EEPROM Emulation Driver for 0.18um SGF Flash

In

S12XS, S12P, and S12HY

User's Manual

REVISION LIST

Version No.	Date	Author	Description
0.0a	05-Jul- 2007	Jiben_Prakash	Initial Version
0.1	07-20-2007	Sindhu R01	Added performance data in Appendix C
0.2	13-Aug-2007	Jiben_Prakash	Updated the function prototypes to support Cosmic Compiler
1.0	17-Jul-2008	Sindhu R01	Modified the Brown-out handling (Margin checks and DEAD sectors). Added functions FlashSetUserMargin(), FSL_CheckMarginLevel() and FSL_ReadDFlash(). Added section regarding Configuration Macros. Modified the prototypes of all the function
1.1	26-Aug-2008	Sindhu R01	Updated Table 75, 76 and 77 with the performance data calculated using the 40MHz bus clock.
2.0	13-Nov-2008	Sindhu R01	Updated for interrupt support. Added functions FSL_Main(),FSL_ISRhandler(), FSL_UpdtCacheTable(),FSL_InitFirstTime(),FSL_ PgmSectorStatus(), FSL_GetErasingCycles(), FlashInterruptSet(). Updated FSL_InitEeprom(), FSL_WriteEeprom(),FSL_DeinitEeprom() and FSL_SwapSector(). Removed the Performance related data.
2.1	26-Nov-2008	Sindhu R01	Added the Performance related data.
2.3	04-Mar-2009	Sindhu R01	Modified to support the multiple byte programming and converting 32-bit addressing to 16-bit addressing. Updated the performance data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 SCOPE

1.1 Document Overview

This document is the user manual of the EEPROM emulation driver for S12XS/S12P microcontroller family using interrupts. The roadmap for the document is as follows:

Section 1.2 gives an overview and features of the System. Section 2 Lists the abbreviations and references used in the document. Section 3.1 details the software architecture. Section 3.2 details the Configuration parameters used. Section 3.3 and 3.4 details the Driver memory layout. Section 3.5 details of calling the '*CallBack()*' function. Section 3.6 gives details of the return codes of the EED. Section 3.7 details the Macros used by the EED. Section 4 explains EEPROM Emulation Operations. Section 5 gives functional hierarchy and calling conventions and Section 6 gives the API Specifications. Appendix gives a list of error codes from high level, middle level and low level APIs and the return codes of low level APIs

1.2 System Overviews

EEPROM (electrically erasable programmable read only memory), which can be byte or word programmed and erased, is often used in automotive electronic control units (ECUs). This flexibility for program and erase operations makes it suitable for data storage of application variables that must be maintained when power is removed and need to be updated individually during run-time. For the devices without EEPROM memory, the page-erasable Flash memory can be used to emulate for EEPROM through EEPROM emulation software.

The EEPROM emulation driver for S12XS/S12P implements the fixed-length data record scheme emulation on SGF Flash. The EEPROM functionalities to be emulated include organizing data records initializing and de-initializing EEPROM reporting EEPROM status reading and writing data records.

Four or more sectors shall be involved in emulation with a round robin scheduling scheme.

1.3 Features

S12XS EEPROM Emulation Driver using interrupts provides the following features:

- High speed program operations.
- Position-independent and ROM-able - all driver functions can run from RAM or FLASH.
- User configurable emulated EEPROM size.
- Larger life for the flash memory used for EEPROM emulation.
- Unique feature of Cache Table which makes data retrieval fast.
- Concurrency support via callback.
- Release format is C source code format.
- Ready-to-use demos illustrating the usage of the driver
- The Flash Erase operation is interrupt based. The EED does not wait for the erase operation to complete. It launches the erase and returns control to the User application.

2 ACRONYMS AND REFERENCES

2.1 Acronyms

Table 1 Acronyms

Abbreviation	Complete Name
EEPROM	Electrically Erasable Programmable Read Only Memory
ECU	Electronic Control Unit
EED	EEPROM Emulation Driver
SGF	Split Gate Flash

2.2 Reference

Table 2 Reference

Sl No.	Document Name	Version	Document Identifier (If any)
1.	Using the S12XE-Family as a Development Platform for the S12XS-Family	2.0	AN3327.pdf
2.	256-Kbyte Flash Module (S12XFTMRK1V0)	0.2	S12XFTMRK1_V0.pdf

3 SOFTWARE ARCHITECTURE

3.1 System Architecture

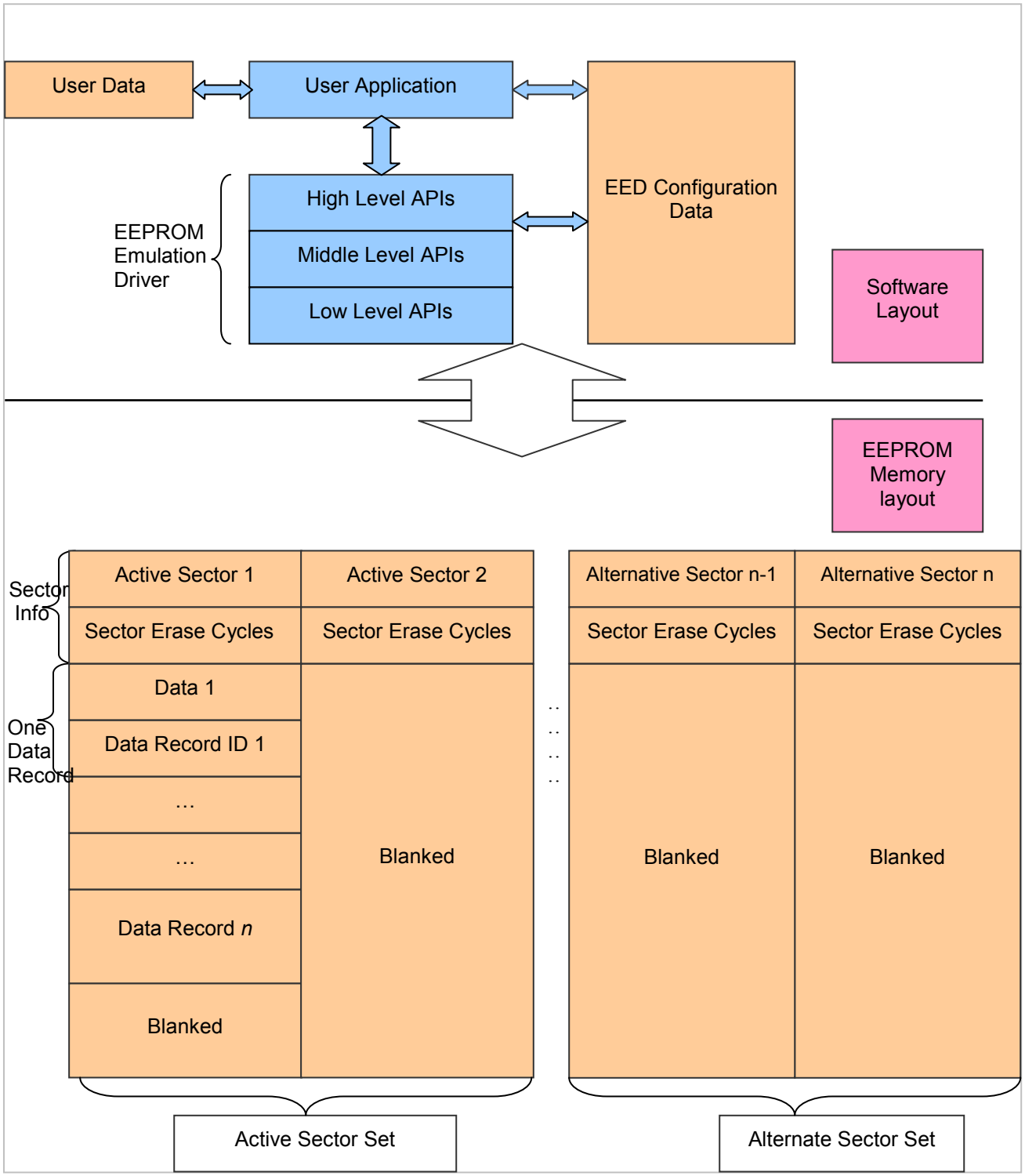

Figure 1 System Architecture

3.2 Configuration Marcos

The HCS12XS/S21P EEPROM Emulation Driver has some configurations macros. These macros are placed in the SSD_SGF18.h. Depending upon the macro selected there will be change in some internal macros and source code.

3.2.1 Derivative Selection

Based on the value assigned to the macro **`'SGF18_SELECT'`**, the other macros will be given the corresponding values and some parts of the code will be conditionally compiled.
If the code is used for S12XS family then, define **`'SGF18_SELECT'`** as **`'S12XS_SGF18'`**.
If the code is used for S12P family then, define **`'SGF18_SELECT'`** as **`'S12P_SGF18'`**.

3.2.2 Other Configuration Macros

Table 3 Configuration Macros used by the low level functions

Macro Name	Description
SSD_MCU_REGISTER_BASE	Base address of MCU register block
SSD_BUS_CLOCK	Bus clock. The unit is 10KHz
SSD_OSCILLATOR_CLOCK	Oscillator clock. The unit is 10KHz

3.3 EEPROM Emulation Software Layout

3.3.1 EEPROM emulation driver

The EEPROM emulation driver has three level APIs: high level, middle level and low level.

- High level (User level) APIs provide the user's interface and program flow controlling
- Middle level APIs provide the relative independent task unit
- Low level APIs use the Standard Software Driver to provide the fundamental Flash operations.

3.3.2 Flash Interrupt Support

The Flash Erase operations of a sector are interrupt based. The EED driver launches the erase operation and returns. On completion of the erase operation the ISR handler is called which sets the status of the erase flag as done. The user should call the *'FSL_Main()'* function which completes the initialization of sectors and other operations to make it ready for EEPROM emulation, after an erase is initiated by other high level APIs.

