EEPROM Emulation with Wear-Leveling for 8-Bit Flash MCUs

1. Introduction

This application note demonstrates a way to use the flash memory of an 8-bit flash MCU to emulate single-variable-rewritable EEPROM memory through firmware. The example API provided enables reading and writing of single variables to non-volatile flash memory. The erase-rewrite algorithm implements wear-leveling on the flash by distributing page erases.

AN568: *EEPROM Emulation for Flash Microcontrollers* discusses an alternate method for EEPROM emulation. The differences between the method proposed in this document and AN568 are:

1. The firmware implementation included with AN568 requires an update to the sector area whenever a byte is updated. The method described in this document updates the data byte only, which helps with wear leveling.
2. The EEPROM emulation method in this document contains a logical-to-physical mapping, where firmware can access a logical address using the EEPROM interface functions. The EEPROM emulation uses a transition method to read/write flash physical addresses that match the logical addresses. This method has been widely used in NAND flash data management.
3. In the method described in this document, firmware uses byte units to access the EEPROM, just as with a real EEPROM.

2. Relevant Documentation

Silicon Labs 8-bit application notes and software are available on the website: www.siliconlabs.com/8bit-appnotes.

- **AN568: *EEPROM Emulation for Flash Microcontrollers*** — This document discusses an alternate method for EEPROM emulation.

3. General Theory

3.1. EEPROM and Flash-Based Memory

EEPROM stands for Electrically Erasable Programmable Read-Only Memory and is a type of nonvolatile memory that is byte erasable. Therefore, it is often used to store small amounts of data that must be saved when power is removed. The main difference between the flash memory used by the C8051Fxxx MCU families for non-volatile data storage and EEPROM is the erasable unit size. Flash memory is block-erasable, which means that bytes cannot be erased individually. Instead, a block consisting of several bytes (typically 512 or 1024 on the C8051Fxxx devices) must be erased at the same time. By using firmware, it is possible to emulate individually erasable and rewritable byte memory using block-erasable flash memory.

To provide EEPROM functionality with an 8-bit flash-based MCU in an application, there are many implementation options available. A hardware option is to include an external EEPROM module when designing the hardware layout of the application. Another is to use the on-chip flash memory and emulate EEPROM functionality through a firmware API. There are some key differences between these two methods:

1. The write access time for flash memory is shorter than for an external EEPROM. This means that writing to emulated EEPROM is faster than writing to an external EEPROM.
2. While a standalone EEPROM will be able to complete a write operation even if the system is reset, the emulated EEPROM will need the CPU to be active throughout the entire flash operation. The consequences of an aborted flash operation should be taken into account when designing an application. The flash-based EEPROM emulation could use checksums and logging to ensure the integrity of written data.

3. The emulated EEPROM will regularly need to erase pages in flash to free space and be able to write to the same page more than once. On a standalone EEPROM, there is no need for a dedicated erase operation, since all bytes can be erased and rewritten independently.
4. The firmware library emulating the EEPROM must also disable interrupts and enable the VDD monitor while performing the flash write/erase operations. With an external hardware EEPROM, interrupts can still occur during the EEPROM-writing process.

3.2. Flash Limitations

Flash memory is limited to a finite number of program-erase cycles. This means that the embedded flash memory of the MCU can be erased only a certain number of times before the wear will begin to affect the integrity of the storage. This deterioration applies to all kinds of flash memory. All 8-bit flash MCUs are guaranteed to withstand a number of erase cycles, which can be found in the data sheet for each device family.

All flash memory is divided into pages, and each page must be erased as a single unit. The amount of on-chip flash memory and the page size varies from one 8-bit MCU family to another. See the device data sheet for more information about the page size. Because the erase operation erases only whole pages, it is important to write as much data as possible to a single page of flash before erasing the page.

4. Implementation

There are different ways to implement an EEPROM emulator using flash memory. The idea behind the implementation discussed in this document is to allocate a certain number of pages of flash memory for the entire lifetime of the application. The wear is then leveled among these by alternating the pages used. The number of pages allocated must reflect the amount of data that will be written throughout the application lifetime.

4.1. Pages and Their States

In every page allocated to the EEPROM emulator, the first 4 bytes are reserved for page head data. This head data contains the status of the page and the erase count, which is the number of times the page has been erased. Each page will always be in one of three different states: **Active**, **Receiving**, or **Erased**.

- After a page is **Erased**, all bits in the entire page are 1's except the head data of the page.
- When a page is **Receiving**, a transfer of variables from a full active page is in progress. After the transfer is complete, the **Receiving** page becomes the new **Active** page, and the firmware erases the old active page.
- The **Active** page is the page that contains the currently valid data. All read and write operations occur on the **Active** page. There should never be more than one **Active** or **Receiving** page at any time in this implementation.

