EEPROM Emulation Software Driver for C55/C90FL/C90LC Flash Modules

User's Manual

REVISION LIST

Version No.	Date	Author	Description
0.1	01-21-2014	FPT Team	Initial Version
1.0	02-28-2014	FPT Team	Add performance data and supported device list
1.12	12-23-2014	FPT Team	Add FSL_RecoverEepromSystem API and note for D-Cache and flash controller buffer
1.2.0	06-16-2015	FPT Team	- Remove some global variables - Add some functions in the middle level - Move driver level definitions from the ‘ee_emulation.h’ file to a new header file ‘ee_internal.h’
1.2.1	10-28-2015	FPT Team	Improve the initialization time
1.3.0	12-28-2015	FPT Team	- Update solution for handling unrecoverable brownout cases, FSL_InitEeprom return EE_ERROR_CANNOT_IDENTIFY_VALID_BLOCK, in 2 blocks configuration in earlier versions - Remove the FSL_RecoverEmulationSystem function because it’s used to handling the error EE_ERROR_CANNOT_IDENTIFY_VALID_BLOCK

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

1 OVERVIEW

1.1 Document Overview

This document is to describe the user manual for the EEPROM Emulation driver embedded on single bank of C55/C90FL/C90LC flash modules on Freescale MPC5xxx devices with both fix length and variable length record schemes. The roadmap of document is as follows:

Section 1 gives a brief overview of the system for general background knowledge of the driver.

Section 2 lists the documents referred and abbreviations used in this document.

Section 3 includes information about driver configuration parameters as physical memory layout in EEPROM emulation system.
Section 4 provides the detailed design of all APIs.

Section 5 provides the performance indexes and some important notes, limitations of the EED.

1.2 System Overview

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 block-erasable flash memory can be used to emulate the EEPROM through EEPROM emulation software.

The EEPROM emulation driver implements both fix length and variable-length data record schemes. Two or more blocks of flash can be used to implement the emulation scheme. The EEPROM functionalities to be emulated include organizing data records; initializing; de-initializing; reporting EEPROM status; reading; writing; and deleting data records.

1.3 Main features

The driver is implemented in the way that supports the following main features:

1. Support C55/C90FL/C90LC flash modules with different hardware ECC detections and handlings
2. Support fixed length and variable length record schemes corresponding with 4/8/16/32 bytes ECC read invalidation boundary.
3. Support re-erase the flash block if the previous erase operation was failed
4. Support re-write the data record to the next location if failed
5. Support dead block elimination
6. Concurrency support via callback function
7. Support code relocation ability: user can run code both from internal ram or from different internal flash partition
8. Support immediate read/write data records while erasing a block in swapping process
9. Increase swapping performance by adding a cache region dedicated for swap purpose.
10. Ready-to-use demos illustrating the usage of the driver with CodeWarrior, GreenHills compilers

2 ACRONYMS AND REFERENCES

2.1 Acronyms

Table 2-1 Acronym

Abbreviation	Complete name
API	Application Programming Interface
ECU	Electronic Control Unit
EED	EEPROM Emulation Driver
EE	EEPROM Emulation
ECC	Error Correction Code
MSB	Most Significant Bit
Word	A word is 4 bytes of data.
Dword	A Dword (double word) is 8 bytes of data
Page	A page is 16 bytes of data

2.2 Terms

Table 2-2 Terms

Term	Definition
Flash Block	It is the smallest portion of flash that can be erased. The minimum block size 16KB
EE Block	It is a cluster used for emulation. It can be a flash block or a combination of several consecutive flash blocks with constrain that they must belong in the same block space (low, middle, high). If user chooses an EE block as a flash block only, then all information of EE block will be totally matched to that of flash block. In this document, the block concept is used to refer to EE block. If we want to mention to flash block concept, the term “flash block” will be used to emphasize it.
Record	This is part of block and it contains the user raw data field and a data id field. Besides these, it has a status field that is used for the emulation purpose and an optional size field to support variable data record schemes.

2.3 References

Table 2-3 References

No	Document Name	Version	Document Identifier (If any)
1.	c55_BlockGuide	Rev.2	
2.	MPC5746M_McKinley_RM	Rev.1	
3.	MPC5744M_Panther _RM	Rev.0	
4.	K2_RM	Rev.2	
5.	MPC5775K RM	Rev.1.1	

3 CONFIGURATION PARAMETERS AND MEMORY LAYOUT

3.1 Configuration Parameters

The configuration parameters, which are handled as structures are given in this section.

3.1.1 EEPROM configuration Structure definition

The structure EEPROM_CONFIG defines the number of blocks that are used for EEPROM emulation, pointer to cache table and an indicator for preventing multiple write at the same time.

Table 3-1 EEPROM Configuration Structure Field Definition

Field	Type	Description
numberOfBlock	UINT32	Total number of blocks used for EEPROM emulation
numberOfDeadBlock	UINT32	Total number of blocks which are make to DEAD in emulation
activeBlockIndex	UINT32	Active block index indicating which block is current active
blockWriteFlag	UINT32	Block write lock for erasing and programming without disturbance
cacheEnable	BOOL	Use cache table flag to speed up reading time or not. - TRUE: use cache. - FALSE: don't use cache.
cTable	CACHE_TABLE*	Cache table pointer
flashBlocks	BLOCK_CONFIG**	Block configuration array pointer

3.1.2 Block configuration structure definition

The structure BLOCK_CONFIG defines block start address, block size, blank space, block space and the bit map of specific block selected as well as its partition information.

Table 3-2 Block Configuration Structure Field Definition

Field	Type	Description
enabledBlock	UINT32	The bit map flash block in physical space (block 0 is corresponding to bit 0; block 1 is corresponding to bit 1 and so on)
blockStartAddr	UINT32	Block start address
blockSize	UINT32	Block size

Field	Type	Description
blankSpace	UINT32	The address pointer to the blank space
blockSpace	UINT8	The space (low, middle, high) for the block in physical address
partSelect	UINT32	The bit map of partition information for this block (partition 0 is corresponding to bit 0; partition 1 is corresponding to bit 1 and so on)

3.1.3 Cache table configuration structure definition

The structure CACHE_TABLE defines start address of cache table and the table size.

Table 3-3 Cache Table Configuration Structure Field Definition

Field	Type	Description
startAddress	UINT32	Start address of Cache table
size	UINT32	Size of cache table

3.2 Callback notification

The EEPROM Emulation Driver facilitates the user to supply a pointer to CallBack() function so that time-critical events can be serviced during EEPROM operations. The service watchdog timer 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 a NULL function macro.

#define NULL_CALLBACK ((void *) 0xFFFFFFFF)

The job processing callback notifications shall have no parameters and no return value. If a job processing callback notification is configured as null pointer, the corresponding callback routine shall not be executed.

3.3 Return Codes

The return values will be returned to the caller function. The following table lists all of possible return values:

Table 3-4 Return Value

Name	Value	Description	Trouble Shootings
EE_OK	0x0000	Function executes successfully.	None
EE_INFO_HVOP_INPROGRESS	0x0001	The high voltage operation is in progress.	Waiting for the completion of the program/erase operation

EE_INFO_PROGRAM_SUSPEND	0x0002	Program operation has been suspended.	None
EE_INFO_ERASE_SUSPEND	0x0004	Erase operation has been suspended.	None
EE_ERROR_WRITE_IN_PROGRESS	0x0008	EPRPOM operation is in progress and cannot launch any other operation.	Waiting for the completion of the operation
EE_ERROR_PE_OPT	0x0010	Cannot perform high voltage operation successfully.	Check the EPROM/BLOCK configurations and hardware status.
EE_ERROR_MISMATCH	0x0020	It indicates that there is at least one double word is not same with source data.	None
EE_ERROR_BLOCK_STATUS	0x0040	The block status is invalid.	Call FSL_InitEeprom to synchronize EEPROM system.
EE_ERROR_BLOCK_CONFIG	0x0080	The block configurations are incorrect.	Check block address spaces in the configuration structures
EE_ERROR_DATA_NOT_FOUND	0x0100	The required data is not found in the EEPROM emulation.	None
EE_ERROR_NOT_IN_CACHE	0x0200	Required data is not in the cache table.	None
EE_ERROR_NO_ENOUGH_SPACE	0x0400	The data is too big to fit in any of the block.	Use large size of EE block.
EE_ERROR_PROGRAM_BLOCK_INDICATOR	0x0800	Failed to make block indicator word to nonblank for several times	Call FSL_InitEeprom to synchronize EEPROM system. If still failed, replace the block.