3.4 EEPROM Emulation Memory Layout

3.4.1 EEPROM Sectors and scheduling

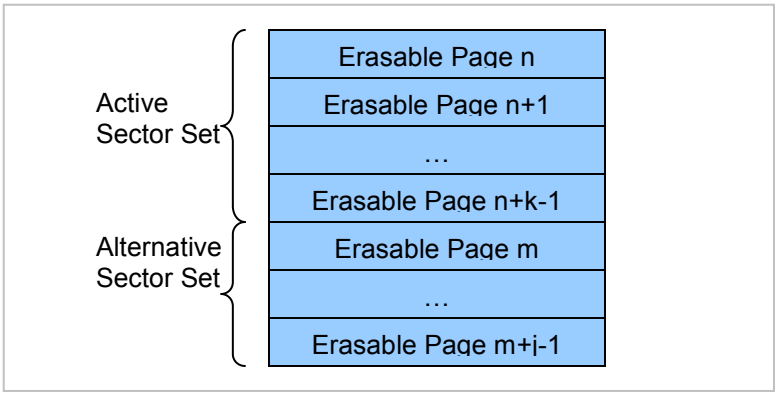

Figure 2 EEPROM Sector

S12XS/S12P EEPROM emulation driver adopts S12XS/S12P Family Flash to be emulated as EEPROM. Minimum two sector sets are needed to emulate EEPROM. One is called the active sector set and the other the alternative sector set. There can be more than one active and alternative sectors used for emulation. This emulation scheme follows the SailFish (HCS12XEPxx) hardware EEPROM emulation scheme.

3.4.1.1 Sector Allocation

The number of sectors used for active and alternative sector set can be different. The minimum number of sectors used as active sector is a decided by the number of bytes the user wants to store in the EEPROM; this is **`'EED_ACTIVE_SECTOR_REQUIRED'`**. The user can allocate more sectors as active sectors; this is **`'EED_EXTRA_ACTIVE_SECTORS'`**. The number of alternative sectors to be used is user configurable. At least two alternative sectors are necessary for the emulation scheme to work. The user can allocate more than two alternative sectors depending on the availability of the memory and the requirement of his application.

The following macros shall be used to calculate the number of sectors used for emulation:

Table 4 Sector Allocation macro

Macro Name	Description
EED_SECTOR_SIZE	Size of sector
EED_SECTOR_CAPACITY	Number of data records that can be stored in a sector
EED_ACTIVE_SECTOR_REQUIRED	Minimum number of ACTIVE sectors required for the emulation.
EED_EXTRA_ACTIVE_SECTORS	Number of extra ACTIVE sector allocated by the user.
EED_ACTIVE_SECTORS	Total number of active sectors that are used for emulation.(equal to EED_EXTRA_ACTIVE_SECTORS + EED_ACTIVE_SECTOR_REQUIRED)
EED_ACTUAL_READY_SECTORS	Number of alternative sectors needed for Round Robin scheme

EED_EXTRA_READY_SECTORS	Number of extra alternative sectors needed to overcome the problem of DEAD sectors
EED_READY_SECTORS	Number of alternative sectors that are used for emulation.(equal to EED_ACTUAL_READY_SECTORS + EED_EXTRA_READY_SECTORS)
EED_SECTORS_ALLOTED	Total number of flash sectors that are used for emulation. (equal to EED_READY_SECTORS+ EED_ACTIVE_SECTORS)
EED_SECTOR_SIZE	Size of EED sector (Multiple of D-Flash sector size; Can include more than one D-Flash Sector)

The **'EED_EXTRA_READY_SECTORS'** and **'EED_ACTUAL_READY_SECTORS'** can be configured by the user. The **'EED_EXTRA_READY_SECTORS'** should be a minimum of one sector. This is to help recover from a situation where one of the sectors becomes corrupted, so can't be used for emulation and is declared as a 'DEAD' sector. The **'EED_ACTUAL_READY_SECTORS'** should be a minimum of two sectors, to help recover from a brown-out scenario. Consider a situation where there is only one alternative sector. And a sector swapping copies all the data records from the oldest active sector to the alternative sector, and makes the updated alternative sector as Active. If a brownout occurred now, then the '*FSL_InitEeprom()'* API will not be able to decide which of the Active sectors should be erased to make it an alternative sector.

The '**EED_MAX_RECORD_NUMBER'** is the total number of data records that are to be stored which depends on the EEPROM size required to be emulated.

There shall be '**EED_ACTIVE_SECTORS'** sectors grouped as 'Active Sectors set' while, '**EED_READY_SECTORS'** sectors shall be grouped as 'Alternative sector set'.

If any of the sectors get corrupted, then the sector is declared as a dead sector. This will result in the reduction of the number of ALTERNATIVE sectors available for emulation.

3.4.1.2 Sector Scheduling

All the sectors in the 'Active' sector set shall be marked active. These sectors shall be scheduled in a round robin queue. If one sector gets filled up, the consecutive active sector shall be used to store the EEPROM data. If there are no active sectors available, then the data from the oldest active sector shall be compressed to the oldest alternative sector which shall now be used as active sector. The oldest active sector shall be erased and declared as alternative.

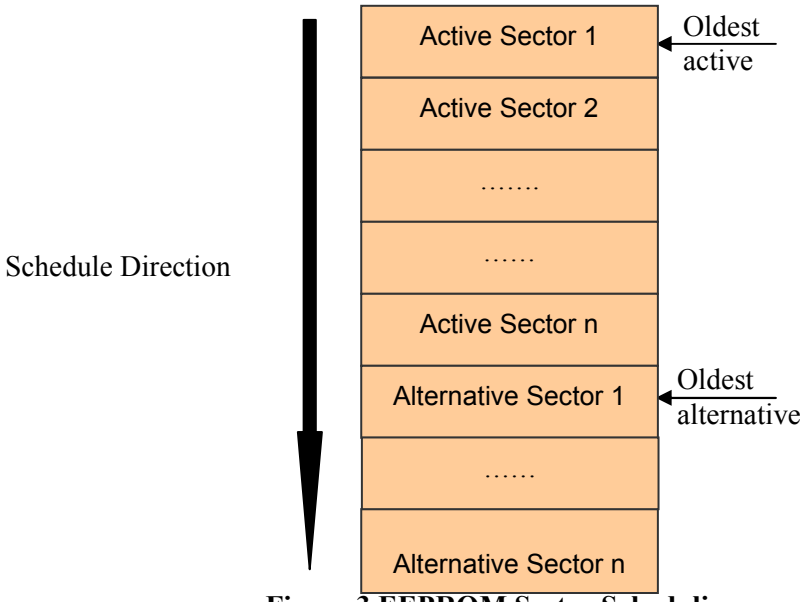

Figure 3 EEPROM Sector Scheduling

3.4.2 EEPROM Data Organization

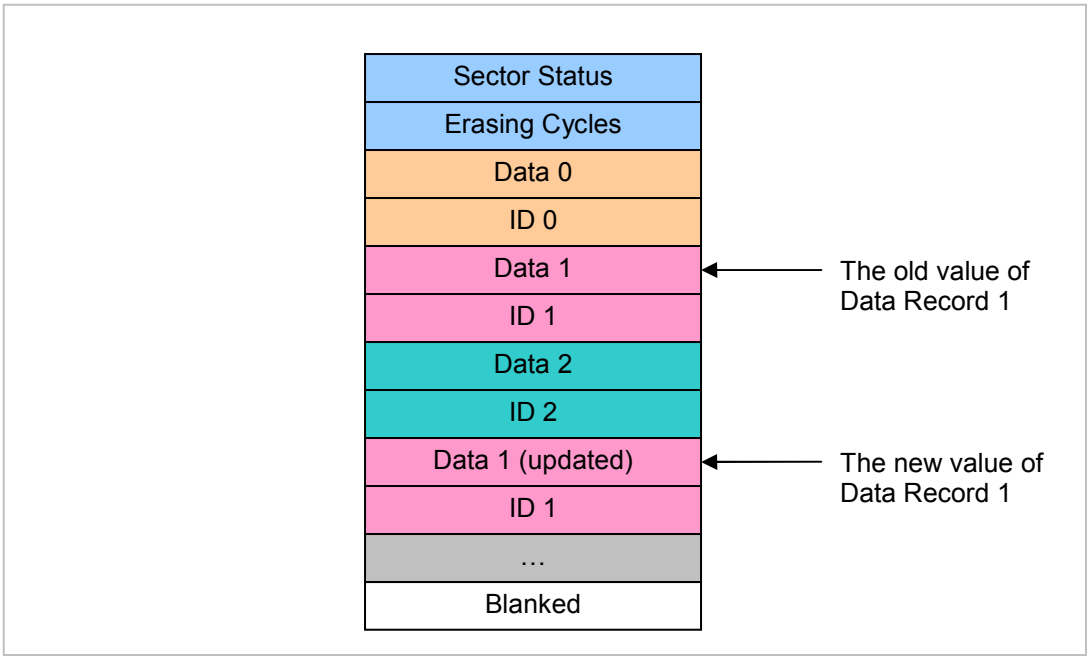

Figure 4 EEPROM Sector Memory Layouts

Each emulation sector contains:

- Sector status field: Store the Sector status, the actual status depends on a combination of the status byte, erased cycles and the first data record ID byte.
- Erasing cycles: Store the sector's erasing cycles since the EEPROM emulation is set up. It will be incremented after each erasure.
- Data records field: Each data record has two fields:
  - Data record ID: the data record identifier
  - Data: user's raw data
- Blank field: It is used for storing new data records.

3.4.2.1 Data Record Configuration

Because the EEPROM emulation driver adopts the fixed length scheme, each record will have the same data length. The next data record location can be obtained by adding the current record start address with the record length.

Table 5 Data Record macro

Macro Name	Description
EED_DATA_ID_SIZE	Data ID Size is 2 bytes.
EED_DATA_VALUE_SIZE	Size of data stored in a data record. (Size of the data corresponding to a data ID).
EED_TOTAL_DATA_SIZE	Total EEPROM size needed by the user to store data. This is user configurable.

EED_MAX_RECORD_NUMBER	Number of Data Records is calculated if we know the total Data Size.
EED_RECORD_LENGTH	Data Record Size can be calculated as sum of ID and data size.

3.4.2.2 Data Updation and status accounting

The data record cannot be updated directly on the same location. Instead, a new data record with the new value will be written to the EEPROM. While reading, the read routine will read the latest occurrence of the ID in the active sectors.