Figure 1 shows the state flow for an implementation using two pages. The flow would be similar if more pages are allocated, with more pages simultaneously in the **Erased** state.

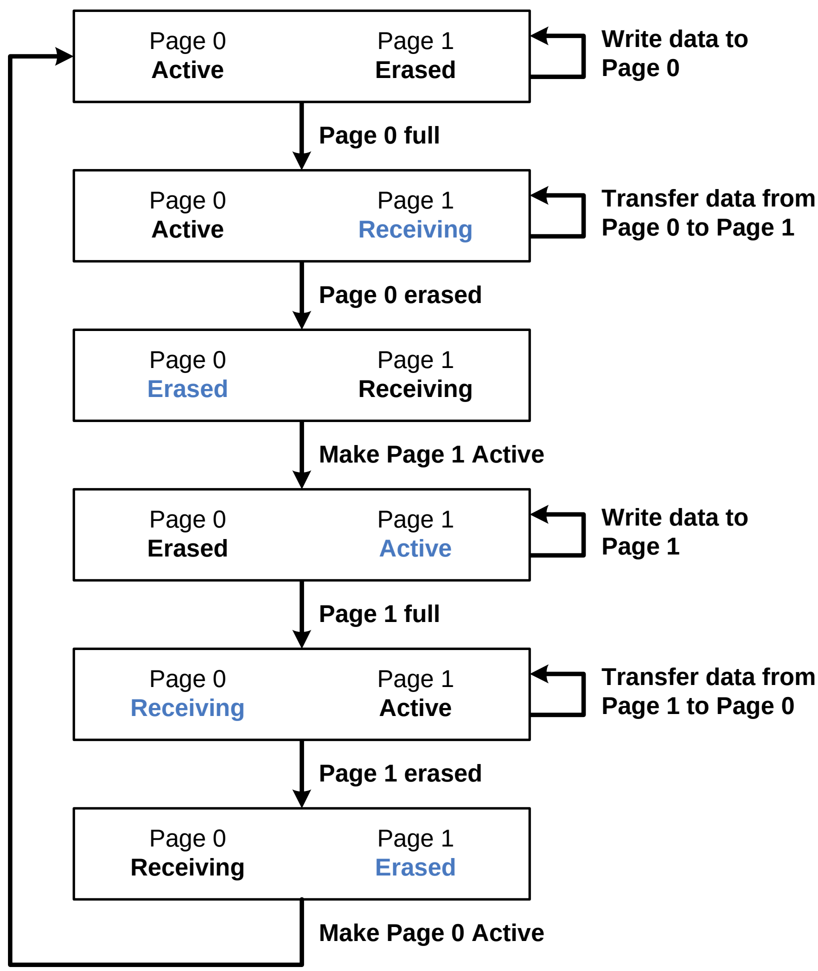


Figure 1. EEPROM Emulation Page Status Flow

During initialization, the firmware checks the page status for the selected number of pages to ensure that a legal set of page-states are present. If there is more than one **Active** page, any pages that are full will be erased. Any **Receiving** status pages will also be erased. This minimizes the probability for data collisions with application program instructions, which are located at the base of the flash. In applications where much of the available flash memory is in use, it can be critical to know where all data is stored in the flash to avoid collisions.

The remaining words of each page after the page head data are free to be used for data storage. Each data storage word is divided into two parts: an 8-bit virtual address field and an 8-bit data field. The emulation firmware initiates a page transfer whenever a page is full. This transfer operation consists of several steps to always ensure that all variables are saved in case of an external interruption event.

1. First, an **Erased** page is located to store the valid data present in the full page. The new page is marked as **Receiving**.
2. Next, the most recent data associated with each variable is transferred to the top of the new page.
3. The old **Active** page is erased, the erase count is written to the old active page header, and the **Receiving** page is labeled as the new **Active** page.

This process is illustrated in Figure 2.

