EEPROM Emulation Driver for M68HC908 Microcontrollers

by: Chen He
Technology Solution Organization, Libraries & Memories

1 Introduction

1.1 Overview

Electrically erasable, programmable, read-only memory (EEPROM), 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 the EEPROM through EEPROM emulation software.

The EEPROM emulation driver for the M68HC908 implements the fixed-length data record scheme on 0.5um SGF Flash. The EEPROM functionalities to be emulated include the following: organizing data records, initializing and de-initializing EEPROM, reporting EEPROM status, and reading, writing, and deleting data records. The demo code shows how to use the EEPROM emulation driver.

Table of Contents

1.2 Features

The EEPROM emulation driver for the M68HC908 provides the following features:

- Implements the fixed-length record scheme with two Flash sector clusters.
- Hierarchical design supports standalone/synchronous applications.
- All driver functions can run directly from Flash, except that the low-level, high-voltage (program/erase) functions have to run from RAM, to minimize RAM usage.
- Supports computer operating properly (COP) service every 240us @ a 8-MHz bus clock.
- Assembly source code release.
- C calling convention compliant.
- Ready-to-use demo illustrates the usage of the driver.

1.3 References

The following references were used to write this document:

1. AN2302r1: “EEPROM Emulation for the MC9S12C32”
2. AN2183: “Using FLASH as EEPROM on the MC68HC908GP32”
3. MPC5500EEWP: “EEPROM Emulation with MPC5500 Family Microcontrollers”
4. *CPU08 Central Processor Unit Reference Manual*
5. “MC68HC08JL3/H Technical Data,” Rev. 4
6. “MC68HC08GP32/H Technical Data,” Rev. 6, 08/2002
7. *HC908 SGF NVM Standard Software Driver User’s Manual*, V1.2

1.4 Acronyms and Abbreviations

The following references are used in this document:

- EE—EEPROM
- EED—EEPROM emulation driver
- SSD—Standard software driver

2 EEPROM Emulation Scheme

2.1 System Architecture

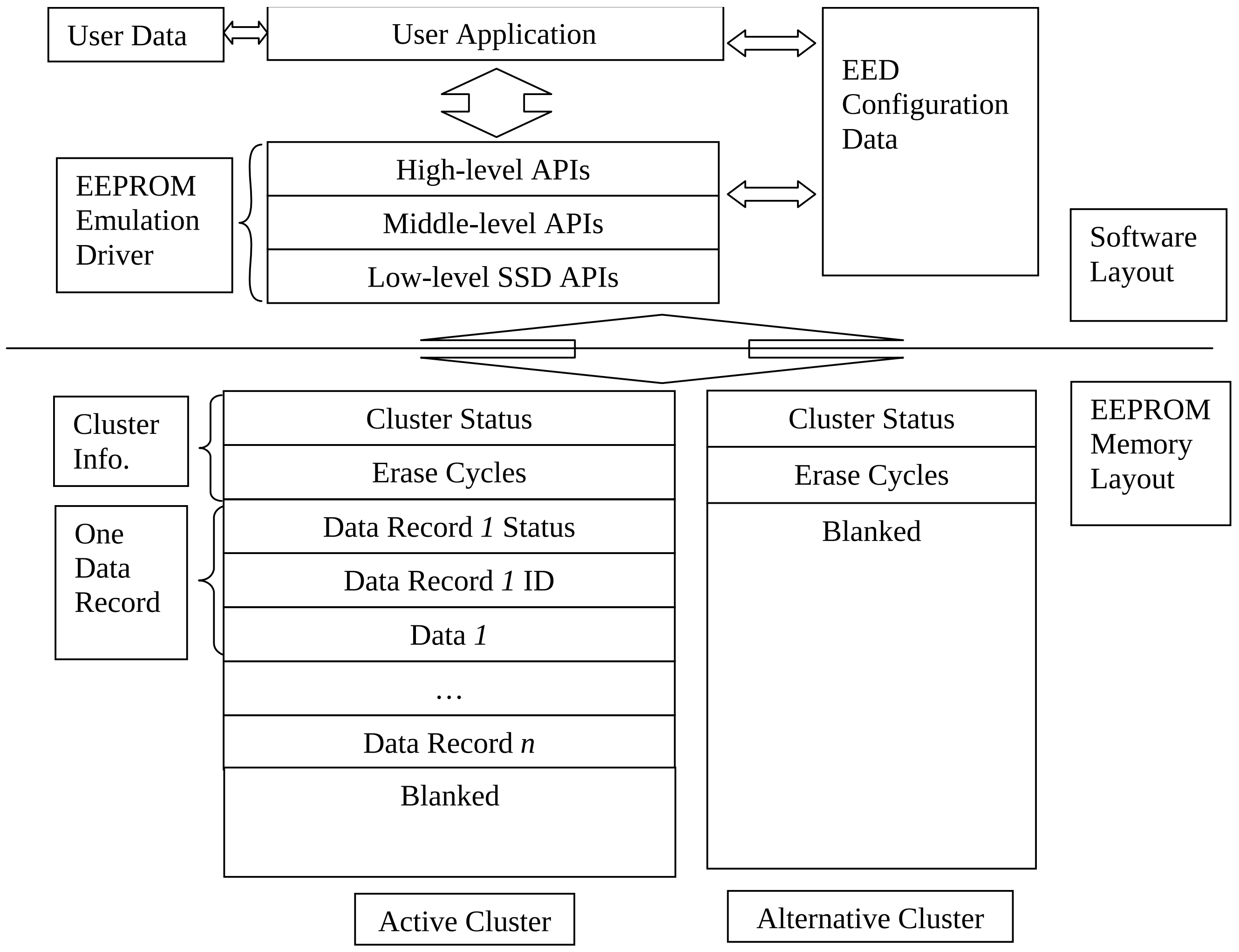

Figure 1 EEPROM Emulation Driver System Architecture

2.2 EEPROM Emulation Software Layout

2.2.1 EEPROM Emulation Configuration Data

The EEPROM emulation driver uses a set of global data. This global data is classified into the following categories:

- User interface data
- Active cluster configuration

- Global data used for low-level SSD functions
- Internally used variables

Considering the code size constraint, all global data is located in the direct page of M68HC908 parts (address from 0x00 to 0xFF), so that direct instructions can be used. The addresses of this global data can be varied within the direct page at compile time, but they cannot be moved to other locations during the run time (not position-independent data).

The total size of EEPROM configuration data is 31 bytes. Figure 2 depicts the memory layout of the global parameters.

Category	MSB LSB	Size
User Interface Global Data	recID	8 Bit
	erasingCycles	16 Bit
	failedAddress	16 Bit
	source	16 Bit
Active Cluster Configuration Data	activeIndex	8 Bit
	emuStartAddr	16 Bit
	emuEndAddr	16 Bit
	emuBlank	16 Bit
SSD Parameter Data	CLOCKSCALAR	16 Bit
	STARTADDR	16 Bit
	ENDADDR	16 Bit
	BUFFER	16 Bit
	FLASHCR	16 Bit
	FLASHPR	16 Bit
Internal Used Variables	hvType	8 Bit
	hvPosition	16 Bit
	nextRecID	8 Bit
	emuBuffer	16 Bit

Figure 2 Memory Layout of EEPROM Emulation Configuration Data

Before a user's application calls the high-level EEPROM emulation driver, the user interface global data has to be provided by the user. The active cluster configuration data is used by the EED to record the active cluster information. The EED driver calls low-level SSD functions to implement Flash operations. The EED sets the SSD parameter data before calling SSD functions. The internal used variables are used by the EED driver internally.

All of the above global parameters are defined in a parameter section named EMUParaSec in EED_Para.asm. For a detailed description of each parameter, please refer to Section 4.3, "Global Parameters and Macros."

2.2.2 User's Data Buffer

The RAM resource of M68HC908 parts is limited. Some parts only have 128 bytes of RAM. To satisfy the part with the least RAM, the RAM memory layout should be arranged carefully. The RAM was used mainly by the following items:

- Global parameters data
- Low-level, high-voltage functions, like FlashEraseCOP and FlashProgram
- Stack consumption in function and function calling chain

The user stack will grow from the higher address to the lower address, so the stack should be allocated at as high an address as possible. Figure 3 depicts an example of the RAM memory layout for the 128-byte MC68HC908JL3 part with a RAM range of 0x80~0xFF.

