Application Note

EEPROM Emulation Type B Design

Zhao Yuhao

ABSTRACT

This application note introduces the library of type B EEPROM emulation. The document describes the structure and behavior of the type B EEPROM emulation. Then the document introduces the use of the associated APIs. In addition, the relevant code is provided to the users. Users can call functions in applications to write, read, and modify data.

Table of Contents

List of Figures

List of Tables

Trademarks

1 Introduction

Many applications require store data in non-volatile memory, so that the application can be reused or modified even after the system is powered up again. EEPROM is design for such applications. Although the MSPM0 MCUs does not have an internal EEPROM, the MSPM0 internal Flash supports EEPROM emulation. Compared to using an external serial EEPROM, EEPROM emulation using the internal Flash saves pin usage and cost.

Different applications require different storage structures. The type B design described in this article is designed for storing small *variable* data. If the application needs to store large *block* of data, the type A design can be referenced.

1.1 Difference Between EEPROM and On-Chip Flash

The EEPROM can erase and write to a single byte of memory multiple times, allowing programmed locations to retain data for long periods of time even if the system is powered off.

Flash memory has a higher density than EEPROM, allowing larger memory arrays (sectors) to be implemented on chip. The flash erase and write cycle is performed by applying a time-controlled voltage to each unit. Each cell (bit) reads the logical value 1 under erase conditions. Therefore, each Flash location reads 0xFFFF After erase. By programming, the cell can be changed to a logic 0. Any word can be overwritten to change a bit from logical 1 to 0; however, the reverse is not possible. In addition, Flash memory has the restriction that the memory is erased by a region. For MSPM0 MCUs, the erase resolution is *sector* with the size of 1k bytes.

One major difference between EEPROM and Flash is cycling capability. Flash endurance is typically 10 thousand cycles, far less than real EEPROM. EEPROM emulation is a software based on Flash to provide the equivalent endurance that can be satisfied for the application. Flash also simulates the behavior of the EEPROM, simplifying the operation of reading and writing data.

A typical emulation scheme involves using a portion of the Flash memory and dividing it into multiple areas. These areas are used alternately to store data. Due to the block erase requirement of Flash, entire Flash sector has to be reserved for the EEPROM emulation. To achieve wear-leveling, at least two sectors are used.

2 Implementation

2.1 Principle

In the solution of this article, Flash are divided into areas which called *data item*. Each data item is 8 bytes, containing data, end-of-write flag and identifier. Identifier is used to identify and distinguish data, similar to variable name or virtual address. Two data items in flash can have the same identifier, but only the most recent data item is valid. Since data items are written sequentially, old and new data items can be distinguished based on position.

When a read operation is performed, the corresponding data item is found based on the identifier. If there are multiple data items with the same identifier, only the latest data item is valid. When a write operation is performed, data and identifiers are input, and assembled as new data item. Therefore, when the user wants to update the data item corresponding to an identifier, the write operation does not modify the previous data item, but continues to create new data item in the unused area. Typically, if the sector is full, the most recent data item corresponding to each identifier is transferred to the next sector, and the sector is then erased. As an extreme example, when the user only updates the data corresponding to a certain identifier, after the sector is full, the identifiers of all data items are the same, and the transfer operation will only transfer the last data item.

It should be noted that since the sector size is fixed, the number of data items that a single sector can hold is limited and fixed. To store more data items, sectors are grouped into *group*. Sectors in the same group will be erased together. When a group is full, the latest data items will be transferred to next group, and this group will be erased then. Figure 2-1 shows the structure of EEPROM emulation.

To mark the status of the group, the first 8 bytes of the group are used as the header. The remainder of the group (total group size minus the 8-byte size of the header) is used for storing data items. The number of data items in one sector is (SectorSize × Number of Sectors in One Group / DataItemSize - 1). For example, if one group has 1 sector, it can store 127 data items. If one group has 2 sectors, it can store 255 data items. If one group has 3 sectors, it can store 383 data items.

It should be emphasized that although users can use as many different identifiers as possible (at most equal to the number of data items), this might lead to frequent transfer operations and erase operations, which might lead to increased system overhead. The recommended number of identifiers is one-half to one-third of the maximum number of data items.

There are three user-configurable parameters, which can be configured in eeprom_emulation_type_b.h due to the application requirements. These parameters affect maximum number of data items and cycling capability, which will be analyzed later.

- Number of groups: at least 2
- Number of sectors in one group: at least 1
- Sector address

Additionally, in the structure of data item, end-of-write flag is used to ensure the write integrity of data items. The flag is set to 0x0000 after the data and identifier are written.

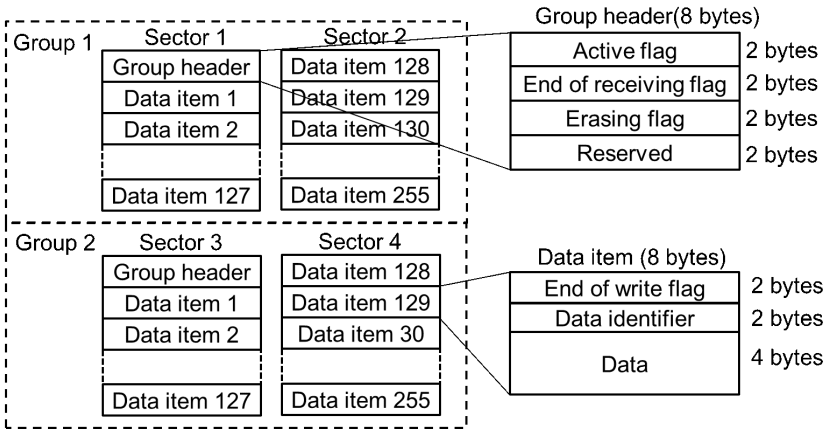

Figure 2-1. The Structure of EEPROM Emulation

The basic behaviors of EEPROM emulation can be seen in Figure 2-2. In A, although there are two Var3, meaning they have the same identifier, only the latter one is valid because it is newer. If var3 is read, x6 rather than x3 will be read. From A to B, write operation is performed and the group 1 is full, so the transfer operation will be performed. In C, Group 2 is marked as *Receiving*, and the latest data items are transferred to group 2. In D, after transfer group 2 is updated to *Active*. Group 1 is marked as 'Erasing' and waiting to be erased. The erase operation is performed only when the users call the function.

