Freescale Semiconductor
Application Note

Document Number: AN4903
Rev. 0, 03/2014

EEPROM Emulation Driver for the Kinetis E Series Microcontrollers

by Wang Peng

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 needs 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 demo code shows how to emulate EEPROM on Flash.

Contents

1 Introduction

The Kinetis E series microcontrollers (except KE02) do not have an on-chip EEPROM, however, the devices can store non-volatile data in the on-chip Flash memory using the software described in this application note, thus saving the cost of an external EEPROM.

One erasable Flash unit is equivalent to one sector. Because the Flash programming only works on an erased address, the Flash memory must be erased before programming. Directly programming data to Flash without the software algorithm results in frequent erasing of the Flash. This frequent erasing reduces the life of the Flash and increases the writing time of data.

The EEPROM emulation driver for the Kinetis E series implements the fixed-length data record scheme on Flash and includes the following features:

- organizing data items
- initializing EEPROM
- checking EEPROM status
- reading, writing and deleting data items;
  including an algorithm to save data to avoid directly and frequently writing to Flash

2 Software Architecture

Figure 1 illustrates a simple API function for customers to implement an EEPROM emulation driver from Flash memory.

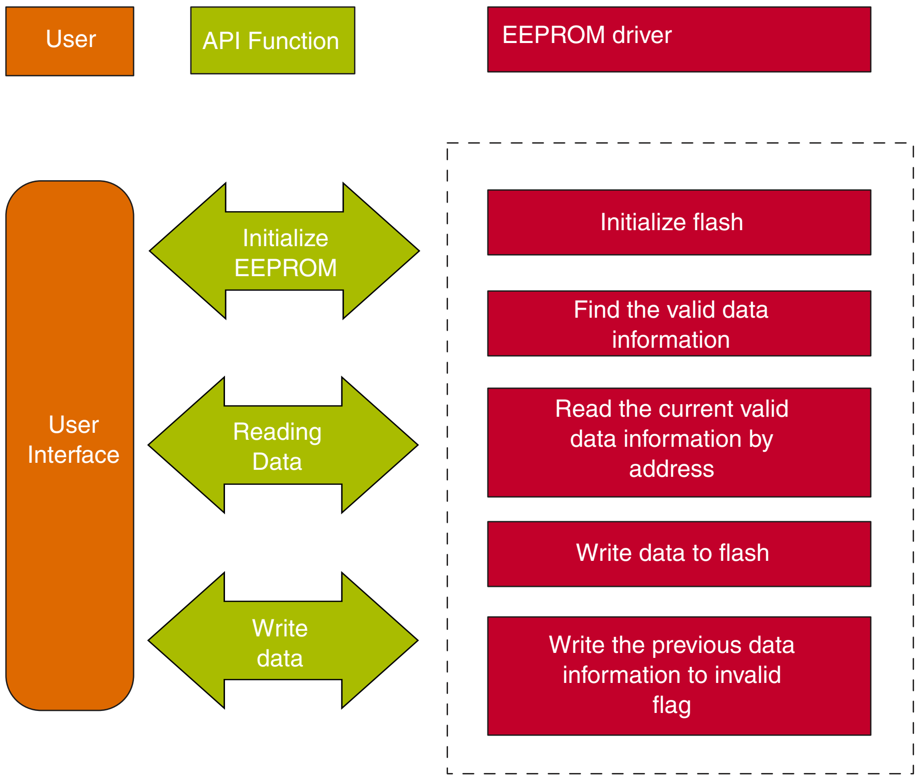
User
API Function
EEPROM driver
User Interface
Initialize EEPROM
Reading Data
Write data
Initialize flash
Find the valid data information
Read the current valid data information by address
Write data to flash
Write the previous data information to invalid flag

Figure 1. Timing diagram elements

API functions are listed as follows:

```
uint8_t EE_Init(uint32_t *pCurrentAddress,uint32_t u32BusClock);

uint8_t EE_Write(EE_ItemInfoPtr pWrItemInfo,uint32_t *p32CurrentAddress);

uint8_t EE_Read(EE_ItemInfoPtr pRdItemInfo,uint32_t u32CurrentAddress);

uint8_t EE_SearchIndex(uint32_t *pCurrentAddress);
```

In the EEPROM driver, four states are defined. Each state is referred to as an *item* in the following and throughout this document:

```
#define EE_ITEM_INFO_NULL          0xff
#define EE_ITEM_INFO_PROCESSING    0xe7
#define EE_ITEM_INFO_VALID         0xa5
#define EE_ITEM_INFO_INVALID       0x00
```

The description of each of the four states follows. Use this information to implement a state machine to identify valid information.

