Using Data Flash to Emulate EEPROM

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	This application note describes how to use Data Flash to emulate EEPROM and increase the reading speed by SRAM buffer. This document includes the technology introduction, performance data analysis and use recommendation. Sample code and libraries are provided in Chapter 3 and Chapter 4.
Apply to	NuMicro® Cortex®-M0/M4 series (NuMicro® M051 DE series is used as an example).

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Overview

The EEPROM supports byte read and byte write function and up to 1 million times of reliable write/erase cycles. User usually stores non-volatile data in EEPROM. In general, a microcontroller (MCU) is not equipped with EEPROM but Flash for storing data. Hence, it recommended to use the software solution to emulate the EEPROM by Data Flash for implementing up to 1 million times of reliable write/erase cycles.

This application note issues a new mechanism to use Data Flash to emulate EEPROM by combining more than 2 pages of Data Flash. This mechanism can increase the reading speed by using the SRAM buffer.The endurance of write/erase emulated EEPROM is up to 1 million times of reliable write/erase cycles. The number of written/erased cycles is recorded and the data is divided into several smaller data sets to reduce the number of Data Flash pages.

- Increasing the reading speed by SRAM buffer

  During data write operation, the data will be written into Data Flash and SRAM data bank synchronously. Once the Data Flash page selected is full, it will switch to the next Data Flash page. Then the emulated EEPROM will store the current SRAM data bank into new Data Flash page. In this way, user only needs the time that moving the SRAM data bank into new Data Flash page instead of sorting all of the data in old Data Flash pages and move it into the new page.

  User can read the data from SRAM data bank instead of reading it from Data Flash for reducing the sorting time. The data inside the Data Flash is only for initializing the data bank in SRAM.

- Up to 1 million times of write/erase cycles

  Through this mechanism, the emulated EEPROM can do byte read/byte write and be erased up to 1 million times.

- Recording the number of written/erased cycles

  User can get the written/erased cycles of Data Flash pages by reading the counter value.

- Reducing the number of Data Flash pages by dividing data into several smaller data sets

  For the NuMicro® Cortex®-M0 series, as shown in Figure 1-1, if the data that user wants to store is increased, to achieve the target endurance of write/erase cycles, the number of Data Flash pages will be increased, too. Hence, it is recommended to divide the target data into several smaller data sets and use less Data Flash to achieve the target endurance of write/erase cycles.

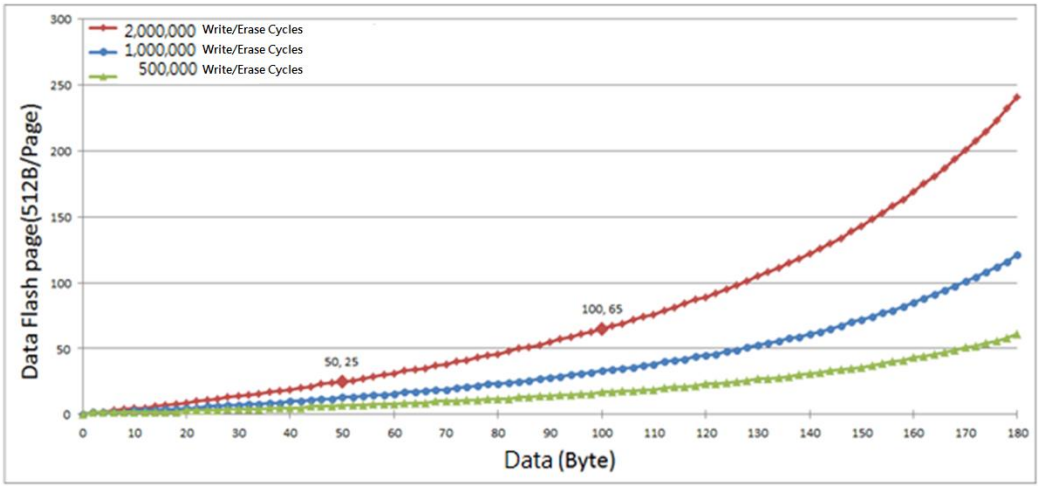

Figure 1-1 Data vs. Data Flash Page - NuMicro® Cortex®-M0 Series

Data (Byte)	Data Flash Page (512B/Page)	Data (Byte)	Data Flash Page (512B/Page)
32	15	50	25
34	16	52	26
36	17	54	27
38	18	56	29
40	19	58	30
42	20	60	31
44	21	62	33
46	23	64	34
48	24		