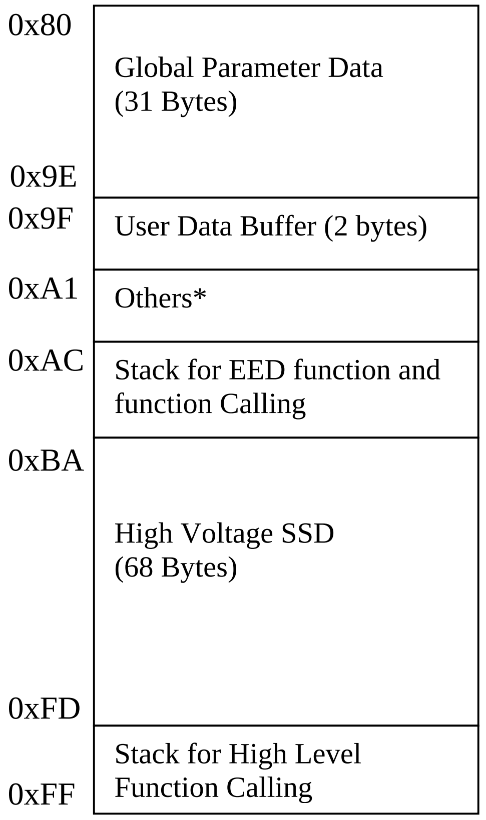
0x80
Global Parameter Data (31 Bytes)
0x9E
0x9F
User Data Buffer (2 bytes)
0xA1
Others*
0xAC
Stack for EED function and function Calling
0xBA
High Voltage SSD (68 Bytes)
0xFD
Stack for High Level Function Calling
0xFF

Figure 3 RAM Memory Layout Example for 128-byte Part

For 128-byte parts, 2 bytes of user buffer is reserved for the user, from 0x9F to 0xA0. Note that "Others" in Figure 3 is for a user employing Metrowerks CodeWarrior to debug into the low-level SSD functions; this 11-byte region will be occupied by CodeWarrior debugger. Otherwise, this region can be reserved as user buffer as well. The user can also adopt other memory layouts—Figure 3 is just one example. For a part with more RAM, the user can reserve more RAM for user data buffer.

2.2.3 EEPROM Emulation Driver

The EEPROM emulation driver has three levels of API: high, middle, and low:

- 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 provides the fundamental Flash operations.

The high level (user level) APIs provide the following EEPROM operations:

- FSL_InitEeprom—Initializes the Flash memory used for EEPROM emulation
- FSL_ReadEeprom—Reads the specific data record from emulated EEPROM
- FSL_WriteEeprom—Writes a data record to emulated EEPROM
- FSL_DeleteRecord—Deletes a data record from emulated EEPRO
- FSL_ReportEepromStatus—Reports the status of the emulated EEPROM
- FSL_DeinitEeprom—De-initializes the Flash memory used for EEPROM emulation

2.3 EEPROM Emulation Memory Layout

2.3.1 EEPROM Cluster

Erasable Page n
Erasable Page n+1
…
Erasable Page n+k–1
Erasable Page m
…
Erasable Page m+k–1

Figure 4 EEPROM Cluster

The EEPROM emulation driver adopts the M68HC908 family's Flash to emulate the EEPROM. Two clusters are needed for EEPROM emulation: active and alternative. Each cluster contains one or more contiguous erasable pages. These two clusters cannot overlap each other, but they do not need to be contiguous. They should contain the same number of pages, and this number is user configurable.

2.3.2 EEPROM Data Organization

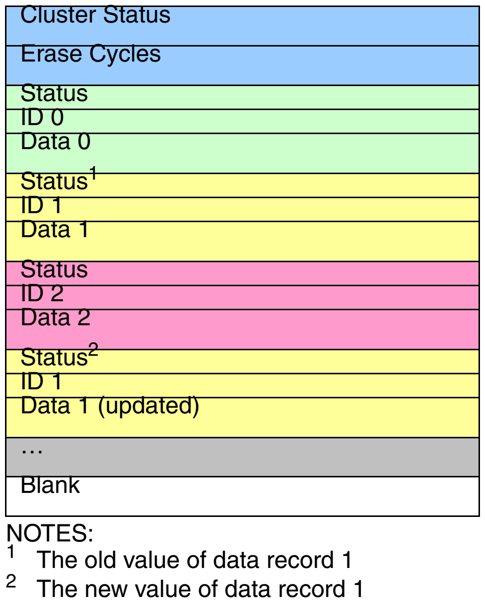

Figure 5 EEPROM Cluster Memory Layout

Each emulation cluster contains the following:

- Cluster status field

  Stores the cluster status, whose values are selected properly, so that it can be programmed several times
- Erasing cycles

  Stores the cluster erasing cycles because the EEPROM emulation is set up; will be accumulated after each erasure
- Data records field

  One data record has three fields:

  — Data record status field: the data record status

  — Data record ID: the data record identifier

  — Data: user’s raw data
- Blank field

  Free space for storing new data records

There should only be one cluster marked as active, and the other should be marked as alternative. If two clusters are marked as active, the cluster with more blank space will be the final active cluster.

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

The data record cannot be updated directly on the same location. Instead, a new record with the new value will be written to the EEPROM, and the read routine will check the latest valid one as the final value of specific data.

2.4 EEPROM Emulation Operations

2.4.1 Initialize EEPROM

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

- Using EEPROM for the first time

  The EED formats the two clusters, then assigns one as active and the other as alternative.
- Continuing to use EEPROM

  The EED determines which cluster is active and initializes the alternative one.

  When there are two clusters marked as active (power down during swapping), the initialization routine will scan through the two clusters and determine the free space size of each cluster. The one with more free space becomes the final active cluster.

2.4.2 Write EEPROM Data

Because the SGF Flash memory cell cannot be erased individually, the EED must write a new data record with the same data ID and updated value from the EEPROM blank area when the data needs updating.

After updating several times, the active cluster may not have enough free space to write a new data record. It has to copy all the latest data records to the alternative cluster to clean up the EEPROM. This procedure is called ‘swapping’. Afterwards, the alternative cluster will become the new active cluster, and the old active cluster will become the new alternative cluster.

During cluster swapping, there are several cluster statuses and swapping stages to recover from an accident:

Cluster statuses:

- CLUSTER_STATUS_ERASED = $FFFF

  The cluster is fully erased and not initialized as the alternative cluster.
- CLUSTER_STATUS_BLANKED = $0FFF

  The cluster is initialized as the alternative cluster and ready for swapping.
- CLUSTER_STATUS_STARTED = $00FF

  The latest data records are copied from the active cluster to this cluster, but the data record cannot be accessed through this cluster.
- CLUSTER_STATUS_ACTIVE = $000F

  All the latest data records are copied completely, and the data accessing can target this cluster.

Swapping stages: (assuming cluster A is the active cluster and cluster B is the alternative cluster)

- Cluster A is CLUSTER_STATUS_ACTIVE and cluster B is CLUSTER_STATUS_BLANKED.

  This is the initial state of cluster swapping.
- Cluster A is CLUSTER_STATUS_ACTIVE and cluster B is CLUSTER_STATUS_STARTED.

  The data record copying is in progress. If swapping fails in this stage, all the data record accesses can also direct to the original active cluster, cluster A. Cluster B should be re-initialized as the alternative one (CLUSTER_STATUS_BLANKED state) and perform the swapping again.
- Cluster A is CLUSTER_STATUS_ACTIVE and cluster B is also CLUSTER_STATUS_ACTIVE.

  All the latest data records are copied completely and the data accessing can target cluster B. If swapping fails in this stage, the user only needs to initialize cluster A as the alternative cluster (*CLUSTER_STATUS_BLANKED* state).
- Cluster A is CLUSTER_STATUS_ERASED and cluster B is CLUSTER_STATUS_ACTIVE.

  Cluster A is fully erased but not ready for emulation. It is needed to continue initializing cluster A as the alternative cluster (CLUSTER_STATUS_BLANKED state).
- Cluster A is CLUSTER_STATUS_BLANKED and cluster B is CLUSTER_STATUS_ACTIVE.

  This is the final state of cluster swapping.

2.4.3 Read EEPROM Data

There will be several data records in EEPROM with same data ID (because of data updating), so the reading routine should identify the latest copy of the data record by scanning the entire active cluster from the first data record to the blank region.

Each data record will have a status field to identify the state of this record:

- RECORD_STATUS_ERASED = $FF; no data record
- RECORD_STATUS_STARTED = $CF; data record is invalid and record ID and record data may be partially programmed.
- RECORD_STATUS_COMPLETED = $0F; data record is valid.
- RECORD_STATUS_DELETED = $0C; data record has been deleted

If the data record state is RECORD_STATUS_DELETED, the data record is not physically removed in Flash. This record is invalid now, so skip this record by its fixed length.

If the data record state is RECORD_STATUS_STARTED or another value that cannot be recognized, this record is partially programmed or corrupted. Skip the fixed length data record to get the next data record start address.

2.4.4 Delete EEPROM Data

If the data is not needed, it can be deleted from the emulated EEPROM. The EED does not physically remove this record at the time a user wants to delete it. Instead, the EED will only change the record's state to RECORD_STATUS_DELETED so that it is regarded as unnecessary data and will be removed from emulated EEPROM in cluster swapping. This method can shorten the deleting time.