EE_ERROR_PROGRAM_ERASE_CYCLE	0x1000	Failed to program erase cycle word	Call FSL_InitEeprom to synchronize EEPROM system. If still failed, replace the block.
EE_ERROR_NOT_ENOUGH_BLOCK_FOR_ROUND_ROBIN	0x2000	The left "good" block numbers is not enough for round robin	Replace the dead blocks and re-initialize EEPROM system.
EE_ERROR_PROGRAM_BLOCK_INDICATOR_FOR_DEAD	0x4000	Failed to program dead block indicator	Eliminate the failed block.
EE_MAKE_DEAD_OK	0x8000	Make the block to dead successfully and can continue round robin	None

3.4 User defined Macros

These following macros need to be defined in "user_cfg.h"

Table 3-5 User defined Macros

Name	Value	Description
NUMBER_OF_ACTIVE_BLOCKS	Any integer value Default: 1	The number of the active blocks configured by user
VLE_IS_ON	1 0	To specify which instruction set is used: - 0: BOOKE - 1: VLE
SCHEME_SELECT	0: ECC4_FIXLENGTH 1: ECC4_VARLENGTH 2: ECC8_FIXLENGTH 3: ECC8_VARLENGTH 4: ECC16_FIXLENGTH 5: ECC16_VARLENGTH 6: ECC32_FIXLENGTH 7: ECC32_VARLENGTH	User must set the macro to select record scheme for emulation. ECC4_xxx, ECC8_xxx, ECC16_xxx, ECC32_xxx are corresponding with 4, 8, 16, 32 ECC read invalidation boundary respectively.
MAX_REERASE	Any integer value Default: 1	Maximum number of times to allow re-erasing a block if it is failed to erase.
MAX_REPGM_BLK_IND	Any integer value Default: 1	Maximum number of times to allow re-programming block indicator words if it is failed to make to nonblank
DATA_SIZE	Integer value range from 1 to 0xFFFE	To specify the data size of record which is mandatory for fixed length record schemes
SWAP_CACHE_SIZE	Any integer value greater	To specify the size of cache in byte

	than 4 and aligned by 4 (word size)	which uses during swapping to speed up swapping time.
EER_OPTION	0x00: EER_MCR 0x01: IVOR_EXCEPTION	To specify the method used for handling ECC error: - 0: via EER bit in MCR register and ADD register - 1: via IVOR2/IVOR1 exception
FLASH_REG_BASE	Depending on hardware	To specify the flash register base address
For C55 devices (except MPC5777C) which handle ECC via EER bit in MCR register and ADD register (in ee_blocks.h)		
BASE_ADDR_aL_a16K		Start address of 16K block in low space
BASE_ADDR_aL_a32K		Start address of 32K block in low space
BASE_ADDR_aL_a64K		Start address of 64K block in low space
BASE_ADDR_aM_a16K		Start address of 16K block in middle space
BASE_ADDR_aM_a32K		Start address of 32K block in middle space
BASE_ADDR_aM_a64K		Start address of 64K block in middle space
BASE_ADDR_aH_a16K		Start address of 16K block in high space
BASE_ADDR_aH_a32K		Start address of 32K block in high space
BASE_ADDR_aH_a64K		Start address of 64K block in high space
Users need to provide the start address of the data flash blocks. For example, on MPC5775K: BASE_ADDR_aL_a16K = 0x00800000 BASE_ADDR_aL_a32K = 0x00808000 BASE_ADDR_aL_a64K = 0xFFFFFFFF It is not necessary to determine the value for the flash block which is not in use. In this case, user can leave default value of 0xFFFFFFFF for them.		

3.5 EEPROM Emulation Memory Layout

3.5.1 EEPROM Data Organization

The block and record schemes are constructed based on length-type record scheme and ECC read invalidation boundary. Hence, there are different contributions as follows:

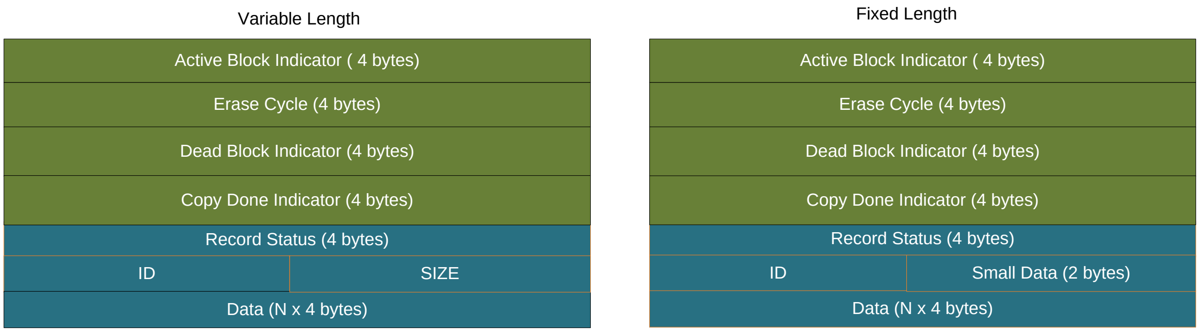

Figure 3-1: Record schemes for 4 bytes ECC read invalidation boundary

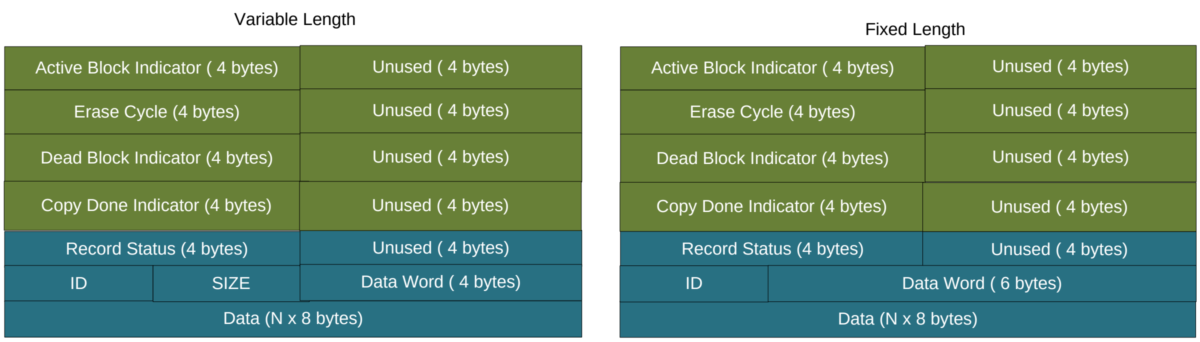

Figure 3-2: Record schemes for 8 bytes ECC read invalidation boundary

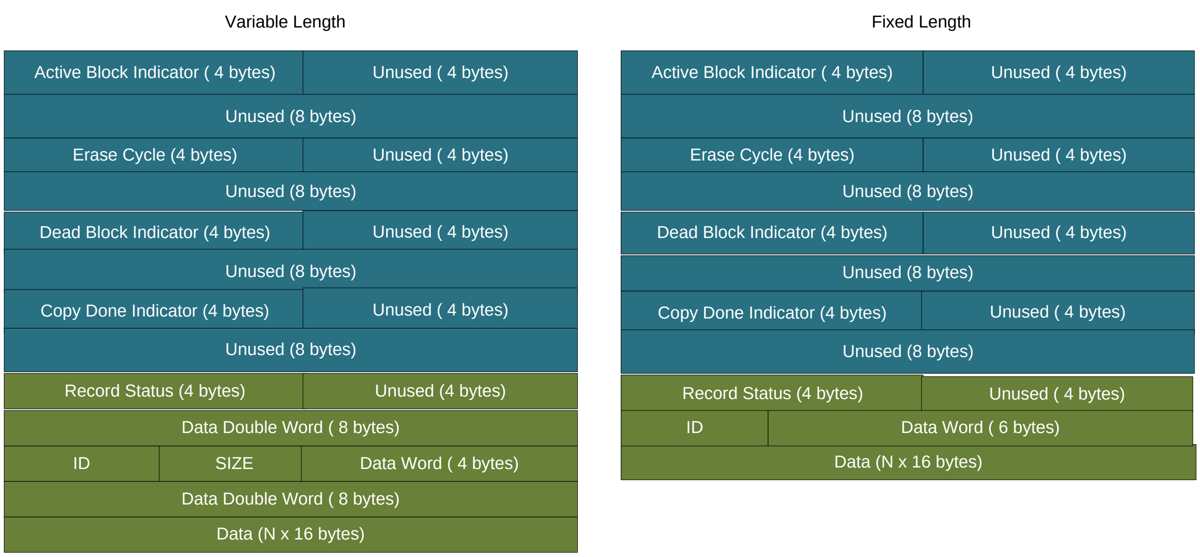

Figure 3-3: Record schemes for 16 bytes ECC read invalidation boundary

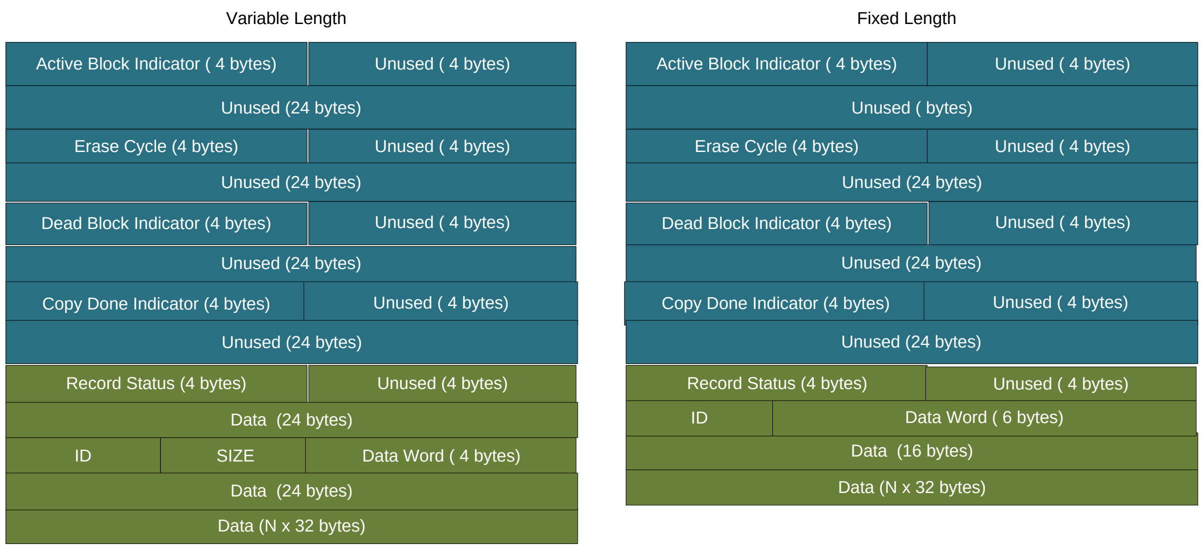

Figure 3-4: Record schemes for 32 bytes ECC read invalidation boundary

Each emulation block contains:

1. Block header

This section stores erase cycle and block indicators. To successfully recover from brownout, each field in this section must be located in the addresses that are aligned by the ECC read boundary.

2. Data record space

Data records are organized as several sections:

- Record status (4 bytes in size): indicate status of the record

  If it is 0xFFFF0000, the record is completed state

  If it is 0x00000000, the record is in deleted state

  Other values, the record is in invalid state.

  Only completed records are considered as the valid one for read/write operations.

- Record identifier (2 bytes in size): Its most significant bit (MSB) will be used to identify whether this record ID is immediate data or not. MSB bit is 1 denotes that this data ID is immediate data. Otherwise, this data is normal data.

- Size (2 bytes in size): It is used only on variable length record schemes and to define actual size of data record.

- Small data: this is the data section that is padded for the record status and identifier/size to make they aligned by ECC read boundary size.

- Remaining data: the remaining data and must be an ECC size boundary alignment component

The block status is specified as the combination of several fields as following table:

Table 8 Block State Definition

State	Dead Block Indicator	Erase Cycle	Active Block Indicator	Copy Done Indicator	Record Space
Dead	Nonblank	Don’t care	Don’t care	Don’t care	Don’t care
Erased	Blank	Blank	Blank	Blank	Blank
Alternate	Blank	Valid	Blank	Blank	Blank
Active	Blank	Valid	Nonblank	Don’t care	Don’t care
Update	Blank	Valid	Blank	Blank	Nonblank
Copy done	Blank	Valid	Blank	Nonblank	Nonblank
Invalid	Not in Dead, Erased, Alternate, Active, Update, Copy done states				

3.5.2 EEPROM Emulation Operation

3.5.2.1 Initialize EEPROM

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

- The first time of using EEPROM: In this case, the EED will format all blocks then assign one as the active block and the others as alternative blocks. At last, clears the contents of the cache table if enabled.
- Continue using EEPROM: In this case, the EED should determine which block is the current active one, do recovery and update blank space for all emulated blocks. Then, it fills the cache table with expected data if enabled.

Initializing EEPROM also does the brownout handling to recover from accident. Normally, the block status transition follows the below figures:

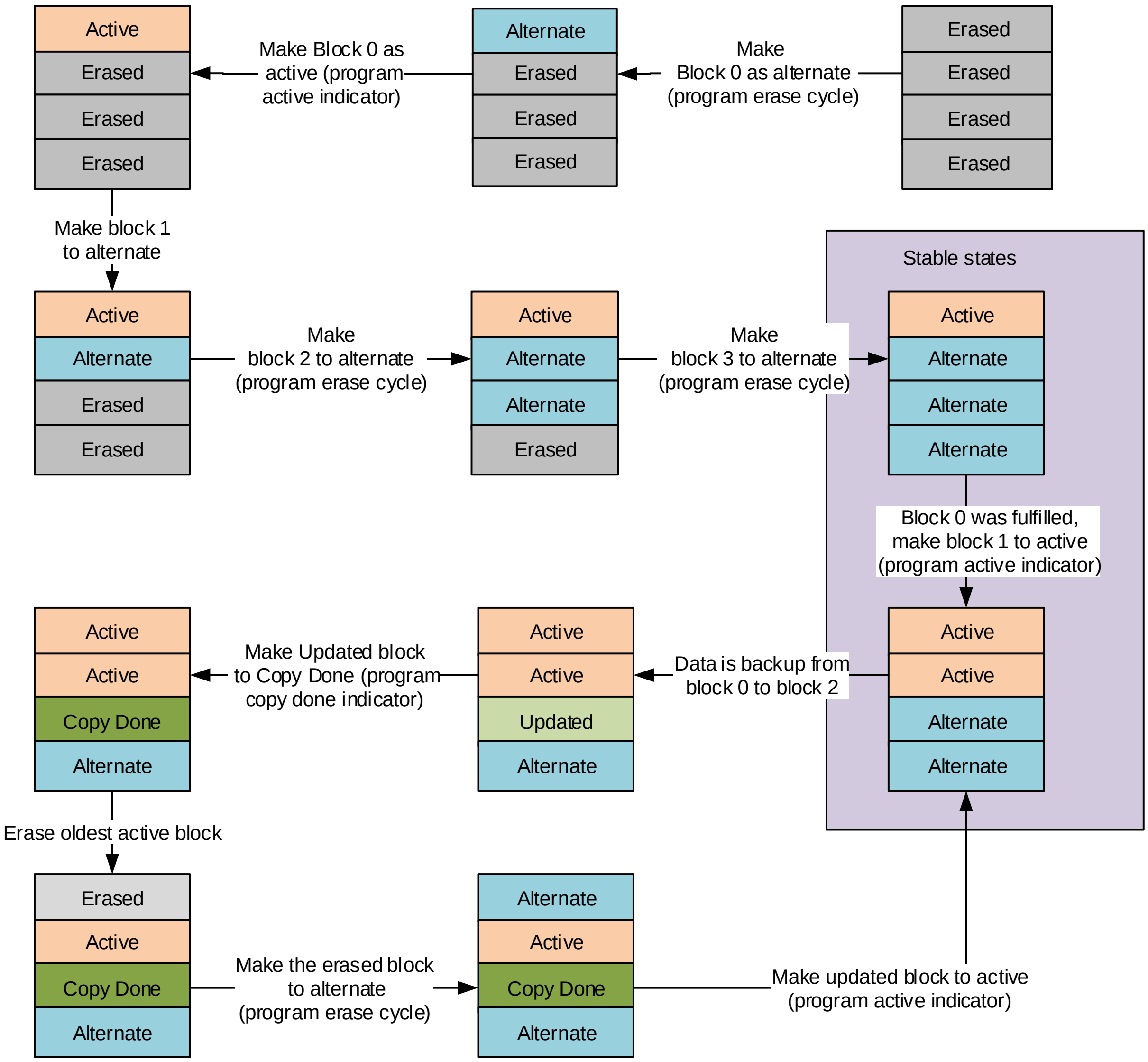

Figure 3-5: Block Transition 1

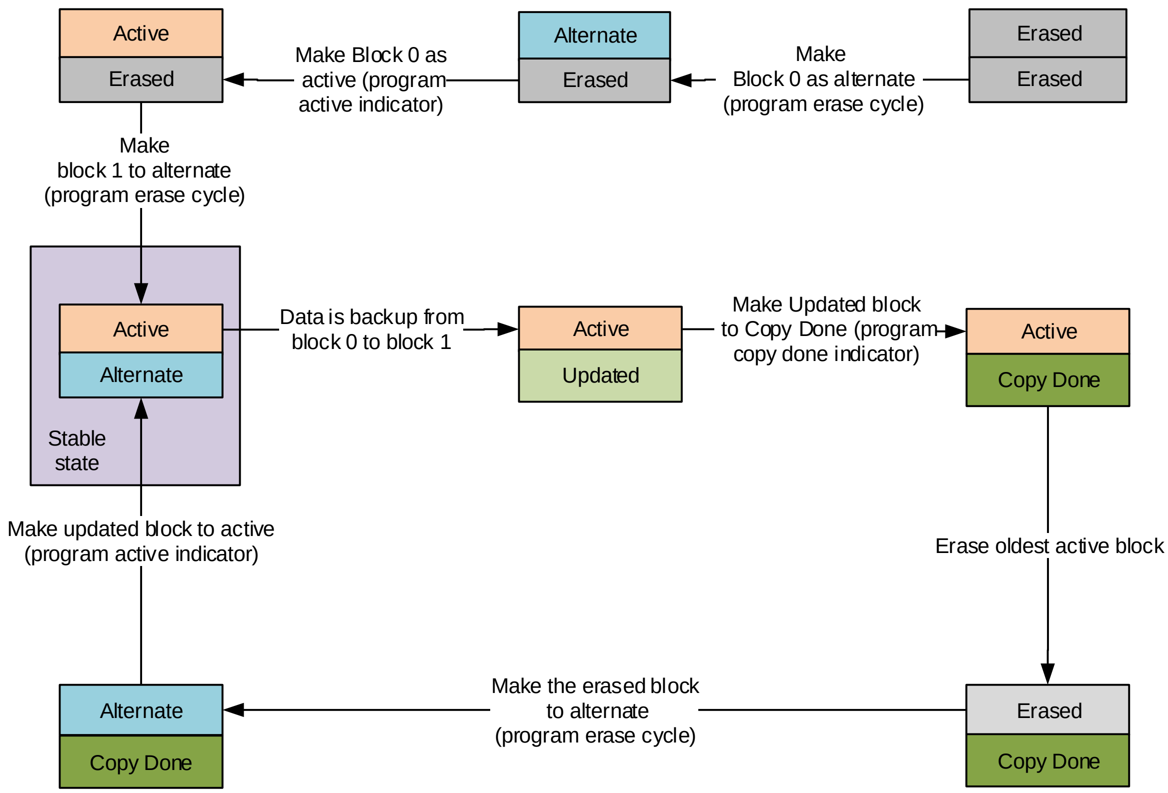

Figure 3-6: Block Transition 2

EEPROM emulation driver is responsible for keeping EEPROM in the stable states.
If there is a brownout occurs during block status transition, to recovery block status as well as data integrity, the driver will handle:

- If EEPROM system is in unstable state as defined in above figures, follow the steps in these figures to make it in stable state
- If failed to program block indicators (includes active block indicator, and copy done indicator) to nonblank value after number of trying times, the driver will return EE_ERROR_PROGRAM_BLOCK_INDICATOR, EE_ERROR_PROGRAM_BLOCK_INDICATOR_FOR_DEAD if that failure is in dead block indicator. In the situation of dead block indicator, it is user's responsibility to eliminate the error block from emulation. Otherwise, FSL_InitEeprom API should be invoke.
- If failed to program erase cycle, the driver will return error EE_ERROR_PROGRAM_ERASE_CYCLE to inform user. In addition, the user must re-initialize emulate system by calling FSL_InitEeprom API.
- During emulation, if number of dead blocks is too large (in other words, number of good blocks is not enough for round robin), it will return error EE_ERROR_NOT_ENOUGH_BLOCK_FOR_ROUND_ROBIN and the user must specify additional blocks to continue using emulation system

3.5.2.2 Write EEPROM Data

Because the flash memory cell cannot be erased individually, EED must write a new data record with same data ID for the updated value to the EEPROM blank area. The followings describe several extreme situations, which may take place as well as corresponding handling during writing operation:

- **Program operation fails**: If program operation fails during programming block information including block indicator and erase cycle, the proper error code will be returned and stop writing. If program operation fails during programming data record, this data will be re-programmed by skipping a suitable address until successfully. If this operation consecutively takes place with too many times such that there is no enough space to write on available blocks, the error of EE_ERROR_NO_ENOUGH_SPACE may be returned and finish writing.
- **Immediate data request:** MSB will be used to distinguish immediate data or normal data. If an immediate data requested (MSB = 1) while an erase operation is going on, this high voltage operation need to be suspended to serve that request. Otherwise, (MSB=0) new normal data requested while an erase operation is going on, it will be returned EE_INFO_HVOP_INPROGRESS error.
- **Swapping:** After several record writings, the active block may not have enough free space for a new data record. It is needed to copy all the latest data records to alternative block to clean up the EEPROM. This procedure is called “swapping” and after swapping, the alternative block will become the new active block and the old active block will be formatted as new alternative block.

3.5.2.3 Read EEPROM Data

Read routine will first search in cache table if enabled. If founded, it should retrieve address of that record from cache table. Otherwise, it will identify the latest copy of data record by scanning the entire the current active block from the first data record to the blank region in case of adopting variable length record schemes. If that record ID is not found in current active block, it will search in entire all other active blocks in the ageing order. For fixed length record scheme, to increase searching performance, an optimized search algorithm is implemented to enable search from blankSpace address back to beginning of blocks. Finally, it will return EE_ERROR_DATA_NOT_FOUND if there is no data ID in cache table as well as all active blocks.

3.5.2.4 Delete EEPROM Data

If does not need a data record, user can delete it from the emulated EEPROM system. EED does NOT physically remove this record at the time users want to delete it. Instead, EED will only change the record’s state to “DELETE” so that it is regarded as un-used data and will be removed from emulated EEPROM in block swapping.

However, the deleted data record can be re-written into the EEPROM. The read routine will determine the latest data record.

3.5.2.5 Report EEPROM Status

The block erasing cycles will be retrieved from the current active block and it reflects the erasure times since the EEPROM has been setup. It is only an approximate number and will be set to one when first time using EEPROM.

3.5.2.6 Remove EEPROM

If the emulated EEPROM is not required, the flash memory for EEPROM emulation should be released. The removing routine will erase all the blocks used for emulation.