Table 1-1 Common Use for Data and Data Flash Page- NuMicro® Cortex®-M0 Series

For the NuMicro® Cortex®-M4 series, as shown in Figure 1-2, if the data that user wants to store is increased, to achieve the target endurance of write/erase cycles, the number of Data Flash pages is increased, too. Because the number of Data Flash page is not increased significantly, it is recommended that do not divide the target data into several smaller data sets.

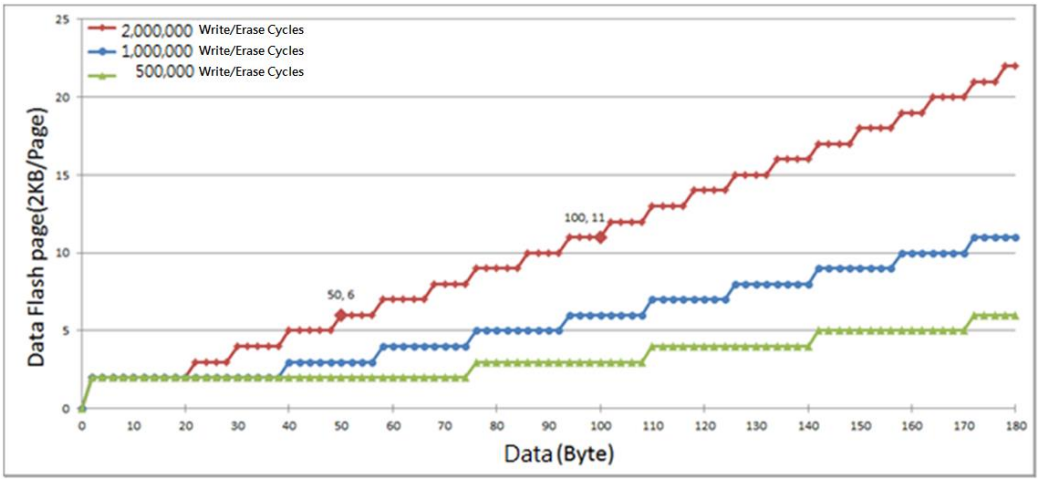

Figure 1-2 Data vs. Data Flash Page - NuMicro® Cortex®-M4 Series

Data (Byte)	Data Flash Page (2 KB/page)	Data (Byte)	Data Flash Page (2 KB/Page)
32	4	50	6
34	4	52	6
36	4	54	6
38	4	56	6
40	5	58	7
42	5	60	7
44	5	62	7
46	5	64	7
48	5		

Table 1-2 Common Use for Data and Data Flash Page - NuMicro® Cortex®-M4 Series

2 Emulated EEPROM

This chapter illustrates how to use Data Flash to emulate EEPROM and increase the reading speed by SRAM.

2.1 Principle of Emulating EEPROM by Data Flash

When using Data Flash to emulate EEPROM, it needs at least 2 pages. Each Data Flash page is divided into several blocks by two bytes. As shown in Figure 2-1, the page size of Data Flash in the NuMicro® Cortex®-M0 series is 512 bytes. The first block records the number of write/erase cycles, and others record the data address and the data value. Besides, there is a data bank in SRAM for storing the recorded data.

When user wants to store data, user needs to write an address and a value into the first page of Data Flash and the corresponding address in SRAM data bank. If the first page is full, the page of Data Flash will switch to the new page and copy the SRAM data bank into the new page (0xFF is invalid). Then, the old Data Flash page will be erased.

Because the time of reading data from Data Flash is longer than from SRAM data bank, using SRAM buffer to store data can reduce the data access time. Once Data Flash page selected currently is full, the Data Flash page will switch to the new page, and the new Data Flash page will copy the SRAM data bank instead of sorting bank data of old page and move it into the new page. In this way, copying SRAM data bank can reduce more time than sorting bank data on old Data Flash page when switching the page.