While updating the data, the data ID and the data bytes are both updated. The order of updation is as follows:

1. Program Data fields.
2. Program Data ID field

Table 6 Data Record Status

Data Id Field	Data Fields	Status of Data Record
0xFFFF	0xFFFF	Record in Erased state. Can be used for holding data
0xFFFF	0xXXXX	Data record is under updation. Further writes possible in the next record location only.
0xXXXX	0xXXXX	Data record contains a valid data.
0x0000	0xXXXX	Data record is invalid.

3.4.2.3 Sector status accounting

The status byte of the Sector is a two bytes field. It can hold only two values (Viz) 0xFFFF, 0x0000 or 0xFACF. The status of the sector is determined by a combination of the sector status field, the sector erase cycle field and the sector's first data field as follows:

Table 7 Sector Status

Sector Status Field	Sector erase cycles	First Data Fields	Status of sector
0xFFFF	0xFFFF	0xFFFF	This sector is blank.
0xFFFF	0xXXXX	0xFFFF	This sector is an alternative sector.
0xFFFF	0xXXXX	0xXXXX	This sector is under updation. Specifically, it is in the process of sector compression.
0xFACF	0xXXXX	0xXXXX	This is the active sector. It will take new data to be stored in EEPROM.
0x0000	0xXXXX	0xXXXX	This is the dead sector. It will not be used by the EED for emulation

3.4.3 Cache table configuration

A Cache table that holds the address of the latest occurrence of the most frequently used data IDs will be used to speed up reading of data and also to speed up sector compression. This feature can be enabled or disabled using the macro '**`EED_CACHETABLE_ENABLE`**'.

The cache table size is configured by the user using the macro '**EED_MAX_CACHETABLE_ENTRY**'. It is recommended that the user may not configure a huge size for the cache table.

The start address of the cache table is RAM is a user configurable macro '**EED_CACHETABLE_START_ADDRESS'**.

This table shall hold the address of IDs starting from ID no.1 to ID no. **EED_MAX_CACHETABLE_ENTRY**. It is the user's responsibility to use these IDs to store the most commonly/frequently used data records.

3.5 Callback notification

The EEPROM Emulation Driver facilitates the user to set the application's 'CallBack' function to the global *'CallBack()'* function pointer, so that time-critical events can be serviced during EEPROM operations. Servicing watchdog timers is one such time critical event. If it is not necessary to provide the '*CallBack()'* service, the user will be able to disable it by setting the macro **CALLBACK_ENABLE** to **FALSE**.

The job processing callback notifications shall have no parameters and no return value.

3.6 Return codes

The following return codes shall be used.

Table 8 Return Codes of High LevelAPIs

Name	Value	Description
EED_OK	0x0000	The requested operation was successful
EED_ERROR_NOT_BLANK	0x0001	The flash memories are not blank
EED_ERROR_SIZE	0x0002	Size is not word aligned
EED_ERROR_NOFND	0x0003	Record not found
EED_ERROR_RANGE	0x0004	Size or destination or end address for program exceeds the valid range
EED_ERROR_SSTAT	0x0005	Sector status error
EED_ERROR_VERIFY	0x0006	Corresponding source data and content of destination location mismatch.
EED_ERROR_IDRNG	0x0007	Record identifier exceeds the valid range
EED_ERROR_ACCERR	0x0008	Access error flag is set while operating the Flash.
EED_ERROR_PVIOL	0x0010	Protection violation flag is set while operating the Flash.
EED_ERROR_MGSTAT0	0x0020	Non-correctable errors have been encountered during the verify operation(Hardware Error)
EED_ERROR_MGSTAT1	0x0040	Errors have been encountered during the verify operation(Hardware Error)
EED_ERROR_INVALIDCLK	0x0800	The clock divider value is not correct.

Note:

The access error, protection violation error, size not aligned error, address range error and the hardware generated errors are not specifically mentioned in any of the middle and low level functions. Yet, it is likely that this error will be thrown up for any API that deals with direct operation on the hardware like a write or an erase.

3.7 Macros Used

The macros that are used in this EED are as follows:

Table 9 Macros Used

Name	Value	Description
EED_SECTOR_ACTIVE	0x0000	Sector status is Active
EED_SECTOR_ALTERNATIVE	0x5555	Sector status is Alternative
EED_SECTOR_BLANK	0xFFFF	Sector status is Blank
EED_SECTOR_UPDATE	0xAAAA	Sector status is Partially Updated
EED_SECTOR_INVALID	0x5A5A	Sector status is Invalid
EED_SECTOR_DEAD	0xA5A5	Sector status is Dead
EED_FLASH_START_ADDRESS	-	Starting address of flash allocated for Emulation of EEPROM.
EED_FLASH_END_ADDRESS	-	End address of flash allocated for Emulation of EEPROM.
EED_CACHETABLE_ENABLE	-	Enable or disable use of cache table.
EED_MAX_CACHETABLE_ENTRY	-	Number of entries that the cache table will hold. This also represents the maximum record ID that the Cache table will hold.
EED_CACHETABLE_START_ADDRESS	-	Start address of the Cache table in RAM

4 EEPROM EMULATION OPERATIONS

4.1 Initialize EEPROM

Before using the EEPROM, it needs to be initialized. The initialization will deal with two kinds of states:

- The first usage of EEPROM:
  In this case, the EED will format the number of sectors user wants as active and number of ready sectors user wants.

- Continue usage of EEPROM:
  In this case, the EED will determine which set of sectors is the active and initialize other set as alternative and update the erase cycles of all the sectors.

- Initialization of Cache Table:

Cache table will also be initialized, all entries of cache table will be made 0xFFFF if the data is not present in the EEPROM.

- Brown-Out handling:

The following brown out scenarios will be handled during initialization of EEPROM:

- Brownout during Initialization:
  - During initialization if there is a reset, which results in few sectors uninitialized. The EED initializes the remaining the blank sectors as 'Active' or 'Alternative'. The driver also does a margin read on the last initialized sector. If the margin read does not succeed, then the sector is re-initialized.
- Brownout during Swapping:
  - During swapping if there is a reset, which results in a sector's status as under updation, that is brownout has happened in between swapping of records. The EED erases the updated sector and makes it alternative again, so that swapping can start again.
  - During swapping if there is a reset, which results in one more sectors as 'Active' that is total number of 'Active' sectors is one more than the actual number. The EED does a margin read on the current ACTIVE sector whose status was the last programmed word. If, the status read is not 100% clean, then the EED erases that sector and initialize it as ALTERNATIVE. Then the previous ACTIVE sector is set as the current ACTIVE sector. If, the status read is 100% clean, then the EED erases the oldest 'Active' sector and makes it alternative, so that the total number of 'Active' sectors remains the same.
  - During swapping if there is a reset, which results in one sector as completely blank, that is, Brownout happened when the oldest 'Active' was being erased after swapping of data. The EED does an erase verify on the blank sector to ensure that it is completely erased, and then makes this erased sector as alternative by programming its erasing cycles to zero.
- Brownout during Writing:
  - During writing there is a reset, which results in a situation where during initialization there is 'Active' sector partially filled and there are 'Active' sectors completely blank/Full. The EED initializes the partially filled sector as 'current Active'. It also does a margin read of the ID of the last record written. If the record is not 100% clean, it is invalidated by writing 0x0000 to the ID field.

4.2 Write EEPROM Data

Because the SGF Flash memory cell cannot be erased individually, EED must write a new data record with same data ID and updated value in the EEPROM blank area when the data need updating. After several times updating, the active sectors may not have enough free space for writing a new data record. It is needed to copy all the latest data records from the active sector to the oldest alternative sector to clean up the EEPROM. This procedure is called “swapping” and after swapping, the alternative sector will become the new active sector and the oldest active sector will be formatted as new alternative sector. During the sector swapping, there are several sector status and swapping stages for recovering from accident (brownout):

Sector status:

- EED_SECTOR_BLANK
  The sector is fully erased and has not been initialized as alternative sector.

- EED_SECTOR_ALTERNATIVE
  The sector is initialized as alternative sector and ready for swapping.

- EED_SECTOR_UPDATE
  The latest data records are being copied from the active sector to this sector, but the data record cannot be accessed through this sector.

- EED_SECTOR_ACTIVE
  All the latest data records have been copied completely and the data accessing can target to this sector.

- EED_SECTOR_DEAD
  The sector is corrupted and cannot be used for emulation purposes.

Swapping stages: (assuming sector 1 is active sector and sector 6 is alternative sector)

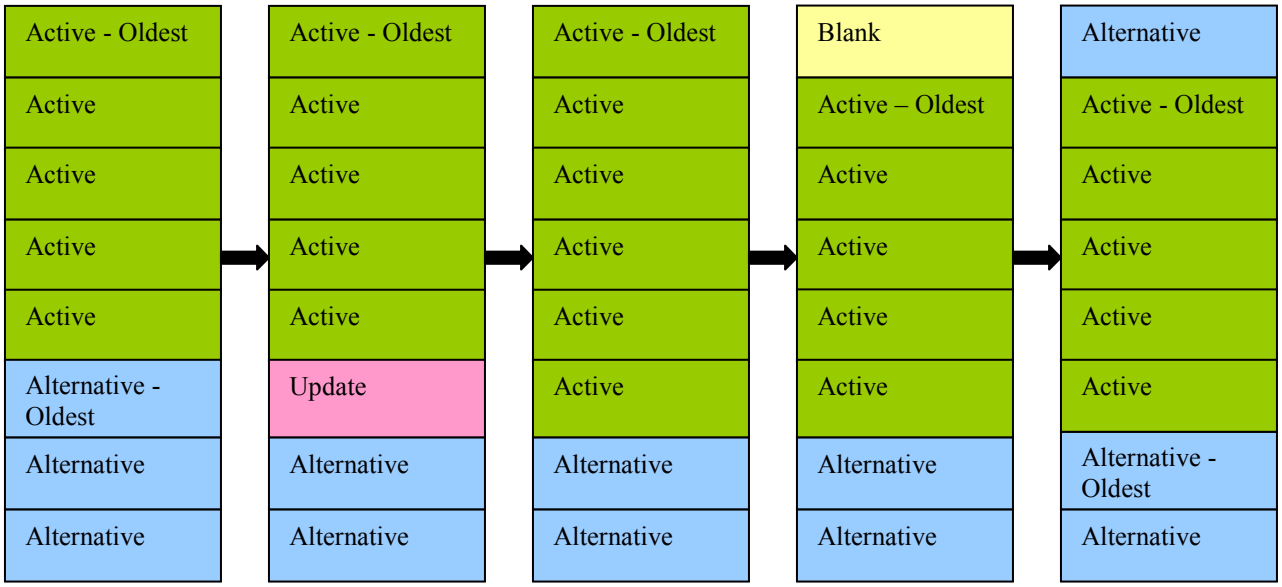

Figure 5 Sector Status

- Sector 1 is EED_SECTOR_ACTIVE and sector 6 is EED_SECTOR_ALTERNATIVE
  This is the initial state of sector swapping.