3.6 EEPROM Emulation Software Cache

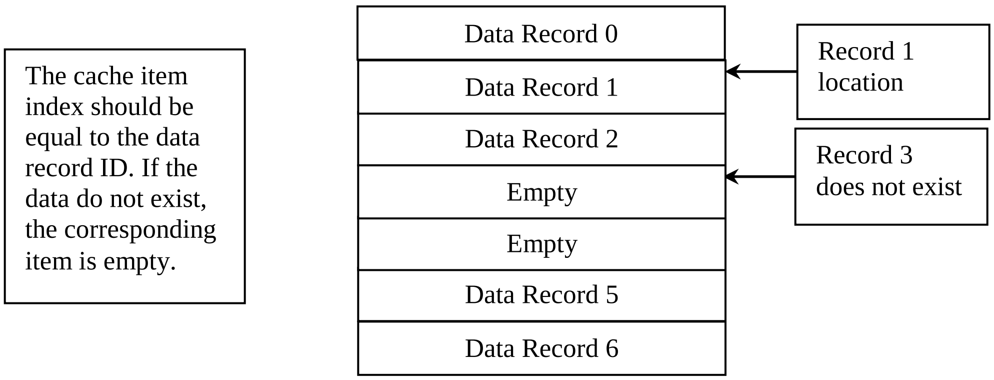

Figure 3-7: EEPROM Emulation Software Cache Layout

In order to speed up the data record searching, the EED provides the software cache for buffering the data records locations. Both the start address and size of the cache table are user configurable.

The cache table is one dimension array. Each item of this array is 32-bit length and saves the latest location of the data record which has ID equals to this item index. The cache items are filled with 0xFFFF_FFFF to indicate the corresponding data records do NOT exist. The total array item number depends on the size of the cache table:

Item Number = Cache Table Size in bytes / 4

If the ID of a data record is larger than the item number, it can only be searched by going through the entire active blocks.

This cache algorithm can save not only the EED code size, but also the reading time. However, it is required to define the most frequently accessed data IDs within the table item number (from 0 to item_number-1).

If the cache table is enabled, the initialization routine will fill the cache table by scanning the active blocks. The cache table will be updated after deleting or writing a new copy of data record. Deleted IDs are filled in the cache table by value of 0xFFFFFFFE.

The cache table can be disabled when the user's resource is limited.

It is not permitted to enable the cache table after the EEPROM initialization. When it is needed, EEPROM initialization routine should be re-called.

4 API SPECIFICATION

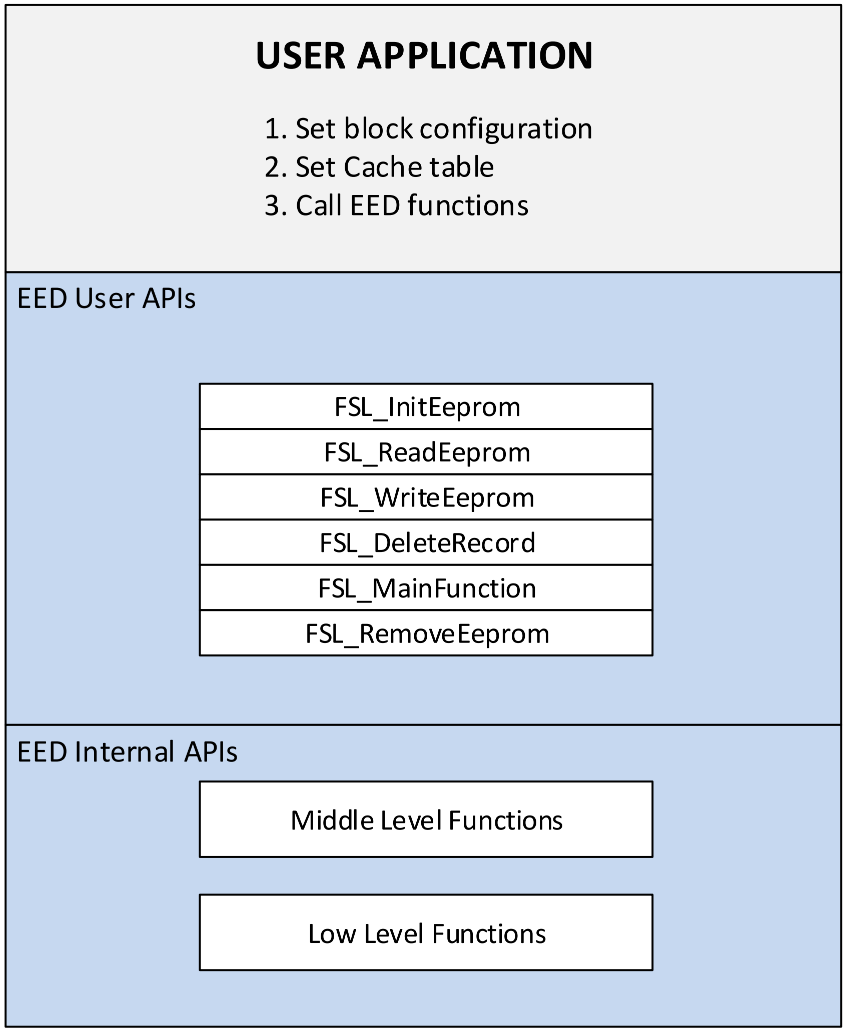

Figure 4-1: EED Architecture

The EEPROM Emulation Driver will have three levels of functions: high level, middle level and low level.

- High level (User level) APIs provide the user’s interface and program flow controlling.
- Middle level functions provide the relatively independent task unit.
- Low level functions interface with hardware to provide the fundamental Flash operations.

4.1.1 FSL_InitEeprom

This API initializes the EEPROM Emulation driver (software) and all EEPROM memory relevant registers (hardware) with parameters provided in the given configuration set.

This API also does the brownout recovering to avoid losing data due to brownout and determines the active block index as well. Refer to section 3.5.2.1 for more information.

Table 4-1: FSL_InitEeprom

Prototype	UINT32 FSL_InitEeprom (EEPROM_CONFIG* eepromConfig, void(*CallBack)(void))	
Parameter	EEPROM_ CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	void *	***CallBack***: function pointer of callback
Return value	UINT32	***EE_OK***: successful completion
		EE_ERROR_PROGRAM_BLOCK_INDICATOR_FOR_DEAD: cannot make the dead block indicator to nonblank
		EE_ERROR_WRITE_IN_PROGRESS: an EEPOM operation is in progress
		EE_ERROR_NO_ENOUGH_SPACE: not enough space to copy the latest copy of data record from oldest active block to update block
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress
		EE_ERROR_PROGRAM_BLOCK_INDICATOR: cannot make the block indicator to nonblank
		EE_ERROR_PROGRAM_ERASE_CYCLE: program erase cycle unsuccessfully
		EE_ERROR_BLOCK_CONFIG: block configurations are incorrect.
		EE_ERROR_NOT_ENOUGH_BLOCK_FOR_ROUND_ROBIN: number of "good" blocks left after dead block elimination is not enough for round robin.

Note: *The FSL_InitEeprom will be synchronous in behavior and it will not support re-entrance*

4.1.2 FSL_ReadEeprom

This API is to read the specific data record. The data record size to be read is determined by the ***dataSize*** variable.

This API can be called when an erase is ongoing on such as a swapping is being done. If the erased block is in different partition with the targeted read block, it will read the expected data record without suspending that high voltage operation. Otherwise, if the erased block is in the same partition, it will read the expected data record after suspending this high voltage. However, FSL_MainFunction still need to be called after that to update block status for all blocks.

Refer to section 3.5.2.3 for more information.

Table 4-2: FSL_ReadEeprom

Prototype	UINT32 FSL_ReadEeprom(EEPROM_CONFIG *eepromConfig, UINT16 dataID, UINT16 dataSize, UINT32 source, void (*CallBack)(void))

Parameter	EEPROM_CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	UINT16	***dataID:*** the required data ID. It can be any value from 0x0 ~ 0xFFFF. The MSB is used to identify whether this is immediate data or not. - MSB = 1: immediate data - MSB = 0: normal data.
	UINT16	***dataSize***: Size of data to be read in byte. This value can be different from actual data record size.
	UINT32	***source***: address of buffer to store read data
	void *	***CallBack***: function pointer of callback
Return value	UINT32	***EE_OK***: successful completion
		EE_ERROR_DATA_NOT_FOUND : the requested data record is not present in EEPROM
		EE_ERROR_WRITE_IN_PROGRESS: an EEPOM operation is in progress
		EE_INFO_HVOP_INPROGRESS: : a program/erase operation is in progress

Note: *The FSL_ReadEeprom will be synchronous in behavior and it will not support re-entrance. FSL_InitEeprom has been successful execution before calling it.*

4.1.3 FSL_WriteEeprom

This API is to write data records to the EEPROM. It will re-write data record if this program operation fails. If an immediate data request while an erase operation is going on, the operation will be suspended to serve this request in advance.