Besides, there is a counter for recording the number of write/erased cycles, according to this counter value, user can calculate the remaining endurance of write/erase cycles and estimate the product life time.

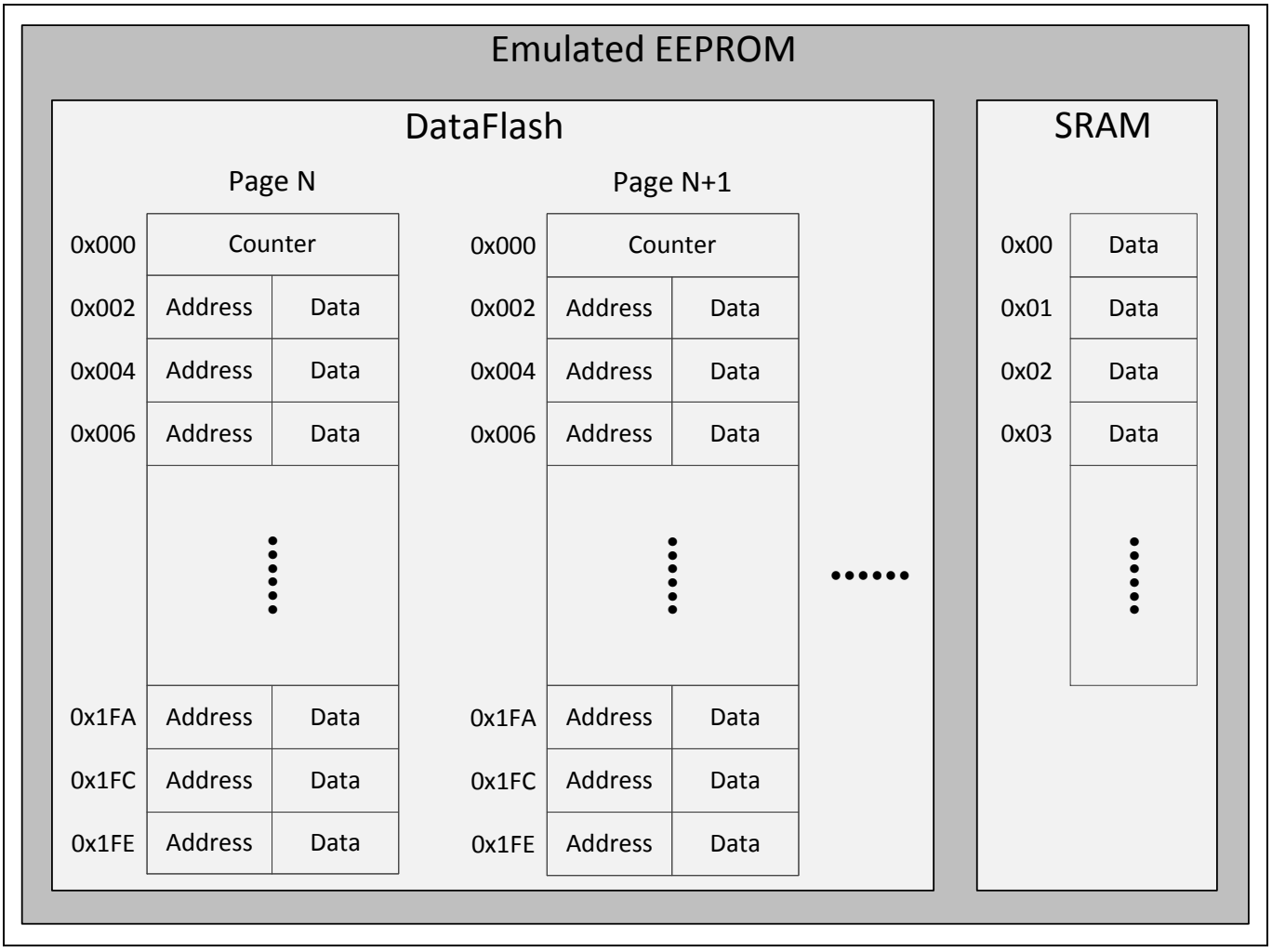

Figure 2-1 Emulated EEPROM

2.2 Initialize Data Flash

Before executing the main program, firstly, user needs to initialize the configured Data Flash. Because only the lastly selected Data Flash page has the valid counter value (not 0xFFFF), user can find the lastly selected Data Flash page by sorting the valid counter value. Secondly, the pointer will point to the next writable address. Finally, copy the content of the lastly selected Data Flash page into SRAM to be the data bank.

