtual page size in bytes.
Default Value	0x8
Parameter Range	Fixed to 0x8
Target File	Fee_cfg.h
Sample Configuration	#define FEE_VIRTUAL_PAGE_SIZE 0x8

3.6.6 Driver Index

Parameter Name	FEE_INDEX
Description	Instance ID of FEE module. Should always be 0x0
Default Value	0x0
Parameter Range	Fixed to 0x0
Target File	Fee_cfg.h
Sample Configuration	#define FEE_INDEX 0x0

3.6.7 Job Error Notification

Parameter Name	FEE_NVM_JOB_ERROR_NOTIFICATION
Description	Call back function to notify a Job Error. This is only applicable if polling mode is OFF.
Default Value	NvM_JobErrorNotification
Parameter Range	User defined function name
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NVM_JOB_ERROR_NOTIFICATION NvM_JobErrorNotification

3.6.8 Job End Notification

Parameter Name	FEE_NVM_JOB_END_NOTIFICATION
Description	Call back function to notify a Job End. This is only applicable if polling mode is OFF.
Default Value	NvM_JobEndNotification
Parameter Range	User defined function name

Target File	Fee_cfg.h
Sample Configuration	#define FEE_NVM_JOB_END_NOTIFICATION NvM_JobEndNotification

3.6.9 FEE Operating Frequency

Parameter Name	FEE_OPERATING_FREQUENCY
Description	Device operating frequency in MHz. It is equivalent to the HCLK frequency in the TMS570 clock tree.
Default Value	160
Parameter Range	Device dependent parameter. Refer to the device datasheet to know the range.
	Fee_cfg.h
Sample Configuration	#define FEE_OPERATING_FREQUENCY 160

3.6.10 Polling Mode

Parameter Name	FEE_POLLING_MODE
Description	Indicates if polling mode is enabled/disabled. Currently, this parameter should always be STD_ON.
Default Value	STR0.48MO

3.6.12 Error Correction Handling

Parameter Name	FEE_FLASH_ERROR_CORRECTION_HANDLING
Description	Indicates desired action to be taken on detection of bit errors. Currently only Fee_None is supported.
Default Value	Fee_None
Parameter Range	Fee_None or Fee_Fix
Target File	Fee_cfg.h
Sample Configuration	#define FEE_FLASH_ERROR_CORRECTION_HANDLING Fee_None

3.6.13 Block Write Counter Save

Parameter Name	FEE_WRITECOUNTER_SAVE
Description	Used to enable/disable saving of write/erase counter value in to block header.
Default Value	STD_OFF
Parameter Range	STD_ON/ STD_OFF
Target File	Fee_cfg.h
Sample Configuration	#define FEE_WRITECOUNTER_SAVE STD_OFF

3.6.14 Enable CRC

Parameter Name	FEE_ CRC_ENABLE
Description	Used to enable/disable 16 bit CRC. If enabled, 16 bit CRC is calculated and written into Block header.
Default Value	STD_OFF
Parameter Range	STD_ON/ STD_OFF
Target File	Fee_cfg.h
Sample Configuration	#define FEE_ CRC_ENABLE STD_OFF

3.6.15 NumberOfEEPs

Parameter Name	FEE_NUMBER_OF_EEPS
Description	Used to configure number of emulations on a single bank.
Default Value	1
Parameter Range	1-2
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NUMBER_OF_EEPS 1

3.6.16 Number of Blocks

Parameter Name	FEE_NUMBER_OF_BLOCKS
Description	Defines the number of Data Blocks used for EEPROM emulation.
Default Value	0x1
Parameter Range	0x1 to 0xFFFE
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NUMBER_OF_BLOCKS 0x10

3.6.17 Number of Virtual Sectors

Parameter Name	FEE_NUMBER_OF_VIRTUAL_SECTORS
Description	Defines the number of Virtual Sectors used for FEE..
Default Value	0x2
Parameter Range	Min :0x2 Max : 0x4,For TMS570LS01227/TMS570LS1113. Min : 0x2 Max : 16, For TMS570LS05xx, TMS570LS07xx, TMS570LS09xx.
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NUMBER_OF_VIRTUAL_SECTORS 0x2