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

2.4.5 Report EEPROM Status

The cluster erasing cycles will be fetched from the cluster and it reflects the erasure times since the EEPROM has been setup. It is only an approximated number and will be set to 0 when first time using EEPROM.

2.4.6 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.

2.5 Limitation of COP Support

As mentioned before, the EEPROM emulation driver for the M68HC908 can service COP during the execution of driver. However the EEPROM emulation driver can only use software logic, instead of the hardware timer, to control the frequency of servicing COP. The time scale of this software logic depends on the bus clock frequency. On the other hand, COP runs at the oscillator clock (OSC) directly. So there might be some combinations of OSC clock and bus clock for which the COP service logic would time out. In order to avoid COP timing out, the user should ensure setting the OSC and bus clocks according to the following formula:

$$\text{OSC Clock} \le 4 * \text{Bus Clock} \qquad \textit{Eqn. 1}$$

2.6 Notes and Limitations

When using the EED, note the following items:

- It is not suggested to use the middle level APIs of the EED directly.
- The Flash module 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.
- All the input parameters for the EED are defined as global variables, which should reside in the direct page of RAM (Z_PAGE, address from 0x00 to 0xFF).
- The EED service COP every 240us @ a 8-MHz bus clock.
- Report EEPROM status routine will return the erasing cycles of the cluster. However, this number might not be accurate since it will be reset to 0 when the cluster status is invalid due to any unexpected failure.
- EEPROM emulation driver cannot be called in any interrupt service routine.
- Interrupt vectors and service routines cannot reside in Flash because Flash is not accessible during EERPOM emulation operations.

- It is strongly recommended not to program or erase the same Flash location while using the EED to operate it.
- The EED is in assembly source code release, so the compiling optimization options do not impact the correctness of the EED.

3 Preparation and Running Environment

3.1 Preparation

3.1.1 EEPROM Emulation Driver Configuration

The EEPROM emulation driver needs the user to provide following information:

- Flash erase page size
- User raw data length in a data record
- Start addresses of the two Flash pages clusters used for emulation
- The number of Flash pages used for each page cluster
- Bus clock
- Addresses of the Flash control and Flash block protection registers

3.1.2 Macros

Before using the EEPROM emulation driver, the following macros should be set properly to meet a specific need:

- EED_ERASE_PAGE_SIZE

  The Flash modules embedded in the M68HC908 series have two sizes of Flash page for erasing: 28 and 64 bytes. Before using the EED Flash driver on a M68HC908 part, it is very important to set the correct size of Flash erase page. The macro EED_ERASE_PAGE_SIZE is used to control the size of Flash page:

  The default setting is:

  EED_ERASE_PAGE_SIZE: EQU $80
- USER_DATA_LENGTH

  The user data length in a fixed length record. The user data length is configurable via this macro before compilation. Its valid value range is [1, 0xFD]. Otherwise, a compilation error message “User data length is out of range [1, 0xFD]!” will be reported.

  The default setting is:

  USER_DATA_LENGTH: EQU $2
- CLUSTER_0_START and CLUSTER_1_START

  The starting address for two clusters. These two values should be page alignment.

The default settings are:

CLUSTER_0_START: EQU $C000

CLUSTER_1_START: EQU $C100

- PAGES_PER_CLUSTER

  The number of pages in each cluster. This value should not be 0. The two cluster ranges decided by macros of CLUSTER_0_START, CLUSTER_1_START, and PAGES_PER_CLUSTER should not be overlapped. Otherwise, a compilation error message "Cluster configuration is incorrect!" will be reported.

  The default settings are:

- PAGES_PER_CLUSTER: EQU $2

  Two macros for cluster ending address CLUSTER_0_END and CLUSTER_1_END were derived based on CLUSTER_0_START, CLUSTER_1_START and PAGES_PER_CLUSTER.

- BUS_CLOCK

  The bus clock in Hz, BUS_CLOCK, was defined in "SSD_Flash.inc". User should configure this macro with appropriate value accordingly before using EEPROM emulation driver. The default value is 2.4576 MHz:

  BUS_CLOCK: EQU 2457600

  This macro was used to set the global variable of CLOCKSCALAR. Please refer to Section 4.3.3, "SSD Global Parameters" for more details.

- FLCR and FLBPR

  The macros for the addresses of the Flash control and Flash block protection registers were defined in EED_Flash.inc. The two default values for FLCR and FLBPR were $FE08 and $FE09 individually.

  FLCR EQU $FE08

  FLBPR EQU $FE09

  These two macros were used to set the SSD global variables FLASHCR and FLASHPR respectively before calling SSD functions. Please refer to Section 4.3.3, "SSD Global Parameters for more details.

3.2 Use EEPROM Emulation Driver

The EEPROM emulation driver is designed to support standalone applications only.

This type of applications calls the EEPROM emulation driver routines and waits until the EEPROM operation complete.

The EEPROM emulation driver should be run in a synchronous environment. It cannot be interrupted until it finishes execution. Please refer to the demo for details.

4 EEPROM Emulation Driver

4.1 Function Introduction

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

Normally the user's application will simply call the high-level APIs directly. If finer scheduling granularity is required, middle-level or low-level APIs can be called in the application.

An example of how to use the APIs can be found in the demos in the release package of EEPROM emulation driver for HC908.

- High level APIs (user level APIs):

  These APIs provide direct operations on emulated EEPROM such as initializing EEPROM, reading record, writing record, deleting data record, reporting EEPROM status, and de-initializing EEPROM.
  - FSL_InitEeprom—Initializes the Flash memory used for EEPROM emulation
  - FSL_ReadEeprom—Reads the specific data record from emulated EEPROM
  - FSL_WriteEeprom—Writes a data record to emulated EEPROM
  - FSL_DeleteRecord—Deletes a data record from emulated EEPROM
  - FSL_ReportEepromStatus—Reports the status of the emulated EEPROM
  - FSL_DeinitEeprom—De-initializes the Flash memory used for EEPROM emulation
- Middle level APIs:

  These APIs provide some individual functionality to support the high level APIs on operating, emulated EEPROM.
  - FSL_Erase—Erases the continuous Flash pages
  - FSL_Program—Programs the data into Flash memory
  - FSL_CopyRecord—Copies one data record to the Flash memory
  - FSL_InitCluster—Initializes one cluster, includes erasing this cluster, blank check, and updating its status to CLUSTER_STATUS_BLANKED
  - FSL_SwapCluster—Copies the latest data records from the active cluster to the alternative cluster while the active cluster is full
  - FSL_SearchRecord—Searches the required data record ID in the cluster
- Low level APIs:

  These APIs are basic Flash operations and composed of a subset of the standard software driver for the M68HC908.
  - FlashEraseCOP—Erases a single Flash logical page with COP service
  - BlankCheck—Checks if a specific Flash range is erased (0xFFs)
  - FlashProgram—Programs data into data Flash
  - ProgramVerify—Verifies that the programmed data is same as the source data

4.2 Function Calling Relationship

Table 4-1 shows the calling relationship among the three hierarchies of APIs.

Table 1. EEPROM Emulation Driver Calling Relationship

API Hierarchies	Function name	Caller functions	Functions to be called
High Level APIs	FSL_InitEeprom	User's applications	FSL_InitCluster (middle) FSL_Program (middle) FSL_SearchRecord (middle)
	FSL_ReadEeprom	User's applications	FSL_SearchRecord (middle)
	FSL_WriteEeprom	User's applications	FSL_SwapCluster (middle) FSL_CopyRecord (middle)
	FSL_DeleteRecord	User's applications	FSL_SearchRecord (middle) FSL_Program (middle)
	FSL_ReportEepromStatus	User's applications	FSL_SearchRecord (middle)
	FSL_DeinitEeprom	User's applications	FSL_Erase (middle)
Middle Level APIs	FSL_Erase	FSL_DeinitEeprom (high) FSL_InitCluster (middle)	FlashEraseCOP (low) BlankCheck (low)
	FSL_Program	FSL_InitEeprom (high) FSL_DeleteRecord (high) FSL_SwapCluster (middle) FSL_CopyRecord (middle) FSL_InitCluster (middle)	FlashProgram (low) ProgramVerify (low)
	FSL_CopyRecord	FSL_WriteEeprom (high) FSL_SwapCluster (middle)	FSL_Program (middle)
	FSL_InitCluster	FSL_InitEeprom (high) FSL_SwapCluster (middle)	FSL_Erase (middle) FSL_Program (middle)
	FSL_SwapCluster	FSL_WriteEeprom (high)	FSL_Program (middle) FSL_SearchRecord (middle) FSL_CopyRecord (middle) FSL_InitCluster (middle)
	FSL_SearchRecord	FSL_InitEeprom (high) FSL_ReadEeprom (high) FSL_DeleteRecord (high) FSL_ReportEepromStatus (high) FSL_SwapCluster (middle)	-
Low Level APIs	FlashEraseCOP	FSL_Erase (middle)	-
	BlankCheck	FSL_Erase (middle)	-
	FlashProgram	FSL_Program (middle)	-
	ProgramVerify	FSL_Program (middle)	-

4.3 Global Parameters and Macros

4.3.1 EEPROM Emulation Interface Global Parameters

The EEPROM emulation interface global parameters are input or output arguments for some high level EED functions.