Users can choose the right time to erase according to the needs. It should be noted that not erasing in time will result in trying to write data into sectors with residual data. This may result in data corruption.

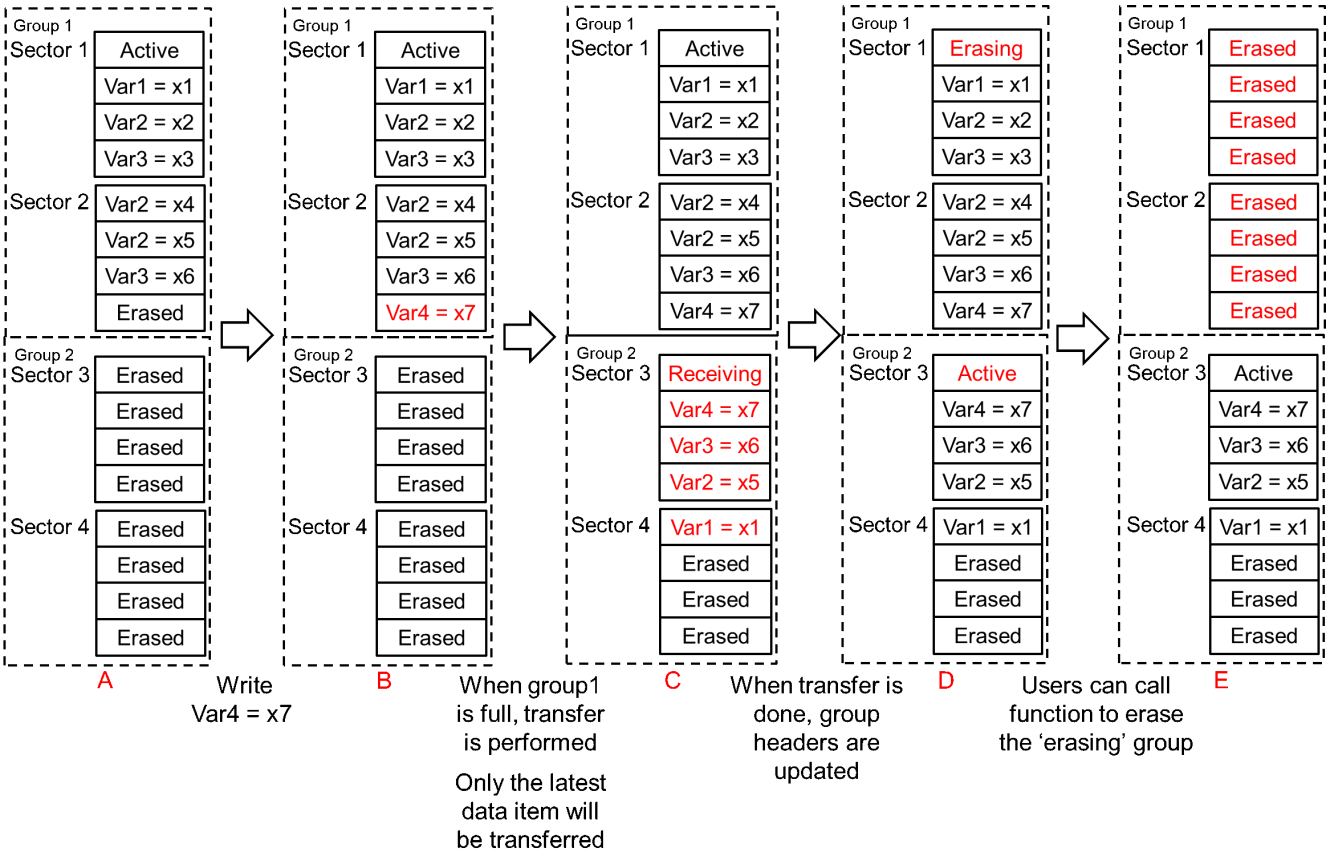

Figure 2-2. The Basic Behaviors of EEPROM Emulation

2.2 Header

Header is designed to manage the group. By checking the header of a single group, the status of the group can be determined. By checking the headers of all groups, the active group can be found and the format of EEPROM emulation can be checked.

Each group has a header to show its status. Header is set to 8 bytes with 3 flags. Depending on the flags, there are four record status in total. The relationship between flags and record status is showed in Table 2-1.

Table 2-1. Relationship Between Flags and Record Status

Group Status	Description	Active flag	End of Receiving Flag	Erasing Flag
Erased	Default state after erase	0xFFFF	0xFFFF	0xFFFF
Receiving	When transferring data items, *Receiving* group receives latest items from the full group	0x0000	0xFFFF	0xFFFF
Active	*Active* group is the group not full of items, and it is waiting for new items to be written	0x0000	0x0000	0xFFFF
Erasing	*Erasing* group is the group waiting to be erased	-	-	0x0000

Figure 2-3 shows how states translate to each other. All flags are erased at first. If *erased* group is written data item to, it will be changed to *Active* and waiting to be written.

If the group is full, next group will be changed to *Receiving*, and the latest data items are transferred to the *Receiving* group. After transfer, the old full group will be changed to *Erasing* and waiting to be erased. The *Receiving* group will be *Active* then.

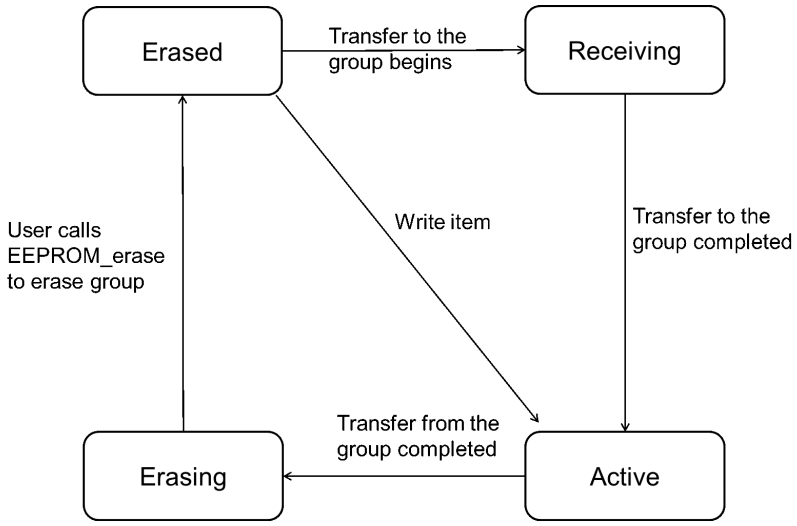

Figure 2-3. How Record Status Change

3 Software Description

This software provides basic EEPROM functionality. At least two sectors are used to emulate the EEPROM. These sectors are organized into groups, and used to store data items. Four global variables are used to trace the EEPROM emulation.