3.6.18 Number of Virtual Sectors on EEP1

Parameter Name	FEE_NUMBER_OF_VIRTUAL_SECTORS_EEP1
Description	Defines the number of Virtual Sectors used for EEP1
Default Value	0x2
Parameter Range	0 to FEE_NUMBER_OF_VIRTUAL_SECTORS-2
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NUMBER_OF_VIRTUAL_SECTORS_EEP1 0x2

3.6.19 Number of Eight Byte Writes

Parameter Name	FEE_NUMBER_OF_EIGHTBYTEWRITES
Description	Defines the number of 8 byte writes to be performed in Main Function.
Default Value	0x1
Parameter Range	1 to 255
Target File	Fee_cfg.h

3.6.20 Maximum Number of non configured blocks to copy

Parameter Name	FEE_NUMBER_OF_UNCONFIGUREDBLOCKSTOCOPY
Description	Defines the maximum number non configured blocks to copy.
Default Value	0x0
Parameter Range	0 to 255
Target File	Fee_cfg.h
Sample Configuration	#define FEE_NUMBER_OF_ UNCONFIGUREDBLOCKSTOCOPY 0x0

3.6.21 Address Range check during Read/Write

Parameter Name	FEE_CHECK_BANK7_ACCESS
Description	Defines the maximum number non configured blocks to copy.
Default Value	STD_OFF
Parameter Range	STD_ON/STD_OFF
Target File	Fee_cfg.h
Sample Configuration	#define FEE_CHECK_BANK7_ACCESS STD_OFF

3.6.22 Number of blocks and Data Sets

Parameter Name	FEE_TOTAL_BLOCKS_DATASETS
Description	Defines the total data sets in block configuration.
Default Value	1
Parameter Range	1-65536
Target File	Fee_cfg.h
Sample Configuration	#define FEE_TOTAL_BLOCKS_DATASETS 1

3.6.23 Generate Device and Virtual Sector Structures

Parameter Name	FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC
Description	Used to enable/disable generation of Device and Virtual sector structures.
Default Value	STD_OFF
Parameter Range	STD_ON/STD_OFF
Target File	Fee_cfg.h
Sample Configuration	#define FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC STD_OFF

Note: When FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC=STD_ON, device (Device_FlashDevice) and virtual sector configuration(Fee_VirtualSectorConfiguration) structures are populated during runtime. Also, these structures will be placed in RAM. Projects should take care that only FEE driver has access to these structures. When it's STD_OFF, structures are places in flash and are const.

3.6.24 Required Virtual Sector Size

Parameter Name	FEE_VIRTUALSECTOR_SIZE
Description	Defines the size of virtual sector.
Default Value	None
Parameter Range	4-32(see below note)
Target File	Fee_cfg.h
Sample Configuration	#define FEE_VIRTUALSECTOR_SIZE 4

Note: Depending on the device, parameter range can be different.
For TMS570LS12xx/11xx family devices, FEE bank is 4*16KB. Macro can take a value of 16 or 32. For TMS570LS09xx, TMS570LS07xx, TMS570LS05xx family devices, FEE bank is 16*4KB. Macro can take a value of 4 or 8 or 12 or 16 or 32.
FEE_VIRTUALSECTOR_SIZE * FEE_NUMBER_OF_VIRTUAL_SECTORS should not exceed the total available FEE bank size on device. This macro is only used when FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC is STD_ON. Based on FEE_VIRTUALSECTOR_SIZE and FEE_NUMBER_OF_VIRTUAL_SECTORS, elements of the structure Fee_VirtualSectorConfiguration will be populated during runtime.