Table 2. EEPROM Emulation Interface Global Parameters Definitions

Name	Size	I/O Type	Function Used	Description
recID	1 byte	Input	FSL_ReadEeprom FSL_WriteEeprom FSL_DeleteRecord	The identifier of the data record to be operated.
erasingCycles	2 bytes	Output	FSL_ReportEepromStatus	The erasing cycles of the active cluster.
failedAddress	2 bytes	Output	FSL_ReportEepromStatus	The starting address of the first invalid record.
source	2 bytes	Input	FSL_WriteEeprom	The starting address of the data for write or read.
		Output	FSL_ReadEeprom	

4.3.2 Active Cluster Global Parameters

The cluster global parameters provide the following:

- Index of the active cluster
- Starting address of the active cluster
- End address of the active cluster
- Start address of the free Flash memory available for new records in the active cluster

Table 3. Active Cluster Global Parameters Definitions

Name	Size	Description
activeIndex	1 byte	Indicating which cluster is active.
emuStartAddr	2 bytes	Start address of the cluster, included.
emuEndAddr	2 bytes	End address of the cluster, included.
emuBlank	2 bytes	Start address of the free Flash memory available for new records in active cluster.

4.3.3 SSD Global Parameters

The SSD global parameters are for the SGF SSD Flash driver. They should be set properly before calling SSD functions.

Table 4. SSD Configuration Definitions

Name	Size	Description
CLOCKSCALAR	1 byte	The scaling factor based on the bus clock used for Flash operations.[1]
STARTADDR	2 bytes	The start address of the Flash area to be operated.
ENDADDR	2 bytes	The end address of the Flash area to be operated.
BUFFER	2 bytes	The start address of the source data buffer used for programming and verification.
FLASHCR	2 bytes	Address of the Flash control register.[2]
FLASHPR	2 bytes	Address of the Flash protection register. [2]

NOTES:
[1] CLOCKSCALAR is used to manually control the delay timings for Flash operation via software instructions. It equals 8 times the value of the bus clock in MHz and then rounded down to the nearest integer. The formula is shown as following:
CLOCKSCALAR = INT [Bus Clock (MHz) * 8]
For example, if the bus clock is 2.45 MHz, 8 times of the bus clock in MHz is 19.6, so that CLOCKSCALAR should be set to 19.
BUS_CLOCK, the bus clock in Hz was defined in "*SSD_Flash.inc*". Then the macro of NVM_CLOCK_SCALAR, also defined in "*SSD_Flash.inc*", can be used to set the global variable of CLOCKSCALAR.
NVM_CLOCK_SCALAR: EQU ((BUS_CLOCK * 8) / 1000000)
The following assembly instruction shows how to use the above macro to set the value of CLOCKSCALAR.

```
MOV    #NVM_CLOCK_SCALAR, CLOCKSCALAR
```

[2] The two SSD global parameters, FLASHCR and FLASHPR, should be set correctly before calling SSD functions. Two macros, FLCR and FLBPR, were defined in EED_Flash.inc. The following assembly instructions show how to use the above macros to set the value of FLASHCR and FLASHPR.

```
LDHX    #FLCR
STHX    FLASHCR    ; set Flash control register address
LDHX    #FLBPR
STHX    FLASHPR    ; set Flash block protection register address
```

For details about the SSD global parameters, please refer to the *SGF NVM SSD for HC908 User's Manual*.

4.3.4 Internal Used Global Parameters

These parameters are internal used by the EEPROM emulation driver. To save the code size, they are also defined as global variables located in the direct page.

Table 5. Internal Used Global Parameters Definitions

Name	Size	Description
hvType	1 byte	The type of High Voltage operation in stack. 0x5A: FlashProgram, 0xA5: FlashEraseCOP.
hvPosition	2 bytes	The RAM location of High Voltage operation locates.
nextRecID	1 byte	To save the valid record ID for the next parsing of the cluster. During cluster swapping, this parameter can skip the non-existing record IDs.
emuBuffer	2 bytes	2-byte internal buffer, used to hold the data for programming or save the location of latest record.

4.3.5 Macros

4.3.5.1 Record Length Configuration

The macro USER_DATA_LENGTH defines the user data length of a fixed length record in byte. It is defined in head file EED_Flash.inc.

The default macro value is set to 2. The user may modify it according to their special needs, provided that the value doesn't exceed the range [1, 0xFD]. Otherwise, an invalid value will lead to the compilation error message, "User data length is out of range [1, 0xFD]!".

Some other macros derived from USER_DATA_LENGTH are listed below.

Table 6. Macros Derived From USER_DATA_LENGTH

Name	Description
MIN_DATA_LENGTH	The minimum length of user data in a record.
MAX_DATA_LENGTH	The maximum length of user data in a record.
RECORD_LENGTH	The total record length. It is composed of user data, record status and record ID, i.e., USER_DATA_LENGTH + 2.
RECORD_ID_MAX	The maximum value of record identifier. It's reserved for EED internal usage.

4.4 Function Return Code

Table 7. Function Return Code

Name	Value	Description
EE_OK	0x00	The requested operation was successful.
EE_ERROR_NOT_BLANK	0x30	The Flash memories are not blank.
EE_ERROR_VERIFY	0x40	Corresponding source data and content of destination location mismatch.
EE_ERROR_NOMEM	0x50	No enough EEPROM memory.

Table 7. Function Return Code (continued)

EE_ERROR_NOFND	0x60	Record not found in cluster.
EE_ERROR_CSTAT	0x70	Cluster status error.
EE_ERROR_RSTAT	0x80	Record status error.
EE_ERROR_IDRNG	0x90	Record identifier exceeds the valid range.

4.5 High Level Functions (User Level Functions)

4.5.1 FSL_InitEeprom

4.5.1.1 Description

FSL_InitEeprom determines the active cluster based on the user's EEPROM configuration macros.

4.5.1.2 Procedure

1. Allocate some space for high voltage function on stack;
2. Check the cluster status field to find the active cluster;
3. If no active cluster is found, erase, blank check the first cluster, change its status to CLUSTER_STATUS_ACTIVE, and set active cluster index to 0 in global parameters;
4. If the active cluster is found, save its index;
5. Erase, blank check, and then change the alternative cluster status to CLUSTER_STATUS_BLANKED;
6. De-allocate the stack for high voltage function;
7. Return.

4.5.1.3 Definition

```
unsigned char FSL_InitEeprom (void);
```

4.5.1.4 Global Variables Refereed

Table 8. Global Variables for FSL_InitEeprom

Argument	Size	Description	Remark
activeIndex	1 byte	Indicate which cluster is active.	Based on the status of two clusters, the active one should be selected. Used as an output argument.
emuStartAddr	2 bytes	The starting address of the active cluster	Used as an output argument.
emuEndAddr	2 bytes	The end address of the active cluster	Used as an output argument.
emuBlank	2 bytes	The blank location of the active cluster.	Used as an output argument.

4.5.1.5 Return Values

Table 9. Return Values for FSL_InitEeprom

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOT_BLANK EE_ERROR_VERIFY

4.5.1.6 Calling Relationship

FSL_InitEeprom:

- FSL_SearchRecord
- FSL_InitCluster
  - — FSL_Erase
  - — FSL_Program
- FSL_Program

4.5.1.7 Tips

The starting address of both clusters should be configured as page alignment. The two clusters have the same number of Flash pages, and the two ranges of clusters should not be overlapped.

If it is the first time using EEPROM emulation, FSL_InitEeprom will initialize both clusters and select the first one as the active cluster. If continuing to use the emulation, FSL_InitEeprom will determine which cluster is the active one and initialize the alternative cluster.

Once there are two cluster marked as active, FSL_InitEeprom will scan two clusters and choose the one has more free space than the final active cluster, then initialize the other one as alternative cluster.

If any record in the active cluster has an invalid record status, FSL_InitEeprom will set “emuBlank” to “emuEndAddr”, which will result in a cluster swapping in the following record writing.

Interface global parameters “recID”, “source”, and “failedAddress” are reused in this function.

