Freescale Semiconductor
Application Note

Document Number:AN4689
Rev. 0, 02/2013

Using EEPROM on MC56F84xxx DSC

by: Richy Ye

Contents

1 Introduction

The FlexMemory (FlexNVM and FlexRAM) is available on Freescale's MC56F84xxx family of digital signal controllers (DSC). The FlexMemory allows the user to configure the FlexNVM block as either basic data flash, enhanced EEPROM (EEE) flash, or a combination of both.

The EEPROM feature is widely used in applications that store small amount of rapidly changing data required to be saved under system power off.

This application describes the features of FlexMemory and enhanced EEPROM (EEE), EEPROM user perspective, EEPROM system-on-chip implementation, write endurance performance, and design considerations. It also includes an example project of EEPROM design on MC56F84789 device to show how to design EEPROM in a user application.

2 Features of FlexMemory and enhanced EEPROM

Figure 1 shows the flash memory blocks and the FlexMemory components on MC56F84xxx family device. Flash memory blocks include program flash and FlexNVM; FlexMemory blocks are composed of FlexNVM and FlexRAM, and enhanced EEPROM (EEE) is comprised of EEPROM backup

in FlexNVM, FlexRAM and EEE state machine. FlexNVM and FlexRAM are the only memories used for enhanced EEPROM implementation.

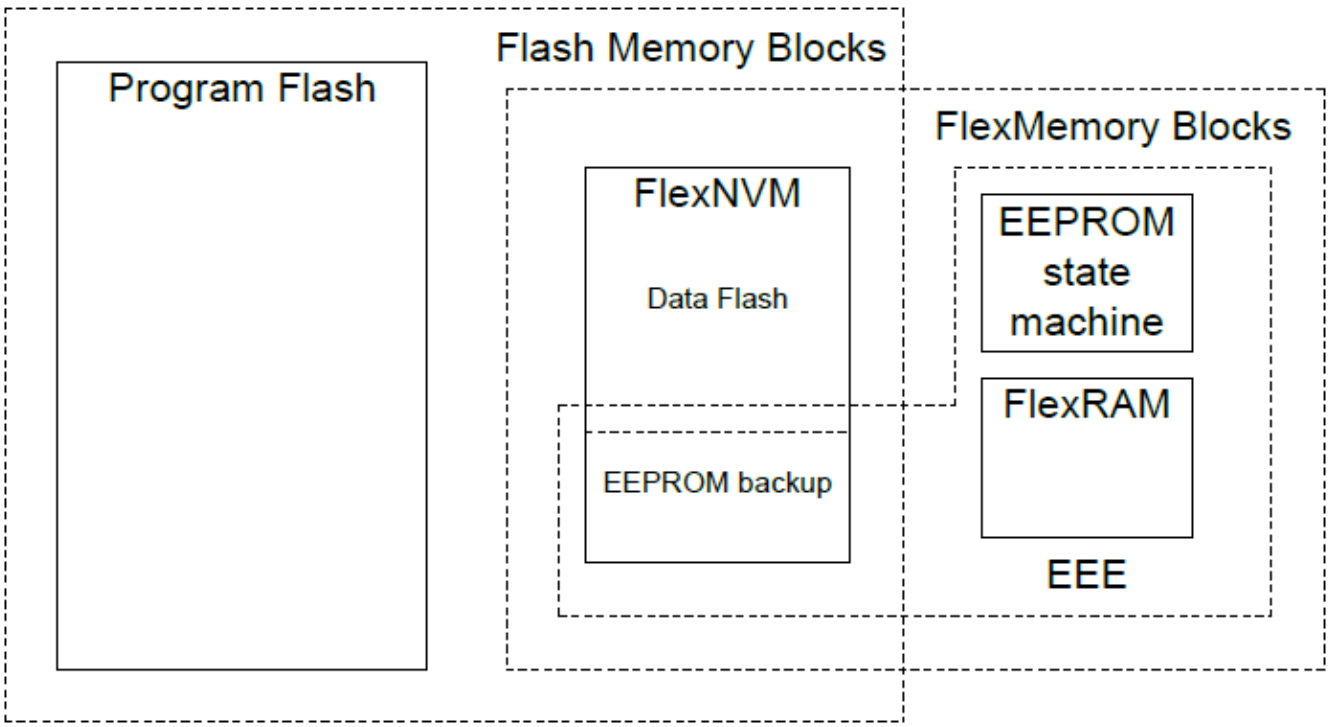


Figure 1. Block diagram of flash memory blocks and FlexMemory components

2.1 FlexMemory features

2.1.1 FlexNVM features

Following are the features of FlexNVM, when it is partitioned for data flash memory.

- Sector size of 1 KB
- Protection scheme prevents accidental programming or erasing of stored data.
- Automated, built-in program and erase algorithms with verify
- Section programming for faster bulk programming times
- Read access to data flash memory is possible while programming or erasing data in the program flash memory.