- EE_ITEM_INFO_NULL
  Indicates that the current data item is NULL. The data item is valid and is programmable to write new data to the current address.
- EE_ITEM_INFO_PROCESSING
  Indicates that the current data item is in process. The processes can be that the current item is being written, that the current item is not complete for the entire updated sequence, or that the power is removed when the current item is being written to Flash.
- EE_ITEM_INFO_VALID
  Indicates that the current data item is valid. In this state, the data item is good for use.
- EE_ITEM_INFO_INVALID
  Indicates that the current data item is invalid. The data item is old and has been replaced.

2.1 Memory map

This EEPROM emulation algorithm saves the data item. Users can also define the length of each data item. Refer to the following macro definition in file “ee_emulation.h” for programming instructions.

```
/* here ensure length divide 4 equal to 0, so that flash operation aligns with 4 bytes */
#define EE_ITEM_INFO_LENGTH    16
#if ((EE_ITEM_INFO_LENGTH%4) != 0 )
#error "please ensure EE information length is align with 4 bytes"
#endif
```

To save the Flash memory space and facilitate the code, the defined length must be divided by four.

Figure 2 illustrates the default memory map of Kinetis E series with a typical allocation of resources for non-volatile data storage.

One data item contains a structure body as follows:

```
typedef struct
{
   uint8_t u8Flag; // indicates item status
   uint8_t u8InfoBuff[EE_ITEM_INFO_LENGTH - 1]; // data item buffer
}EE_ItemInfoType,*EE_ItemInfoPtr;
```

Here the u8Flag indicates the item’s status.

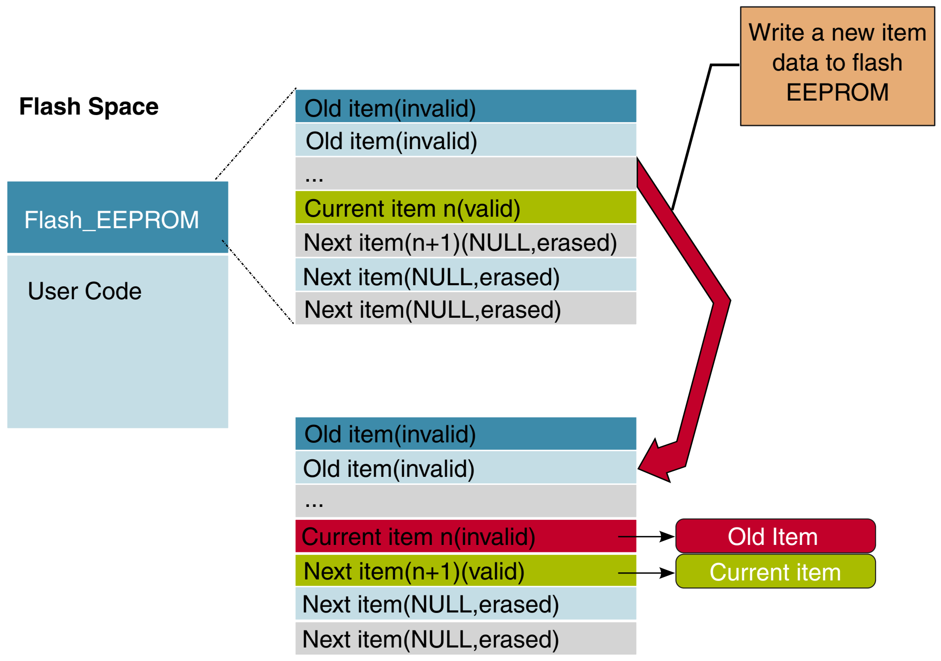

Figure 2. Memory allocation in Flash

2.2 EEPROM life time optimization

Users must specify the start address and end address for the EEPROM emulation driver. It is also necessary to reserve enough space for the EEPROM emulation driver, EEPROM size, and each data item’s length.

```
#define EE_START_ADDRESS  0xC000   // start address

#define FLASH_PAGE_SIZE   512      // number bytes of each page or sector

#define EE_PAGE_NUMBER    2        // reserved pages for EEPROM

#define EE_END_ADDRESS    EE_START_ADDRESS +FLASH_PAGE_SIZE*EE_PAGE_NUMBER-1
```