4.5.1.8 Troubleshooting

Table 10. Troubleshooting for FSL_InitEeprom

Returned Error	Description	Solution
EE_ERROR_NOT_BLANK	The Flash memories are not blank.	Check the Flash memory if it can be erased correctly.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.5.2 FSL_ReadEeprom

4.5.2.1 Description

This function reads the specific data record. The starting address of the record data will be returned.

4.5.2.2 Procedure

1. Check that data ID is within the valid ID range;
2. Check the data in the active cluster;
3. If data ID is not found in the active block, return error code;
4. Get the starting address of the record data;
5. Return.

4.5.2.3 Definition

```
unsigned char FSL_ReadEeprom (void);
```

4.5.2.4 Global Parameters Referred

Table 11. Global Parameters for FSL_ReadEeprom

Argument	Size	Description	Remark
recID	1 byte	The required data ID.	Can be any value from 0 ~ (RECORD_ID_MAX – 1).
source	2 bytes	The address of record data.	Used as an output argument.
emuStartAddr	2 bytes	The starting address of the active cluster.	
emuEndAddr	2 bytes	The end address of the active cluster.	
emuBlank	2 bytes	The blank location of the active cluster.	

4.5.2.5 Return Values

Table 12. Return Values for FSL_ReadEeprom

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOFND EE_ERROR_IDRNG

4.5.2.6 Calling Relationship

FSL_ReadEeprom

- FSL_SearchRecord

4.5.2.7 Tips

This function does not read out the data into the user's buffer. Instead, it returns the starting address of the user data in the data record and saves this address into "source" variable.

Interface global parameter "failedAddress" is reused in this function.

4.5.2.8 Troubleshooting

Table 13. Troubleshooting for FSL_ReadEeprom

Returned Error	Description	Solution
EE_ERROR_NOFND	Record not found in cluster.	Write the data record to the EEPROM first.
EE_ERROR_IDRNG	Record identifier exceeds the valid range.	Check the data ID, which should be within the valid ID range.

4.5.3 FSL_WriteEeprom

4.5.3.1 Description

This function writes a data record to the EEPROM emulation. While there is not enough free space in the active cluster, this routine will perform cluster swapping to clean up the EEPROM.

4.5.3.2 Procedure

1. Allocate some space for high voltage function on stack;
2. Check the validity of user provided record ID;
3. Check if there is enough free space for the data to be written in the active cluster;
4. If not enough free space, perform cluster swapping. After swapping, if still not enough free space in the active cluster, return error code;
5. Write the data record to the active cluster;

6. De-allocate the stack for high voltage function;
7. Return.

4.5.3.3 Definition

```
unsigned char FSL_WriteEeprom (void);
```

4.5.3.4 Global Parameters Referred

\

Table 14. Global Parameters for FSL_WriteEeprom

Argument	Size	Description	Remark
activeIndex	1 byte	Indicate which cluster is active.	If swapping occurred, activeIndex should be updated.
recID	1 byte	The required data ID.	Can be any value from 0 ~ (RECORD_ID_MAX – 1).
source	2 bytes	Address of the record data for writing.	Can be any accessible address. No alignment limitation. This parameter should be set just before calling FSL_WriteEeprom.
emuStartAddr	2 bytes	The starting address of the active cluster.	Used as an input and output argument.
emuEndAddr	2 bytes	The end address of the active cluster.	Used as an input and output argument.
emuBlank	2 bytes	The blank location of the active cluster.	Used as an input and output argument.

4.5.3.5 Return Values

Table 15. Return Values for FSL_WriteEeprom

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_IDRNG EE_ERROR_NOMEM EE_ERROR_CSTAT EE_ERROR_NOT_BLANK EE_ERROR_VERIFY

4.5.3.6 Calling Relationship

FSL_WriteEeprom

- FSL_SwapCluster
  — FSL_Program
  — FSL_SearchRecord

  - FSL_CopyRecord
    - FSL_Program
  - FSL_InitCluster
    - FSL_Erase
    - FSL_Program
- FSL_CopyRecord
  - FSL_Program

4.5.3.7 Tips

The record data length is limited within the values of MIN_DATA_LENGTH and MAX_DATA_LENGTH, which are configured in EED_Flash.inc. Please refer to Section 4.3.5, "Macros" for details. The record data length cannot be set to 0, which will result in a compiling error.

If there is not enough free space for writing a new data record, FSL_WriteEeprom will trigger a cluster swapping to copy all the latest valid data copies to the alternative cluster and remove the old and invalid data records. This procedure may take a little longer (depending on the number of valid data records) to complete the write operation.

Interface global parameters "erasingCycles" and "failedAddress" are reused in this function.

4.5.3.8 Troubleshooting

Table 16. Troubleshooting for FSL_WriteEeprom

Returned Error	Description	Solution
EE_ERROR_IDRNG	Record identifier exceeds the valid range.	Check the data ID, which should be within the valid ID range.
EE_ERROR_NOMEM	No enough EEPROM memory.	Use more Flash pages to enlarge the emulated EEPROM size.
EE_ERROR_CSTAT	Cluster status error.	Call FSL_InitEeprom to re-initialize the EEPROM.
EE_ERROR_NOT_BLANK	The Flash memories are not blank.	Check the Flash memory if it can be erased correctly.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.5.4 FSL_DeleteRecord

4.5.4.1 Description

This function deletes a data record in EEPROM. It does not physically remove the data record from the EEPROM. Instead, it only changes the record status to a special value (RECORD_STATUS_DELETED). This deleted record will be discarded during the next cluster swapping.

4.5.4.2 Procedure

1. Allocate some space for high voltage function on stack;
2. Check the validity of the record ID;
3. Search the record in the active cluster;
4. If it is found, update the record status to *RECORD_STATUS_DELETED*;
5. If it is not found, return error code;
6. De-allocate the stack for high voltage function;
7. Return.

4.5.4.3 Definition

```
unsigned char FSL_DeleteRecord (void);
```

4.5.4.4 Global Parameters Referred

Table 17. Global Parameters for FSL_DeleteRecord

Argument	Size	Description	Remark
recID	1 byte	The required data ID.	Can be any value from 0 ~ (RECORD_ID_MAX – 1).
emuStartAddr	2 bytes	The starting address of the active cluster.	
emuEndAddr	2 bytes	The end address of the active cluster.	
emuBlank	2 bytes	The blank location of the active cluster.	

4.5.4.5 Return Values

Table 18. Return Values for FSL_DeleteRecord

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOFND EE_ERROR_IDRNG EE_ERROR_VERIFY

4.5.4.6 Calling Relationship

FSL_DeleteRecord

- FSL_SearchRecord
- FSL_Program

4.5.4.7 Tips

The deleted data record can be re-written to EEPROM when needed.

Interface global parameter “failedAddress” is reused in this function.

4.5.4.8 Troubleshooting

Table 19. Troubleshooting for FSL_DeleteRecord

Returned Error	Description	Solution
EE_ERROR_NOFND	Record not found in cluster	Check the data record to be deleted.
EE_ERROR_IDRNG	Record identifier exceeds the valid range.	Check the data ID, which should be within the valid ID range.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.5.5 FSL_DeinitEeprom

4.5.5.1 Description

This function releases all the Flash used to EEPROM emulation. After de-initialized, the Flash pages for emulation are fully erased.

4.5.5.2 Procedure

1. Allocate some space for high voltage function on stack;
2. Erase the Flash pages used for EEPROM emulation;
3. De-allocate the stack for high voltage function;
4. Return.

4.5.5.3 Definition

```
unsigned char FSL_DeinitEeprom (void);
```

4.5.5.4 Global Parameters Referred

None.

4.5.5.5 Return Values

Table 20. Return Values for FSL_DeinitEeprom

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOT_BLANK

4.5.5.6 Calling Relationship

FSL_DeinitEeprom

- FSL_Erase

4.5.5.7 Tips

Only the Flash pages used for EEPROM emulation will be erased.

4.5.5.8 Troubleshooting

Table 21. Troubleshooting for FSL_DeinitEeprom

Returned Error	Description	Solution
EE_ERROR_NOT_BLANK	The Flash memories are not blank	Check the Flash memory if it can be erased correctly.

4.5.6 FSL_ReportEepromStatus

Reports cluster erasure cycles and checks the cluster and data record statuses.

4.5.6.1 Procedure

1. Check both cluster statuses, if any invalid value existing, return error code;
2. Fetch the erasing cycles from active cluster erasing cycle field;
3. Go through entire active cluster to check the data record status with record ID of RECORD_ID_MAX. If the status of any data record is not correct, fill its address to "failedAddress" and set the error return code;
4. Return.