3.6.25 FEE bank Physical Sector Size

Parameter Name	FEE_PHYSICALSECTOR_SIZE
Description	Defines the size of one physical sector on the device.
Default Value	None
Parameter Range	4/16(see below note)
Target File	Fee_cfg.h
Sample Configuration	#define FEE_PHYSICALSECTOR_SIZE 4

Note: This macro can only have 4/16 as value.
For TMS570LS12xx/11xx family devices, sector size is 16. For TMS570LS09xx, TMS570LS07xx, TMS570LS05xx family devices, sector size is 4. This macro is only used when FEE_GENERATE_DEVICEANDVIRTUALSECTORSTRUC is STD_OFF. This parameter is used to select device specific files.

3.6.26 Virtual Sector Configuration

Array Name	FEE_VirtualSectorConfiguration	
Description	Used to define a Virtual Sector	
Array Type	Fee_VirtualSectorConfigType. This is a structure having the following members.	
Members	FeeVirtualSectorNumber	Virtual Sector's Number.
	FeeFlashBank	Flash Bank for EEPROM emulation, Only Bank 7 for F021 devices.
	FeeStartSector	Starting Sector in the Bank for this Virtual Sector.
	FeeEndSector	

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.
Target File	Fee_cfg.c

3.6.26.4 FeeEndSector

Parameter Name	FeeEndSector
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.
Target File	Fee_cfg.c

3.6.26.5 Sample Virtual Sector Configuration

The following code snippet indicates one of the possible configurations for the Virtual Sectors:

```
/* 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            */
   },
};
```

Note: All the Virtual Sectors should have the same Flash Bank.
Only Bank 7 is supported for EEPROM emulation on F021 devices.

3.6.27 Block Configuration

Array Name	Fee_BlockConfiguration	
Description	Used to define a Data Block	
Array Type	Fee_BlockConfigType. This is a structure having 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 on which EEP, this block is configured.
Target File	Fee_cfg.c	

The configurations described below are repeated for each Data Block.

3.6.27.1 FeeBlockNumber

Parameter Name	FeeBlockNumber
Description	Each block is assigned a unique number starting from 0x1.
Default Value	0x1
Parameter Range	Min : 0x1, Max : 0xFFFE
Target File	Fee_cfg.c

3.6.27.2 FeeBlockSize

Parameter Name	FeeBlockSize
Description	Indicates the size of the Block in bytes.
Default Value	0x08
Parameter Range	0x1 to 0xFFFE
Target File	Fee_cfg.c

3.6.27.3 FeeImmediateData

Parameter Name	FeeImmediateData
Description	Indicates if the block is used for immediate data.
Default Value	FALSE
Parameter Range	TRUE / FALSE
Target File	Fee_cfg.c

3.6.27.4 FeeNumberOfWriteCycles

Parameter Name	FeeNumberOfWriteCycles
Description	Indicates the number of clock cycles required to write to a flash address location.
Default Value	0x1
Parameter Range	Device or core/flash tech dependent parameter.
Target File	Fee_cfg.c

3.6.27.5 FeeDeviceIndex

Parameter Name	FeeDeviceIndex
Description	Indicates the device index. This will always be 0.
Default Value	0x0
Parameter Range	Fixed to 0x0
Target File	Fee_cfg.c

3.6.27.6 FeeNumberOfDataSets

Parameter Name	FeeNumberOfDataSets
Description	Indicates the number of Datasets for this particular Block .
Default Value	0x1
Parameter Range	0x1 to 0xFF
Target File	Fee_cfg.c

3.6.27.7 FeeEEPNumber

Parameter Name	FeeEEPNumber
Description	Indicates into which EEP, this block is configured.
Default Value	0x0
Parameter Range	0x0 to 0x1
Target File	Fee_cfg.c

3.6.27.8 Sample Block Configuration

The following code snippet indicates one of the possible configurations for the Blocks:

```
/* 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              */
      1                /* 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           */
        0               /* EEP Number                   */
    },

};
```

3.7 API Classification

This section introduces the application-programming interface for the AutoSAR FEE Driver by grouping them into logical units. This is intended for the user to get a quick understanding of the AutoSAR FEE Driver APIs. For detailed descriptions please refer to the API specification section that follows this section.