3.1 Software Functionality and Flow

In total, there are six functions. The first four functions are called directly by the user. The last two functions are called by these functions.

- EEPROM_TypeB_init
- EEPROM_TypeB_write
- EEPROM_TypeB_readDataItem
- EEPROM_TypeB_eraseGroup
- EEPROM_TypeB_findDataItem
- EEPROM_TypeB_transferDataItem

The high-level software flow is shown in Figure 3-1. The device should first go through the initialization code. By calling EEPROM_TypeB_init, it will search active group and check the format of Flash. If active group exists, the global variables are updated to trace the active group and latest data item. If active group does not exist, flash will be initialized.

In the application, user can use EEPROM_TypeB_readDataItem to read the data according to the input identifier. And user can use EEPROM_TypeB_write to write data and identifier. If the group is full, the latest data items are transferred to next group. After transfer, the full group will be marked as *Erasing* with setting the erase flag. In the flow chart below, EEPROM_TypeB_eraseGroup is called immediately when the erase flag is set. Users can select the appropriate point in time to erase based on the requirement of application.

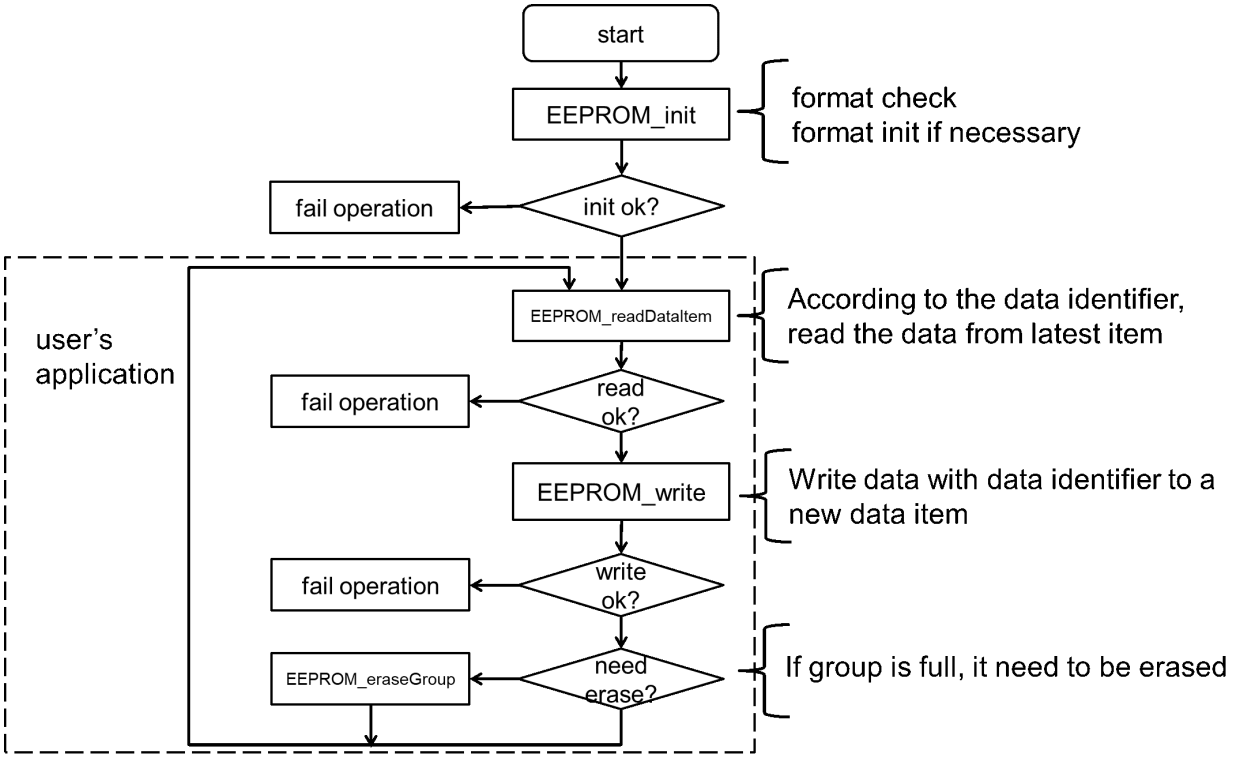

Figure 3-1. The High-Level Software Flow

3.2 EEPROM Functions

To implement this functionality, six functions are required. Four global variables are used to record the status of the EEPROM emulation.

3.2.1 Global Variables

Two global variables are used to trace the active group.

- uint16_t gActiveDataItemNum;
- uint16_t gActiveGroupNum;

gActiveDataItemNum is used to record the number of data items.

gActiveGroupNum is used to record the active group.

Two global variables are used for flags.

- bool gEEPROMTypeBSearchFlag;
- bool gEEPROMTypeBEraseFlag;

gEEPROMTypeBSearchFlag is set when EEPROM_TypeB_readDataItem finds the data item based on input identifier.

gEEPROMTypeBEraseFlag is set when the group is full and needs to be erased.

All global variables are defined in eeprom_emulation_type_b.c

3.2.2 EEPROM_TypeB_readDataItem

EEPROM_TypeB_readDataItem is used to read the data item that matches the input identifier. The software flow is shown in Figure 3-2. It calls EEPROM_TypeB_findDataItem to find the data item.

The input of the function is the identifier. The output of the function is the data. Besides, gEEPROMTypeBSearchFlag is used to show whether the data item is found.