The previous macros define the start address, the number of pages used for the EEPROM, the number of bytes per page—that is, sector, as well as the end address used for EEPROM emulation driver.

The optimal EEPROM life time was evaluated by the formula in Equation 1.

$$\textit{Lift time} = \textit{Fc} \times \textit{EE\_size/Item\_length} \quad \textbf{Eqn. 1}$$

Where:

Fc—Indicates the Flash programming cycle. For Kinetis E series the typical cycle number is 100K.

EE_size—Indicates the Flash space reserved for the EEPROM.

Item_length—Indicates the data length for each item.

2.3 EEPROM initialization

To use the EEPROM emulation driver, it is necessary to call the “EE_Init” function to complete EEPROM initialization. This function performs Flash initialization and searches for the valid Flash address. It will search the entire Flash space assigned to EEPROM, find and return the valid data item’s address, or if no valid data item is found, return a NULL data item address.

For the Kinetis E series microcontrollers, it is necessary to initialize the Flash to configure the Flash clock to about 1 MHz, otherwise a Flash operation error occurs. After the Flash clock is initialized, then the Flash can be erased or programmed.

After calling the initialization function, a valid data item can be informed by “EE_Read” and a new data item can be written to Flash.

Figure 3 provides a detailed initialization flow chart.

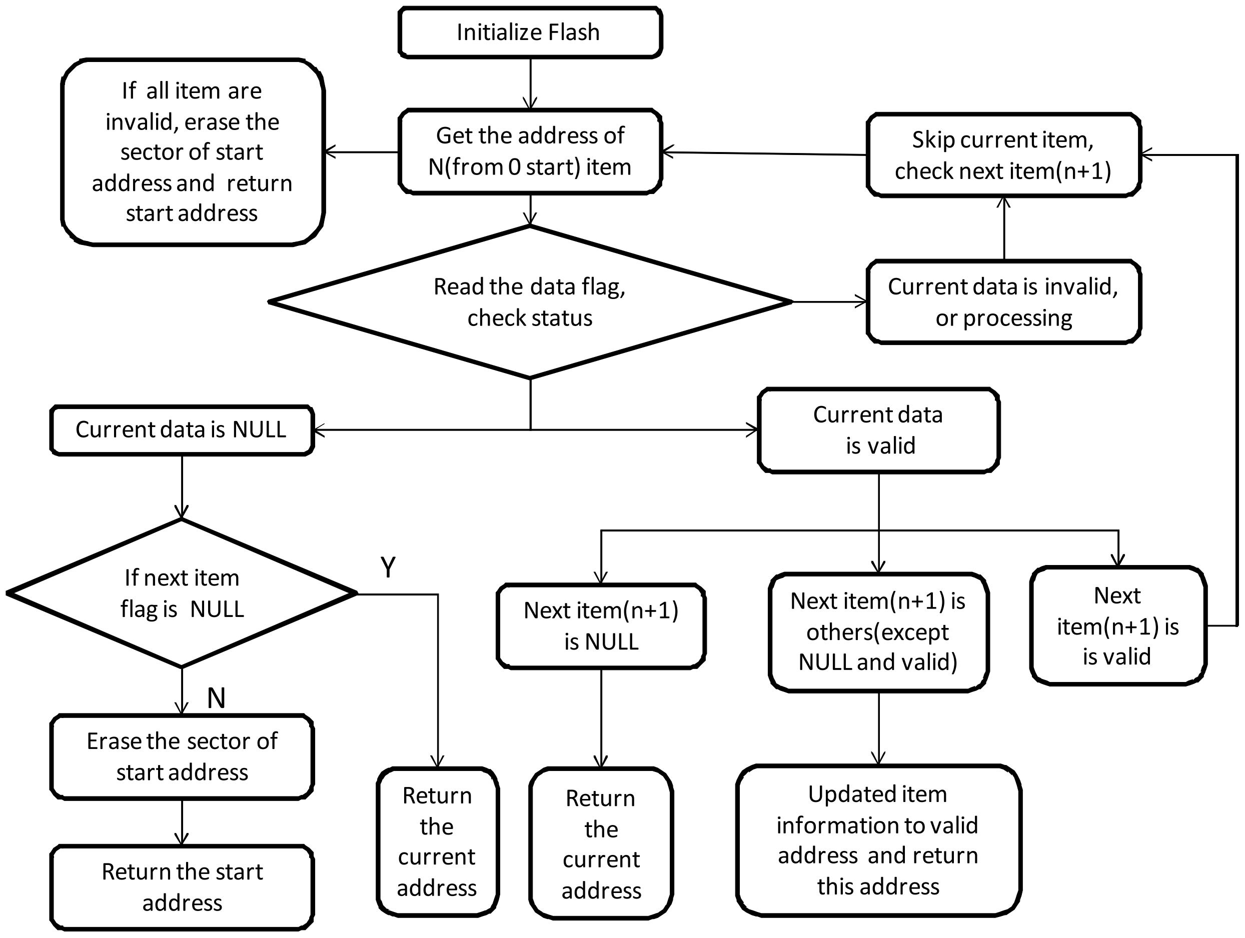
Initialize Flash
Get the address of N(from 0 start) item
If all item are invalid, erase the sector of start address and return start address
Skip current item, check next item(n+1)
Read the data flag, check status
Current data is invalid, or processing
Current data is NULL
Current data is valid
If next item flag is NULL
Y
N
Next item(n+1) is NULL
Next item(n+1) is others(except NULL and valid)
Next item(n+1) is is valid
Erase the sector of start address
Return the start address
Return the current address
Return the current address
Updated item information to valid address and return this address