3.7.1 Initialization

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

Name	Description
Fee_Init	Used to initialize the FEE module

4-4. AutoSAR FEE Driver Initialization APIs

3.7.2 Data Operations

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

Name	Description
Fee_Write	Used to initiate a Write Operation to a Data Block. Fee_MainFunction should be called at regular intervals to finish the Write Operation
Fee_Read	Fee_MainFunction should be called at regular intervals to finish this Operation
Fee_EraseImmediateBlock	Used to initiate an Erase Operation of a Data Block. Fee_MainFunction should be called at regular intervals to finish this Operation
Fee_InvalidateBlock	Used to initiate an Invalidate Operation on a Data Block. Fee_MainFunction should be called at regular intervals to finish this Operation

4-5. AutoSAR FEE Driver Data Operation APIs

3.7.3 Information

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

Name	Description
Fee_GetVersionInfo	Used to get the Driver version.
Fee_GetStatus	Used to get the status of the FEE module.
Fee_GetJobResult	Used to get the job result of a Data Operation.

4-6. AutoSAR FEE Driver Information APIs

3.7.4 Internal Operations

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

Name	Description
Fee_MainFunction	Used to complete the Data Operations initiated by any of the Data Operation functions.

4-7. AutoSAR FEE Driver Internal Operation APIs

3.7.5 Cancel/ Terminate Operations

The AutoSAR FEE Driver APIs that are used to cancel/terminate an ongoing Data Operation are listed below.

Name	Description
Fee_Cancel	Used to cancel an ongoing write, erase, invalidate or read operation.

4-8. AutoSAR FEE Driver Terminate/Cancel Operation APIs

3.7.6 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.

4-9. AutoSAR FEE Driver Error Info and Recovery APIs

3.7.7 Suspend/Resume Erase Sector

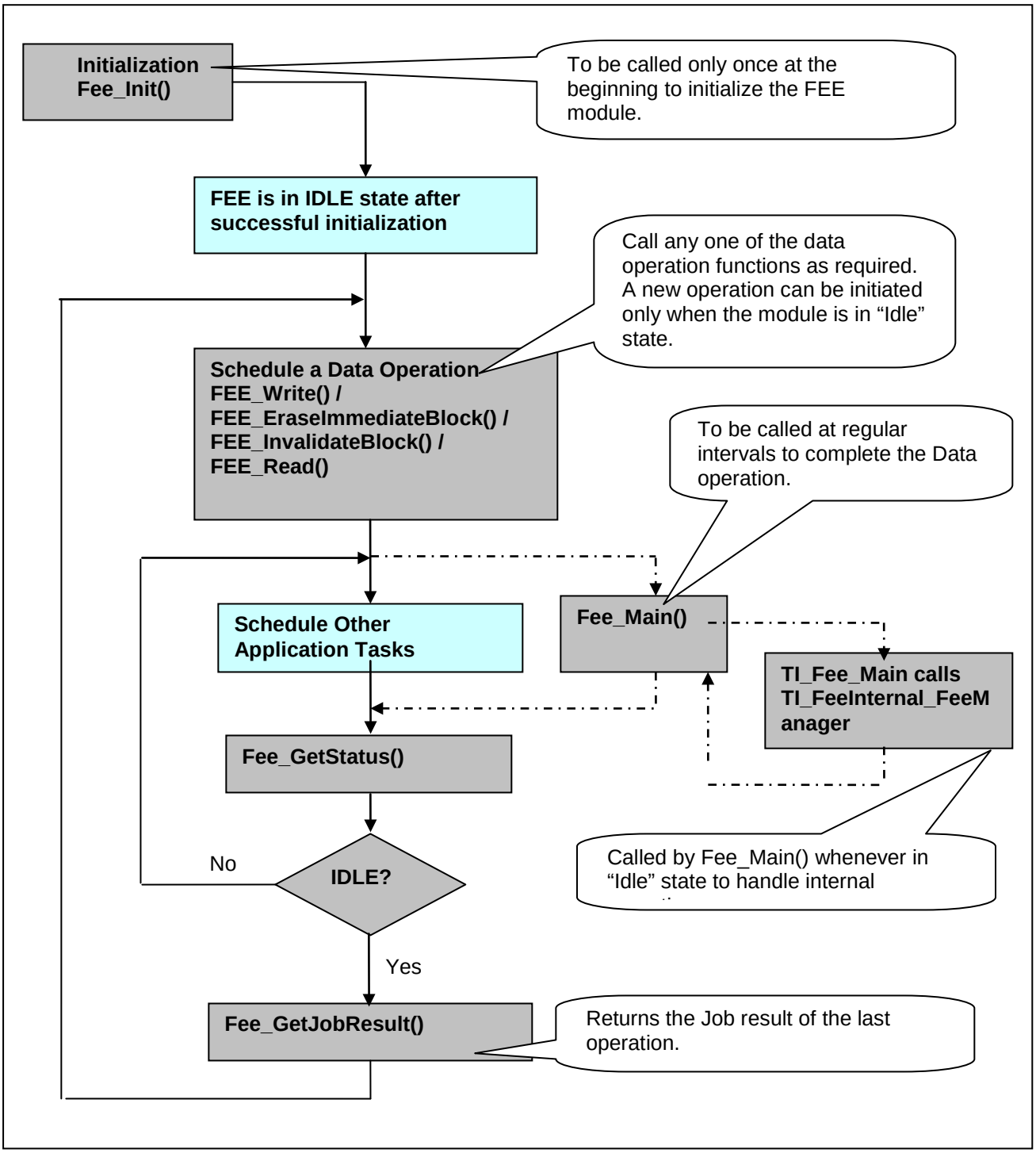
The TI FEE Driver APIs that are used to suspend/Resume erasing of sector.