2.1.2 FlexRAM features

- Memory can be used as traditional RAM or high-endurance EEPROM.
- Up to 2 KB of FlexRAM configured for traditional RAM or EEPROM operations.
- When configured for EEPROM:
  - Protection scheme prevents accidental programming or erasing of data written for EEPROM.
  - Built-in hardware emulation scheme to automate EEPROM record maintenance functions
  - Programmable EEPROM data set size and FlexNVM partition code facilitating EEPROM memory endurance trade-offs

    - Supports FlexRAM aligned writes of 1, 2, or 4 bytes at a time
    - Read access to FlexRAM is possible while programming or erasing data in the program or data flash memory.
- When configured for traditional RAM:
    - Read and write access is possible to the FlexRAM while programming or erasing data in the program or data flash memory.

2.2 Enhanced EEPROM features

The enhanced EEPROM (EEE) capability of the FlexMemory has a number of features that allow the replacement of external EEPROMs and improves upon their performance.

Enhanced EEPROM features include:

- Configurability to allow the designer to make trade-off decisions based on the EEE endurance requirement and the total amount of EEE and data flash memory requirement
- High-endurance EEE memory can reach over 100M cycles (depending on the configuration).
- Supports byte, word, and long word access
- Fast write operation time (about 175 µs for a word-write operation to a pre-erased space, and 385 µs for a word-write operation)
- EEE functionality across the full operating voltage range for the processor

3 How enhanced EEPROM works

The enhanced EEPROM system is shown in the following figure.

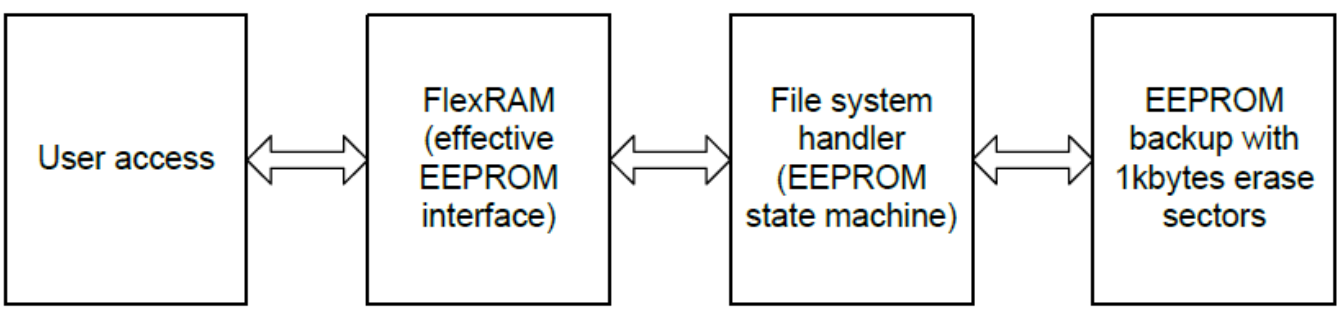


Figure 2. Top-level enhanced EEPROM architecture

To provide enhanced EEPROM (EEE) functionality, the FlexMemory uses a RAM block (FlexRAM), a flash block (FlexNVM), and EEE state machine. When the EEE functionality is enabled, the FlexRAM becomes EEE memory interface, which means that all of the EEE data is accessible at the FlexRAM address space. When the EEE is accessed, the EEE state machine keeps track of the data and backs it up as data records, stored in some portion of the FlexNVM used as EEPROM backup. Using a large partition of FlexNVM to back up the data for a small amount of EEE data allows the EEE implementation to offer extremely high endurance.

The EEE state machine uses 32-bit records to back up data from the EEE into the flash. 16 bits of the record are used for the data, and the other 16 bits are address and status information about the data. The data records are written and erased as needed. This means that if a location within the EEPROM has never been accessed, there will not be a data record for it. This helps to reduce the amount of data that needs to be backed up and can increase the memory endurance.

4 Enhanced EEPROM implementation

4.1 FlexMemory partition

To use the EEE features, the memory has to be partitioned. The partitioning process tells the EEE state machine how much EEE memory will be used and how much the FlexNVM will be used to back up the EEE.