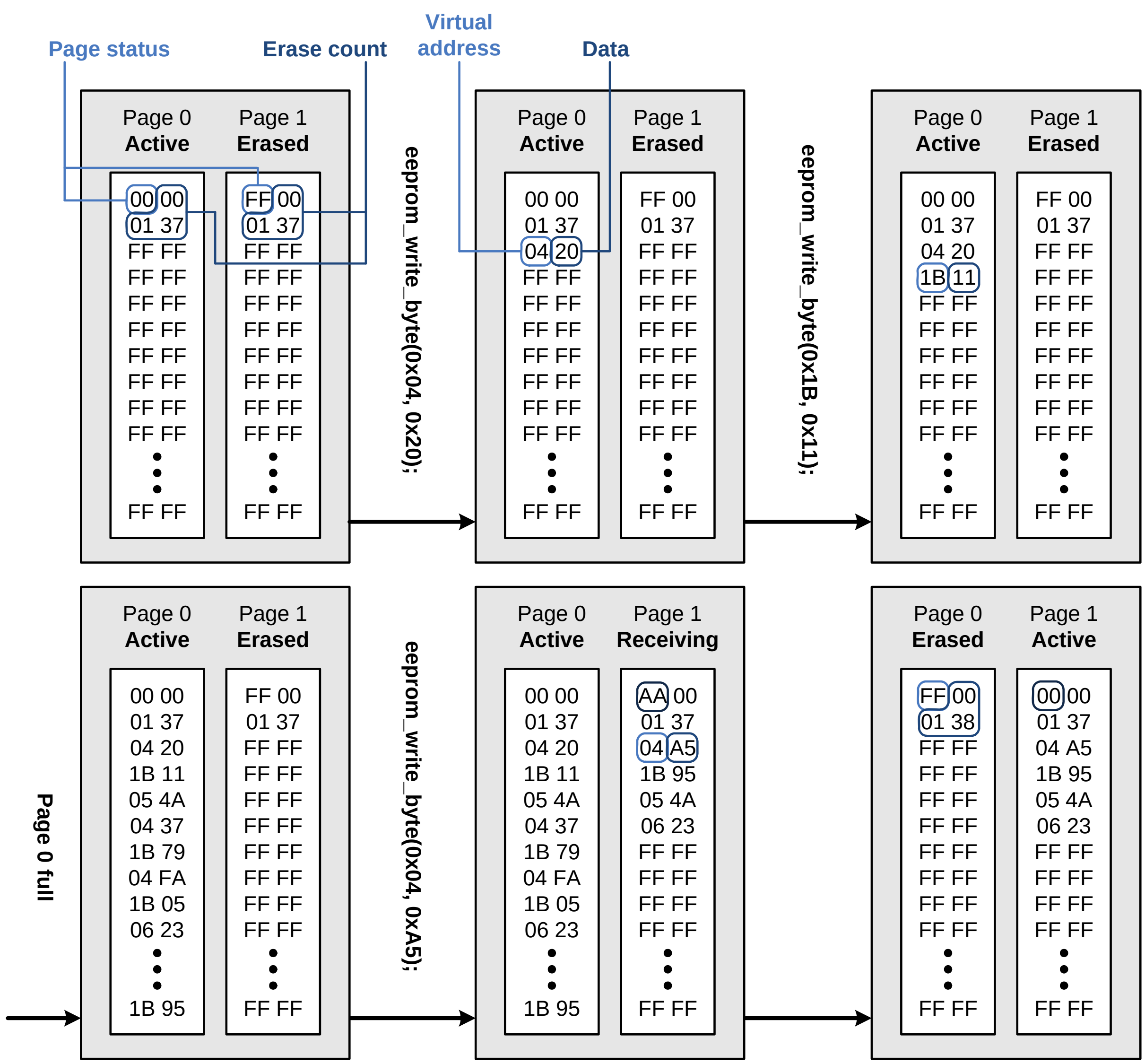


Figure 2. EEPROM Emulation Variable Flow

4.2. Reading and Writing

Reading consists of iterating through the **Active** page starting from the end of the page. When the firmware finds the correct virtual address for the first time, the corresponding data is returned as the currently valid data.

The write operation consists of a similar iteration process starting at the first address of the **Active** page. When an empty word is found, the firmware writes the correct virtual address and data, and returns. If no empty word is found, and the end of the page is reached, the page is considered full.

In this implementation, all valid data has to fit inside one **Active** page. Therefore, the physical flash page size puts a direct limitation on the emulated EEPROM size. Each data slot uses 2 bytes (1 byte for data and 1 byte for the virtual address), and 4 bytes on the page are reserved for the page status header. By default, the EEPROM size is no more than 1/4 of the flash page size, also considering 8-bit alignment. For example, with a 512-byte flash page size, the maximum EEPROM size is:

$$N_{MAX} = \frac{512 - 4}{4} \,\&\, 0xF8 = 127 \,\&\, 0xF8 = 120$$

4.3. Initialization and Recovery

An important part of EEPROM emulation firmware is to ensure correct page states and enable data recovery on system startup. For this reason, the initialization function should be called near the start of the application firmware, before any data is written to or read from the emulated EEPROM. The initializing function decides how many pages will be allocated to the emulator. The firmware will then check the head of each of these pages to ensure that the set of pages is valid. There are several conditions that should be handled:

- **Erased page —** If an **Erased** page is not formatted, the firmware will simply format this page. The format operation erases the page and updates the erase count in the page head data.
- **Pages with Receiving status** — The firmware formats this page. This status can be caused by an incomplete page transfer.
- **Two or more pages with Active status** — The firmware compares with two pages and formats any pages that are full.

Once the status check completes, the firmware will scan the current **Active** page and update the page head information.