- Input: uint16_t data identifier
- Output:uint32_t data

Figure 3-3 shows the different cases of EEPROM_TypeB_readDataItem. If data item is found, the function will return the data, and gEEPROMTypeBSearchFlag is set. If not, the function will return 0, and gEEPROMTypeBSearchFlag is clear. By checking the flag, users can judge whether the data is found and read successfully.

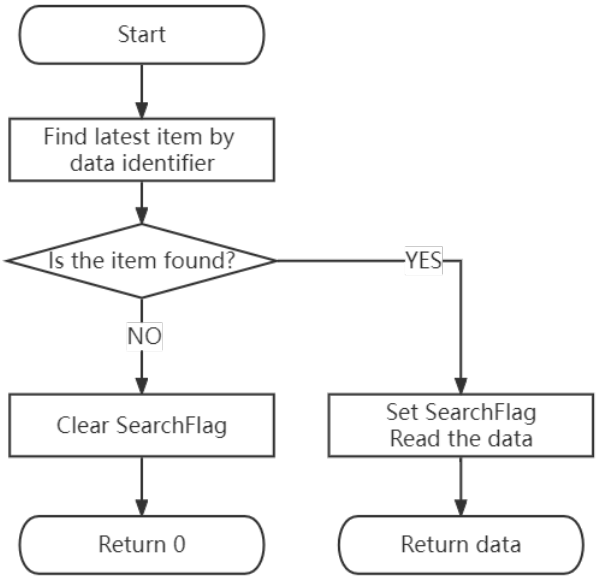

Figure 3-2. The Software Flow of EEPROM_TypeB_readDataItem

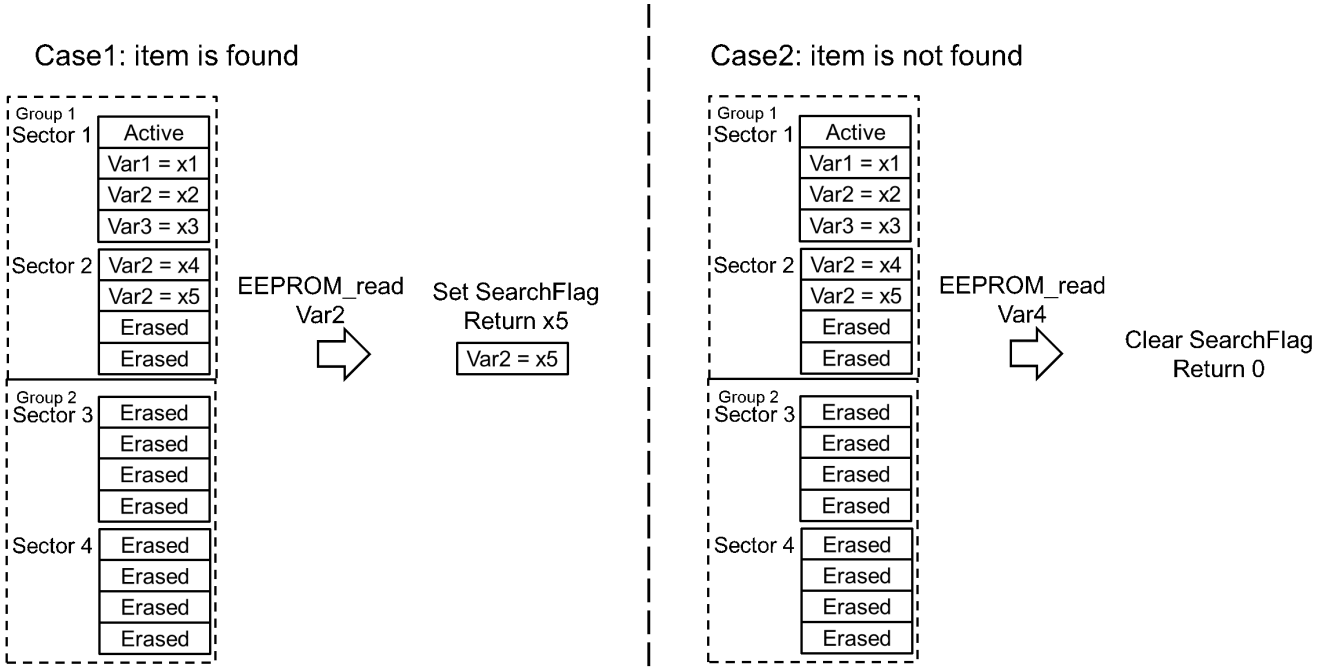

Figure 3-3. The Different Cases of EEPROM_TypeB_readDataItem

3.2.3 EEPROM_TypeB_findDataItem

EEPROM_TypeB_findDataItem is used to search for data items within the specified group. The software flow is shown in Figure 3-4. The search traverses the data items in the group from back to front, so the first data item found that matches the identifier must be the latest data item.

The input of the function is the identifier, GroupNum and DataItemNum. GroupNum specifies the group in which to search. DataItemNum specifies the maximum number of data items to be searched. The output of the function is the address. If data item is found, the function returns the address of the data item. If not, the function returns EEPROM_EMULATION_FINDITEM_NOT_FOUND (value is 0).