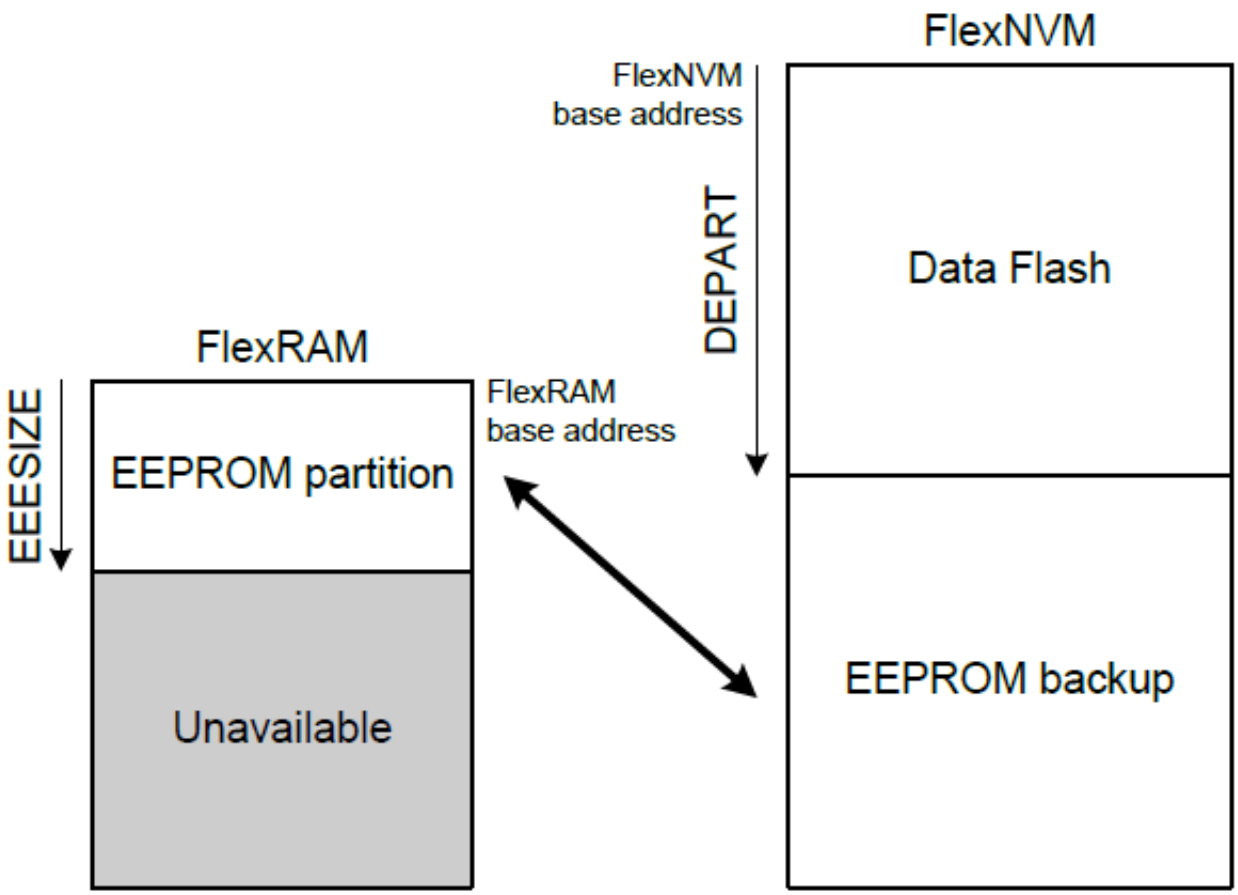


Figure 3. FlexRAM to FlexNVM memory mapping

To handle varying customer requirements, the FlexRAM and FlexNVM blocks can be split into partitions as shown in Figure 3. There are three programmable options that are used to define the exact memory use for a system. These parameters are:

- **EEPROM partition (EEESIZE)**: The amount of FlexRAM used for EEPROM can be set from 0 bytes (no EEPROM) to the maximum FlexRAM size (2 KB). The remainder of the FlexRAM is not accessible while the FlexRAM is configured for EEPROM (use the Set FlexRAM Function command; see Set FlexRAM Function command in Chapter 20 "Flash Memory Module" of *MC56F847XXRM: MC56F847xx Reference Manual*, available on **freescale.com**, for more information). The EEPROM partition grows upward from the bottom of the FlexRAM address space.
- **Data flash partition (DEPART)**: The amount of FlexNVM memory used for data flash can be programmed from 0 bytes (all of the FlexNVM block is available for EEPROM backup) to maximum size of the FlexNVM block (32 KB).
- **FlexNVM EEPROM partition**: The amount of FlexNVM memory used for EEPROM backup, which is equal to the FlexNVM block size minus the data flash memory partition size. The EEPROM backup size must be at least 16 times the EEPROM partition size in FlexRAM.

The partition information (EEESIZE, DEPART) is stored in the FlexNVM IFR (shown in Figure 4, Figure 5, and Figure 6), which can be programmed using the Program Partition command, and read using the Read Resource command (see Program Partition command and Read Resource command in *MC56F847XXRM: MC56F847xx Reference Manual*, available on **freescale.com**, for more information). Typically, the Program Partition command is executed only once in the lifetime of the device.

Address Range	Size (Bytes)	Field Description
0x00 – 0xFB, 0xFE – 0xFF	254	Reserved
0xFD	1	EEPROM data set size
0xFC	1	FlexNVM partition code

Figure 4. FlexNVM IFR map

Data flash IFR: 0x00FD							
7	6	5	4	3	2	1	0
1	1	1	1	EEESIZE			
	= Unimplemented or Reserved						

Figure 5. EEPROM data set size

Data Flash IFR: 0x00FC							
7	6	5	4	3	2	1	0
1	1	1	1	DEPART			
	= Unimplemented or Reserved						

Figure 6. FlexNVM partition code

Data flash memory is useful for applications that need to quickly store large amounts of data or store data that is static. The EEPROM partition in FlexRAM is useful for storing smaller amounts of data that will be changed often.

NOTE

Partitioning must be done only once. If the flash is repartitioned for a different configuration, then recorded data is lost and the user may not get the expected endurance.

4.2 Enhanced EEPROM startup

Out of reset with the FSTAT[CCIF] bit clear, the partition settings (EEESIZE, DEPART) are read from the data flash IFR and the EEPROM file system is initialized accordingly. The EEPROM file system locates all valid EEPROM data records in EEPROM backup and copies the newest data to FlexRAM. The FSTAT[CCIF] and FCNFG[EEERDY] bits are set after data from all valid EEPROM data records is copied to the FlexRAM. After the FCNFG[EEERDY] bit is set, the FlexRAM is available for read or write access operation.