- Sector 1 is EED_SECTOR_ACTIVE and sector 6 is EED_SECTOR_UPDATE
  The data record copying is in progress. If swapping is failed in this stage, all the data record access will be directed to original active sector – sector 1. But sector 6 should be re-initialized as alternative one (EED_SECTOR_ALTERNATIVE state) and perform the swapping again.

- Sector 1 is EED_SECTOR_ACTIVE and sector 6 is also EED_SECTOR_ACTIVE.
  All the latest data records have been copied completely and the data accessing can target to sector 2. So if the swapping is failed in this stage, only need is to initialize the sector 1 as the alternative sector (EED_SECTOR_ALTERNATIVE state).

- Sector 1 EED_SECTOR_BLANK is blank and sector 6 is EED_SECTOR_ACTIVE. The sector 1 is fully erased but not ready for emulation. It is needed to continue initializing the sector as the alternative sector (EED_SECTOR_ALTERNATIVE state).

- Sector 1 EED_SECTOR_ALTERNATIVE is alternative and sector 6 is EED_SECTOR_ACTIVE. This is the finished state of sector swapping.

4.3 Read EEPROM Data

- Read routine should check whether the record ID being searched is within valid range (less than or equal EED_MAX_RECORD_NUMBER) or not. If the record ID is not in the range it should inform the user about it by returning the error EED_ERROR_IDRNG.

- There will be several data records in EEPROM with same data ID (because of data updating).Some of that data IDs will also be present in cache table, so reading routine should search that record ID in cache table.
- If that ID is found in cache table, it should retrieve the address of that record from cache table. Otherwise reading routine should identify the latest copy of data record by scanning the entire active sectors from the first data record to the blank region.
- If record ID is not found it should inform user about the same by returning the error EED_ERROR_NOFND.

.

4.4 Report EEPROM Status

- Report EEPROM routine should determine the status of current active sector.
- If status is other than active return an error EED_ERROR_SSTAT.
- If the sector is active store erasing cycles of the sector and return.

4.5 De-initialize EEPROM

- If the emulated EEPROM is not required, the Flash memory for EEPROM emulation should be released. The de-initialization routine will erase all the Flash memory used for emulation.
- The routine will use emulation start address EED_FLASH_START_ADDRESS and number of sectors allotted EED_SECTORS_ALLOTED.

4.6 Notes and Limitations

Attention should be paid to the following items before using EED:

1. User has to take care to disable the protection and security features before using the EED.
2. It is suggested NOT to use the middle level APIs or low level APIs of EED directly.
3. User has to give the COP feature through '*CallBack()'* function in case he wants to use the feature.
4. Report EEPROM status routine will return the erasing cycles of the sector. This number may not give the exact number of erasing cycles because once the sector status is not correct, the erasing cycles will be re-counted from 0.
5. EEPROM Emulation driver CANNOT be called in any interrupt service routine.
6. It is strongly recommended NOT to program or erase the same Flash location while using EED to operate it.

5 FUNCTIONAL HIERARCHY AND CALLING CONVENTION

EEPROM emulation driver (EED) provides three hierarchies of application programming interfaces (APIs): high level, middle level and low level APIs.

5.1 High level APIs (User level APIs):

These APIs provide direct operations on emulated EEPROM such as initialize EEPROM, read record, write record, report EEPROM status and de-initialize EEPROM.

- **FSL_InitEeprom** – Initialize the Flash memory used for EEPROM emulation

- **FSL_ReadEeprom** – Read the specific data record from emulated EEPROM
- **FSL_WriteEeprom** – Write a data record to emulated EEPROM
- **FSL_ReportEepromStatus** – Report the status of the emulated EEPROM
- **FSL_DeinitEeprom** – De-initialize the Flash memory used for EEPROM emulation
- **FSL_Main** – Completes the initialization of sectors and other operations to make sectors ready for EEPROM emulation

5.2 Middle level APIs:

These APIs provide some individual functionality to support the high level APIs on operating emulated EEPROM

- **FSL_Program** – Program the data into Flash memory.
- **FSL_CopyRecord** – Copy one data record to the Flash memory.
- **FSL_InitSector** – Initialize one sector, including erase this sector, blank check and update the erased cycles field of the sector.
- **FSL_SwapSector** – Copy the latest data records from the oldest active sector to the oldest alternative sector.
- **FSL_SearchLoop** – Loop across all the active sectors and search for the given record ID.
- **FSL_SearchRecord** – Search the required data record ID in a sector.
- **FSL_SectorStatus –** Return the status of the sector.
- **FSL_ReadRecord –** Reads the data and ID fields and the associated ECC and return the status of the record.
- **FSL_CheckMarginLevel –** Sets different margin levels and reads the data from a particular address, to detect partial programming errors, if any.
- **FSL_ReadDFlash** - Reads a D-Flash memory location by explicitly handling the page registers. (This function is required for the S12XS family only).
- **FSL_GetErasingCycles –** Read and returns the erasing cycles of a particular sector.
- **FSL_PgmSectorStatus –** Erase verifies an erased sector and initializes the sector as ACTIVE or ALTERNATIVE sector.
- **FSL_InitFirstTime –** Initializes all the during first time initialization.
- **FSL_UpdtCacheTable –** Updates the cache table if enabled.
- **FSL_ISRHandler –** Services the interrupt service routine once the command complete interrupt occurs after the erase operation.

5.3 Low level APIs

These APIs are basic Flash operations and composed of a subset of Standard Software Driver for S12XE SSD.

- **FlashInit** – This function will be used to set the clock divider during the initialization of flash for the EEPROM emulation.
- **DFlashErase** – launches the erase of a D-Flash sector.
- **DFlashEraseVerify –** verify if the specified destination is blanked
- **DFlashProgram** – program data into data Flash.
- **FlashProgramVerify** – verify the content of the destination with the source.
- **FlashCommandSequence** – This function launches the flash command that has been written into Flash Command Register by '*DFlashProgram()'* function and waits until the command finishes. This will be an internal function that shall only be called by SSD functions such as *'DFlashProgram*()', etc.

- **FlashSetUserMargin** – is used to set the user margin level for margin reads of partially programmed data.
- **FlashInterruptSet** – Disable or enable the Command complete interrupt flag.

5.4 Call Tree

Table 10 Call Tree

API Hierarchy	Function Name	Caller Functions	Functions to be called
High Level APIs	FSL_InitEeprom	User's Applications	FSL_SearchRecord(middle) FSL_SectorStatus(middle) FSL_CheckMarginLevel(middle) FSL_ReadDFlash(middle) FSL_GetErasingCycles(middle) FSL_InitFirstTime(middle) FSL_UpdtCacheTable(middle) FSL_Program(middle) FlashInit(low) DFlashErase(low) DFlashEraseVerify(low)
	FSL_ReadEeprom	User's Applications	FSL_SearchLoop(middle)
	FSL_WriteEeprom	User's Applications	FSL_SwapSector(middle) FSL_CopyRecord(middle) FSL_SectorStatus(middle)
	FSL_ReportEepromStatus	User's Applications	FSL_SectorStatus(middle) FSL_ReadDFlash(middle)
	FSL_DeinitEeprom	User's Applications	DFlashErase(low)
	FSL_Main	User's Applications	FSL_SwapSector(middle) FSL_PgmSectorStatus(middle) FSL_InitFirstTime(middle) FSL_UpdtCacheTable(middle) FSL_CopyRecord(middle) FSL_UpdtCacheTable(middle) DFlashErase(low) DFlashEraseVerify(low)
Middle Level APIs	FSL_Program	FSL_InitEeprom(high) FSL_CopyRecord (middle)	DFlashProgram(low) FlashPorgramVerify(low)
	FSL_CopyRecord	FSL_WriteEeprom (high) FSL_SwapSector(middle)	FSL_Program (middle)
	FSL_SwapSector	FSL_WriteEeprom(high) FSL_Main(high)	FSL_SearchLoop(middle) FSL_CopyRecord(middle) FSL_ReadRecord(middle) FSL_GetErasingCycles(middle) FSL_PgmSectorStatus(middle) FSL_ReadDFlash(middle) DFlashErase(low)
	FSL_SearchRecord	FSL_SearchLoop(middle) FSL_InitEeprom(high)	FSL_ReadRecord(middle) FSL_ReadDFlash(middle)