- Input: uint16_t data identifier

uint16_t GroupNum

uint16_t DataItemNum

- Output:uint32_t address

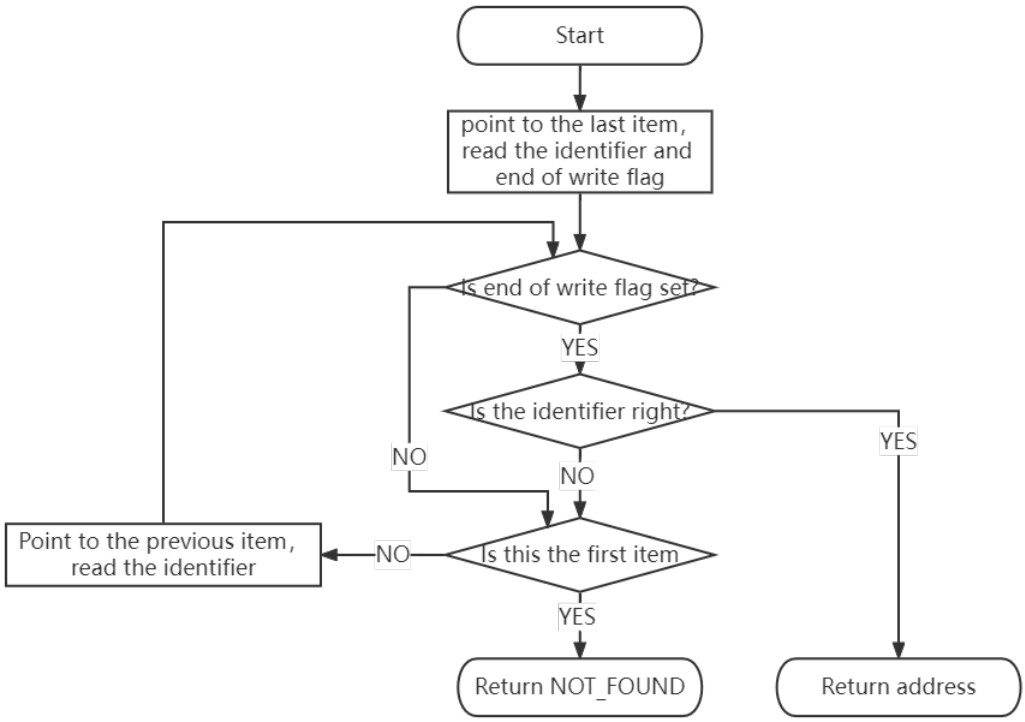

Figure 3-4. The Software Flow of EEPROM_TypeB_findDataItem

3.2.4 EEPROM_TypeB_write

EEPROM_TypeB_write is used to write provided data and identifier to the Flash. Through the function, a new data item is added to the Flash. If group is full, EEPROM_TypeB_transferDataItem is called and transfer is performed. The software flow is shown in Figure 3-5.

First, the function checks whether next data item is erased. Then it brings together the data and the identifier into a new data item, and set the end of write flag to ensure the data integrity. If the end of write flag is not set, the data item is invalid and all functions will skip this item.

Finally, the function checks whether the group is full. If so, transfer is performed. Refer to EEPROM_TypeB_transferDataItem to see more details.

This scheme allows users to choose which identifiers to use, but also requires users to pay attention to the number of identifiers used. The recommended number of identifiers is one-half to one-third of the maximum number of data items. If the number of identifiers is close to the maximum number of data items, frequent transfers and erasures will occur, increasing system overhead. An error is caused if the number of identifiers exceeds the maximum number of data items.

The input of the function is data and data identifier. The output of the function is the operation states. Besides, gActiveGroupNum and gActiveDataItemNum are updated to trace the active group.

- Input: uint32_t data

uint16_t data identifier

- Output:uint32_t operation state

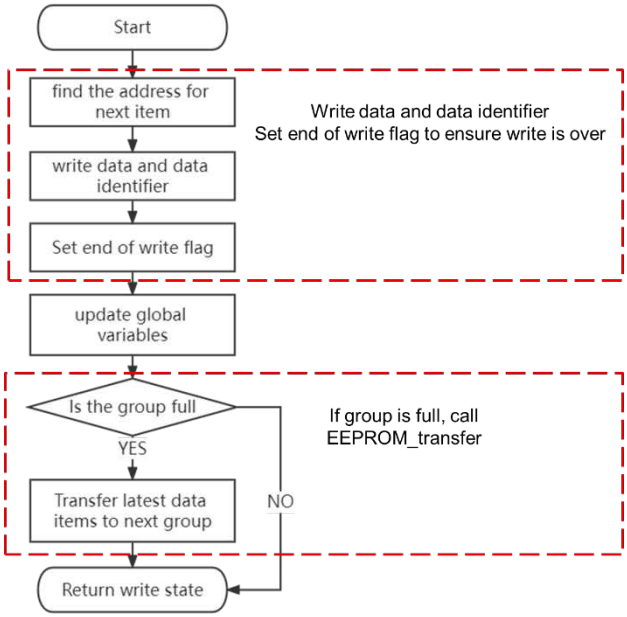

Figure 3-5. The Software Flow of EEPROM_TypeB_write

3.2.5 EEPROM_TypeB_transferDataItem

EEPROM_TypeB_transferDataItem is used to transfer the latest data items from one group to next group. Not all items will be transfer. Only the latest data item corresponding to each identifier is transferred.

The software flow is shown in Figure 3-6. First, it updates next group to be *Receiving*. Then it traverses the current group from back to front, checking if the data item already exists in the *Receiving* group. If not, the data item will be transferred. If it already exists, skip the data item. After transfer, the latest data item has been transferred to the *Receiving* group. Finally, *Receiving* group is updated to *Active* group. The group which is transferred is marked as *Erasing*, and gEEPROMTypeBEraseFlag is set. The process of transfer is shown in Figure 2-2.

By checking gEEPROMTypeBEraseFlag, users can call EEPROM_TypeB_eraseGroup to erase all *Erasing* group. Users can arrange erasure time points according to application needs.

The input of the function is GroupNum to choose which group to be transferred. The output of the function is the operation states. Besides, gEEPROMTypeBEraseFlag is updated to show if erasing is required.