Note that if this API is called to write a normal data while an erase is going on such as a swapping is being done, it will return EE_INFO_HVOP_INPROGRESS.

Refer to section 3.5.2.2 for more information.

Table 4-3: FSL_WriteEeprom

Prototype	UINT32 FSL_WriteEeprom(EEPROM_CONFIG* eepromConfig, UINT16 dataID, UINT16 dataSize, UINT32 source, void (*CallBack)(void))	
Parameter	EEPROM_CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	UINT16	***dataID:*** the required data ID. It can be any value from 0x0 ~ 0xFFFF. The MSB is used to identify whether this is immediate data or not. - MSB = 1: immediate data - MSB = 0: normal data.
	UINT16	***dataSize***: the actual data size in bytes.
	UINT32	***source***: address of data buffer
	void *	***CallBack***: function pointer of callback

Return value	UINT32	***EE_OK***: successful completion
		EE_ERROR_NO_ENOUGH_SPACE: not enough blank space for the requested record
		EE_ERROR_WRITE_IN_PROGRESS: an EEPOM operation is in progress
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress
		EE_ERROR_PROGRAM_BLOCK_INDICATOR: cannot make the block indicator to nonblank

Note: *The FSL_WriteEeprom will be synchronous in behavior and it will not support re-entrance. FSL_InitEeprom has been successful execution before calling it.*

4.1.4 FSL_DeleteRecord

This API is to delete a data record in the EEPROM emulated Flash.

This API can be called when an erase is going on such as a swapping is being done. But it will suspend this high voltage before deleting the data record. However, FSL_MainFunction still need to be called after that to update block status for all blocks.

Table 4-4: FSL_DeleteRecord

Prototype	UINT32 FSL_DeleteRecord(EEPROM_CONFIG* eepromConfig, UINT16 dataID, void (*CallBack)(void))	
Parameters	EEPROM_CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	UINT16	***dataID:*** the required data ID. It can be any value from 0x0 ~ 0xFFFF. The MSB is used to identify whether this is immediate data or not. - MSB = 1: immediate data - MSB = 0: normal data.
	void *	***CallBack***: function pointer of callback
Return values	UINT32	***EE_OK***: successful completion
		EE_ERROR_DATA_NOT_FOUND : the requested data record is not present in EEPROM
		EE_ERROR_WRITE_IN_PROGRESS: an EERPOM operation is in progress
		EE_ERROR_PE_OPT: failed to perform high voltage operation
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress

Note: *The FSL_DeleteRecord will be synchronous in behavior and it will not support re-entrance. FSL_InitEeprom has been successful execution before calling it.*

4.1.5 FSL_RemoveEeprom

This function is to clear all blocks used for EEPROM emulation. Moreover, all the blocks will be fully erased.

Table 4-5: FSL_RemoveEeprom

Prototype	UINT32 FSL_RemoveEeprom(EEPROM_CONFIG * eepromConfig, void (*CallBack)(void))	
Parameters	EEPROM_ CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	void *	***CallBack***: function pointer of callback
Return values	UINT32	***EE_OK***: successful completion
		EE_ERROR_WRITE_IN_PROGRESS: an EEPROM operation is in progress
		EE_ERROR_PE_OPT: failed to perform high voltage operation
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress
		EE_ERROR_BLOCK_CONFIG: block configuration is not correct

Note: *The FSL_RemoveEepom will be synchronous in behavior and it will not support re-entrance.*

4.1.6 FSL_ReportEepromStatus

This API is to report block erasing cycles and check the current Active block status.

Note that if this API is called when an erase is going on such as a swapping is being done, it will return EE_INFO_HVOP_INPROGRESS.

Table 4-6: FSL_ReportEepromStatus

Prototype	UINT32 FSL_ReportEepromStatus(EEPROM_CONFIG* eepromConfig, UINT32* erasingCycles)	
Parameters	EEPROM_ CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	UINT32*	***erasingCycles***: store the erase cycle which is retrieved from current active block
Return value	UINT32	***EE_OK***: successful completion
		EE_ERROR_BLOCK_STATUS: there is a block which is not in erased, copy done, alternate, active states
		EE_ERROR_WRITE_IN_PROGRESS: an EEPROM operation is in progress
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress

Note: *The FSL_ReportEepromStatus will be synchronous in behavior and it will not support re-entrance.*

4.1.7 FSL_MainFunction

This API will help in synchronizing the EEPROM system. It will try to re-erase the old ACTIVE block for defined number of times if previous erase operation was failed. It also updates erase cycles and block status.

Table 4-7: FSL_ MainFunction

Prototype	void FSL_MainFunction(EEPROM_CONFIG *eepromConfig, void (*CallBack) (void))	
Parameter	EEPROM_ CONFIG*	***eepromConfig***: the EEPROM emulation configurations structure pointer.
	void *	***CallBack***: function pointer of callback
Return value	UINT32	***EE_OK***: successful completion
		EE_ERROR_PROGRAM_ERASE_CYCLE: program erase cycle unsuccessfully
		EE_ERROR_NOT_ENOUGH_BLOCK_FOR_ROUND_ROBIN: number of "good" blocks left after dead block elimination is not enough for round robin.
		EE_ERROR_PE_OPT: failed to perform high voltage operation
		EE_INFO_HVOP_INPROGRESS: a program/erase operation is in progress
		EE_ERROR_PROGRAM_BLOCK_INDICATOR_FOR_DEAD: failed in make the dead block indicator to nonblank
		EE_MAKE_DEAD_OK: make the block to dead successfully and can continue emulation
		EE_ERROR_PROGRAM_BLOCK_INDICATOR: failed in make the active block indicator to nonblank

Note: *The FSL_MainFunction will be synchronous in behavior and it will not support re-entrance.*

It is the user's responsibility to poll swap status global enumeration variable ***eraseStatus_Flag*** to quit the calling API loop. User should consider the following possible values:

- ERASE_NOTSTARTED (0x00): the variable keeps that value after successful completing the swapping process.
- ERASE_INPROGRESS (0x03): the FSL_WriteEeprom has just triggered an erase operation in swapping process or the FSL_MainFunction has started a re-erase operation.
- ERASE_SWAPERROR (0x04): the function failed to re-erase the block or failed in programming block indicator/erase cycle. At this situation, it is necessary to call FSL_InitEeprom to synchronize EEPROM system.

5 APPENDIX

5.1 Code sizes of all the APIs and Timing

The below mentioned data are the code size of all the APIs on VLE and BOOKE modes when compiled with CodeWarrior 2.10, Green Hill 6.1.5, Diab 5.9.3.0 compilers

Table 5-1: Code size for C55 devices – VLE mode

API Name	Code Warrior		Green Hills	
	FixLength	VarLength	FixLength	VarLength
FSL_BlockSwapping	628	632	590	590
FSL_CopyDataRecord	142	140	142	140
FSL_DeleteRecord	300	300	270	270
FSL_EraseEEBlock	212	212	234	234
FSL_FlashAbortErase	74	74	42	42
FSL_FlashCheckStatus	106	106	54	54
FSL_FlashEraseStart	88	88	90	90
FSL_FlashProgramStart	222	222	188	188
FSL_FlashRead	230	230	184	184
FSL_FlashResume	98	98	80	80
FSL_FlashSuspend	210	210	128	128
FSL_GetEraseStatus	156	156	154	154
FSL_GetFailedAddr	192	192	126	126
FSL_GetLastJobStatus	1010	1010	928	928
FSL_GetRecordLength	74	76	48	48
FSL_GetWriteRecordOption	148	148	152	152
FSL_InitEeprom	1088	1088	1116	1116
FSL_MainFunction	236	236	250	250
FSL_MakeBlock2Dead	264	264	234	234
FSL_ProcessImmediateRequest	98	98	92	92
FSL_ProgramBlockIndicator	120	120	102	102
FSL_ReadBlockStatus	466	466	358	358
FSL_ReadEeprom	246	246	232	232
FSL_ReadRecordAtAddr	128	162	150	156
FSL_ReadRecordHead	54	46	62	58
FSL_RemoveEeprom	94	94	94	94
FSL_ReportEepromStatus	138	138	136	136
FSL_SearchInAllBlocks	188	192	180	188
FSL_SearchInTable	64	64	66	66
FSL_SearchRecordFromBottom	156	N/A	142	N/A
FSL_SearchRecordFromTop	504	556	388	442
FSL_SyncProgram	204	204	180	180