For the detailed flow, please refer to Figure 2-2, function Init_EEPROM() and function Search_Valid_Page().

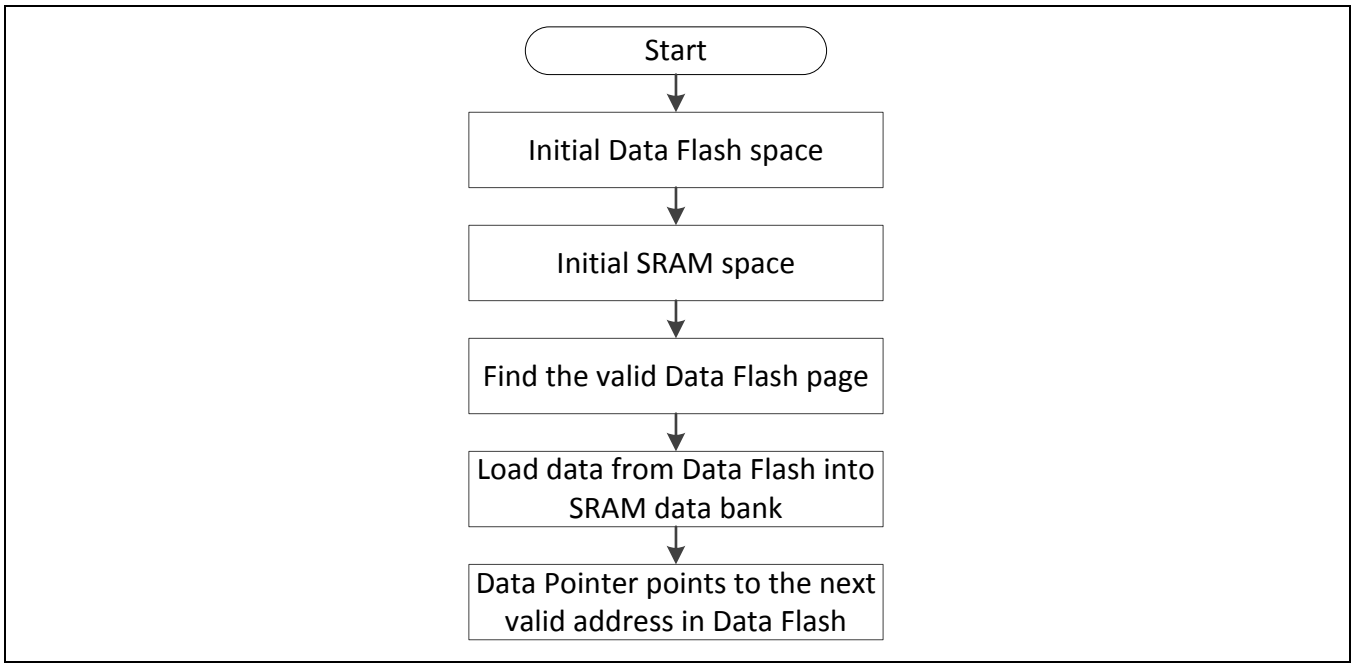

Figure 2-2 Data Flash Initialization Flow

2.3 Write Data

When writing data into the emulated EEPROM on Data Flash, at first, the emulated EEPROM will sort the same address in SRAM data bank and compare the corresponding data in SRAM data bank. If both data are the same, it is not necessary to write it again. If both data are different, the data will be updated.

When doing updates, new data will be written into Data Flash and SRAM data bank at the same time, and then emulated EEPROM will check if the selected Data Flash page is full or not. If it is not full, pointer will point to the next address of page. If it is full, emulated EEPROM will copy the valid data of SRAM data bank (0xFF is invalid) into the new configured Data Flash page.

Compared with the way above, sorting Data Flash and writing to the new page, copying the data of SRAM data bank into new configured Data Flash page can substantially reduce the time of switching to the new page.