- Input: uint16_t GroupNum
- Output:uint32_t operation state

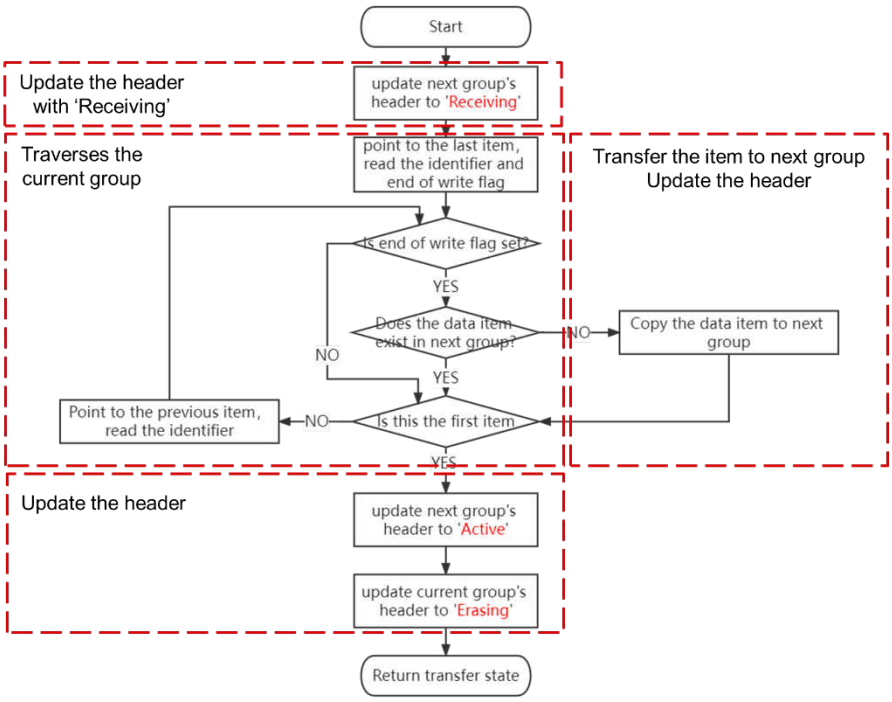

Figure 3-6. The Software Flow of EEPROM_TypeB_transferDataItem

3.2.6 EEPROM_TypeB_eraseGroup

EEPROM_TypeB_eraseGroup is used to erase the *Erasing* groups. After calling EEPROM_TypeB_transferDataItem, gEEPROMTypeBEraseFlag is set. It prompts the user that there is an *Erasing* group. It is recommended to call the EEPROM_TypeB_eraseGroup immediately when gEEPROMTypeBEraseFlag is set, as in Figure 3-1. However, users can change the timepoint to erase by modifying the high-level software flow.

The output of the function is the operation states.

- Input: void
- Output:uint32_t operation state

3.2.7 EEPROM_TypeB_init

The function is used to initialize the EEPROM emulation. It should be done once before using EEPROM emulation, for example after the device is powered up. This ensures that the relevant Flash area is properly formatted and global variables are correctly assigned.

The software flow is shown in Figure 3-7. Firstly, it searches the active group and checks the format by iterating over all group headers. If active group exists, it will erase other groups and transfer the active group. If active group does not exist, all groups are erased.

Figure 3-8 and Figure 3-9 shows different cases for EEPROM_TypeB_init. A is a normal case. B is the initialization after power down during transfer. C is a case where *Erasing* groups is not erased. D is case where all groups are empty. E is the case with invalid data.

The output of the function is the operation states. Besides, gActiveGroupNum and gActiveDataItemNum are updated to trace the active group.

