*Subsystem Design*

# *Emulate EEPROM With Flash (Type B)*

*Yuhao Zhao*

## 1 Description

This subsystem demonstrates how to implement Electrically Erasable Programmable Read-Only Memory (EEPROM) emulation (Type B) in the application. EEPROM emulation allows a device to emulate EEPROM in Flash memory, and make an equivalent durability similar to EEPROM. The following features are available using Flash memory:

- Data retention after unexpected power loss
- Flexible structure for different applications
- User-configured erase operation

Download the code for this example.

There are two types of EEPROM emulation libraries for MSPM0. *Type A* is to store one large block of data (64 bytes, 128 bytes or 256 bytes) with a Static Random Access Memory (SRAM) buffer. See the *EEPROM Emulation Type A Design* application note for details of the library. *Type B* is to store many small data items (16bit or 32bit) with item identifiers. See also the *EEPROM Emulation Type B Design* application note for details of the library.

This subsystem shows the usage of *Type B*. For the *Type A* subsystem, see the *Emulate EEPROM With Flash (Type A)* subsystem design.

Figure 1-1 shows a functional diagram of this subsystem.

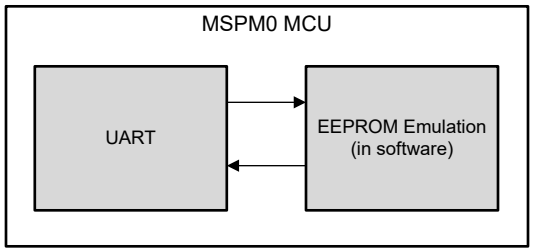


**Figure 1-1. Subsystem Functional Block Diagram**

## 2 Required Peripherals

This application requires Flash.

**Table 2-1. Required Peripherals**

| Subblock Functionality | Peripheral Use | Notes |
|---|---|---|
| Flash API | (1 ×) Flash | Called *FLASHCTL* in code |

## 3 Compatible Devices

Based on the requirements in Table 2-1, this example is compatible with the devices in Table 3-1. The corresponding EVM can be used for prototyping.

**Table 3-1. Compatible Devices**

| Compatible Devices | EVM |
|---|---|
| MSPM0Gxxxx | LP-MSPM0G3507 |
| MSPM0Lxxxx | LP-MSPM0L1306 |
| MSPM0Cxxxx | LP-MSPM0C1104 |
| MSPM0Hxxxx | LP-MSPM0H3216 |

## 4 Design Steps

1. Add the EEPROM emulation library. The MSPM0 software development kit (SDK) has included the EEPROM emulation library.

   **Note**

   The EEPROM emulation library is based on the Flash API so the `drivelib` from SDK is also required.

   For Type B, the following files are needed:
   a. eeprom_emulation_type_b.c
   b. eeprom_emulation_type_b.h

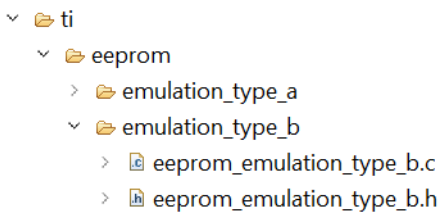


**Figure 4-1. EEPROM Emulation Files**

2. Add the include path in the code for eeprom_emulation_type_b.h.

```
#include <ti/eeprom/emulation_type_b/eeprom_emulation_type_b.h>
```

   Users can modify the start address, the number of groups (at least 2) and the number of sectors in one group (at least 1) in eeprom_emulation_type_b.h. The default Flash address used for EEPROM emulation is 0x00001400, and 6 sectors (3 groups, 2 sectors in every group) are used by default, so 0x00001400-0x00002bff is occupied.

   a. `#define EEPROM_EMULATION_ADDRESS (0x00001400)`
   b. `#define EEPROM_EMULATION_GROUP_ACCOUNT (3)`
   c. `#define EEPROM_EMULATION_SECTOR_INGROUP_ACCOUNT (2)`