4.3 Enhanced EEPROM read and write operations

The EEE data is read and written by accessing the FlexRAM address space. The EEE space is allocated starting at the beginning of the FlexRAM. The addressable space is the FlexRAM base address (0x03_C000, byte address) up to the programmed EEE size. Any space in the FlexRAM that is not used as EEE must not be accessed while the EEE functionality is enabled. For example, if the EEE size is configured as 128 bytes total, then read or write accesses to any address space between 0x03_C000 and 0x03_C07F are allowed, but 0x03_C080 to 0x03_C7FF will generate a bus error.

Because the EEE data is accessed through a RAM, the data is readable and writable at any size, byte, word, or long word. Although any access size is possible, the records used to back up the EEE data use a word sized data field. This means that byte writes are possible, but it makes less efficient use of the EEPROM backup. That is why the endurance Equation 1 on page 6 uses a different efficiency factor for 8-bit access than for 16-bit or 32-bit access.

4.3.1 Enhanced EEPROM write operation

When configured for enhanced EEPROM use, write to the unprotected valid space in FlexRAM launches an EEE operation to store the data within the EEPROM backup in a round-robin fashion. Because this is a flash program operation, the software must test the FSTAT[CCIF] bit (or FCNFG[EEERDY] bit) to determine if any other flash operations are in progress before writing to the EEE space. Because multiple concurrent writes and read-while-write operations within the same flash memory block are not allowed, accesses to the EEE or FlexNVM space are not allowed until the EEE write is complete.

NOTE

The p-flash memory is a completely separate logical block, therefore read/write operation to the p-flash can continue normally while an EEE write is in progress.

4.3.2 Enhanced EEPROM read operation

When the EEE is read, the data is supplied by the FlexRAM, and so no flash operation is trigerred. However, EEE read is not allowed while an EEE write is in progress. The software must either test the FCNFG[EEERDY] bit before read operation or wait for FCNFG[EEERDY] after a write operation before allowing the software to continue. In many cases, it is most efficient for software to test FCNFG[EEERDY] bit (or FSTAT[CCIF] bit) both before and after the write operation and block other EEE operations until FCNFG[EEERDY] bit sets after the write operation. This way a special function is needed for EEE write, but an EEE read doesn't require any special software. Another advantage to this approach is that no additional delay or flag checking is required if there are multiple EEE read operations without EEE write operations in between.

A special case for an EEE read that must be considered is the first access to the EEE after a chip reset event. For the first read of the EEE after reset, the FCNFG[EEERDY] bit may need to be tested to make sure that the EEE state machine has completed the initial load of data from the EEPROM backup to the FlexRAM. If the system startup time is long, this guarantees that the initial data load has enough time to complete before the first EEE read, then a test of the FCNFG[EEERDY] bit before the first read may not be required. However, it is safer to explicitly test the FCNFG[EEERDY] bit before the first read access to the EEE space.

5 Enhanced EEPROM write endurance

The bytes not assigned to data flash via the FlexNVM partition code are used by the FlexMemory module to obtain an effective endurance increase for the EEE data. The built-in EEPROM record management system raises the number of program/erase cycles that can be attained prior to device wear-out by cycling the EEE data through a larger EEPROM backup space.

While different partitions of the FlexNVM are available, the intention is that a single choice for the FlexNVM partition code and EEPROM data set size is used throughout the entire lifetime of a given application. For many applications, the endurance requirement for the EEE data drives the decision of how the FlexMemory must be configured. Understanding how the parameters affect the EEE write endurance is usually required to determine what EEE configuration to use.

The estimated minimum EEE write endurance equation and graph shown below assume that only one configuration is ever used.

$$\text{Writes\_FlexRAM} = \frac{\text{EEPROM} - 2*\text{EEESIZE}}{\text{EEESIZE}} * \text{Write\_efficiency} * n_{\text{nvmcycd}}$$

Equation 1

Where:

- Writes_ FlexRAM—minimum number of writes to each FlexRAM location
- EEPROM—allocated FlexNVM for EEPROM backup based on DEPART of FlexNVM partition code in FlexNVM IFR; programmed using the Program Partition command
- EEESIZE—allocated FlexRAM for EEE data based on EEESIZE of EEPROM data set size in FlexNVM IFR; programmed using the Program Partition command
- Write_efficiency:
  - 0.25 for 8-bit writes to FlexRAM
  - 0.5 for 16-bit or 32-bit writes to FlexRAM
- $n_{nvmcycd}$—FlexNVM memory cycling endurance (the following figure assumes 10,000 cycles)