- Input: void
- Output:uint32_t operation state

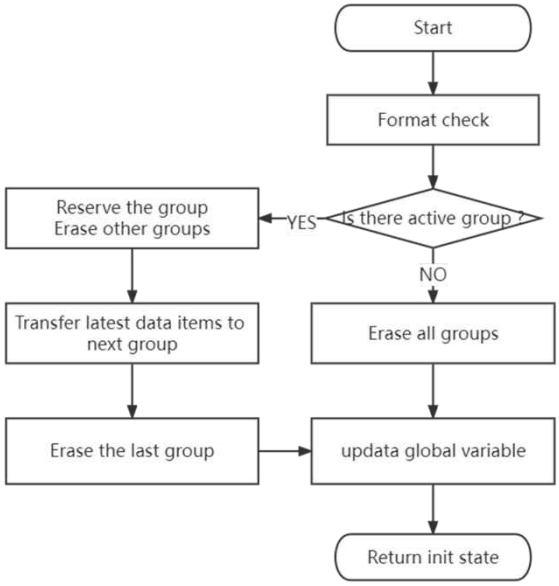

Figure 3-7. The Software Flow of EEPROM_TypeB_init

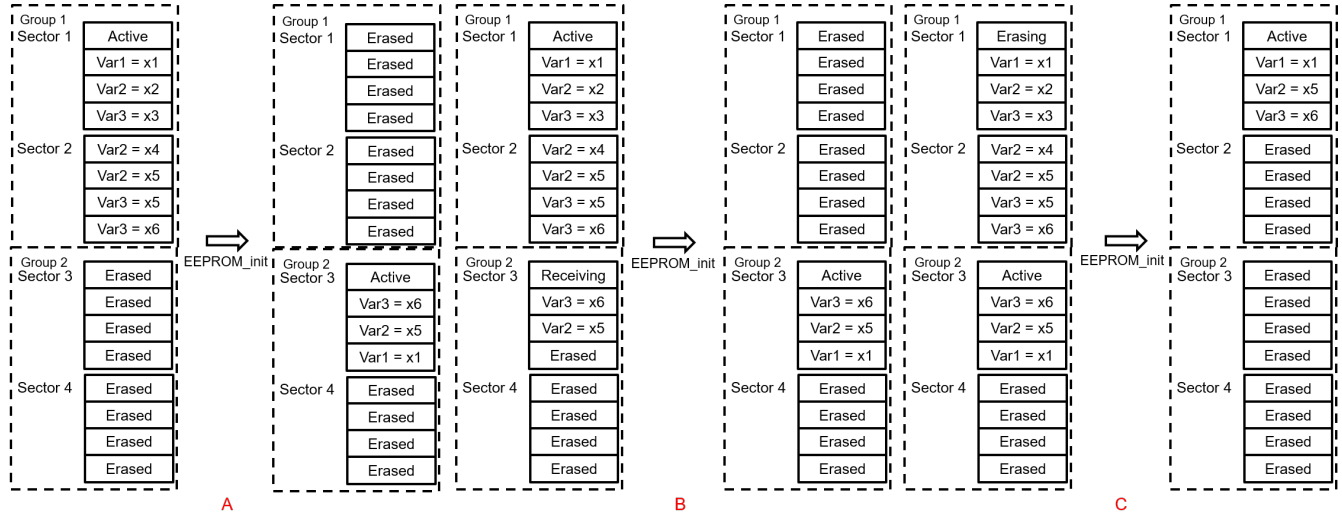

Figure 3-8. EEPROM_TypeB_init for Three Cases When Active Group Exists

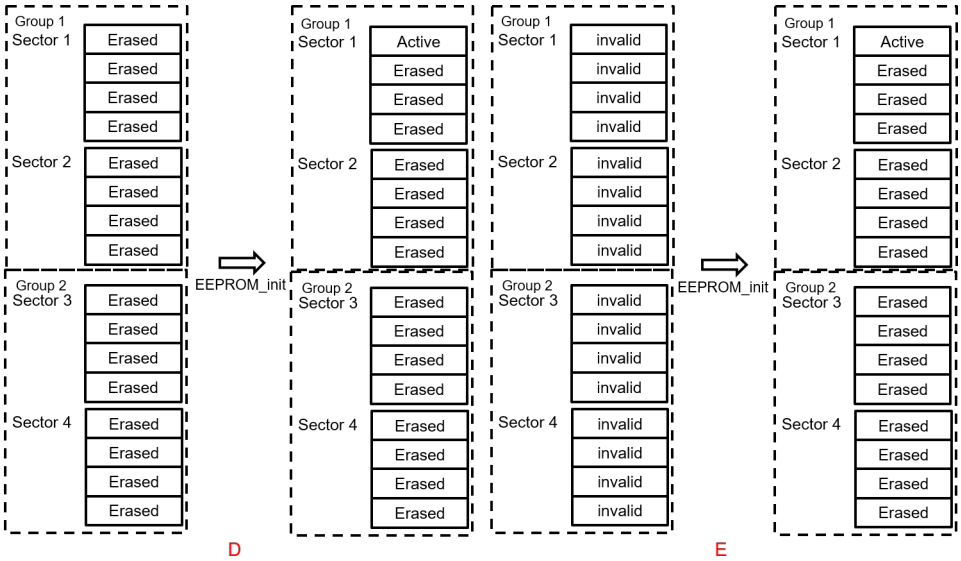

Figure 3-9. EEPROM_TypeB_init for Two Cases When Active Group Does Not Exist

3.3 Application Integration

Applications requiring this functionality need to include the eeprom_emulation_type_b.c and eeprom_emulation_type_b.h files provided for MSPM0 MCUs. The Flash API also needs to be included for the specific device. For example, for the MSPM0G3507/MSPM0L1306 the following files are needed:

- eeprom_emulation_type_b.c
- eeprom_emulation_type_b.h
- ti_msp_dl_config.c
- ti_msp_dl_config.h

EEPROM emulation library has been included in the SDK supporting MSPM0 products.

All Flash API files are also included in the SDK.

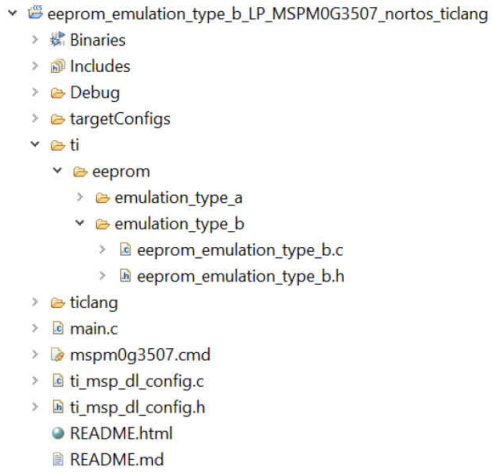

Figure 3-10. Files Required by the Software

3.4 EEPROM Emulation Memory Footprint

Table 3-1 details the footprint of the EEPROM emulation driver in terms of Flash size and RAM size. Table 3-1 have been determined using the Code Composer Studio (Version: 11.2.0.00007) with optimization level 2.

Table 3-1. The Structure of EEPROM Emulation

Mechanism	Minimum Required Code Size (bytes)	
	Flash	SRAM
EEPROM emulation type B	2816	6

3.5 EEPROM Emulation Timing

This section describes the timing parameters associated with the EEPROM emulation driver based on four 1-Kbyte Flash sectors. All timing measurements are performed:

- MSPM0G3507
- System clock at 32 MHz
- With execution from Flash
- At room temperature

The functions are test at the parameters:

- Number of groups: 2
- Number of sectors in one group: 2
- Sector address: 0x00001000

Table 3-2. Timings of EEPROM Emulation Operations.

Operation	Minimum	Typical	Maximum
EEPROM_TypeB_readDataItem	2.2us		89.4 us
EEPROM_TypeB_init with active group	4.65ms		61.38 ms
EEPROM_TypeB_init without active group		4.26 ms	
EEPROM_TypeB_write without transfer		109.5 us	
EEPROM_TypeB_write with transfer	630.8 us		39.09ms
EEPROM_TypeB_eraseGroup		12.2 ms	

4 Application Aspects

This section describes the application-level features of EEPROM emulation solution and how to configure it to meet the application needs.

4.1 Selection of Configurable Parameters

There are three user-configurable parameters in eeprom_emulation_type_b.h. These parameters can be configured accordingly, depending on the requirements of the application.

- Number of groups: at least 2
- Number of sectors in one group: at least 1
- Sector address

When only the number of groups is configured, the change can be seen in Figure 4-1. When only the sectors in one group is configured, the change can be seen in Figure 4-2. Both configuration methods will increase the Flash area used, thereby increasing the cycling capability. But when configuring the number of groups, the maximum number of data items does not change. When configuring the number of sectors in one group, the maximum number of data items will change accordingly.