4.4. Configurable Options

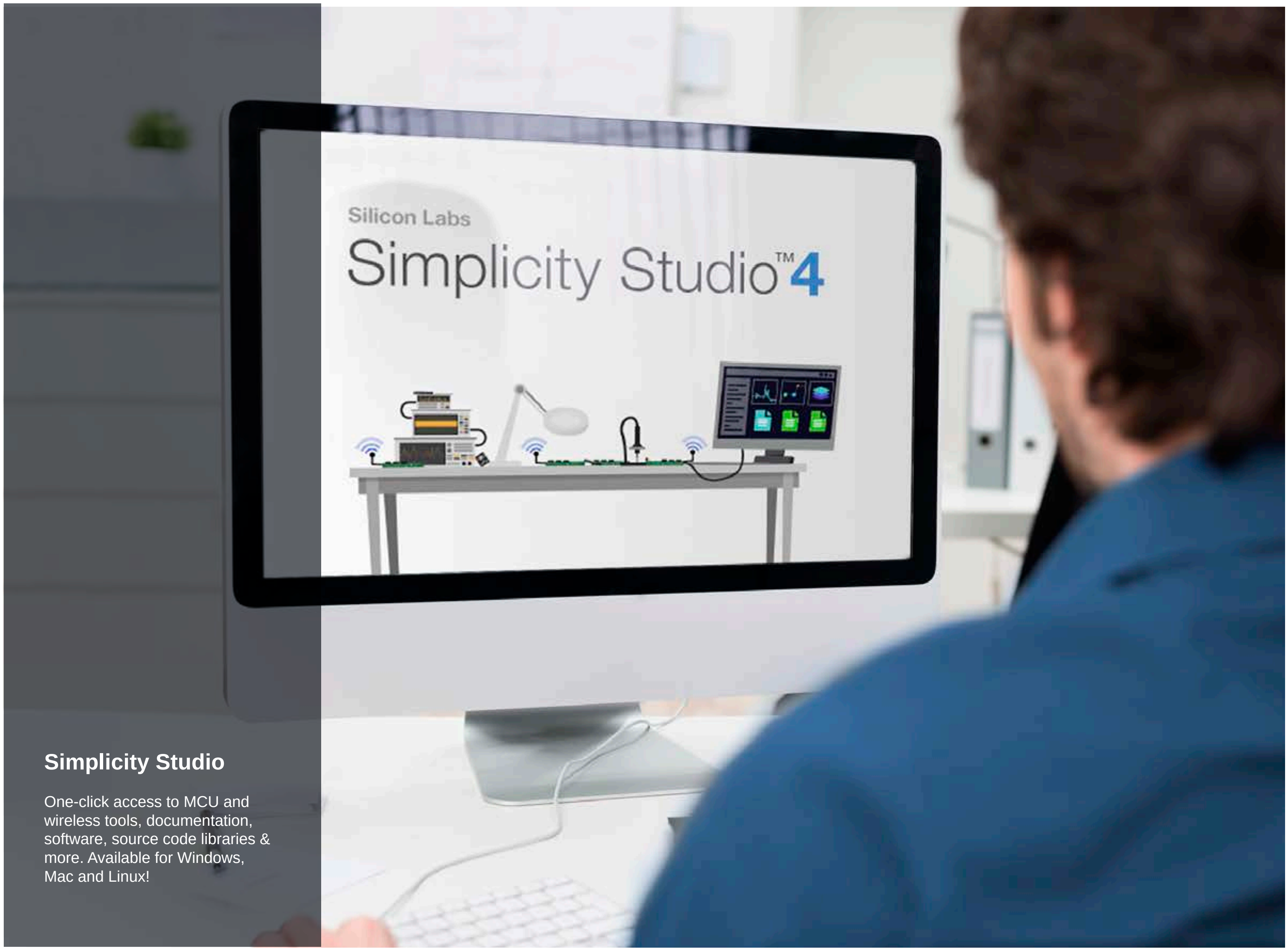
The firmware supports all Silicon Labs 8-bit flash MCU families. Define the device being used in the **flash_parameters.h** file. For example, for a C8051F850 device:

```
#define C8051F850
```

The EEPROM emulation firmware has several configurable parameters listed in Table 1. These parameters are located in the **eeprom_config.h** file.

Table 1. Script File Structure

Parameter	Description
FL_PAGES	Pages used for EEPROM emulation: **2**
EE_BASE_ADDR	EEPROM storage area begins: **LOCK_PAGE - FLPAGE_SIZE * FL_PAGES**
EE_SIZE	EEPROM size in bytes: **16**
EE_TAG_SIZE	Number of bytes for the page head data: **4**
EE_VARIABLE_SIZE	Number of bytes used per EEPROM data slot: **2**



IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com