Subsystem Design

Emulate EEPROM With Flash (Type A)

Yuhao Zhao

1 Description

This subsystem demonstrates how to implement Electrically Erasable Programmable Read-Only Memory (EEPROM) emulation (Type A) 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 A*. For the *Type B* subsystem, see the *Emulate EEPROM With Flash (Type B)* subsystem design.

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

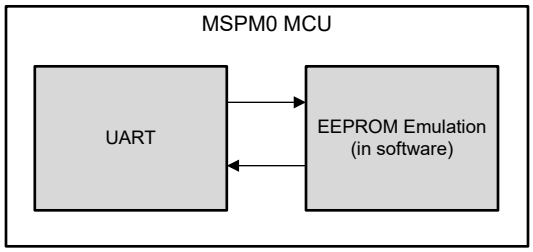

Figure 1-1. Subsystem Functional Block Diagram

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) includes 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 A, the following files are needed:
   a. eeprom_emulation_type_a.c
   b. eeprom_emulation_type_a.h

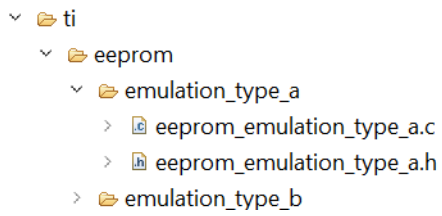

Figure 4-1. EEPROM Emulation Files

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

```
#include <ti/eeprom/emulation_type_a/eeprom_emulation_type_a.h>
```

   Users can modify the start address, the number of sectors to use, and the record size in eeprom_emulation_type_a.h. The default Flash address used for EEPROM emulation is 0x00001000, and 2 sectors are used by default, so 0x00001000-0x000017ff is occupied. Additionally, the default size of the emulated EEPROM is 128 bytes.

   a. #define EEPROM_EMULATION_ADDRESS (0x00001000)
   b. #define EEPROM_EMULATION_SECTOR_ACCOUNT (2)
   c. #define EEPROM_EMULATION_RECORD_SIZE (128)

3. Define a global array as a buffer in random-access memory (RAM). Every time the system powers on, the data of the emulated EEPROM is loaded from Flash to this buffer using the initialize function. The size of array ought to be equal to the record size in step 2.
   - `uint32_t EEPROMEmulationBuffer[EEPROM_EMULATION_DATA_SIZE / sizeof(uint32_t)];`

4. Add the *initialize* function at the beginning of `main()`, typically after `SYSCFG_DL_init()`. This action allows the relevant Flash areas to be formatted correctly and global variables to be allocated correctly. The initialize function `EEPROM_TypeA_init()` also searches the active record and loads the data from Flash to the buffer in step 3.
   - `EEPROM_TypeA_init(&EEPROMEmulationBuffer[0]);`

```
/* Initialize the virtual EEPROM */
EEPROMEmulationState = EEPROM_TypeA_init(&EEPROMEmulationBuffer[0]);
if (EEPROMEmulationState != EEPROM_EMULATION_INIT_OK) {
    //__BKPT(0);
}
```

Figure 4-2. EEPROM Initialization

5. Users can read or modify the buffer in RAM, as needed, after initialization. When the data from the buffer to Flash need to be stored, call `EEPROM_TypeA_writeData()` to create a new *record* in Flash.

Note

After power down, the data in RAM is lost. To implement data storage after power loss, execute this function at least once.

   - `EEPROM_TypeA_writeData(&EEPROMEmulationBuffer[0]);`

```
/* Store the EEPROMEmulationBuffer to the flash to be a new record */
EEPROMEmulationState =
    EEPROM_TypeA_writeData(&EEPROMEmulationBuffer[0]);
if (EEPROMEmulationState != EEPROM_EMULATION_WRITE_OK) {
    //__BKPT(0);
}
```

Figure 4-3. EEPROM Write

6. Add the erase function according to the `gEEPROMTypeAEraseFlag`. Flash needs to be erased before writing data again and the smallest unit of erasure is sector. For EEPROM emulation, after one sector is full, `gEEPROMTypeAEraseFlag` is set. Users can call `EEPROM_TypeA_eraseLastSector()` according to the flag. For example, add the following code after `EEPROM_TypeA_writeData()` from step 5. Users can also choose an appropriate timepoint to erase the full sector, as needed.
   - `EEPROM_TypeA_eraseLastSector();`

```
if (gEEPROMTypeAEraseFlag == 1) {
    /* In this demo, when the sector is full, it will be erased immediately */
    EEPROM_TypeA_eraseLastSector();
    gEEPROMTypeAEraseFlag = 0;
}
```

Figure 4-4. EEPROM Erase

After steps 1 through 6, the EEPROM emulation Type A 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_a.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 A Design* application note.
   a. Record size: 64, 128, or 256 bytes
   b. Number of sectors used: at least 2
   c. Sector address
2. To evaluate the Flash usage and cycling capability, see the *application aspects* section in the *EEPROM Emulation Type A Design* application note.
3. Data or header corruption is possible in case of a power loss during a `EEPROM_TypeA_writeData` or `EEPROM_TypeA_eraseLastSector`.

   To detect, and recover from corruption, implement `EEPROM_TypeA_init`. Call `EEPROM_TypeA_init`immediately after power-up. `EEPROM_TypeA_init` checks all the headers of the records to confirm whether data storage of EEPROM emulation is correct, and performs format-repair, if necessary.