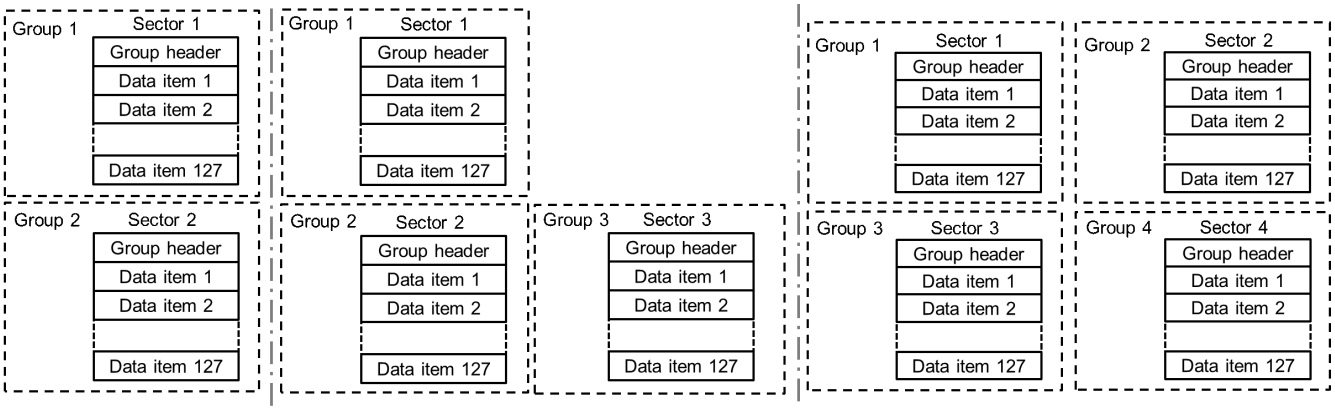

Figure 4-1. Change of Structure When Only the Number of Groups is Configured

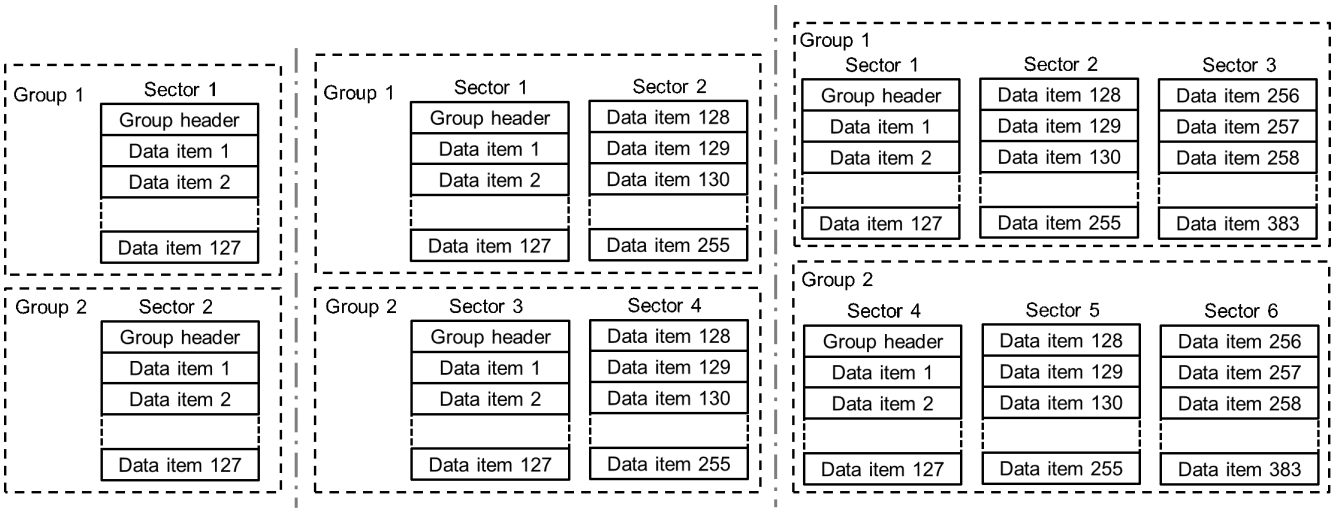

Figure 4-2. Change of Structure When Only the Sectors in One Group is Configured

4.1.1 Number of Data Items

The number of data items is directly related to the number of sectors in the group.

$$\text{Number of data items} = \frac{\text{Sector size}}{\text{Data item size}} \times \text{Number of sectors in one group} - 1$$

Choosing the correct number of data items is critical. The point is to choose based on how many variables your application needs to store. If the number of variables is close to the number of data items, the transfer will occur frequently when updating the values of those variables. If the number of variables is much less than the number of data items, it means that the size of the group is relatively large, and additional time will be spent in operations such as transfer, erasure, and search.

The recommended number of identifiers is one-half to one-third of the maximum number of data items.

4.1.2 Cycling Capability

Flash endurance is typically 10 thousand cycles, far less than real EEPROM. One feature of EEPROM emulation is that its endurance is extended compared to Flash.By dividing the Flash sector into multiple data items and writing data one by one, the Flash sector does not need to erase each time it writes, but only after it is full. And the equivalent endurance is further enhanced by the use of multiple Flash sectors.

Both the number of sectors in one group and the number of groups affect the equivalent erase/write capability. Since the number of sectors in one group has been determined above, the appropriate equivalent erase/write capability is now obtained by adjusting the number of groups.

$$\text{Effective endurance} = \frac{\text{Number of data items}}{\text{user's data items}} \times \text{Number of groups} \times \text{flash endurance}$$

It is recommended that the user evaluate the cycling capability required by the application before selecting the appropriate number of groups.

For three user-configurable parameters, a recommended design process is as follows:

1. Evaluate the number of data items required by application and select the appropriate number of sectors in one group.
2. Evaluate the cycling capability required by application, and select the appropriate the number of groups.
3. Select the appropriate Flash address

For example, if there is an application which needs to update data in EEPROM every 10 minutes. There are totally 20 variables, and 10 years of continuous service is guaranteed. Firstly, each group just need 1 sector (127 data items). Secondly, cycling capability required by application is 525600 cycles (10 years × 365 days × 24 hours × 6 cycles per hour). Nine groups are needed and the equivalent endurance is 571500 cycles.

4.2 Recovery in Case of Power Loss

Data corruption is possible in case of a power loss during EEPROM_TypeB_write or EEPROM_TypeB_eraseGroup.

To detect the corruption and recover from it, EEPROM_TypeB_init is implemented. It should be called immediately after power-up. EEPROM_TypeB_init checks all groups' header to confirm whether data storage of EEPROM emulation is correct.

In the structure of EEPROM emulation, headers show the status of corresponding groups. There are four states in total. The changes between the four states are described in detail in the previous section.

5 References

1. Texas Instruments, *EEPROM Emulation Using Low-Memory MSP430™ FRAM MCUs* application note.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265