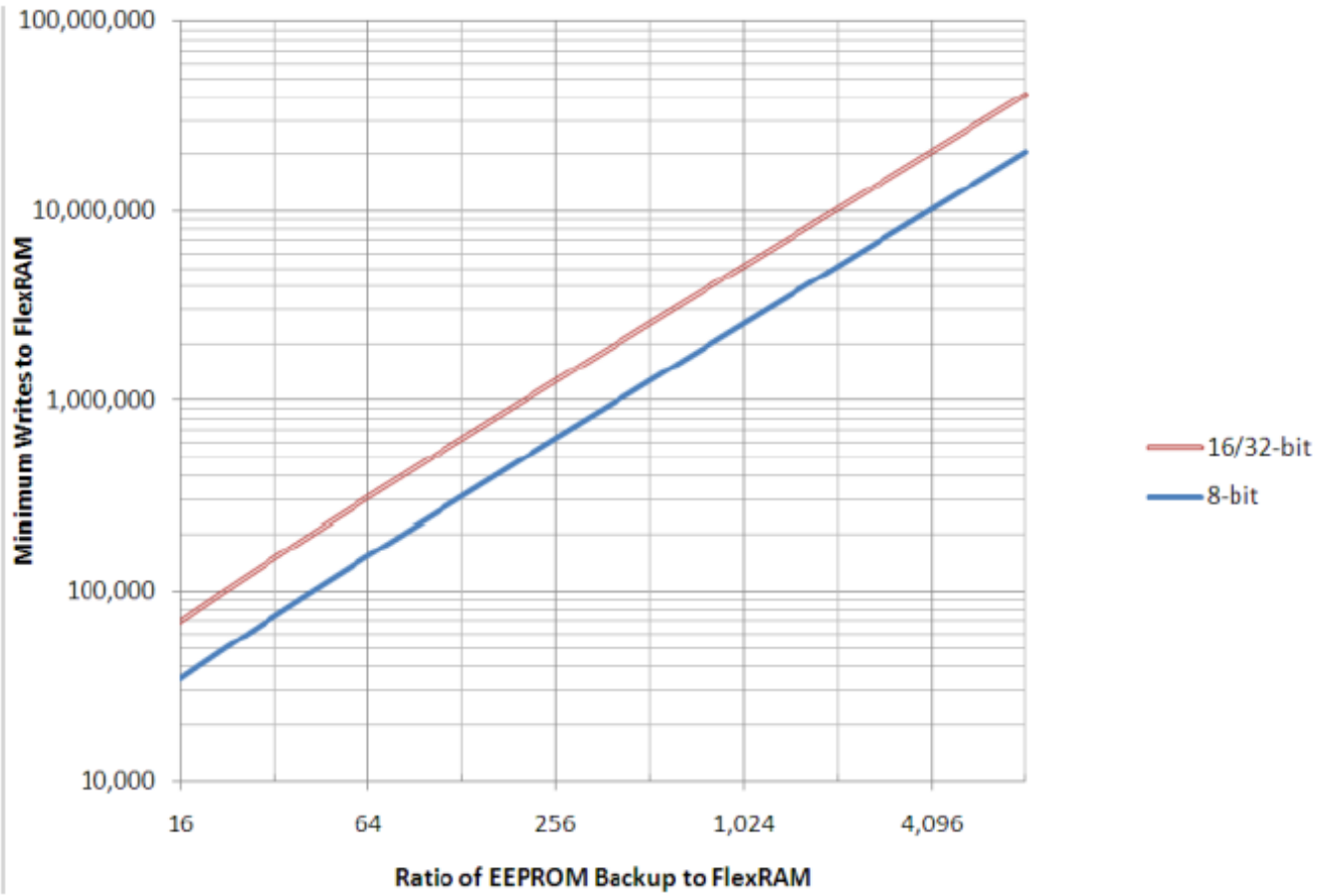


Figure 7. Enhanced EEPROM write endurance

Equation 1 on page 6 can be used to get a general idea of the amount of EEPROM backup memory that may be required for the system. However, the equation is simplified and intended to be used as a general guideline.

6 Example project

This example project discusses enhanced EEPROM operations on MC56F84xxx family DSC for user reference. To easily and visually validate project operation result, a FreeMASTER project is also developed to directly observe the data changes in EEE space.

Example project

Figure 8 shows the code process flow in example project. The process blocks in light green are the key functional modules for EEE operations. And for EEPROM mass write process block, it completely complies with the description of Enhanced EEPROM write operation. The FCNFG[EEERDY] bit is tested before each EEE write operation to assure that no other FlexNVM operation is in progress.

Besides, 8-bit, 16-bit, and 32-bit data written to EEE space are provided, and any type of write operation can be enabled through uncommenting the relevant code.