Name	Description
TI_Fee_SuspendResumeErase	Function to suspend/resume erasing of sectors.

4-10. TI FEE Driver Suspend/Resume Erase Sector APIs

3.8 Integration Example

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



3.9 API Specification

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

For each of the published API listed, the following attributes are specified

- ***Prototype***: The signature of the function and or macro method in question
- ***Description***: The functionality of the procedure or macro
- ***Arguments***: The list of parameters supplied by the user
- ***Return value***: The evaluated return value from the procedure invoked

3.9.1 AutoSAR FEE Driver Functions

3.9.1.1 Fee Initilization Function

Prototype	void Fee_Init( )
Description	Used to initialize the FEE module
Arguments	None
Return value	None

The function Fee_Init() shall initialize the Fee module. This function shall initialize all Flash Memory relevant registers (hardware) with parameters provided in the given configuration set.

The function Fee_Init() shall initialize all Fee module global variables and those controller registers that are needed for controlling the flash device and that do not influence or depend on other (hardware) modules. Registers that can influence or depend on other modules shall be initialized by a common system module.

This function shall set the Fee module state to MEMIF_IDLE after successfully finishing the Fee module initialization.

Note: Do not call Fee_Init API multiple times in a single power cycle. There are chances of some blocks getting lost(if there are different block configurations for different instances of Fee_Init call), if it's called multiple times. If project requirement needs to call Fee_Init multiple times, make sure .data and .bss sections are cleared before calling Fee_Init.

3.9.1.2 Fee Write Function

Prototype		Std_ReturnType Fee_Write(uint16 BlockNumber, uint8* DataBufferPtr)
Description		Used to initiate a Write Operation on a Data Block / DataSet within a Data Block
Arguments	**BlockNumber**	The BlockNumber should comprise of the number of the logical block and the Dataset index within that logical block. The number of the logical block is left shifted by the maximum number of Datasets configures (NVM_DATASET_SELECTION_BITS) and then combined with the Dataset index to obtain the BlockNumber parameter.
	DataBufferPtr	Pointer to data buffer.
Return value		E_OK: The write job was accepted by the Fee module
		E_NOT_OK: The write job was not accepted by the Fee module.