3. Add the `initialize` function at the beginning of `main()`, typically after `SYSCFG_DL_init()`. This allows the relevant Flash areas to be formatted correctly and global variables to be allocated correctly to trace the active group and latest data item.
   - `EEPROMEmulationState = EEPROM_TypeB_init();`

```
/* Initialize */
EEPROMEmulationState = EEPROM_TypeB_init();
if (EEPROMEmulationState == EEPROM_EMULATION_INIT_ERROR ||
    EEPROMEmulationState == EEPROM_EMULATION_TRANSFER_ERROR) {
    /* If there is an error, it will stop here */
    //__BKPT(0);
}
```

**Figure 4-2. EEPROM Emulation Initialization**

4. Use `EEPROM_TypeB_write` to write 32-bit data with a 16-bit identifier.
   - `EEPROMEmulationState = EEPROM_TypeB_write(var_id, var_data);`

```
/* Write operation */
EEPROMEmulationState = EEPROM_TypeB_write(var_id, var_data);
if (EEPROMEmulationState != EEPROM_EMULATION_WRITE_OK) {
    /* If there is an error, it will stop here */
    //__BKPT(0);
}
```

**Figure 4-3. EEPROM Emulation Write**

5. Use `EEPROM_TypeB_readDataItem` to read the data according to the input identifier.
   - `var_data_read = EEPROM_TypeB_readDataItem(var_id);`

```
/* Read operation */
var_data_read = EEPROM_TypeB_readDataItem(var_id);
if (gEEPROMTypeBSearchFlag == 0) {
    /* If identifier is not found, it will stop here */
    //__BKPT(0);
}
```

**Figure 4-4. EEPROM Emulation Read**

6. Add the erase function according to the `gEEPROMTypeBEraseFlag`. Flash needs to be erased before writing data again and the smallest unit of erasure is sector. For EEPROM emulation Type B, after one group is full, `gEEPROMTypeBEraseFlag` is set. Users can call `EEPROM_TypeB_eraseGroup()` according to the flag. For example add the following code after `EEPROM_TypeB_write()` from step 4. Users can also choose an appropriate timepoint to erase the full group, as needed.
   - `EEPROM_TypeB_eraseGroup();`

```
/* Erase operation */
if (gEEPROMTypeBEraseFlag == 1) {
    /*
     * In this demo, gEEPROMTypeBEraseFlag is checked after write
     * operation. If the group is full, it will be erased immediately
     */
    //__BKPT(0);
    EEPROM_TypeB_eraseGroup();
    gEEPROMTypeBEraseFlag = 0;
}
```

**Figure 4-5. EEPROM Emulation Erase**

After steps 1 through 6, the EEPROM emulation Type B is implemented in the application. See Section 6 for the flow.

## 5 Design Considerations

1. 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. To set appropriate parameters, see the *application aspects* section in the *EEPROM Emulation Type B Design* application note.
   a. Number of groups: at least 2
   b. Number of sectors in one group: at least 1
   c. Sector address
2. The number of data items is directly related to the number of sectors in the group.

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

   Choosing the correct number of data items is critical. The point is to choose based on how many variables the application needs to store. If the number of variables is close to the number of data items, the transfer occurs frequently when updating the values of those variables. If the number of variables is much less than the number of data items, this means that the size of the group is relatively large, and additional time is 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.
3. To evaluate the Flash usage and cycling capability, see the *application aspects* section in the *EEPROM Emulation Type B Design* application note.
4. Data corruption is possible in case of a power loss during `EEPROM_TypeB_write` or `EEPROM_TypeB_eraseGroup`.

   To detect and recover from the corruption, implement `EEPROM_TypeB_init`. Call `EEPROM_TypeB_init` immediately after power up. `EEPROM_TypeB_init` checks the header of all the groups 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 Section 4.

## 6 Software Flow Chart

Figure 6-1 shows the code flow chart for *EEPROM Emulation Type B*, which introduces how to add functions in the application code to implement EEPROM emulation. Three functions are required here: `EEPROM_TypeB_init`, `EEPROM_TypeB_write`, `EEPROM_TypeB_readDataItem`, and `EEPROM_TypeB_eraseGroup`.