FSL_UpdateCacheTable	54	54	54	54
FSL_ValidateCopyDoneBlock	148	148	138	138
FSL_ValidateDeadBlocks	186	186	172	172
FSL_WriteDataRecord	370	522	350	474
FSL_WriteEeprom	400	398	376	374
Total	9096	9176	8282	8324

Table 5-2: Code size for C55fp devices – VLE mode – Diab compiler

API Name	FixLength	VarLength
EER_exception_handler	48	48
FSL_BlockSwapping	556	556
FSL_CopyDataRecord	122	122
FSL_DeleteRecord	276	276
FSL_EraseEEBlock	180	180
FSL_FlashAbortErase	74	74
FSL_FlashCheckStatus	100	100
FSL_FlashEraseStart	86	86
FSL_FlashProgramStart	204	204
FSL_FlashRead	156	156
FSL_FlashResume	96	96
FSL_FlashSuspend	160	160
FSL_GetEraseStatus	154	154
FSL_GetLastJobStatus	840	840
FSL_GetRecordLength	60	60
FSL_GetWriteRecordOption	148	148
FSL_InitEeprom	970	970
FSL_MainFunction	232	232
FSL_MakeBlock2Dead	198	198
FSL_ProcessImmediateRequest	92	92
FSL_ProgramBlockIndicator	92	92
FSL_ReadBlockStatus	354	354
FSL_ReadEeprom	226	226
FSL_ReadRecordAtAddr	110	110
FSL_ReadRecordHead	40	30
FSL_RemoveEeprom	82	82
FSL_ReportEepromStatus	124	124
FSL_SearchInAllBlocks	172	174
FSL_SearchInTable	60	60
FSL_SearchRecordFromBottom	142	N/A
FSL_SearchRecordFromTop	396	436
FSL_SyncProgram	170	170

FSL_UpdateCacheTable	52	52
FSL_ValidateCopyDoneBlock	124	124
FSL_ValidateDeadBlocks	170	170
FSL_WriteDataRecord	304	400
FSL_WriteEeprom	350	350
cReadAndClearEei	24	24
cRestoreEei	16	16
Total	7760	7746

Table 5-3: Code size for C90 devices

API Name	Code Warrior				Green Hills			
	BOOKE		VLE		BOOKE		VLE	
	Fix	Var	Fix	Var	Fix	Var	Fix	Var
EER_exception_handler	28	28	36	36	60	60	44	44
FSL_BlockSwapping	936	936	626	626	880	880	590	590
FSL_CopyDataRecord	208	204	142	140	216	212	142	140
FSL_DeleteRecord	468	468	300	300	456	456	270	270
FSL_EraseEEBlock	444	444	272	272	412	412	286	286
FSL_FlashAbortErase	112	112	90	90	80	80	46	46
FSL_FlashCheckStatus	160	160	130	130	108	108	58	58
FSL_FlashDepletionRecover_C	332	332	226	226	332	332	226	226
FSL_FlashEraseStart	136	136	102	102	148	148	94	94
FSL_FlashProgramStart	312	312	218	218	280	280	182	182
FSL_FlashRead	288	288	194	194	252	252	164	164
FSL_FlashResume	160	160	108	108	128	128	84	84
FSL_FlashSuspend	312	312	226	226	224	224	132	132
FSL_GetEraseStatus	264	264	162	162	256	256	160	160
FSL_GetLastJobStatus	1700	1700	1002	1002	1412	1412	928	928
FSL_GetRecordLength	100	100	74	74	72	72	48	48
FSL_GetWriteRecordOption	260	260	148	148	264	264	152	152
FSL_InitEeprom	1752	1752	1094	1094	1660	1660	1120	1120
FSL_MainFunction	412	412	236	236	380	380	248	248
FSL_MakeBlock2Dead	408	408	264	264	372	372	234	234
FSL_ProcessImmediateRequest	152	152	98	98	148	148	92	92
FSL_ProgramBlockIndicator	200	200	120	120	168	168	102	102
FSL_ReadBlockStatus	684	684	458	458	580	580	358	358
FSL_ReadEeprom	384	384	246	246	376	376	232	232
FSL_ReadRecordAtAddr	196	196	128	128	252	252	150	150
FSL_ReadRecordHead	96	80	54	46	112	104	60	56
FSL_RemoveEeprom	172	172	94	94	180	180	94	94
FSL_ReportEepromStatus	228	228	138	138	228	228	136	136

FSL_SearchInAllBlocks	284	292	188	192	280	288	180	188
FSL_SearchInTable	104	104	64	64	104	104	66	66
FSL_SearchRecordFromBottom	240	N/A	156	N/A	220	N/A	140	N/A
FSL_SearchRecordFromTop	692	752	504	550	564	628	388	422
FSL_SyncProgram	308	308	202	202	280	280	178	178
FSL_UpdateCacheTable	84	84	54	54	84	84	54	54
FSL_ValidateCopyDoneBlock	256	256	148	148	244	244	138	138
FSL_ValidateDeadBlocks	292	292	184	184	280	280	172	172
FSL_WriteDataRecord	560	728	366	480	508	708	346	466
FSL_WriteEeprom	640	636	400	398	608	604	372	370
cReadAndClearEei	12	12	10	10	16	16	14	14
cRestoreEei	8	8	6	6	16	16	12	12
Total (1)	14384	14356	9268	9264	13240	13276	8492	8506
Total (2)	14024	13900	9012	8942	12872	12896	8224	8230

Note:
(1) : For C90FL
(2) : For C90LC – Don't include FSL_FlashDepletionRecover_C

5.1.1 Initialization/Read/Write Timings

The timing is measured in **millisecond unit (ms)** and under following common configuration:

- Number of blocks is 3
- Maximum number of active blocks is 2
- Size of each EPROM block is 0x4000 (16 Kb)
- Cache size (if enabled) is 48 bytes (12 elements)
- Swap cache size is 0x28 bytes (10 elements)
- Data record has ID from 0 to 11

For the initialization operation:

- Best case is defined as:
    - Cache is disable
    - The EEPROM system has been finished initialization for the first time
- Worse case is defined as:
    - Cache is disable
    - The EEPROM system has just been started swapping

For the read operation:

- Best case is defined as:
    - Cache is enabled
    - The read record is in the enabled cache
- Worse case is defined as:
    - Cache is enabled
    - The read record at the start of the oldest ACTIVE block
    - Current active block has been fulfilled with data records

For the write operation:

- Best case is defined as:
    - Cache is disabled
    - Current active block still has space for the new record
- Worse case is defined as:
    - Cache is disabled
    - Current active block has not space for the new record, so need proceed a swapping operation
    - Swap cache size = 4 bytes

Table 5-4: Initialization Timing

Data Size				16 Bytes	32 Bytes	64 Bytes
Scheme 32 (MPC5775K) System clock is 260 MHz	VLE	Fixed - Length	Best Case	48.84	48.84	48.84
			Worst Case	85.5	67.6	61.4
		Variable Length	Best Case	48.86	48.86	48.86
			Worst Case	94.1	94.1	94.1
Scheme 16 (MPC5604P) System clock is 80 MHz	VLE	Fixed - Length	Best Case	40	40	40
			Worst Case	288	270	256
		Variable Length	Best Case	38	38	38
			Worst Case	296	297	297
Scheme 8 (MPC5643L) System clock is 64 MHz	VLE	Fixed - Length	Best Case	81	81	81
			Worst Case	520	489	465
		Variable Length	Best Case	73	73	73

			Worst Case	510	510	510
	BookE	Fixed - Length	Best Case	85	85	85
			Worst Case	531	499	472
		Variable Length	Best Case	85	85	85
			Worst Case	532	534	533
Scheme 4 (MPC5602D) System clock is 80 MHz	VLE	Fixed - Length	Best Case	52	52	52
			Worst Case	183	155	137
		Variable Length	Best Case	47	47	47
			Worst Case	174	174	171

Table 5-5: Read Timing

Data Size				16 Bytes	32 Bytes	64 Bytes
Scheme 32 (MPC5775K) System clock is 260 MHz	VLE	Fixed - Length	Best Case	0.008	0.009	0.011
			Worst Case	4.47	2.24	1.5
		Variable Length	Best Case	0.015	0.016	0.02
			Worst Case	6.03	6.03	6.02