The function Fee_Write() shall take the block start address and calculate the corresponding memory write address. This function shall copy the given/computed parameters to module internal variables, initiate a write job, set Fee module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.

The Fee module shall execute the job of the function Fee_Write() asynchronously within the Fee module's main function. This function shall write one or more complete flash pages to the Flash device. The job of this function shall program a Flash memory block with data provided via DataBufferPtr.

This function shall check the following:

1. That the write length is greater than 0. If this check fails, this function shall reject the Write job and return with E_NOT_OK.
2. That the Fee module has been initialized. If this check fails, this function shall reject the Write job and return with E_NOT_OK.
3. That the Fee module is currently not busy. If this check fails, this function shall reject the Write job and return with E_NOT_OK.
4. The given data buffer pointer for not being a null pointer. If this check fails, this function shall reject the Write job and return with E_NOT_OK.

Projects can configure FEE_NUMBER_OF_EIGHTBYTEWRITES to suitable value. By default this is 0x1. This means 8 bytes of data are written for every main function call. If this parameter is configured to 0x2, 16 bytes of data are written.

3.9.1.3 Fee Read Function

Prototype		Std_ReturnType Fee_Read(uint16 BlockNumber, uint16 BlockOffset, uint8* DataBufferPtr, uint16 Length)
Description		Used to perform a Read Operation on a Data Block / DataSet within a Data Block
Arguments	**BlockNumber**	The BlockNumber should comprise of the number of the logical block and the Dataset index within that logical block. The number of the logical block is left shifted by the maximum number of Datasets configures (NVM_DATASET_SELECTION_BITS) and then combined with the Dataset index to obtain the BlockNumber parameter.
	BlockOffset	Read address offset inside the block.
	DataBufferPtr	Pointer to data buffer.
	Length	Number of bytes to read.
Return value		E_OK: The Read job was completed successfully.
		E_NOT_OK: The Read job was not completed successfully.

The function Fee_Read() shall take the block start address and offset and calculate the corresponding memory read address. The address offset and length parameter can take any value within the given types range. This allows reading of an arbitrary number of bytes from an arbitrary star address inside a logical block.

The Fee module shall execute the job of the function Fee_Read() asynchronously within the Fee module's main function. This function shall initiate a read operation which copies a continuous Flash memory block starting from the computed start address to the size of Length to the buffer pointed to DataBufferPtr. This function shall set Fee module status to MEMIF_BUSY, set the job result to MEMIF_JOB_PENDING and return with E_OK.

This function shall check the following:

1. That the read length is greater than 0. If this check fails, this function shall reject the Read job and return with E_NOT_OK.
2. That the Fee module has been initialized. If this check fails, this function shall reject the Read job and return with E_NOT_OK.
3. That the Fee module is currently not busy. If this check fails, this function shall reject the Read job and return with E_NOT_OK.
4. The given data buffer pointer for not being a null pointer. If this check fails, this function shall reject the Read job and return with E_NOT_OK.

Main function will do complete data read in single main function call.

3.9.1.4 Fee Erase Function

Prototype		Std_ReturnType Fee_EraseImmediateBlock(uint16 BlockNumber)
Description		Used to initiate a logical Erase operation on a Data Block / DataSet within a Data Block
Arguments	**BlockNumber**	The BlockNumber should comprise of the number of the logical block and the Dataset index within that logical block. The number of the logical block is left shifted by the maximum number of Datasets configures (NVM_DATASET_SELECTION_BITS) and then combined with the Dataset index to obtain the BlockNumber parameter.
Return value		E_OK: The Erase job was accepted by the Fee module
		E_NOT_OK: The Erase job was not accepted by the Fee module**.**

The function Fee_ EraseImmediateBlock() shall take the block number calculate the corresponding memory address for that block.