Figure 3. Initialization flow chart

2.4 Writing data to EEPROM

To update a new data item with the EEPROM emulation driver, call the “EE_Write” function. The write function updates the new data to next data item’s address and then changes the current data item’s flag to invalid. This prevents the new data item from being corrupted during the Flash write process due to unexpected causes such as power removal. The state machine provides an algorithm to enable restoration of data information—that is, old valid information. Figure 4 illustrates writing data to the EEPROM emulation driver.

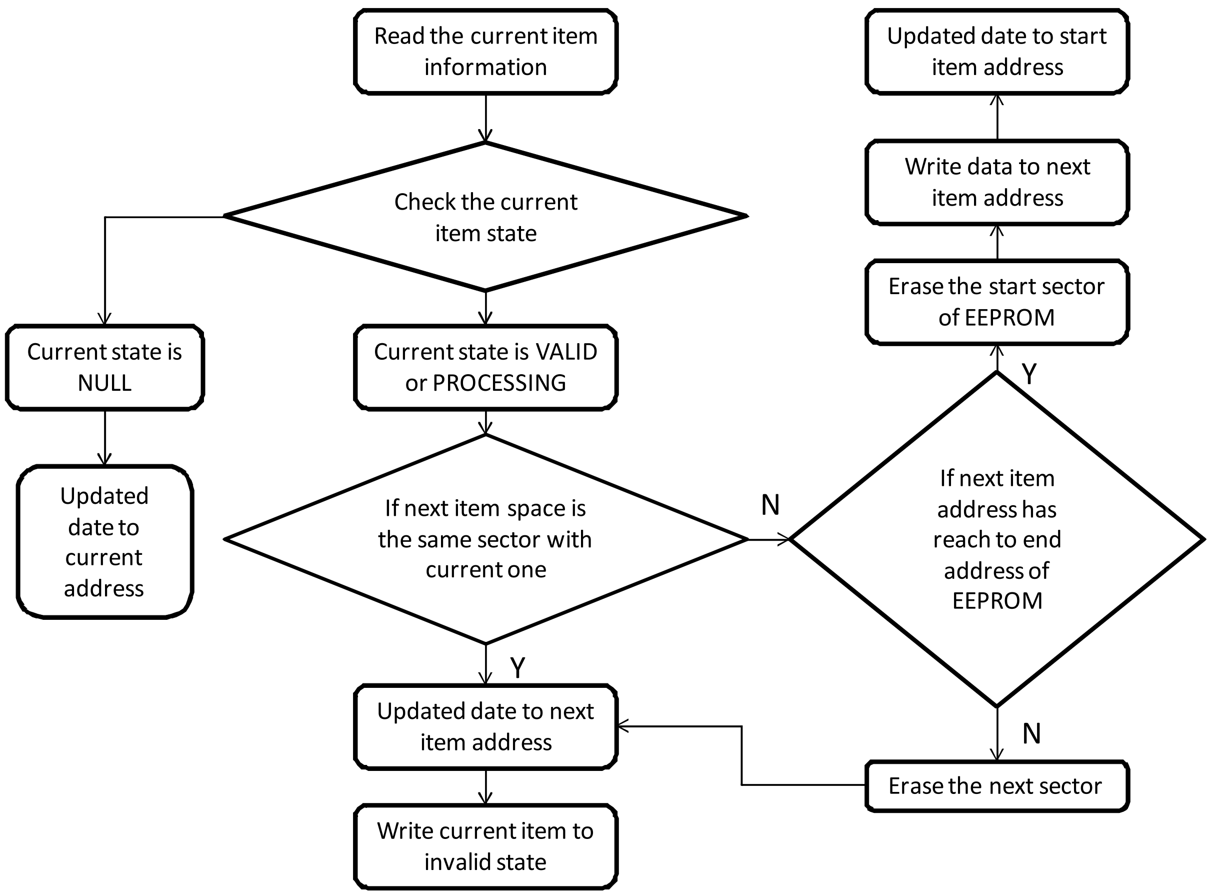
Read the current item information
Check the current item state
Current state is NULL
Updated date to current address
Current state is VALID or PROCESSING
If next item space is the same sector with current one
Y
Updated date to next item address
Write current item to invalid state
N
If next item address has reach to end address of EEPROM
Y
Erase the start sector of EEPROM
Write data to next item address
Updated date to start item address
N
Erase the next sector

Figure 4. Writing to EEPROM flow chart

The EEPROM emulation driver first checks the current data item's state and then determines how to update a new data item to the next address:

- If the current state is NULL, then it directly updates the next data item's address.
- If the current state is VALID, then it determines the next data item's state, and if that state is NULL, it saves the new data item to next address, and then changes the current data item's flag to INVALID.

When the data item is updated to Flash, it first writes the flag of "EE_ITEM_INFO_PROCESSING" to the new data item. After the data item update completes, it changes the flag to "EE_ITEM_INFO_VALID". If the power is removed during the "EE_ITEM_INFO_PROCESSING" updating process, the flag of the current data item is updated to "EE_ITEM_INFO_INVALID". When powered resumes, the flag of the previous data item shows "EE_ITEM_INFO_VALID" first, and then restores to the new address. It changes the used data item's address to "EE_ITEM_INFO_INVALID" by using the "EE_Init()". Figure 5 illustrates how to restore a data item's information from a bad update.