4.5.6.2 Definition

```
unsigned char FSL_ReportEepromStatus (void);
```

4.5.6.3 Global Parameters Referred

Table 22. Global Parameters for FSL_ReportEepromStatus

Argument	Size	Description	Remark
activeIndex	1 byte	Indicate which cluster is active.	
erasingCycles	2 bytes	Used to save the erasing cycles read from the Flash block array.	It is similar value of erasing cycles. And it is only valid when the block status is valid.
failedAddress	2 bytes	Used to save the failed data record address.	It is only valid while function returns EE_ERROR_RSTAT
emuStartAddr	2 bytes	The starting address of the active cluster.	
emuEndAddr	2 bytes	The end address of the active cluster.	
emuBlank	2 bytes	The blank location of the active cluster.	If invalid record existing in the active cluster, emuBlank will be set to the end of the cluster.

4.5.6.4 Return Values

Table 23. Return Values for FSL_ReportEepromStatus

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_CSTAT EE_ERROR_RSTAT

4.5.6.5 Calling Relationship

FSL_ReportEepromStatus

- FSL_SearchRecord

4.5.6.6 Tips

The erasure cycles of the cluster is counted from the time EEPROM is set up and will be accumulated after each erasure. Once the EEPROM is de-initialized, the erasure cycles will be reset to 0.

This function can only check the cluster and data record status. It cannot report error when the user's data are incorrect.

If any of the records have invalid record status, FSL_ReportEepromStatus will set “emuBlank” to “emuEndAddr”, which will result in a cluster swapping in the following record writing.

The value of failedAddress is valid only when the return code is EE_ERROR_RSTAT. And the value of erasingCycles is valid only when the return code is not EE_ERROR_CSTAT.

4.5.6.7 Troubleshooting

Table 24. Troubleshooting for FSL_ReportEepromStatus

Returned Error	Description	Solution
EE_ERROR_CSTAT	Cluster status error.	Call FSL_InitEeprom to re-initialize the EEPROM.
EE_ERROR_RSTAT	Record status error.	Re-write the record into EEPROM.

4.6 Middle Level Functions

4.6.1 FSL_Erase

4.6.1.1 Description

FSL_Erase will erase a range of contiguous Flash pages and verify them. It encapsulates two low-level SSD functions: FlashEraseCOP and BlankCheck. Input parameters are used directly without any check.

4.6.1.2 Procedure

1. Check if FlashEraseCOP function is already in stack. If not, copy it into stack;
2. Call SSD FlashEraseCOP to erase a single Flash page;
3. Call SSD BlankCheck to verify the Flash page;
4. Repeat steps 3 and 4 until all Flash pages are erased and verified;
5. Return.

4.6.1.3 Definition

```
unsigned char FSL_Erase (void);
```

4.6.1.4 Global Parameters Referred

Table 25. Global Parameters for FSL_Erase

Argument	Size	Description	Remark
STARTADDR	2 bytes	The starting address of the first Flash pages to be erased.	It should be page alignment.
ENDADDR	2 bytes	The end address of the last Flash pages to be erased.	

4.6.1.5 Return Values

Table 26. Return Values for FSL_Erase

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOT_BLANK

4.6.1.6 Calling Relationship

FSL_Erase

- FlashEraseCOP (SSD)
- BlankCheck (SSD)

4.6.1.7 Tips

At the entry of this function, it checks if the required high voltage function is on stack or not by evaluating the value of hvType. If not, copy the required high voltage function onto stack specified by hvPostion.

This function requires that the STARTADDR should be on Flash page alignment.

FlashEraseCOP can erase only one Flash page per call. The ENDADDR is required to be pushed onto stack before calling FlashEraseCOP. FSL_Erase will erase Flash pages one by one.

4.6.1.8 Troubleshooting

Table 27. Troubleshooting for FSL_Erase

Returned Error	Description	Solution
EE_ERROR_NOT_BLANK	The Flash memories are not blank	Check the Flash memory if it can be erased correctly.

4.6.2 FSL_Program

4.6.2.1 Description

FSL_Program will program specified data to destination address in Flash and verify the data. It encapsulates two low-level SSD functions: FlashProgram and ProgramVerify. Input parameters are used directly without any check. At the most, 2 bytes can be programmed in a single call.

4.6.2.2 Procedure

1. Check if the FlashProgram function is already in stack. If not, copy it into stack;
2. Program the data contained in emuBuffer to Flash;
3. Verify the data programmed in Flash;
4. Return.

4.6.2.3 Definition

```
unsigned char FSL_Program (void);
```

4.6.2.4 Global Parameters Referred

Table 28. Global Parameters for FSL_Program

Argument	Size	Description	Remark
STARTADDR	2 bytes	The starting address in Flash to be written to.	Used as an input and output argument.
ENDADDR	2 bytes	The end address in Flash to be written to.	The range determined by STARTADDR and ENDADDR should be within a same Flash row.
emuBuffer	2 bytes	Source data to be programmed.	At most 2 bytes can be programmed per call.

4.6.2.5 Return Values

Table 29. Return Values for FSL_Program

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_VERIFY

4.6.2.6 Calling Relationship

FSL_Program

- FlashProgram (SSD)
- ProgramVerify (SSD)

4.6.2.7 Tips

At the entry of this function, it checks if the required high voltage function is on stack or not by evaluating the value of hvType. If not, copy the required high voltage function onto stack specified by hvPostion.

FlashProgram will update the value of STARTADDR. To verify the Flash result after programming, STARTADDR should be pushed onto stack before calling FlashProgram.

Since the Flash requires row programming (i.e., only up to one row can be programmed per call), the programming range specified by STARTADDR and ENDADDR should be within the same row.

4.6.2.8 Troubleshooting

Table 30. Troubleshooting for FSL_Program

Returned Error	Description	Solution
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.6.3 FSL_CopyRecord

4.6.3.1 Description

FSL_CopyRecord writes user data into Flash in a record format.

4.6.3.2 Procedure

1. Program and verify the record status and record status is changed to RECORD_STATUS_STARTED;
2. Program and verify the record ID;
3. Program and verify the record data byte by byte;

4. Change the record status to RECORD_STATUS_COMPLETED;
5. Return.

4.6.3.3 Definition

```
unsigned char FSL_CopyRecord (void);
```

4.6.3.4 Global Parameters Referred

Table 31. Global Parameters for FSL_CopyRecord

Argument	Size	Description	Remark
recID	1 byte	ID of the record to be copied.	Any value from 0 to RECORD_ID_MAX - 1
emuBlank	2 byte	The destination address in Flash memory to be written to.	Used as an input parameter.
failedAddress	2 byte	Buffer address of record data.	*failedAddress* is reused to save the buffer address of the record data.

4.6.3.5 Return Values

Table 32. Return Values for FSL_CopyRecord

Returned Value	Description
EE_OK	Operation finished successfully.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.

4.6.3.6 Calling Relationship

FSL_CopyRecord

- FSL_Program

4.6.3.7 Tips

To reduce the RAM usage, global variable “failedAddress” is reused here to save the buffer address of the record data.

4.6.3.8 Troubleshooting

Table 33. Troubleshooting for FSL_CopyRecord

Returned Error	Description	Solution
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.6.4 FSL_InitCluster

4.6.4.1 Description

FSL_InitCluster initializes one cluster to make it ready for EEPROM emulation.

4.6.4.2 Procedure

1. Check cluster status field;
2. If the cluster status is CLUSTER_STATUS_BLANKED, return directly;
3. If the cluster status is CLUSTER_STATUS_STARTED or CLUSTER_STATUS_ACTIVE, its erasing cycles will be add 1;
4. Otherwise, its erasing cycles will be 1;
5. Erase and verify the cluster;
6. Write the new erasing cycles;
7. Change the cluster status to CLUSTER_STATUS_BLANKED;
8. Return.

4.6.4.3 Definition

```
unsigned char FSL_InitCluster (void);
```

4.6.4.4 Global Parameters Referred

Table 34. Global Parameters for FSL_InitCluster

Argument	Size	Description
emuStartAddr	2 bytes	The starting address of the cluster.
emuEndAddr	2 bytes	The end address of the cluster.

4.6.4.5 Return Values

Table 35. Return Values for FSL_InitCluster

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_NOT_BLANK EE_ERROR_VERIFY

4.6.4.6 Calling Relationship

FSL_InitCluster

- FSL_Erase
- FSL_Program

4.6.4.7 Tips

The cluster after initialization contains only the cluster status field and erasure cycle field.

If the cluster status to be initialized is CLUSTER_STATUS_BLANKED, its erasing cycles will be kept the same. If the cluster status is CLUSTER_STATUS_STARTED or CLUSTER_STATUS_ACTIVE, its erasing cycles will be add 1 based on its current erasing cycles. For all other cases, its erasing cycles will be reset to 1.

4.6.4.8 Troubleshooting

Table 36. Troubleshooting for FSL_InitCluster

Returned Error	Description	Solution
EE_ERROR_NOT_BLANK	The Flash memories are not blank.	Check the Flash memory if it can be erased correctly.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.6.5 FSL_SwapCluster

4.6.5.1 Description

FSL_SwapCluster copies the latest valid records from the active cluster to the alternative cluster.