If the Data Flash page is the last configured Data Flash page, when switching it to the next page, the counter will add 1. Then emulated EEPROM will copy the SRAM data bank into the first configured Data Flash page.

User can read the counter value to check the used write/erase cycles.

After writing data into the new page, the old full page will be erased.

For the detailed flow, please refer to Figure 2-3, function Write_Data() and function Manage_Next_Page().

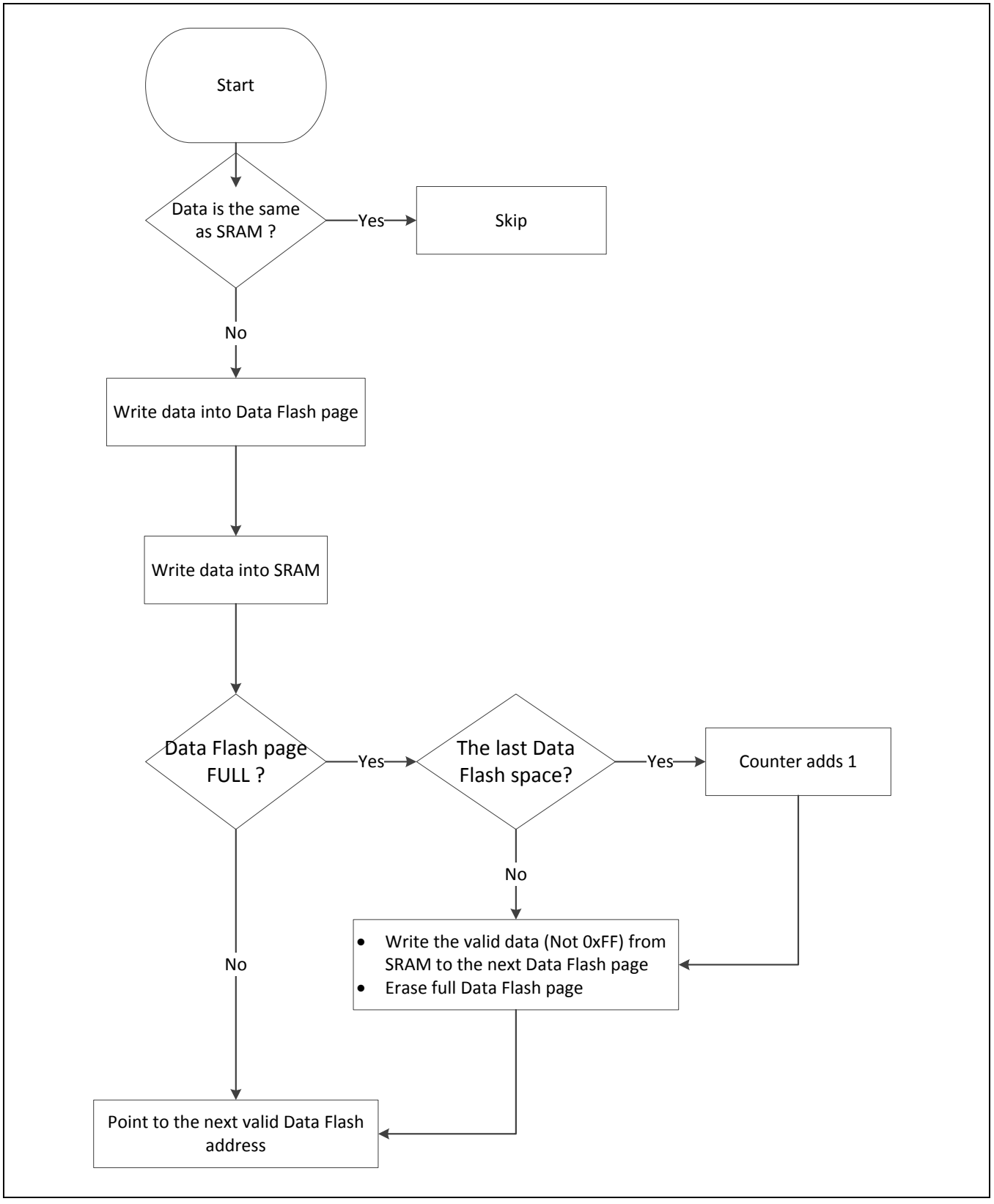

Figure 2-3 Write Data Flow

2.4 Read Data

Because there is the data bank in SRAM, if user wants to read data from the emulated EEPROM, the user only needs to read the data bank from SRAM. In this way, it can reduce much time because the speed of reading data from SRAM is fast than reading from Data Flash.