	FSL_SectorStatus	FSL_InitEeprom(high) FSL_WriteEeprom(high) FSL_ReportEepromStatus–(high)	FSL_ReadRecord(middle)
	FSL_SearchLoop	FSL_ReadEeprom(high) FSL_SwapSector(middle)	FSL_SearhRecord(middle) FSL_ReadDFlash(middle)
	FSL_ReadRecord	FSL_SectorStatus(middle) FSL_SwapSector(middle) FSL_SearchRecord (middle)	-
	FSL_CheckMarginLevel	FSL_InitEeprom(high)	FlashSetUserMargin(low)
	FSL_ReadDFlash(for S12XS only)	FSL_InitEeprom(high) FSL_ReportEepromStatus (high) FSL_SwapSector(middle) FSL_SearchLoop(middle)	-
	FSL_GetErasingCycles	FSL_InitEeprom(high) FSL_SwapSector(middle)	-
	FSL_PgmSectorStatus	FSL_Main(high) FSL_SwapSector(middle) FSL_InitFirstTime(middle)	DFlashEraseVerify(low) FSL_Program(middle)
	FSL_InitFirstTime	FSL_InitEeprom(high) FSL_Main(high)	FSL_PgmSectorStatus(middle)
	FSL_UpdtCacheTable	FSL_InitEeprom (high) FSL_Main(high)	FSL_SearhRecord(middle) FSL_ReadDFlash (middle)
	FSL_ISRHandler	-	FlashInterruptSet(low)
Low Level APIs	FlashInit	FSL_InitEeprom(high)	-
	DFlashErase	FSL_InitEeprom(high) FSL_DeinitEeprom(high) FSL_SwapSector(middle) FSL_Main(high)	FlashInterruptSet(low)
	DFlashEraseVerify	FSL_PgmSectorStatus(middle) FSL_Main(high)	-
	DFlashProgram	FSL_Program (middle)	-
	FlashProgramVerify	FSL_Program (middle)	
	FlashCommandSequence	DFlashErase(low) DFlashEraseVerify(low) DFlashProgram(low) FlashSetUserMargin(low)	-
	FlashSetUserMargin	FSL_CheckMarginLevel (middle)	FlashCommandSequence(low)
	FlashInterruptSet	FSL_ISRHandler(middle) DFlashErase(low)	-

6 API SPECIFICATIONS

6.1 High Level API

6.1.1 FSL_InitEeprom

6.1.1.1 Overview

'FSL_InitEeprom()' will perform the flash module clock initialization. This function will also determine active, alternative and brown out affected sectors and erase/update the sectors. Initializing variables that hold active sector related information like the start address of the current active sector and the blank space available is also done in this function. Cache table is also initialized in this function. If no sectors are initialized then, this function shall initialize all the sectors in a round robin queue. If during initialization, erasing of a sector is necessary as part of the brown-out handling, then the function launches the erase operation and returns. The initialization is then completed by the *'FSL_Main()'*.

6.1.1.2 Prototype

```
UINT16 FSL_InitEeprom ()
```

6.1.1.3 Arguments

None

6.1.1.4 Return Values

The return values of this function include those that are escalated from the functions that it calls. The possible return values of this function are:

Table 11 Return Values for FSL_InitEeprom()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Source and Destination data mismatch	EED_ERROR_VERIFY
	clock setting failed.	EED_ERROR_INVALIDCLK

6.1.1.5 Troubleshooting

Table 12 Troubleshooting for FSL_InitEeprom()

Returned Error Bits	Description	Solution
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Refer Appendix A
EED_ERROR_INVALIDCLK	Error in clock setting.	Refer Appendix A

6.1.1.6 Global Variables used

- *gInitFirstTime* – BOOL varaiable which specifies the first time initialization,
- *gActiveCount* – number of active sectors initialized,
- *gAltCount* – number of alterntive sectors initialized,
- *gAddressToMain* – address of the sector on which operations need to be done by *'FSL_Main()'*,
- *gEraseCycles* – stores erase cycles before erasing the cluster,
- *gJobToMain* – sets the job to be done by *'FSL_Main'* function,
- *currentActiveSector* – start address of Current Active sector,
- *freeSpaceAddress* – address of the blank space in the current active sector ,
- *EE_Status* – flag to keep track of the status of the emulated Eeprom,
- CallBack function pointer

6.1.1.7 Comments

None.

6.1.1.8 Assumptions

The user application should call the *'FSL_Main()'* while the *EE_Status* flag is *BUSY*.

6.1.2 FSL_ReadEeprom

6.1.2.1 Overview

This function is to read the specific data record. The starting address of the record data will be returned.

6.1.2.2 Prototype

```
UINT16 FSL_ReadEeprom (UINT16 dataID, UINT16 *recordAddr)
```

6.1.2.3 Arguments

Table 13 Arguments for FSL_ReadEeprom()

Argument	Type	Description
dataID	UINT16	The ID whose data value is required should be with in the range; from 1 to less than or equal EED_MAX_RECORD_NUMBER.
recordAddr	UINT16*	The starting address of the data record if found

6.1.2.4 Return Values

The possible return values of this function are:

Table 14 Return Values for FSL_ReadEeprom()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK

	record not found	EED_ERROR_NOFND
	Record identifier exceeds the valid range	EED_ERROR_IDRNG

6.1.2.5 Troubleshooting

Table 15 Troubleshooting for FSL_ReadEeprom()

Returned Error Bits	Description	Solution
EED_ERROR_NOFND	Record ID not found.	Refer Appendix A
EED_ERROR_IDRNG	Record ID not in range.	Refer Appendix A

6.1.2.6 Global variables used

None

6.1.2.7 Comments

The function does not return the data to be read. It saves the address of the data to be read in the output variable *'recordAddr'*.

6.1.2.8 Assumptions

None

6.1.3 FSL_WriteEeprom

6.1.3.1 Overview

This function will be used to encapsulate the user data in a record and write it to the emulated EEPROM. If there is no enough free space in active sector, this routine shall check if the next sector available is an 'active sector'. Otherwise, this routine will initiate sector swapping to clean up the EEPROM. If a swap is initiated, then the swap will launch the erase of the old sector and return. The completion of the swap process and writing the record that caused the swap will be done by the *'FSL_Main()'*.

6.1.3.2 Prototype

```
UINT16 FSL_WriteEeprom (UINT16 dataID, UINT16 dataValue)
```

6.1.3.3 Arguments

Table 16 Arguments for FSL_WriteEeprom()

Argument	Type	Description
dataID	UINT16	The ID should be with in the range; from 1 to less than or equal to EED_MAX_RECORD_NUMBER.
dataValue	UINT16	The data to be stored

6.1.3.4 Return Values

The return values of this function include those that are escalated from the functions that it calls. The possible return values of this function are:

Table 17 Return Values for FSL_WriteEeprom()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Record identifier exceeds the valid range	EED_ERROR_IDRNG
	Source data and the destination data just written have a mismatch	EED_ERROR_VERIFY

6.1.3.5 Troubleshooting

Table 18 Troubleshooting for FSL_WriteEeprom()

Returned Error Bits	Description	Solution
EED_ERROR_IDRNG	Record ID not in range.	Refer Appendix A
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Refer Appendix A

6.1.3.6 Global Parameters Referred

- *gDataRecord* – record used in copying data between *'FSL_WriteEeprom()'* and *'FSL_Main()'* ,
- *currentActiveSector* – start address of Current Active sector,
- *freeSpaceAddress* – address of the blank space in the current active sector ,
- CallBack function pointer.

6.1.3.7 Comments

None

6.1.3.8 Assumptions

The user application should call the *'FSL_Main()'* while the *EE_Status* flag is *BUSY.*

6.1.4 FSL_ReportEepromStatus

6.1.4.1 Overview

This function reports statistics like active emulation sector erasure cycles, check the emulation sector status.

6.1.4.2 Prototype

```
UINT16 FSL_ReportEepromStatus(UINT16 *erasingCycles)
```

6.1.4.3 Arguments

Table 19 Arguments for FSL_ ReportEepromStatus()

Argument	Type	Description
erasingCycles	UINT16*	The erasing cycles of the current active sector

6.1.4.4 Return Values

The possible return values of this function are:

Table 20 Return Values for FSL_ReportEepromStatus()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Sector Status error	EED_ERROR_SSTAT

6.1.4.5 Troubleshooting

Table 21 Troubleshooting for FSL_ ReportEepromStatus()

Returned Error Bits	Description	Solution
EED_ERROR_SSTAT	Sector Status error.	Refer Appendix A

6.1.4.6 Global Parameters Referred

- *currentActiveSector* – start address of Current Active sector

6.1.4.7 Comments

Value of erasing cycles is valid only when the return code is not EED_ERROR_SSTAT

Erasure Cycles is counted only after initialization of EEPROM. When it is de-initialized, the erasure cycles count is lost. So if the sector is re-initialized, the erasure cycle will start from 1.

6.1.4.8 Assumptions

None

6.1.5 FSL_DeinitEeprom

6.1.5.1 Overview

This function is to release all the Flash used to EEPROM emulation. After de-initialized, the Flash pages for emulation will be fully erased. This function launches the erase operation on the first D-Flash sector used for emulation and sets parameter so that the *'FSL_Main()'* can erase the remaining sector used for emulation.

6.1.5.2 Prototype

```
UINT16 FSL_DeinitEeprom ()
```

6.1.5.3 Arguments

None

6.1.5.4 Return Values

The return values of this function include those that are escalated from the functions that it calls. The possible return values of this function are:

Table 22 Return Values for FSL_DeinitEeprom()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	The flash memories are not blank	EED_ERROR_NOT_BLANK

6.1.5.5 Troubleshooting

Table 23 Troubleshooting for FSL_DeinitEeprom()

Returned Error Bits	Description	Solution
EED_ERROR_NOT_BLANK	The flash memories are not blank.	Refer Appendix A

6.1.5.6 Global Parameters Referred

- *gJobToMain – sets the job to be done by 'FSL_Main' function,*
- *EE_Status – flag to keep track of the status of the emulated Eeprom.*

6.1.5.7 Comments

Only the Flash pages which are used for EEPROM Emulation will be erased

6.1.5.8 Assumptions

The user application should call the *'FSL_Main()'* while the *EE_Status* flag is *BUSY*.

6.1.6 FSL_Main

6.1.6.1 Overview

This function completes the initialization of erased sectors and performs other operations to make the sectors ready for EEPROM emulation, after an erase is initiated by *'FSL_InitEeprom*()', '*FSL_WriteEeprom()'* or *'FSL_DeinitEeprom()'* APIs.