4.6.5.2 Procedure

1. If the status of alternative cluster is not CLUSTER_STATUS_BLANKED, return cluster status error code;
2. Change the alternative cluster status to CLUSTER_STATUS_STARTED;
3. Push the recID onto stack;
4. Fetch the latest data record of a specific ID from the active cluster;
5. Copy this record to the alternative cluster;
6. Update the alternative cluster blank space pointer and next record ID;
7. Repeat steps 4–6 until all latest data records are copied to alternative cluster;
8. Change the alternative cluster status to CLUSTER_STATUS_ACTIVE;
9. Initialize the old active cluster;
10. Pull recID from stack;
11. Update the active cluster index in the EEPROM configuration;
12. Return.

4.6.5.3 Definition

```
unsigned char FSL_SwapCluster (void);
```

4.6.5.4 Global Parameters Referred

Table 37. Global Parameters for FSL_SwapCluster

Argument	Size	Description	Remark
activeIndex	1 byte	Indicating which cluster is active.	After swapping finished successfully, it should be updated.
emuStartAddr	2 bytes	The starting address of the active cluster.	It will be updated after swapping. Used as an input and output argument.
emuEndAddr	2 bytes	The end address of the active cluster.	It will be updated after swapping. Used as an input and output argument.
emuBlank	2 bytes	The blank location of the active cluster.	It will be updated after swapping. Used as an input and output argument.

4.6.5.5 Return Values

Table 38. Return Values for FSL_SwapCluster

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_CSTAT EE_ERROR_NOT_BLANK EE_ERROR_VERIFY

4.6.5.6 Calling Relationship

FSL_SwapCluster

- FSL_Program
- FSL_SearchRecord
- FSL_CopyRecord
  - — FSL_Program
- FSL_InitCluster
  - — FSL_Erase
  - — FSL_Program

4.6.5.7 Tips

In this function, all the latest valid records should be copied from the active cluster to the alternative cluster. The global variable recID is reused. To save its original value, recID is pushed onto the stack before it is reused and restored from the stack before exiting.

4.6.5.8 Troubleshooting

Table 39. Troubleshooting for FSL_SwapCluster

Returned Error	Description	Solution
EE_ERROR_CSTAT	Cluster status error.	Re-initialize the EEPROM.
EE_ERROR_NOT_BLANK	The Flash memories are not blank.	Check the Flash memory if it can be erased correctly.
EE_ERROR_VERIFY	Corresponding source data and content of destination location mismatch.	Check the Flash memory if it can be programmed correctly.

4.6.6 FSL_SearchRecord

4.6.6.1 Description

FSL_SearchRecord searches the record with a specific record ID in a cluster. This function will parse the whole cluster to determine the latest copy of data record.

4.6.6.2 Procedure

1. Clear the emuBuffer with 0xFFFF;
2. Initialize nextRecID to RECORD_ID_MAX;
3. Fetch one record from the cluster;
4. If the recID is RECORD_ID_MAX, go to step 5, otherwise, go to step 9;
5. Check the record status;
6. If it is RECORD_STATUS_COMPLETED, RECORD_STATUS_STARTED or RECORD_STATUS_DELETED, skip this data record;
7. If it is RECORD_STATUS_ERASED, it is the end of the valid data record region;
8. Otherwise, it's an invalid record, set return code to EE_ERROR_RSTAT and go to step 16;
9. If the record ID of the current record is not the one being searching for, skip this data record;
10. Otherwise, check the record status;
11. If it is RECORD_STATUS_COMPLETED, save the record address to emuBuffer;
12. If it is RECORD_STATUS_STARTED, skip this wrong data record;
13. Otherwise, clear emuBuffer and skip this wrong data record;
14. For the steps 9–13, if the ID of current record is greater than recID and less than nextRecID, update nextRecID as the current record ID;
15. Repeat steps 3 and 14 until you reach the end of the cluster or encounter the blank location;
16. Return (The location of error record or the starting address of cluster free space is saved in failedAddress.).

4.6.6.3 Definition

```
unsigned char FSL_SearchRecord (void);
```

4.6.6.4 Global Parameters Referred

Table 40. Global Parameters for FSL_SearchRecord

Argument	Size	Description	Remark
recID	1 byte	Record ID to search for	It should be within record ID range from 0 to RECORD_ID_MAX. RECORD_ID_MAX is used specially for parse the cluster to the start of free space or checking the record status in active cluster.
nextRecID	1 byte	To save the record ID for the next parsing of the cluster	For each parse of the cluster, the next valid record ID can be returned via this parameter. This parameter is only used for speeding up the cluster swapping procedure.
emuBuffer	2 bytes	To save the searching result.	2-byte global variable used as an output parameter.
failedAddress	2 bytes	To save the location of error record or the first free location in cluster.	This parameter provides a way to return the location of the corrupt record or to return the first free location of the cluster when the parameter *recID* is of value RECORD_ID_MAX.
emuStartAddr	2 bytes	The starting address of the active cluster.	
emuEndAddr	2 bytes	The end address of the active cluster.	

4.6.6.5 Return Values

Table 41. Return Values for FSL_SearchRecord

Type	Description	Possible Values
unsigned char	Successful completion or error value.	EE_OK EE_ERROR_RSTAT

4.6.6.6 Calling Relationship

N/A.

4.6.6.7 Tips

None.

4.6.6.8 Troubleshooting

Table 42. Troubleshooting for FSL_SearchRecord

Returned Error	Description	Solution
EE_ERROR_RSTAT	Cluster status error.	Re-write the record into EEPROM.

4.7 Low Level Functions

4.7.1 FlashEraseCOP

4.7.1.1 Description

This function supports page erase with COP service enabled. Furthermore, this function only provides the 4ms-page erase timing. The page erase size depends on the hardware and can either be 64 or 128 bytes.

This function supports COP service period of minimum 240 us at a 8-MHz bus clock by splitting a long erase pulse into pieces of shorter pulses. COP service period is scaled with bus clock if the user selects a different bus clock. For instance, if a 4-MHz bus clock is used, minimum COP period to be supported is then 480 us.

To minimize code size, this function does not perform any range checking on parameters. So the user must ensure the relevant global parameters are correctly initialized.

This function does not have return code. Please use BlankCheck to confirm the target region is erased.

4.7.1.2 Procedure

1. Set the erase loop counter based on required timing.
2. Set ERASE bit in the Flash control register;
3. Read the Flash protection register;
4. Bump COP;
5. Write to any Flash address within the erase region with any data;
6. Wait for time Tnvs (>=10 us);
7. Set HVEN bit in the Flash control register;
8. Delay for Tcop;
9. Clear ERASE bit in the Flash control register;
10. For page erase, wait for Tnvh (>=5 us);
11. Clear HVEN bit in the Flash control register;
12. Bump COP;
13. Decrease the erase loop counter. If it is not 0, go to step2, else go to next step;
14. Return.

4.7.1.3 Definition

```
void FlashEraseCOP (void);
```

4.7.1.4 Global Parameters Referred

Table 43. Global Parameters for FlashEraseCOP

Argument	Size	Description	Remark
FLASHCR	2 bytes	Address of the Flash control register.	Please consult hardware spec for actual address. Typical value is 0xFE08.
FLASHPR	2 bytes	Address of the Flash protection register.	Please consult hardware spec for actual address. Typical value is 0xFF7E.
CLOCKSCALAR	1 byte	The scaling factor based on the bus clock.	8 times of bus clock scaled in MHz and rounded down to the nearest integer.
STARTADDR	2 bytes	The start address of Flash area to be erased.	Should be within valid Flash region.

4.7.1.5 Return Values

None.

4.7.1.6 Tips

Set the Flash protection bits correctly before calling erase function.

Writing to the COP control register when erase voltage is turned on will interfere with erase operation, because the COP register is located in the Flash array. Thus writing to COP can only occur between erase pulses.

The function uses the STARTADDR as the erase interlock write address. Because there is no error checking on parameters, the user must ensure the STARTADDR is within the Flash region to be erased.

This function cannot erase the Flash array in which it resides.

It is highly recommended that interrupts be disabled during program/erase operations.

4.7.1.7 Troubleshooting

This section is intentionally blank.

4.7.1.8 Affected Register

Table 44. Register Affected in FlashEraseCOP

Name	Bit	Description
Flash Control Register	HVEN, MASS, ERASE and PGM	Read, Write
Flash Block Protection Register	BPR[7:0]	Read
COP	BIT[7:0]	Write

4.7.2 BlankCheck

4.7.2.1 Description

This function reads the memory and compares against 0xFF (logic state of erased bit cells).

If the blank check fails, the error code of SGF_ERROR_NOT_BLANK (0x30) will be returned in register A. The first non-blank address can be derived from the global variable of STARTADDR.