The Fee module shall execute the job of the function Fee_ EraseImmediateBlock() asynchronously within the Fee module's main function.
The function Fee_EraseImmediateBlock() ensures that the Fee module can write immediate data. As the Fee module implementation should always be ready to write data, unless the block is marked as FeeImmediateData == FALSE, it shall return E_OK. If the block is marked as FeeImmediateData == FALSE it will return E_NOT_OK.

This function shall check the following:
1. That the Fee module has been initialized. If this check fails, this function shall reject the Erase job and return with E_NOT_OK.
2. That the Fee module is currently not busy. If this check fails, this function shall reject the Erase job and return with E_NOT_OK.
3. The given Block Number is marked for Immediate Data Write. If this check fails, this function shall reject the Erase job and return with E_NOT_OK.

3.9.1.5 Fee Invalidate Function

Prototype		Std_ReturnType Fee_InvalidateBlock(uint16 BlockNumber)
Description		Used to initiate a logical Invalidate operation on a Data Block / DataSet within a Data Block
Arguments	**BlockNumber**	The BlockNumber should comprise of the number of the logical block and the Dataset index within that logical block. The number of the logical block is left shifted by the maximum number of Datasets configures (NVM_DATASET_SELECTION_BITS) and then combined with the Dataset index to obtain the BlockNumber parameter.
Return value		E_OK: The Invalidate job was accepted by the Fee module
		E_NOT_OK: The Invalidate job was not accepted by the Fee module**.**

The function Fee_ InvalidateBlock() shall take the block number calculate the corresponding memory address for that block.
The Fee module shall execute the job of the function Fee_ InvalidateBlock() asynchronously within the Fee module's main function. This command shall change the valid bit in the Data Block header to invalidate it.

This function shall check the following:
1. That the Fee module has been initialized. If this check fails, this function shall reject the Invalidate job and return with E_NOT_OK.
2. That the Fee module is currently not busy. If this check fails, this function shall reject the Invalidate job and return with E_NOT_OK.

3.9.1.6 Fee Get Version Info Function

Prototype	void Fee_GetVersionInfo(Std_VersionInfoType* VersionInfoPtr)
Description	Function to return the version information of the Fee module**.**
Arguments	None
Return value	None

The function Fee_GetVersionInfo() shall return the version information for the Fee module. The version information includes:

- Module Id
- Vendor Id
- Fee specific version numbers MM.mm.rr
  - MM – Major Version
  - mm – Minor Version
  - rr – Revision

The function Fee_GetVersionInfo() shall be pre-compile time configurable On/Off by the configuration parameter FEE_VERSION_INFO_API.

3.9.1.7 Fee Get Status Function

Prototype	MemIf_StatusType Fee_GetStatus()
Description	Function to return the status of the Fee module**.**
Arguments	None
Return value	MEMIF_UNINIT: The Fee Module has not been initialized.
	MEMIF_IDLE: The Fee Module is currently idle.
	MEMIF_BUSY: The Fee Module is currently busy.
	MEMIF_BUSY_INTERNAL: The Fee Module is currently busy with internal management operations

The function Fee_ GetStatus() shall return the status information for the Fee module.

The function Fee_ GetStatus() shall be pre-compile time configurable On/Off by the configuration parameter FEE_GET_STATUS_API.

3.9.1.8 Fee Get Job Result Function

Prototype	MemIf_JobResultType Fee_GetJobResult()
Description	Function to get the job result from the Fee module**.**
Arguments	None
Return value	MEMIF_JOB_OK: The last job has finished successfully.
	MEMIF_JOB_PENDING: The last job is waiting for execution or is currently being executed.
	MEMIF_JOB_FAILED: The last read/erase/write/compare job failed.
	MEMIF_BLOCK_INCONSISTENT: The requested block is inconsistent, it may contain corrupted data.
	MEMIF_JOB_CANCELLED: The last job has been cancelled.
	MEMIF_BLOCK_INVALID: The requested block has been invalidated. The requested read operation cannot be performed.