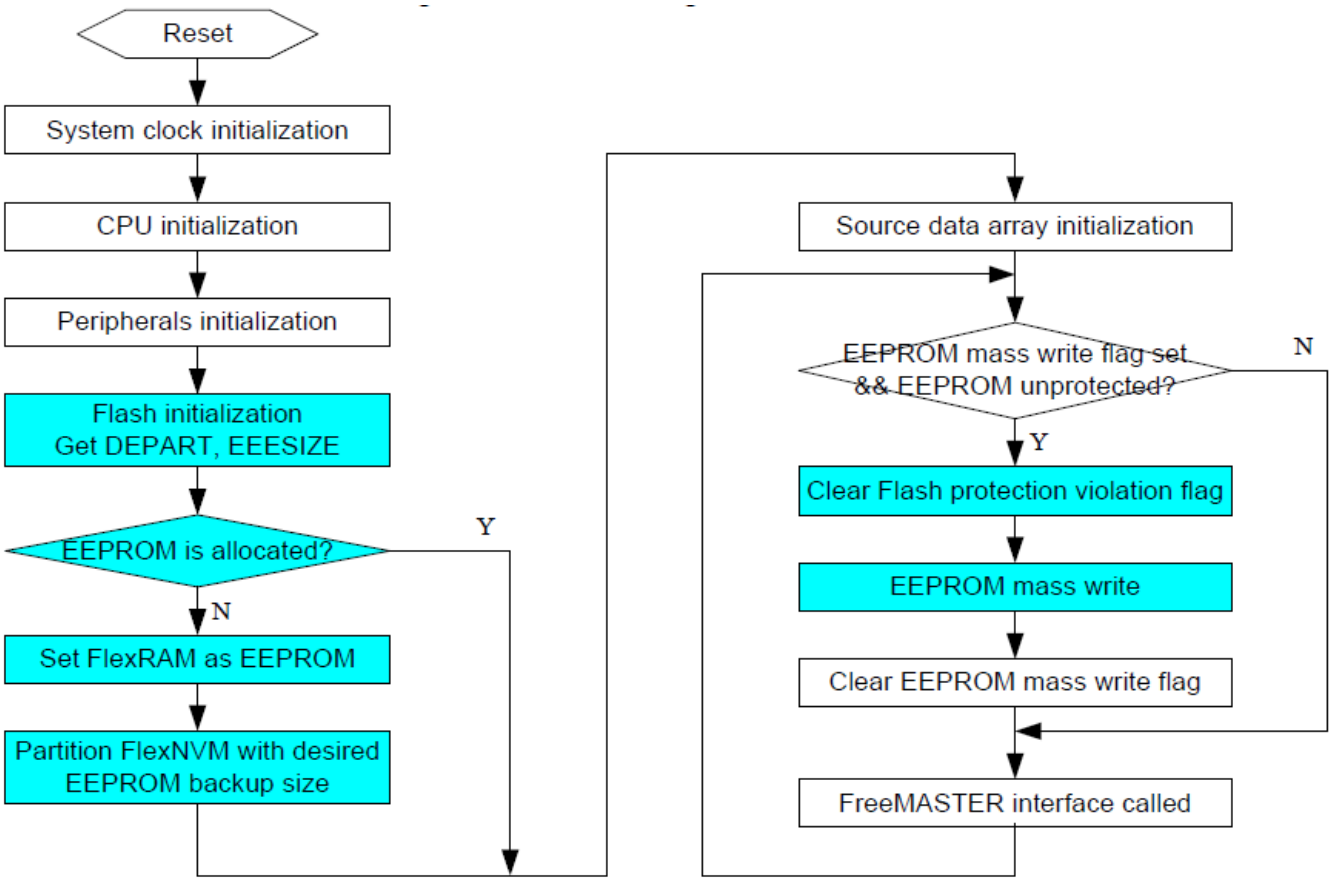


Figure 8. Flowchart of example project for EEE operations

The FreeMASTER project interface (open by Freescale free FreeMASTER application, which connects with target device through USBTAP) is shown in Figure 9. All variables are listed in watch grid window, which are divided into three parts:

- The first part (in purple rectangle) lists the variable of EEPROM mass write enable flag. When this flag is set to 1, all valid EEE space will be written by a predefined data array (source[SIZE_OF_BYTES])
- The secondary part (in red rectangle) lists the variables of on-chip memory configuration information, which help verify the correctness of flash memory initialization function in code.
- The third part (in blue rectangle) lists all the variables in valid EEE space; this variable data is organized via a 256-byte data array (eedata[SIZE_OF_BYTES]), which is assigned to valid EEE space (0x03C000 – 0x03C0FF, byte address) in code.