To minimize code size, this function does not perform any range checking on parameters. The user must ensure the relevant global parameters are correctly initialized.

There is no COP service in this function.

4.7.2.2 Procedure

1. Compare STARTADDR with ENDADDR; if greater then go to step 4;
2. Compare the data at STARTADDR against 0xFF and update STARTADDR at the same time;
3. If not equal, then load error code SGF_ERROR_NOT_BLANK to A and go to step 5; otherwise go to step 1;
4. Load SGF_OK to A;
5. Return.

4.7.2.3 Definition

```
unsigned char BlankCheck(void);
```

4.7.2.4 Global Parameters Referred

Table 45. Global Parameters for BlankCheck

Argument	Size	Description	Remark
STARTADDR	2 bytes	The start address of area to be checked.	Can be any readable location.
ENDADDR	2 bytes	The end address of area to be checked.	Should equal to or greater than the start address. Otherwise, no checking will be performed.

4.7.2.5 Return Values

Table 46. Return Values for BlankCheck

Type	Description	Possible Values
unsigned char	Successful completion or error value.	SGF_OK SGF_ERROR_NOT_BLANK

4.7.2.6 Tips

If ENDADDR is less than STARTADDR, the function returns SGF_OK without any checking.

If SGF_ERROR_NOT_BLANK is returned in A, the first non-blank address is equal to the value of STARTADDR minus 1.

4.7.2.7 Troubleshooting

Table 47. Troubleshooting for BlankCheck

Return Value	Description	Solution
SGF_ERROR_NOT_BLANK	There is a non-blank byte (i.e. not 0xFF) within the region.	Erase the region again.

4.7.2.8 Affected Register

None.

4.7.3 FlashProgram

4.7.3.1 Description

This function is used to program source data to the specified Flash area. It only supports row programming. That is, only up to one row (32 bytes or 64 bytes depending on hardware) can be programmed per call. Users should ensure all the addresses to be programmed are within one row; otherwise, programming may fail. Please refer to the User’s Manual of M68HC908 parts for more information.

To minimize code size, this function does not perform any range checking on parameters. The user must ensure the relevant global parameters are correctly initialized.

COP is not serviced inside this function.

This function does not have return code. Please use ProgramVerify to confirm the target region is programmed correctly.

4.7.3.2 Procedure

1. Set PGM bit in the Flash control register and clear other bits;
2. Read the Flash protection register;

3. Write to any Flash address within the programming row with any data;
4. Wait for time Tnvs (>=10 us);
5. Set HVEN bit in the Flash control register;
6. Wait for time Tpgs (>=5 us);
7. Write source data byte to the Flash destination address to be programmed;
8. Wait for time Tprog (30 us to 40 us);
9. Increment the source address and destination address; go to step 7 until the destination address exceeds the end address;
10. Clear PGM bit in the Flash control register;
11. Wait for time Tnvh (>=5 us);
12. Clear HVEN bit in the Flash control register;
13. Return.

4.7.3.3 Definition

```
void  FlashProgram (void);
```

4.7.3.4 Global Parameters Referred

Table 48. Global Parameters for FlashProgram

Argument	Size	Description	Remark
FLASHCR	2 bytes	Address of the Flash control register.	Please consult hardware spec for actual address. Typical value is 0xFE08.
FLASHPR	2 bytes	Address of the Flash protection register.	Please consult hardware spec for actual address. Typical value is 0xFF7E.
CLOCKSCALAR	1 bytes	The scaling factor based on the bus clock.	8 times of bus clock scaled in MHz and rounded down to the nearest integer.
STARTADDR	2 bytes	The start address of Flash area to be programmed.	Should be within valid Flash region.
ENDADDR	2 bytes	The end address of Flash area to be programmed.	Should be within valid Flash region. And must be equal to or greater than STARTADDR.
BUFFER	2 bytes	The start address of the source data buffer used for programming.	Should be within readable memory.

4.7.3.5 Return Values

None.

4.7.3.6 Tips

Set the Flash protection bits correctly before calling program function.

Because the function does not perform any checking on parameters, the user must ensure that ENDADDR is not less than STARTADDR and the program region is within valid Flash range.

The time required to program an entire row may be longer than the COP timeout period (the smallest possible COP timeout period is about 240 microseconds). Based on the programming performance data shown in the appendix, users decide how many bytes to be programmed per call in order to avoid COP timeout.

It is highly recommended that interrupts be disabled during program/erase operations.

4.7.3.7 Troubleshooting

This section is intentionally blank.

4.7.3.8 Affected Register

Table 49. Register Affected in FlashProgram

Name	Bit	Description
Flash Control Register	HVEN, MASS, ERASE and PGM	Read, Write
Flash Block Protection Register	BPR[7:0]	Read

4.7.4 ProgramVerify

4.7.4.1 Description

This function is used to verify that the data programmed into the Flash or EEPROM matches the source data.

If the verification fails, the error code of SGF_ERROR_VERIFY (0x40) will be returned in the register A. The first failed address can be derived from the global variable of STARTADDR.

To minimize code size, this function does not perform any range checking on parameters. The user must ensure the relevant global parameters are correctly initialized.

There is no COP service in this function.

4.7.4.2 Procedure

1. Set SGF_OK to A;
2. Compare STARTADDR with ENDADDR; if greater then go to 9;
3. Get the data at STARTADDR and increment STARTADDR at the same time;
4. Compare the data with the content of the source buffer;
5. If not equal, go to step 8;
6. Update the source data buffer address;
7. Go to step 1;
8. Load error code SGF_ERROR_VERIFY to A;
9. Return.

4.7.4.3 Definition

```
unsigned char ProgramVerify (void);
```

4.7.4.4 Global Parameters Referred

Table 50. Global Parameters for ProgramVerify

Argument	Size	Description	Remark
STARTADDR	2 bytes	The start address of area to be verified.	Can be any readable location.
ENDADDR	2 bytes	The end address of area to be verified.	Should equal to or greater than the start address. Otherwise, no checking will be performed.
BUFFER	2 bytes	The start address of the source data buffer used for verification.	Should be within readable memory.

4.7.4.5 Return Values

Table 51. Return Values for ProgramVerify

Type	Description	Possible Values
unsigned char	Successful completion or error value.	SGF_OK SGF_ERROR_VERIFY

4.7.4.6 Tips

If ENDADDR is less than STARTADDR, the function returns SGF_OK without any checking.

If SGF_ERROR_VERIFY is returned in A, the first failed address is equal to the value of STARTADDR minus 1.

4.7.4.7 Troubleshooting

Table 52. Troubleshooting for ProgramVerify

Return Value	Description	Solution
SGF_ERROR_VERIFY	There is a mismatch between the source data and programmed Flash region.	Erase the region and re-program the source data.

4.7.4.8 Affected Register

None.

Appendix A: Performance Data

The performance data in Appendix A was collected using the CodeWarrior for HCS08, V3.0.

A.1 Code Size and Stack Usage

Table 53. Code Size

Hierarchies	Code Size (byte)
High-Level Driver	524
Middle-Level Driver	608
Low-Level Driver	173
Total Code Size	1305

Table 54. Stack Usage

Function Name	Stack (byte)[1]
FSL_InitEeprom	79
FSL_WriteEeprom	82
FSL_ReadEeprom	2
FSL_DeleteRecord	76
FSL_ReportEepromStatus	2
FSL_DeinitEeprom	77
MAX Stack Usage	82

NOTES:
[1] The stack usage contains the stack allocated for high-voltage SSD functions.

A.2 Read / Write Times

The common conditions for collecting read/write time performance data are listed below:

- The data are collected on MC68HC908JL3 parts;
- 9.8304-MHz oscillator clock;
- 2.4576-MHz bus clock;
- Two 64-byte Flash pages configured for each cluster;
- All records adopt 2 bytes fixed user data length.

Table 55. Read / Write Times

Function Name		Typical Time (us)
FSL_WriteEeprom	No swap	1985.67
	Swap	74257.62
FSL_ReadEeprom	Min time	81.38
	Max time	914.71
FSL_DeleteRecord		2010.09
FSL_ReportEepromStatus		693.36

NOTE

The data above was collected based on the MC68HC908JL3 parts. For other M68HC908 parts, similar performance data is expected.

The “No swap” writing time for FSL_WriteEeprom is collected when the write will not trigger cluster swapping.

The “Swap” writing time for FSL_WriteEeprom is collected when the write will trigger cluster swapping (The cluster is full of records, only two of them have a same record ID. All other records have different record ID).

The minimal time for FSL_ReadEeprom is collected when the active cluster contains only 1 record, while the maximum read time is collected when the active cluster is full of records.

The timings for FSL_DeleteRecord and FSL_ReportEepromStatus are collected when the active cluster is full of records.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com