6.1.6.2 Prototype

```
void FSL_Main ()
```

6.1.6.3 Arguments

None

6.1.6.4 Return Values

None

6.1.6.5 Troubleshooting

None

6.1.6.6 Global Parameters Referred

- *gJobToMain* – sets the job to be done by *'FSL_Main'* function,
- *gAddressToMain* – address of the sector on which operations need to be done by *'FSL_Main()',*
- *gEE_Error* – global variable to keep track of error code,
- *EE_Status* – flag to keep track of the status of the emulated Eeprom,
- *eraseStatusFlag* – flag to keep track of Erase State,
- *gEraseCycles* – stores erase cycles before erasing the cluster,
- *gInitFirstTime* – BOOL varaiable which specifies the first time initialization,
- *gDataRecord* – record used in copying data between *'FSL_WriteEeprom()'* and *'FSL_Main()'* ,
- *currentActiveSector* – start address of Current Active sector,
- *freeSpaceAddress – address of the blank space in the current active sector*

6.1.6.7 Comments

Only the Flash pages which are used for EEPROM Emulation will be erased

6.1.6.8 Assumptions

The user application should call the *'FSL_Main()'* while the *EE_Status* flag is *BUSY.*

6.2 Middle Level API

6.2.1 FSL_Program

6.2.1.1 Overview

'FSL_Program()' will program specified data to destination address in Flash and verify the data. It encapsulates two low-level SSD functions: *'DFlashProgram()'* and *'FlashProgramVerify()'*. Input parameters are used directly without any check.

6.2.1.2 Prototype

```
UINT16 FSL_Program (UINT16 destination, UINT16 size, UINT16
*SSD_SGF18_NEAR srcPtr)
```

6.2.1.3 Arguments

Table 24 Arguments for FSL_Program()

Argument	Type	Description
destination	UINT16	Start address for the D flash Program operation.
Size	UINT16	Size to be programmed in bytes.
srcPtr	UINT16*	Pointer to source data

6.2.1.4 Return Values

The return values of this function include those that are escalated from the functions that it calls. The possible return values of this function are:

Table 25 Return Values for FSL_Program()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Source data and the corresponding destination data do not match	EED_ERROR_VERIFY

6.2.1.5 Troubleshooting

Table 26 Troubleshooting for FSL_Program()

Returned Error Bits	Description	Solution
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Refer Appendix A

6.2.1.6 Global Parameters Referred

- CallBack function pointer

6.2.1.7 Comments

None

6.2.1.8 Assumptions

None

6.2.2 FSL_CopyRecord

6.2.2.1 Overview

'FSL_CopyRecord()' writes user data into Flash in a record format.

6.2.2.2 Prototype

```
UINT16 FSL_CopyRecord(DATA_RECORD dataRecord, UINT16 destination)
```

6.2.2.3 Argument

Table 27 Arguments for FSL_CopyRecord()

Argument	Type	Description
dataRecord	DATA_RECORD	A structure containing ID and a pointer to data fields of a data record
destination	UINT16	Address where the record is to be written.

6.2.2.4 Return Values

The return values of this function include those that are escalated from the functions that it calls. The possible return values of this function are:

Table 28 Return Values for FSL_CopyRecord()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Source data and the corresponding destination data do not match	EED_ERROR_VERIFY

6.2.2.5 Troubleshooting

Table 29 Troubleshooting for FSL_CopyRecord()

Returned Error Bits	Description	Solution
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Refer Appendix A

6.2.2.6 Global Parameters Referred

- CallBack function pointer

6.2.2.7 Comments

None.

6.2.2.8 Assumptions

None

6.2.3 FSL_SwapSector

6.2.3.1 Overview

This function copies the latest valid records from the active sector to the blank alternative sector. It initializes the updated sector as ACTIVE sector. It launches the erase operation on the oldest Active sector and returns. The *'FSL_Main()'* function initializes the oldest ACTIVE sector as ALTERNATIVE and writes the record that initiated the swap. If the updated ACTIVE sector is full it does another swap.

6.2.3.2 Prototype

```
UINT16 FSL_SwapSector()
```

6.2.3.3 Argument

None

6.2.3.4 Return Values

The return values of this function include those that are escalated from the functions that it calls.

Table 30 Return Values for FSL_SwapSector()

Size	Description	Possible Values
16 bit	Successful Initialization.	EED_OK
	Source data and the corresponding destination data do not match	EED_ERROR_VERIFY

6.2.3.5 Troubleshooting

Table 31 Troubleshooting for FSL_SwapSector()

Returned Error Bits	Description	Solution
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Refer Appendix A

6.2.3.6 Global Parameters Referred

- *gEraseCycles* – stores erase cycles before erasing the cluster,
- *gJobToMain* – sets the job to be done by 'FSL_Main' function,
- *gAddressToMain* – address of the sector on which operations need to be done by *'FSL_Main()'*,
- *currentActiveSector* – start address of Current Active sector,

- *freeSpaceAddress* – address of the blank space in the current active sector ,
- *EE_Status* – flag to keep track of the status of the emulated Eeprom,
- CallBack function pointer.

6.2.3.7 Comments

None

6.2.3.8 Assumptions

None

6.2.4 FSL_SearchRecord

6.2.4.1 Overview

'FSL_SearchRecord()' searches record with a specific record ID in a sector. This function will parse the whole sector to find out the latest copy of data records.

6.2.4.2 Prototype

```
UINT16 FSL_SearchRecord (UINT16 dataID, UINT16 sectorAddress, UINT16
*SSD_SGF18_NEAR recordAddress)
```

6.2.4.3 Argument

Table 32 Arguments for FSL_SearchRecord()

Argument	Type	Description
recID	UINT16	The record ID to be found
sectorAddress	UINT16	Address of the sector to be searched.
recordAddress	UINT16 *	Pointer to the address of the record found

6.2.4.4 Return Values

Table 33 Return Values for FSL_SearchRecord()

Size	Description	Possible Values
16 bit	Record ID found in the sector	EED_OK
	Record ID not found in the sector	EED_ERROR_NOFND

6.2.4.5 Troubleshooting

Table 34 Troubleshooting for FSL_SearchRecord()

Returned Error Bits	Description	Solution
EED_ERROR_NOFND	Record ID not found.	Refer Appendix A

6.2.4.6 Global Parameters Referred

- CallBack function pointer

6.2.4.7 Comments

None

6.2.4.8 Assumptions

None

6.2.5 FSL_SearchLoop

6.2.5.1 Overview

'FSL_SearchLoop()' searches record with a specific record ID in all active sector.

6.2.5.2 Prototype

```
UINT16 FSL_SearchLoop (UINT16 dataID, UINT16 *SSD_SGF18_NEAR
recordAddress)
```

6.2.5.3 Argument

Table 35 Arguments for FSL_SearchLoop()

Argument	Type	Description
recID	UINt16	The record ID to be searched
recordAddress	UINT16 *	Pointer to the address of the record found

6.2.5.4 Return Values

The possible return values of this function are:

Table 36 Return Values for FSL_SearchLoop()

Size	Description	Possible Values
16 bit	Record ID found.	EED_OK
	Record ID not found	EED_ERROR_NOFND

6.2.5.5 Troubleshooting

Table 37 Troubleshooting for FSL_SearchLoop()

Returned Error Bits	Description	Solution
EED_ERROR_NOFND	Record ID not found.	Refer Appendix A

6.2.5.6 Global Parameters Referred

- *currentActiveSector* – start address of Current Active sector
- CallBack function pointer

6.2.5.7 Comments

None

6.2.5.8 Assumptions

None

6.2.6 FSL_ReadRecord

6.2.6.1 Overview

This function is used to read the data and ID fields and the associated ECC and return the status of the record.

6.2.6.2 Prototype

```
UINT16 FSL_ReadRecord (UINT16 *SSD_SGF18_NEAR dataID, UINT16
readAddress)
```

6.2.6.3 Argument

Table 38 Arguments for FSL_ReadRecord()

Argument	Type	Description
dataID	UINT16*	Pointer to the dataID
readAddress	UINT16	Address of the record to be read

6.2.6.4 Return Value

Table 39 Return Values for FSL_ReadRecord()

Size	Description	Possible Values
16 bit	The record is valid.	EED_RECORD_VALID
	The record is invalid.	EED_RECORD_INVALID
	The record is blank	EED_RECORD_BLANK

6.2.6.5 Troubleshooting

None

6.2.6.6 Global Parameters Referred

- CallBack function pointer

6.2.6.7 Comments

None

6.2.6.8 Assumptions

None

6.2.7 FSL_SectorStatus

6.2.7.1 Overview

This function returns the status of the sector.

6.2.7.2 Prototype

```
UINT16 FSL_SectorStatus (UINT16 sectorAddress)
```

6.2.7.3 Argument

Table 40 Arguments for FSL_SectorStatus()

Argument	Type	Description
sectorAddress	UINT16	Start address of the sector

6.2.7.4 Return Values

The possible return values of this function are:

Table 41 Return Values for FSL_SectorStatus()

Size	Description	Possible Values
16 bit	The sector is active.	EED_SECTOR_ACTIVE
	The sector is alternative	EED_SECTOR_ALTERNATIVE
	The sector is blank	EED_SECTOR_BLANK
	The sector is under updation	EED_SECTOR_UPDATE
	The sector is invalid	EED_SECTOR_INVALID

6.2.7.5 Troubleshooting

None

6.2.7.6 Global Parameters Referred

- CallBack function pointer

6.2.7.7 Comments

None

6.2.7.8 Assumptions

None

6.2.8 FSL_CheckMarginLevel

6.2.8.1 Overview

This function checks a memory location for partial programming errors. It reads the data at the address being checked. It should set the margin level of the D-Flash sector to different levels and then, read the location again for the different margin levels. If the data read is not the same as before then the function returns with an error code**.**

6.2.8.2 Prototype

```
UINT16 FSL_CheckMarginLevel (UINT16 readAddress)
```

6.2.8.3 Argument

Table 42 Arguments for FSL_CheckMarginLevel ()

Argument	Type	Description
readAddress	UINT16	The Address which has to be checked for partial programming errors.

6.2.8.4 Return Values

The possible return values of this function are:

Table 43 Return Values for FSL_CheckMarginLevel ()

Size	Description	Possible Values
16 bit	No partial programming error at the address passed.	EED_OK
	There is a partial programming error at the address passed.	EED_NOT_OK

6.2.8.5 Troubleshooting

None

6.2.8.6 Global Parameters Referred

- CallBack function pointer

6.2.8.7 Comments

None

6.2.8.8 Assumptions

None

6.2.9 FSL_ReadDFlash

6.2.9.1 Overview

This function read a D-Flash location by calculating the logical address of the global address passed to the function. It saves the value of the EPAGE register and manipulates it to read the D-Flash address and then restores the original value. This function is used by S12XS derivatives only. It is a conditionally compiled code**.**

6.2.9.2 Prototype

```
UINT16 FSL_ReadDFlash (UINT16 destination)
```

6.2.9.3 Argument

Table 44 Arguments for FSL_ReadDFlash ()

Argument	Type	Description
destination	UINT16	The D-Flash memory location which has to be read.