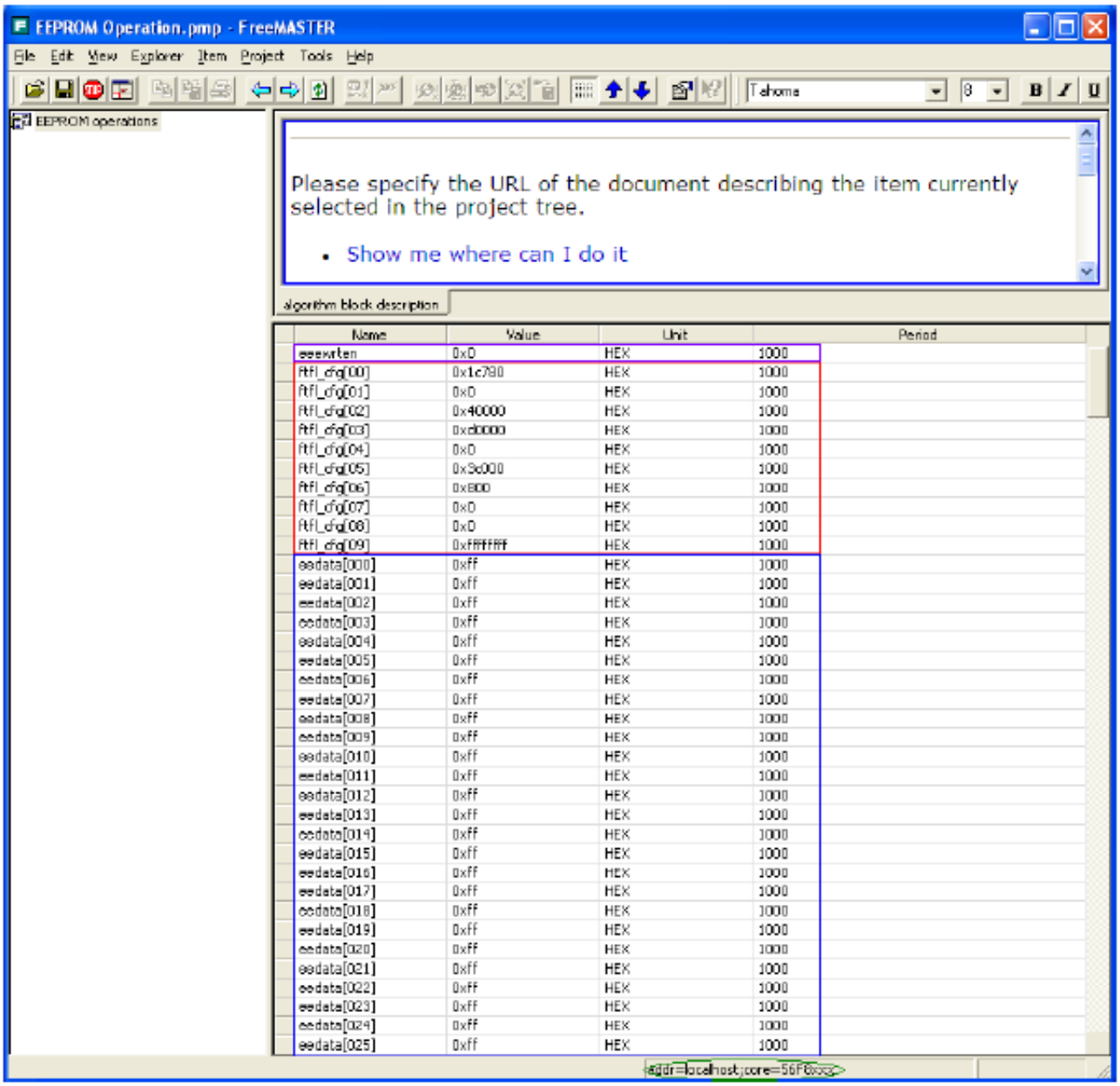
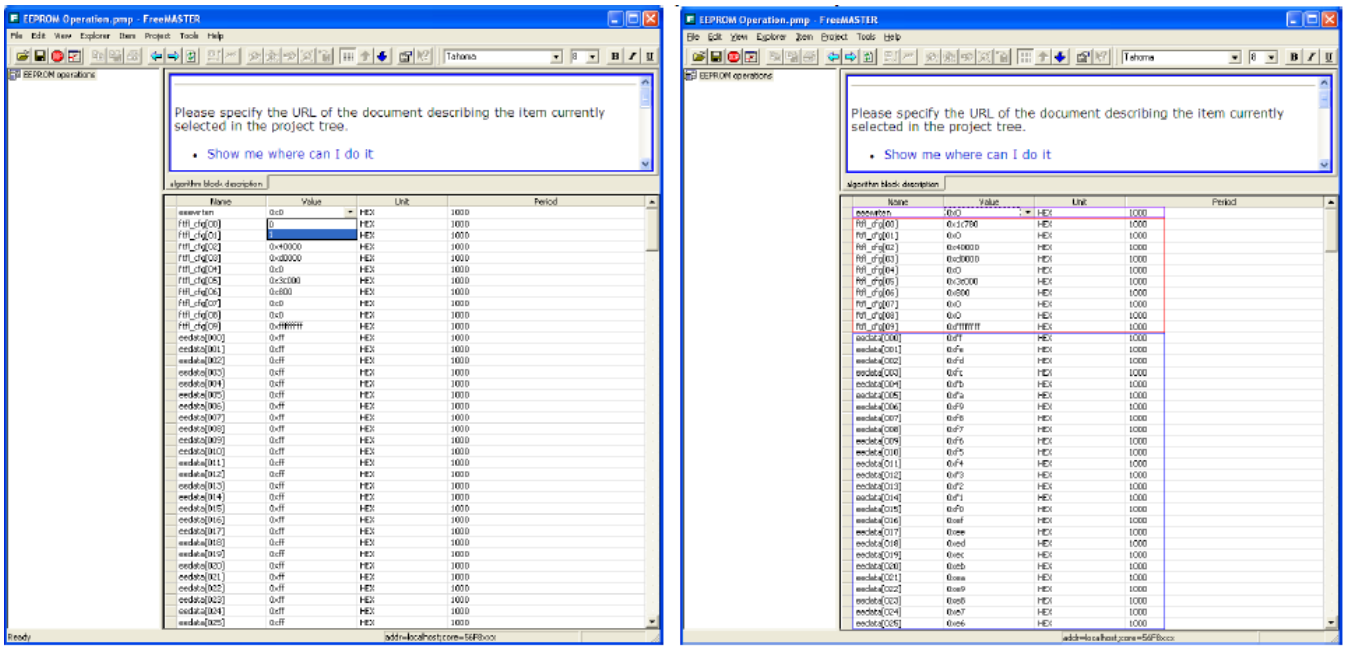


Figure 9. FreeMASTER project interface for EEE operations code

Figure 10 shows the EEPROM mass write operation. All valid EEE data is mass written by the equation of 0xFF minus its index when setting the variable of EEPROM mass write enable flag to 1. This operation can be observed in Figure 10 (a) and Figure 10 (b). When the chip is repowered after power off, these EEE data will keep the last updated value.