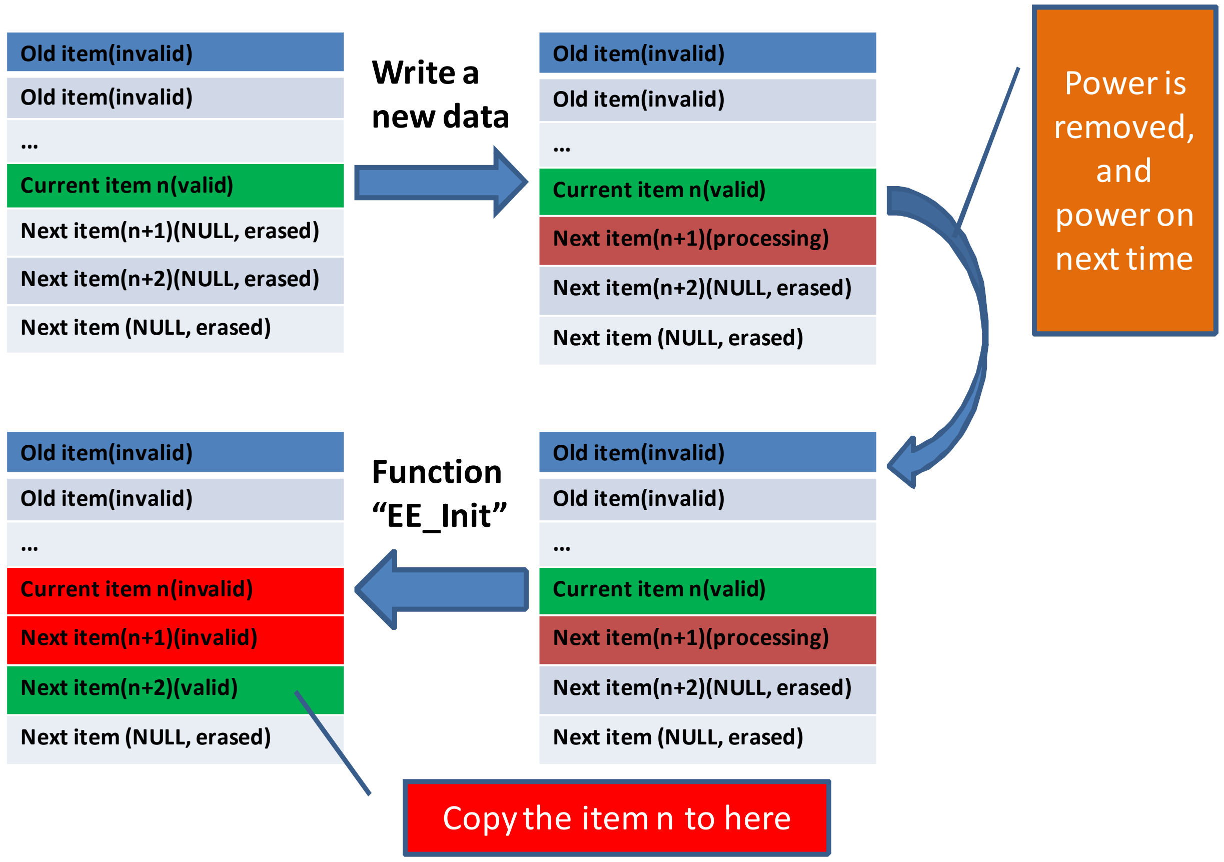

Figure 5. Restore Data from a old data item

2.5 Reading data from EEPROM

This API function reads the current data item from Flash and copies it to the RAM address specified by user.

```
uint8_t EE_Read(EE_ItemInfoPtr pRdItemInfo,uint32_t u32CurrentAddress)
 {
        if( pRdItemInfo->u8Flag == EE_ITEM_INFO_INVALID )
        {
        return FALSE;
        }
        memcpy((void *)pRdItemInfo,
        (uint8_t   *)u32CurrentAddress,sizeof(EE_ItemInfoType));
        return TRUE;
 }
```

3 EEPROM emulation driver example

Figure 6 shows how to add the files in this project to use the EEPROM emulation driver.

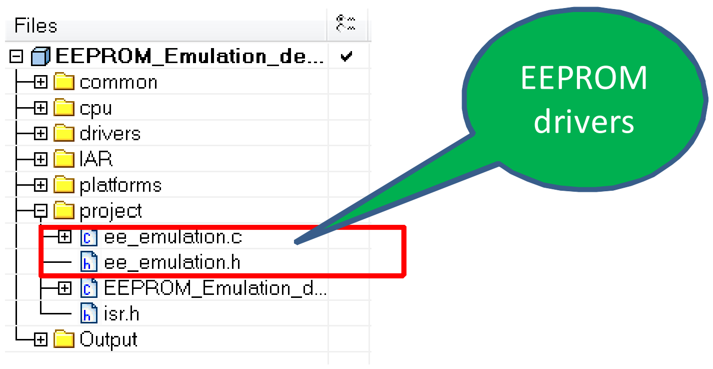

Figure 6. EEPROM drivers

Both the EEPROM emulation drivers and the non-volatile memory drivers are necessary. The following API functions are required for the Flash operations, including the initialization, programming, and erasing.

```
uint16_t FLASH_Init(uint32_t BusClock);

uint16_t FLASH_Program(uint32_t wNVMTargetAddress, uint8_t *pData, uint16_t sizeBytes);

uint16_t EEPROM_EraseSector(uint32_t wNVMTargetAddress);
```

Figure 7 illustrates the basic flow of using EEPROM emulation driver.