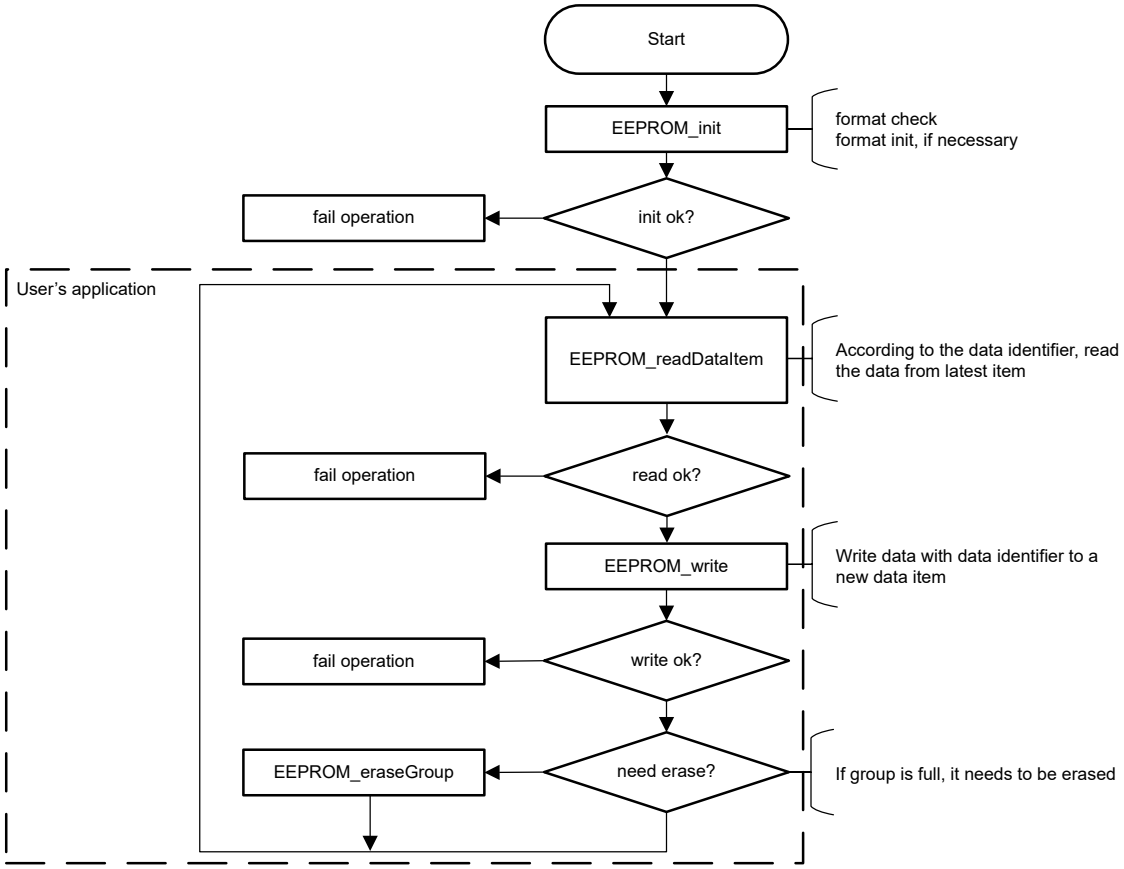


**Figure 6-1. Application Software Flow Chart**

# 7 Application Code

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

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

The following 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`.

# 8 Additional Resources

- Texas Instruments, *EEPROM Emulation Type A Design Application Note*
- Texas Instruments, *EEPROM Emulation Type B Design Application Note*
- Texas Instruments, *Emulate EEPROM With Flash (Type A) Subsystem Design*
- Texas Instruments, Download the MSPM0 SDK
- Texas Instruments, Learn more about SysConfig

## Trademarks

All trademarks are the property of their respective owners.

# 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