   In the structure of EEPROM emulation, headers show the status of the corresponding records. 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 diagram for *EEPROM Emulation Type A* which introduces how to add functions in application code to implement EEPROM emulation. Three functions are required here: `EEPROM_TypeA_init`, `EEPROM_TypeA_writeData`, `EEPROM_TypeA_eraseLastSector`.

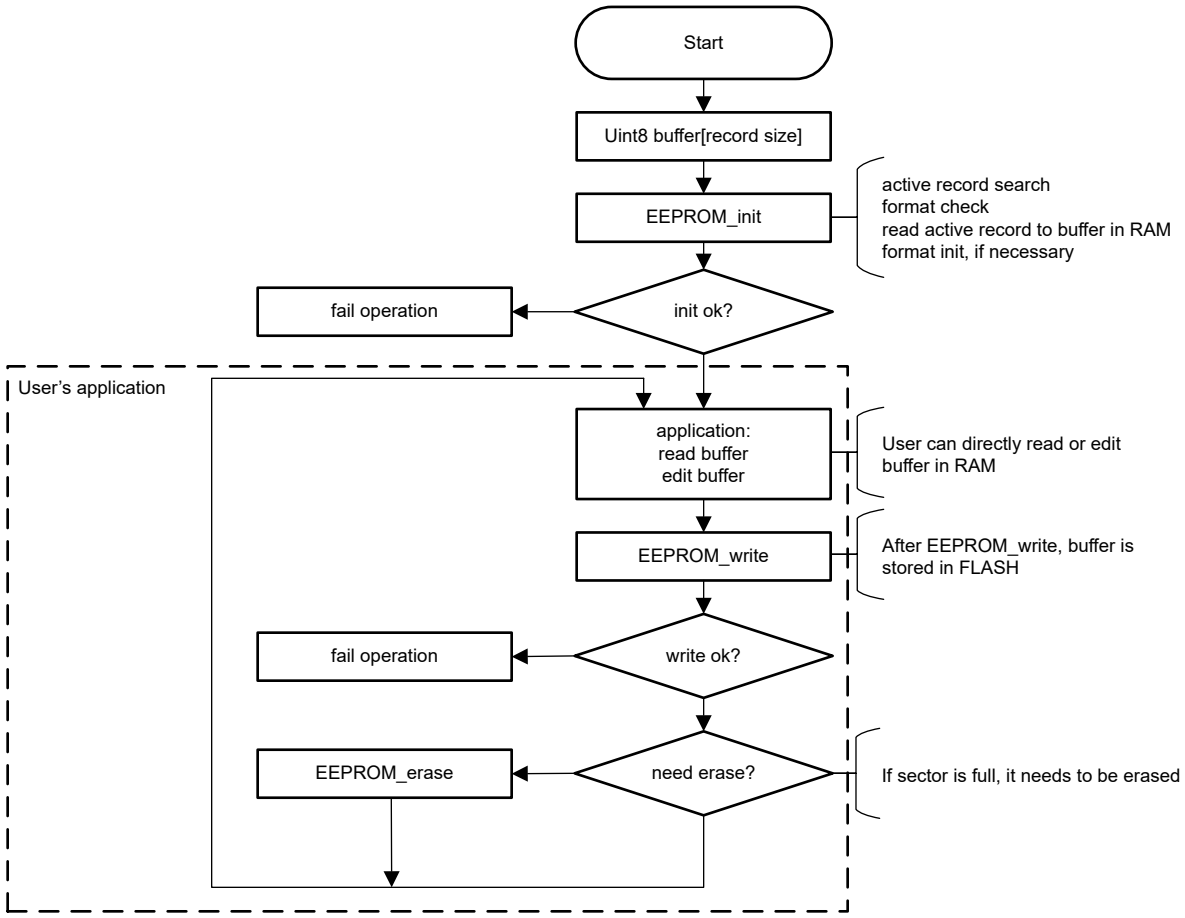

Figure 6-1. Application Software Flow Chart

7 Application Code

To implement this functionality, six functions are required. In addition to the three functions mentioned in Section 6, the remaining three functions are called primarily by `EEPROM_TypeA_init`.

- `EEPROM_TypeA_init`
- `EEPROM_TypeA_writeData`
- `EEPROM_TypeA_eraseLastSector`
- `EEPROM_TypeA_readData`
- `EEPROM_TypeA_searchCheck`
- `EEPROM_TypeA_repairFormat`

Additionally, seven global variables are used to record the status of the EEPROM emulation. Four global variables are used to trace the active record.

- `uint32_t gActiveRecordAddress`
- `uint32_t gNextRecordAddress;`
- `uint16_t gActiveRecordNum;`
- `uint16_t gActiveSectorNum;`

`gActiveRecordAddress` and `gNextRecordAddress` are used to store the address about active record.

`gActiveRecordNum` and `gActiveSectorNum` are used to trace the position of active record.

Three global variables are used for flags.

- bool `gEEPROMTypeASearchFlag;`
- bool `gEEPROMTypeAEraseFlag;`
- bool `gEEPROMTypeAFormatErrorFlag;`

`gEEPROMTypeASearchFlag` is set when the active record exists.

`gEEPROMTypeAEraseFlag` is set when the sector is full and needs to be erased.

`gEEPROMTypeAFormatErrorFlag` is set When format error is found.

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 B) 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