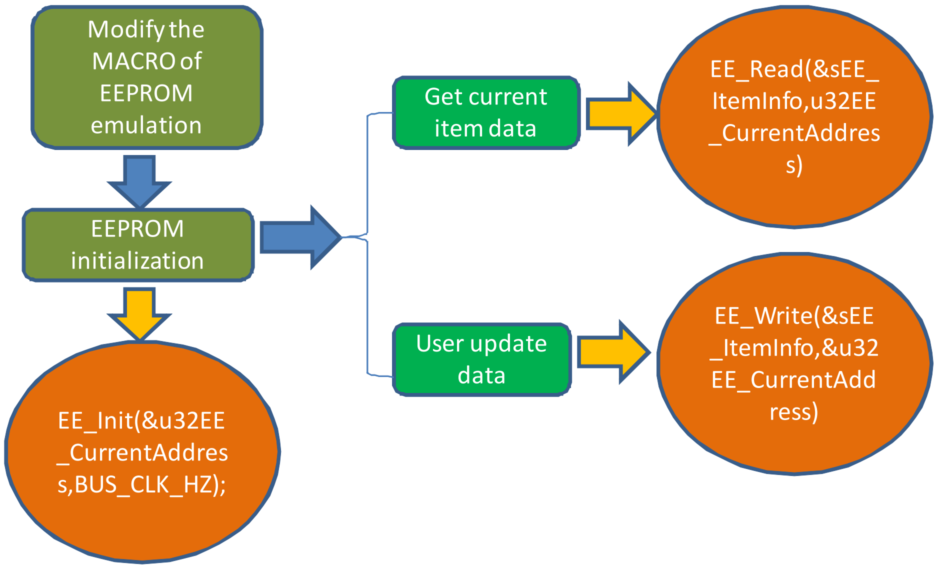
Modify the MACRO of EEPROM emulation
EEPROM initialization
EE_Init(&u32EE_CurrentAddress,BUS_CLK_HZ);
Get current item data
EE_Read(&sEE_ItemInfo,u32EE_CurrentAddress)
User update data
EE_Write(&sEE_ItemInfo,&u32EE_CurrentAddress)

Figure 7. Flow chart to use EEPROM emulation driver

A software demo using the EEPROM emulation driver on the Kinetis KE06 is available in the accompanying AN4903SW. The software driver runs on the FRDM-KE06Z board. The demo shows a simple application to write new data to the EEPROM.

4 Conclusion

This document introduces a way to implement EEPROM on Flash. It enables users to rapidly establish the EEPROM function on the Kinetis E series microcontrollers and to integrate software algorithms to increase access speed and optimize the life of non-volatile memory. It also describes the features of the restore machine when unexpected conditions occur such as power removal during the update operation. In summary, it is easy for customers to use and migrate to different platforms.

5 References

- EEPROM Emulation Driver for the Kinetis E Series Microcontrollers Software (Document number: AN4903SW)
- KE04 Sub-Family Reference Manual (Document number: MKE04P24M48SF0RM)
- KE06 Sub-Family Reference Manual (Document number: MKE06P80M48SF0RM)
- KE04 Sub-Family Data Sheet (Document number: MKE04P24M48SF0)
- KE06 Sub-Family Data Sheet (Document number: MKE06P80M48SF0)
- AN2302 EEPROM Emulation for the MC9S12C32
- AN3040 EEPROM Emulation Driver for M68HC908 Microcontrollers

6 Glossary

NVM	Non-Volatile Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
FCCOB	Flash Common Command OBject
WDOG	Watchdog
MCG	Multipurpose Clock Generator

7 Revision history

Revision 0 is the initial release of this document.

Document Number: AN4903
Rev. 0
03/2014