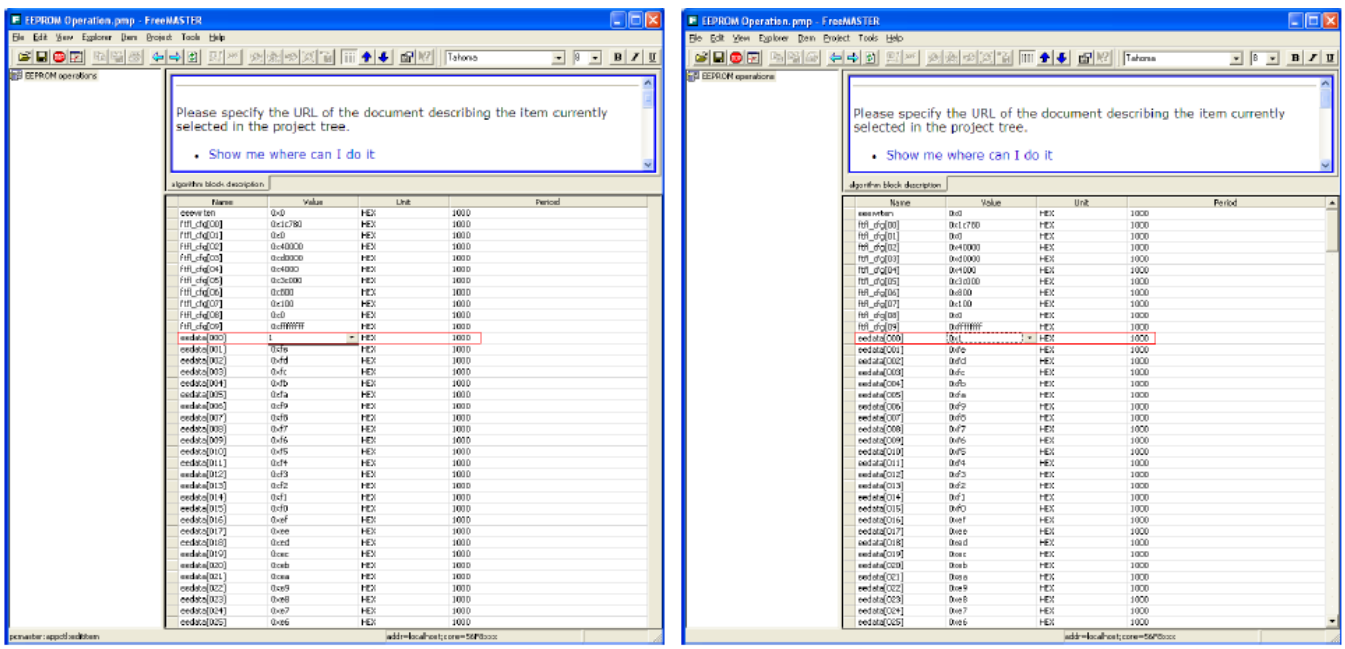


a) Before EEPROM mass write operation

b) After EEPROM mass write operation

Figure 10. EEPROM mass write operation for EEE operations example code

The individual of EEE space can also be updated independently. The value in EEE address of 0x03C000 (eedata[0]) is changed from 0xFF to 0x01 through directly changing the value in FreeMASTER interface as shown in Figure 11 (a) and Figure 11 (b). When the chip is repowered after power off, these EEE data will keep the last updated value.



a) Before EEPROM byte write operation

b) After EEPROM byte write operation

Figure 11. EEPROM byte write operation for EEE operations example code

7 Design considerations

As known from Enhanced EEPROM features, the EEE write operation to a pre-erased space can help reduce the program time because an erase cycle is not needed. This feature allows for quick data logging in time-critical situations. One typical case is that fault data and operating parameters need be stored when an imminent system power off is detected. The amount of data that needs to be saved times the maximum write time determines how much decoupling capacitors must be provided in this system to maintain minimum operating power long enough for the data to be stored. The significant decrease in the EEE program time by pre-erased data locations means that less decoupling capacitors is required and more data is stored in this situation before power off.

The EEE state machine can detect if any EEE data has not been fully programmed. Any situation where EEE data is detected as not fully programmed due to either a brownout or any reset during the write operation is treated the same. If a reset event happens while an EEE write is in progress, then the data can become corrupted. The EEE state machine tests vulnerable EEE data record for values that may not be fully programmed. If an incomplete record is detected, the state machine marks the data record as compromised and replaces it with the previous valid data record for the associated EEE address during the next EEE write operation. This ensures that if an EEE write is interrupted for any reason, the user will get the last value that was properly written to the EEE space. Depending on how far into the write the reset occurred, this value can be either the previous value or the new value, but the user will not get a corrupted value.

8 Conclusion

This application note summarizes the features, user perspective, SoC implementation and write endurance performance of enhanced EEPROM function on MC56F84xxx DSC, and provides example code as attached (see AN4689SW.zip file associated with this application note) to help better understand functional implementation, and then make it easy for readers to use this feature in their applications. Finally, some design considerations are shared with readers for better use.

9 Annexure

The example project is developed on MC56F84789 in CodeWarrior v10.3 IDE and the associated FreeMASTER project is also attached in the package.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4689
Rev. 0, 02/2013