The function Fee_GetJobResult() shall return the result of the last job synchronously. The erase, write, read and invalidate functions shall share the same job result, therefore, only the result of the last job can be queried.

The function Fee_GetJobResult shall be pre-compile time configurable On/Off by the configuration parameter FEE_GET_JOB_RESULT_API.

3.9.1.9 Fee Main Function

Prototype	void Fee_MainFunction()
Description	Function to handle the requested read/write/erase jobs and the internal management operations of the Fee module.
Arguments	None
Return value	None

The function Fee_MainFunction() shall asynchronously handle the requested read/write/ erase jobs respectively and the internal management operations.

The function shall accept only one read, write, or erase job at a time. When a job has been initiated, the Fee module's environment shall call the function Fee_MainFunction() cyclically until the job is finished. This function shall only process as much data in one call cycle as statically configured for the current job type (read, write or erase).

After a read, erase or write job has been finished; the function shall set the Fee module's job result to MEMIF_JOB_OK if it is currently in state

MEMIF_JOB_PENDING. Otherwise, it shall leave the result unchanged. Furthermore, the function shall set the Fee module's state to MEMIF_IDLE and call the job end notification function.

This function shall at most issue one sector erase command (to the hardware) in each cycle.

3.9.1.10 Fee Manager Function

Prototype	Fee_StatusType TI_FeeInternal_FeeManager(void)
Description	Function to handle the internal operations of the FEE driver.
Arguments	None
Return value	FEE_OK : Function detected No Error
	FEE_ERROR: Function detected an Error condition and returned.

The function TI_FeeInternal_FeeFeeManager() manages the Flash EEPROM Emulation and is called when no other job is pending by the Fee_MainFunction. 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 MEMIF_IDLE state. It should set the Fee module to MEMIF_BUSY_INTERNAL state.

3.9.1.11 Fee Cancel Function

Prototype	void Fee_Cancel(void)
Description	Function to cancel/terminate an ongoing operation.
Arguments	None
Return value	None

The function Fee_Cancel() provides functionality for cancelling/terminating an ongoing operation. This function shall cancel an ongoing flash read, write or erase job and shall reset the internal variables making the module ready to accept a new job. This function shall operate synchronously so after returning from this function a new job can be started.

3.9.1.12 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.

Prototype	TI_FeeErrorCodeType TI_FeeErrorCode(uint8 u8EEPIndex)
Description	Function to know the error type.
Arguments	EEP Index
Return value	Error code

3.9.1.13 TI_Fee_ErrorRecovery

This function provides functionality to recover from any severe errors.

Prototype	Void TI_Fee_ErrorRecovery(TI_Fee_ErrorCodeType Error Code, uint8 u8VirtualSector)	
Description	Function to recover from severe errors.	
Arguments	**Error Code**	**Error_TwoActiveVS**
		Error_TwoCopyVS
		Error_SetupStateMachine
		Error_NoActiveVS
		Error_CopyButNoActiveVS
		Error_NoFreeVS
	Virtual Sector Number	
Return value	**None**	

3.9.1.14 TI_Fee_SuspendResumeErase

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

Prototype	void TI_Fee_SuspendResumeErase(TI_Fee_EraseCommandType Command)
Description	Function to suspend/Resume erasing of sector.
Arguments	Suspend_Erase/Resume_Erase
Return value	none

Note: This API has to be called once after 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.

3.10 Privilege Mode access

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

- Fee_Init
- TI_FeeInternal_WriteDataF021
- TI_Fee_Read

3.11 Deviations from Autosar3.x requirements

- 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 is completed and then erasing of a sector is completed. If there is a powerloss during copying of blocks, then the next Fee_Init will resume COPY operation.)

3.12 Important Notes

- If projects are using bootloader, make sure the active bank in FMAC register is same as before the start of bootloader and after the completion of bootloader.
- Projects should not call Fee_Init mutiple times in one power cycle. If it is required to call, make sure all the global, static variable sections are cleared before calling Fee_Init.