Scheme 16 (MPC5604P) System clock is 80 MHz	VLE	Fixed - Length	Best Case	0.013	0.016	0.022
			Worst Case	6.5	4.4	2.6
		Variable Length	Best Case	0.016	0.019	0.025
			Worst Case	8.8	8.8	8.8
Scheme 8 Scheme 8 (MPC5643L) System clock is 64 MHz	VLE	Fixed - Length	Best Case	0.02	0.025	0.034
			Worst Case	11	7	4.24
		Variable Length	Best Case	0.019	0.023	0.031
			Worst Case	11.45	11.45	11.45
	BookE	Fixed - Length	Best Case	0.023	0.28	0.038
			Worst Case	11.8	7.9	4.8
		Variable Length	Best Case	0.023	0.027	0.37
			Worst Case	13.8	13.8	13.8
Scheme 4 (MPC5602D) System clock is 80 MHz	VLE	Fixed - Length	Best Case	0.012	0.15	0.02
			Worst Case	7.9	4.7	2.65
		Variable Length	Best Case	0.011	0.014	0.02

			Worst Case	8.6	8.6	8.6

Table 5-6: Write Timing

Data Size				16 Bytes	32 Bytes	64 Bytes
Scheme 32 (MPC5775K) System clock is 260 MHz	VLE	Fixed - Length	Best Case	0.06	0.08	0.11
			Worst Case	46.3	30.6	25.5
		Variable Length	Best Case	0.09	0.1	0.12
			Worst Case	53.75	53.75	53.78
Scheme 16 (MPC5604P) System clock is 80 MHz	VLE	Fixed - Length	Best Case	0.14	0.2	0.32
			Worst Case	57	41	28
		Variable Length	Best Case	0.15	0.2	0.33
			Worst Case	65	65	65
Scheme 8 (MPC5643L) System clock is 64 MHz	VLE	Fixed - Length	Best Case	0.17	0.23	0.35
			Worst Case	96	70	49
		Variable Length	Best Case	0.18	0.23	0.35
			Worst Case	89	89	89

	BookE	Fixed - Length	Best Case	0.18	0.25	0.37
			Worst Case	103.6	87.4	53.9
		Variable Length	Best Case	0.19	0.25	0.37
			Worst Case	106	107	107
Scheme 4 (MPC5602D) System clock is 80 MHz	VLE	Fixed - Length	Best Case	0.1	0.15	0.26
			Worst Case	72	48	32
		Variable Length	Best Case	0.1	0.16	0.26
			Worst Case	65	65	65

5.2 Record Scheme vs. Device Mapping

The following table lists all the supported devices as well as corresponding properly record schemes

Table 5-7: Device – Record scheme mapping

<table>
<tr><th>No.</th><th>Flash module</th><th>Record scheme</th><th>ECC handing method</th><th>Devices</th><th>Configuration macro</th></tr>
<tr><td rowspan="2">1</td><td rowspan="2">C55 Data Flash</td><td>ECC32</td><td>Via MCR and ARRD registers</td><td>MPC5744P
MPC5746M
MPC5746R
MPC5775K
MPC5777M
MPC5748G</td><td>FLASH_MODULE = C55
SCHEME_SELECT = ECC32_VARLENGTH
or
SCHEME_SELECT = ECC32_FIXLENGTH
EER_OPTION = EER_MCR</td></tr>
<tr><td>ECC8</td><td>Via exception handler</td><td>MPC5777C</td><td>FLASH_MODULE = C55
SCHEME_SELECT = ECC8_VARLENGTH
or
SCHEME_SELECT = ECC8_FIXLENGTH
EER_OPTION = IVOR_EXCEPTION</td></tr>
<tr><td>2</td><td>C90FL Code Flash</td><td>ECC8</td><td>Via exception handler</td><td>MPC5668
MPC5674F
MPC5644A
MPC564xS
MPC564xL
MPC5642A
MPC5676R</td><td>FLASH_MODULE = C90FL
SCHEME_SELECT = ECC8_VARLENGTH
or
SCHEME_SELECT = ECC8_FIXLENGTH
EER_OPTION = IVOR_EXCEPTION</td></tr>
<tr><td>3</td><td>C90LC Data Flash</td><td>ECC16</td><td>Via exception handler</td><td>MPC5604B
MPC5607B
MPC5606B
MPC5605P
MPC5604P
MPC560xS
MPC563xM</td><td>FLASH_MODULE = C90LC
SCHEME_SELECT = ECC16_VARLENGTH
or
SCHEME_SELECT = ECC16_FIXLENGTH
EER_OPTION = IVOR_EXCEPTION</td></tr>
<tr><td>4</td><td>C90LC Data Optimized Flash</td><td>ECC4</td><td>Via exception handler</td><td>MPC5602D
MPC564xB
MPC564xC
MPC5602P
MPC567xK
MPC560xE</td><td>FLASH_MODULE = C90LC_DFO
SCHEME_SELECT = ECC4_VARLENGTH
or
SCHEME_SELECT = ECC4_FIXLENGTH
EER_OPTION = IVOR_EXCEPTION</td></tr>
</table>

5.3 Notes and Limitations

Please pay attention to the following items while using the EED:

1. The flash protections are NOT changed by EED functions, even if it is required to perform an erase or program operation. It is up to the user to unprotect the flash region to allow these functions to work.

2. Please ensure the macro for CallBack function is defined with properly value to meet the specific time requirement.
3. Report EEPROM status routine will return the erasing cycles of the current ACTIVE block. This number is not an accurate value. If brownout occurs during updating erase cycle, this erasing cycle will be re-counted from the erase cycle value of other block.
4. EEPROM Emulation driver CANNOT be called in any interrupt service routine.
5. Interrupt vectors and service routines CANNOT reside in flash partitions used for emulation since these flash partitions are not accessible during EERPOM emulation operations.
6. It is strongly recommended that do NOT program or erase the same flash location while using EED to operate it.
7. Cache table is optional and if internal RAM size is large enough, it is suggested to enable it and provide with cache size = total number of EEPROM variables * 4 bytes.
8. Read buffer which use in read EEPROM operation should be large enough to store data size need to be read.
9. EED is in source code release, so the compiling optimization options may impact the correctness of the EED. Please make sure those options do not change the code logic.
10. EED will apply for an internal buffer from the stack, so the user's compiler should ensure this temporary buffer is on at least 4-byte alignment.
11. User needs to ensure that FSL_MainFunction() is called after every swap. User can check swap status global enumeration variable ***eraseStatus_Flag*** after writing data record to decide when needs to call the function.
12. User has to ensure that macro NUMBER_OF_ACTIVE_BLOCKS in "user_cfg.h" should always be less than number of blocks specified in EEPROM_CONFIG structure. Numbers of Alternative blocks are determined by subtracting total number of blocks by total number of Active blocks.
13. When ECC errors occurred during Flash data reads, either IVOR1 (Z7 or Z4D core) or IVOR2 (other Zen cores) exception will be invoked. It is recommended that the ME and EE bits in the MSR register are set to avoid check stop state. By default, when exiting from the exception handler, the instruction pointer will point to the instruction causing the exception to retry that violating instruction. This will cause an endless invoking of the exceptions since Flash ECC errors will be persistent until that Flash region is erased and reprogrammed.

Therefore, in the IVOR1/IVOR2 exception handlers for Flash ECC errors, we must increment the instruction pointer to point to the next instruction following the one causing the exception before. For applications using BookE instruction set, this is straightforward. We can always increment the instruction pointer by 4. For applications using VLE instruction set, there are 2 options: either we have to decode the instruction causing exception to determine to increment the instruction pointer by 2 or 4 in the exception handler (see example VLE exception handlers included in the release package), or we have to allocate all the Flash read instructions to a non-VLE section so that we can always increment the instruction pointer by 4 in the exception handler. For the latter option, we have call the function FSL_FlashRead to isolate all flash read instructions.

14. It is highly recommended that the D-cache of core should be disable at the initialization code to make sure the program/erase functions work properly

15. Flash controller buffer shall be disabled in the beginning of application for reading and writing to flash. And trying to re-configure flash controller during runtime can cause an unexpected behavior.