6.2.9.4 Return Values

The possible return values of this function are:

Table 45 Return Values for FSL_ReadDFlash ()

Size	Description	Possible Values
16 bit	The data at the D-Flash memory location which that was read	Any 16-bit value.

6.2.9.5 Troubleshooting

None

6.2.9.6 Global Parameters Referred

- CallBack function pointer

6.2.9.7 Comments

None

6.2.9.8 Assumptions

None

6.2.10 FSL_GetErasingCycles

6.2.10.1 Overview

This function reads and returns the erasing cycles of a particular sector. If the read causes a double bit fault, it sets the error flag.

6.2.10.2 Prototype

```
UINT16 FSL_GetErasingCycles (UINT16 sectorAddress, BOOL*
pErrorGettingEC)
```

6.2.10.3 Argument

Table 46 Arguments for FSL_GetErasingCycles()

Argument	Type	Description

sectorAddress	UINT16	Address of the sector whose erasing cycles are required.
pErrorGetting EC	BOOL	Set the flag if a double bit fault occurs while reading the erasing cycles. If the flag is set the value returned is ignored.

6.2.10.4 Return Values

The possible return values of this function are:

Table 47 Return Values for FSL_GetErasingCycles ()

Size	Description	Possible Values
16 bit	The erasing cycles of the sector.	Range from 0 to 0xFFFE

6.2.10.5 Troubleshooting

None

6.2.10.6 Global Parameters Referred

None

6.2.10.7 Comments

None

6.2.10.8 Assumptions

None

6.2.11 FSL_PgmSectorStatus

6.2.11.1 Overview

This function erase verifies an erased sector and initializes sector as ACTIVE or ALTERNATIVE.

6.2.11.2 Prototype

```
UINT16 FSL_PgmSectorStatus (UINT16 sectorAddress, UINT16 status, UINT16
erasingCycles, BOOL ersVerify)
```

6.2.11.3 Argument

Table 48 Arguments for FSL_ PgmSectorStatus()

Argument	Type	Description
sectorAddress	UINT16	Address of the sector to be programmed.
Status	UINT16	Status to be initialized.
erasingCycles	UINT16	This is the erasing cycles of the sector that is to be programmed.
ersVerify	BOOL	Specifies erase verify is required or not.

6.2.11.4 Return Values

The possible return values of this function are:

Table 49 Return Values for FSL_PgmSectorStatus ()

Size	Description	Possible Values
UINT16	Successful Initialization.	EED_OK
	Source data and the corresponding destination data do not match	EED_ERROR_VERIFY

6.2.11.5 Troubleshooting

None

6.2.11.6 Global Parameters Referred

- *gInitFirstTime* – BOOL varaiable which indicates first time initialization.

6.2.11.7 Comments

None

6.2.11.8 Assumptions

None

6.2.12 FSL_InitFirstTime

6.2.12.1 Overview

This function will initialize the sectors used for EEPROM emulation for the first time. If a brown-out occurred during the first time initialization of the sectors, it initializes the remaining sectors.

6.2.12.2 Prototype

```
UINT16 FSL_InitFirstTime()
```

6.2.12.3 Argument

None

6.2.12.4 Return Values

The possible return values of this function are:

Table 50 Return Values for FSL_InitFirstTime()

Size	Description	Possible Values
UINT16	Successful Initialization.	EED_OK
	Source data and the corresponding destination data do not match	EED_ERROR_VERIFY

6.2.12.5 Troubleshooting

None

6.2.12.6 Global Parameters Referred

- *gActiveCount* – number of active sectors initialized
- *gAltCount* – number of alterntive sectors initialized
- *gAddressToMain* – address of the sector on which operations need to be done by *'FSL_Main()'*

6.2.12.7 Comments

None

6.2.12.8 Assumptions

None

6.2.13 FSL_UpdtCacheTable

6.2.13.1 Overview

This function will initialize the cache table entries. This function is called only if the Cache Table in enabled.

6.2.13.2 Prototype

```
void FSL_UpdtCacheTable()
```

6.2.13.3 Argument

None

6.2.13.4 Return Values

None

6.2.13.5 Troubleshooting

None

6.2.13.6 Global Parameters Referred

- *currentActiveSector* – start address of Current Active sector

6.2.13.7 Comments

None

6.2.13.8 Assumptions

None

6.2.14 FSL_ISRHandler

6.2.14.1 Overview

This function will be invoked when the command complete interrupt flag is set after erasing a sector. The interrupt is not invoked for any other flash operations like program or erase verify. This function will check if the erase was done successfully. If not, it sets the error code in the global error variable and erase status flag as FAIL.

6.2.14.2 Prototype

```
void SSD_SGF18_NEAR FSL_ISRHandler()
```

6.2.14.3 Argument

None

6.2.14.4 Return Values

If any error occurred during the erase operation, the error code is set in a global error variable *gEE_Error.*

6.2.14.5 Troubleshooting

None

6.2.14.6 Global Parameters Referred

- *gEE_Error* – If an EED operation fails it will contain the error code.
- *eraseStatusFlag* – flag to keep track of Erase State.

6.2.14.7 Comments

None

6.2.14.8 Assumptions

None

6.3 Low Level API

6.3.1 FlashInit()

6.3.1.1 Overview

This function checks and initializes Flash module for the other Flash APIs. It will check the enabling state of Flash module, initialize the Flash clock, and initialize Flash status register. SGF_OK will be returned if everything is OK. For other cases a bit-mapped return code combined with the flash status will be returned to user application.

6.3.1.2 Prototype

```
UINT16 FlashInit()
```

6.3.1.3 Arguments

None

6.3.1.4 Return Values

Table 51 Return Values for FlashInit ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value.	SGF_ERR_INVALIDCLK

6.3.1.5 Troubleshooting

Table 52 Troubleshooting for FlashInit ()

Returned Error Bits	Description	Solution
SGF_ERR_INVALIDCLK	Error in clock setting.	Refer Appendix A

6.3.1.6 Comments

The '*FlashInit*()' will be synchronous in behavior and it will not support re-entrancy.

6.3.1.7 Assumptions

None

6.3.2 DFlashErase

6.3.2.1 Overview

This function is used to perform sector erase operation on D flash. It launches the erase operation on a sector, then enable the command complete interrupt and returns.

6.3.2.2 Prototype

```
UINT16 DFlashErase(UINT16 destination)
```

6.3.2.3 Arguments

Table 53 Arguments for DFlashErase ()

Argument	Type	Description
destination	UINT16	Start address for the D flash erase operation.

6.3.2.4 Return Values

Table 54 Return Values for DFlashErase ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_RANGE

6.3.2.5 Troubleshooting

Table 55 Troubleshooting for DFlashErase ()

Returned Error Bits	Description	Solution
SGF_ERR_RANGE	Address exceeds the range.	Refer Appendix A

6.3.2.6 Global Variables used

- Command Array
- Erase Status Flag

6.3.2.7 Comments

The '*DFlashErase*()' will be synchronous in behavior and it will not support re-entrancy.

6.3.2.8 Assumptions

None

6.3.3 DFlashEraseVerify ()

6.3.3.1 Overview

1. This function is used to perform erase verify operation on entire D flash block or sections (multiple of words) of D Flash.

6.3.3.2 Prototype

```
UINT16 DFlashEraseVerify(UINT16 destination, UINT16 size)
```

6.3.3.3 Arguments

Table 56 Arguments for DFlashEraseVerify ()

Argument	Type	Description
Destination	UINT16	Start address for the D flash erase verify operation.
Size	UINT16	Size to be verified in bytes.

6.3.3.4 Return Values

Table 57 Return Values for DFlashEraseVerify ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_ACCERR SGF_ERR_SIZE SGF_ERR_MGSTAT0 SGF_ERR_MGSTAT1

6.3.3.5 Troubleshooting

Table 58 Troubleshooting for DFlashEraseVerify ()

Returned Error Bits	Description	Solution
SGF_ERR_ACCERR	Access error flag is set while operating the Flash.	Refer Appendix A
SGF_ERR_SIZE	Error due to size mismatch	Refer Appendix A
SGF_ERR_MGSTAT0	Hardware Error	Refer Appendix A
SGF_ERR_MGSTAT1	Hardware Error	Refer Appendix A

6.3.3.6 Global Variables used

- Command Array

6.3.3.7 Comments

None

6.3.3.8 Assumptions

None

6.3.4 DFlashProgram ()

6.3.4.1 Overview

This function is used to perform program operation on D flash.

6.3.4.2 Prototype

```
UINT16 DFlashProgram(UINT16 destination, UINT16 size, UINT16
*SSD_SGF18_NEAR source)
```

6.3.4.3 Arguments

Table 59 Arguments for DFlashProgram ()

Argument	Type	Description
destination	UINT16	Start address for the D flash program operation.
size	UINT16	Size to be programmed in bytes.

source	UINT16 *	Pointer to the address of the data to be programmed.

6.3.4.4 Return Values

The return value is an 8-bit value in accumulator. Each bit refers to a specific error. Refer to Table 74 Return Code for details of the bits. The possible bit positions that can be set while returning from *'FlashErase()'* are:

Table 60 Return Values for DFlashProgram ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_ACCERR SGF_ERR_SIZE SGF_ERR_RANGE SGF_ERR_MGSTAT0 SGF_ERR_MGSTAT1

6.3.4.5 Troubleshooting

Table 61 Troubleshooting for DFlashProgram ()

Returned Error Bits	Description	Solution
SGF_ERR_ACCERR	Access error flag is set while operating the Flash.	Refer Appendix A
SGF_ERR_SIZE	Error due to size mismatch	Refer Appendix A
SGF_ERR_RANGE	Address exceeds the range.	Refer Appendix A
SGF_ERR_MGSTAT0	Hardware Error	Refer Appendix A
SGF_ERR_MGSTAT1	Hardware Error	Refer Appendix A

6.3.4.6 Global Variables used

- Command Array

6.3.4.7 Comments

The '*DFlashProgram*()' will be synchronous in behavior and it will not support re-entrancy.

6.3.4.8 Assumptions

None

6.3.5 FlashProgramVerify ()

6.3.5.1 Overview

This function is used to verify programmed data serially.

6.3.5.2 Prototype

```
UINT16 FlashProgramVerify(UINT16 destination, UINT16 size, UINT16
*SSD_SGF18_NEAR source)
```

6.3.5.3 Arguments

Table 62 Arguments for FlashProgramverify ()

Argument	Type	Description
destination	UINT16	Start address of the Flash range to be verified
size	UINT16	Size in byte of the Flash range to be verified
source	UINT16 *	Pointer to the Source data buffer

6.3.5.4 Return Values

Table 63 Return Values for FlashProgramVerify ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_ADDR SGF_ERR_SIZE SGF_ERR_RANGE SGF_ERR_DATAMISMATCH

6.3.5.5 Troubleshooting

Table 64 Troubleshooting for FlashProgramVerify ()

Returned Error Bits	Description	Solution
SGF_ERR_ADDR	Error due address mismatch.	Refer Appendix A
SGF_ERR_SIZE	Error due to size mismatch.	Refer Appendix A
SGF_ERR_RANGE	Address exceeds the range.	Refer Appendix A
SGF_ERR_DATAMISMATCH	Error due to mismatch in source and destination data.	Refer Appendix A

6.3.5.6 Global Variables used

- CallBack function pointer

6.3.5.7 Comments

The '*FlashProgramVerify()'* will be synchronous in behavior and it will not support re-entrancy. The program verification will be done in word (2 bytes at a time).

6.3.5.8 Assumptions

None

6.3.6 FlashCommandSequence ()

6.3.6.1 Overview

This function is used to perform command write sequence on the flash blocks.

6.3.6.2 Prototype

```
UINT16 FlashCommandSequence(UINT8 index)
```

6.3.6.3 Arguments

Table 65 Arguments for FlashCommandSequence ()

Argument	Type	Description
index	UINT8	Maximum index value for the FCCOBIX register for a particular command

6.3.6.4 Return Values

Table 66 Return Values for FlashCommandSequence ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_ACCERR SGF_ERR_PVIOL SGF_ERR_MGSTAT1 SGF_ERR_MGSTAT0

6.3.6.5 Troubleshooting

Table 67 Troubleshooting for FlashCommandSequence ()

Returned Error Bits	Description	Solution
SGF_ERR_ACCERR	Access error flag is set while operating the Flash.	Refer Appendix A
SGF_ERR_PVIOL	Protection violation flag is set while operating the Flash.	Refer Appendix A
SGF_ERR_MGSTAT1	Hardware Error	Refer Appendix A
SGF_ERR_MGSTAT0	Hardware Error	Refer Appendix A

6.3.6.6 Global Variables used

- Command Array
- CallBack function pointer

6.3.6.7 Comments

The '*FlashCommandSequence*()' is an internal function of the SSD. User should not call this API from the application.

6.3.6.8 Assumptions

None

6.3.7 FlashSetUserMargin ()

6.3.7.1 Overview

This function is used to set the user margin level.

6.3.7.2 Prototype

```
UINT16 FlashSetUserMargin(UINT16 marginValue)
```

6.3.7.3 Arguments

Table 68 Arguments for FlashSetUserMargin ()

Argument	Type	Description
marginValue	UINT16	The value of the margin to be set in the FlashSetUserMargin

6.3.7.4 Return Values

Table 69 Return Values for FlashSetUserMargin ()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK
	Error value	SGF_ERR_ACCERR

6.3.7.5 Troubleshooting

Table 70 Troubleshooting for FlashSetUserMargin ()

Returned Error Bits	Description	Solution
SGF_ERR_ACCERR	Access error flag is set while operating the Flash.	Refer Appendix A

6.3.7.6 Global Variables used

- Command Array

6.3.7.7 Comments

The *'FlashSetUserMargin()'* will be synchronous in behavior and it will not support re-entrancy.

6.3.7.8 Assumptions

None

6.3.8 FlashInterruptSet()

6.3.8.1 Overview

This function is used to set the user margin level.

6.3.8.2 Prototype

```
UINT16 FlashInterruptSet(UINT8 InterruptState)
```

6.3.8.3 Arguments

Table 71 Arguments for FlashInterruptSet()

Argument	Type	Description
InterruptState	UINT16	Interrupt state that needs to be set.

6.3.8.4 Return Values

Table 72 Return Values for FlashInterruptSet()

Type	Description	Possible Values
UINT16	Successful completion	SGF_OK

6.3.8.5 Troubleshooting

None

6.3.8.6 Global Variables used

None

6.3.8.7 Comments

None

6.3.8.8 Assumptions

None

APPENDIX A: TROUBLESHOOTING FOR HIGH, MIDDLE LEVEL AND LOW LEVEL APIS

Table 73 Troubleshooting for High, Middle level and Low level APIs

Return Value	Description	Reason	Solution
EED_ERROR_NOT_BLANK	The flash memories are not blank.	Memory location allotted for emulation might be corrupt.	Allot some other memory location for EEPROM emulation.
EED_ERROR_NOFND	Record ID not found.	Data ID entered is not present in the active sectors.	Enter a record ID present the active sectors.
EED_ERROR_RANGE	Address exceeds the range.	Size or destination or end address for program exceeds the valid range	For load data field command end address should not cross into the next block.
EED_ERROR_SSTAT	Sector Status error.	Function is called without initialization or after a brown out.	Reinitialize the EEPROM.
EED_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Memory location allotted for emulation might be corrupt.	Allot some other memory location for EEPROM emulation.
EED_ERROR_IDRNG	Record ID not in range.	The ID entered is greater than EED_MAX_RECORD_NUMBER or the ID is 0x0000	Enter a valid record ID.
EED_ERROR_ACCERR	Access error flag is set while operating the Flash.	Set if Clock is not initialized	Check the FCLKDIV register and set the register and set the register with the correct value required.
EED_ERROR_PVIOL	Protection violation flag is set while operating the Flash.	Set due to program or erase in the protection area.	Reset the flag and write give the destination address with is not in the protection area.
EED_ERROR_INVALIDCLK	Error in clock setting.	The clock divider value is not correct	Check the FCLKDIV register and set the register and set the register with the correct value required. Check the oscillator clock and bus clock values. The values should be such that the Flash clock is between 150 to 200 KHz.
EED_ERR_SIZE	Error due to size mismatch.	Size is not sector aligned	The size should be sector aligned
EED_ERR_MGSTAT0	Hardware Error	Set if any non-correctable errors have been encountered during the verify operation	Hardware Error

EED_ERR_MGSTAT1	Hardware Error	Set if any errors have been encountered during the verify operation	Hardware Error
EED_ERR_DATAMISMATCH	Error due to mismatch in source and destination data.	Source and destination data mismatch	Check the failed address, failed data and source data for the exact error.

APPENDIX B: RETURN CODES OF LOW LEVEL APIS

The following Return Codes will be returned to the caller function/application. The return codes shall be a 16-bit value and each bit position corresponds to a specific error. If more than one bit position is set, it means that more than one error has occurred. The bit positions and the corresponding error are detailed below.

Table 74 Return Code

Name	Value	Description
SGF_OK	0x00	Called function succeeded.
SGF_ERR_SIZE	0x0002	Size is not word aligned
SGF_ERR_RANGE	0x0004	Size or destination or end address for program exceeds the valid range
SGF_ERR_ACCERR	0x0008	This Flash operation caused an access violation.
SGF_ERR_PVIOL		Set if any area of the Flash protected
SGF_ERR_MGSTAT0	0x0020	Set if any non-correctable errors have been encountered during the verify operation(Hardware Error)
SGF_ERR_MGSTAT1	0x0040	Set if any errors have been encountered during the verify operation(Hardware Error)
SGF_ERR_DATAMISMATCH	0x0400	Source and destination data mismatch
SGF_ERR_INVALIDCLK	0x0800	The clock divider value is not correct.

Since the SSD is reused, the return values are not changed. But the EED middle and high level return values are so chosen that they coincide with these low level return values. The only change is that for erase verify. The return code of anything other than SGF_OK is handled in the Middle/High level such that a distinct error code is thrown up at the middle and high level.

Note:

The access error, protection violation error, size not aligned error, address range error and the hardware generated errors are not specifically mentioned in any of the middle and high level functions. Yet, it is likely that this error will be thrown up for any API that deals with direct operation on the hardware like a write or an erase.

APPENDIX C: PERFORMANCE DATA

C.1 Operational Timings

Testing Conditions:

Bus clock : 40 Mhz
No of Active sectors : 2
No of Alternative sectors : 3
No of IDs stored in Cache Table: 8
Data Length of an ID : 6 bytes
Cache Table size : 32 bytes (8*2)

Note: The D-Flash sector erase operation is interrupt based.

Table 75 Operational Timing for Read

Case	Time(ms)	Comments
Cache Table Entry	0.003	DataID is found in the cache table
Best	0.008	DataID is the first record searched, (i.e) the last record written.
Average	0.148	DataID is found in the middle of the active, (i.e) about half the records should be searched.
Worst	0.268	DataID is found at the top of the active sectors,(i.e) all the records in the active sectors shoud be searched

Table 76 Operational Timing for Write with Cache Table enabled

Case	Time(ms)	Comments
Without Swap	0.370	The data record is written into the free space of the active sector
With Swap	20.4	The oldest active sector is compressed to the oldest alternative sector
Double Swap	53.0	This is the worst case scenario. When the oldest active sector contains all unique data, then the newest active sector will become full and another swap is required.

Table 77 Operational Timing for Write with Cache Table disabled

Case	Time(ms)	Comments
Without Swap	0.370	The data record is written into the free space of the active sector
With Swap	23.6	The oldest active sector is compressed to the oldest alternative

		sector
Double Swap	56.4	This is the worst case scenario. When the oldest active sector contains all unique data, then the newest active sector will become full and another swap is required.

Timing is measured using the port toggling on the S12XS128 EVB.