For the detailed flow, please refer to Figure 2-4, function Write_Data() and function Manage_Next_Page().

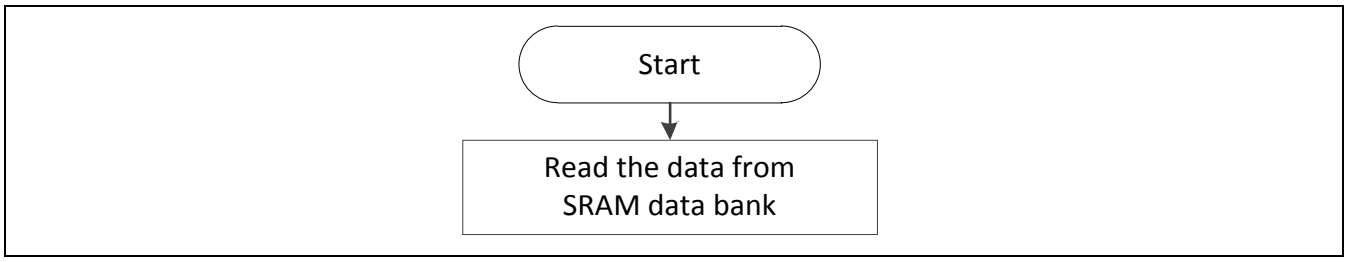

Figure 2-4 Read Data Flow

2.5 Read Counter Value

User can get the written/erased cycles of Data Flash pages through reading the counter value and estimate the remaining write/erase cycles. Please refer to Figure 2-5 and function Get_Cycle_Counter()**.**

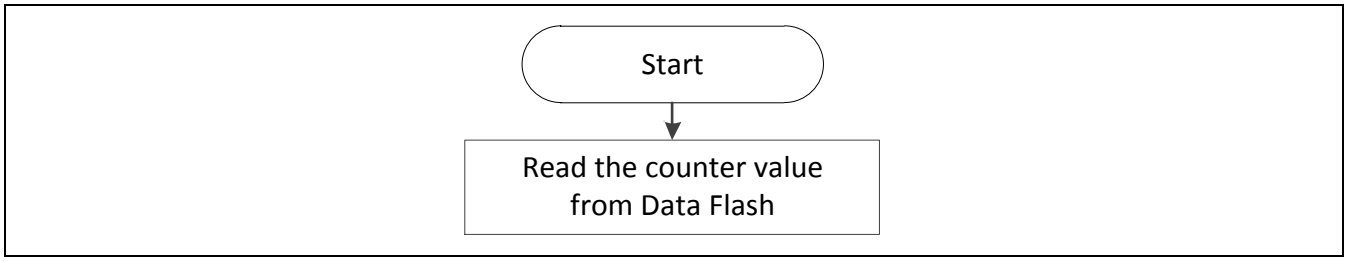

Figure 2-5 Read Counter Value Flow

2.6 Read/Write Data Example

Before main loop processing, Data Flash needs to be initialized. As shown in Figure 2-6, 2 pages of Data Flash are used to emulate EEPROM. In the initial step, the emulated EEPROM will search the valid data (not 0xFF) in Data Flash page and puts it into SRAM. The data size is 8 bytes in this example. The pointer will point to the next address that is valid to write data.

To write the data 0x07[0x68] (Addr [Data]), as shown in Figure 2-7, the emulated EEPROM will check if the data is the same as SRAM data. Because the data, 0x68, is not the same as SRAM data, the emulated EEPROM will write the data address and data value into Data Flash page and write the data into the corresponding address, 0x07 in SRAM. Then, the emulated EEPROM will check if the Data Flash page is full. Because the Data Flash page is not full, the emulated EEPROM just needs to move the pointer to the next address.

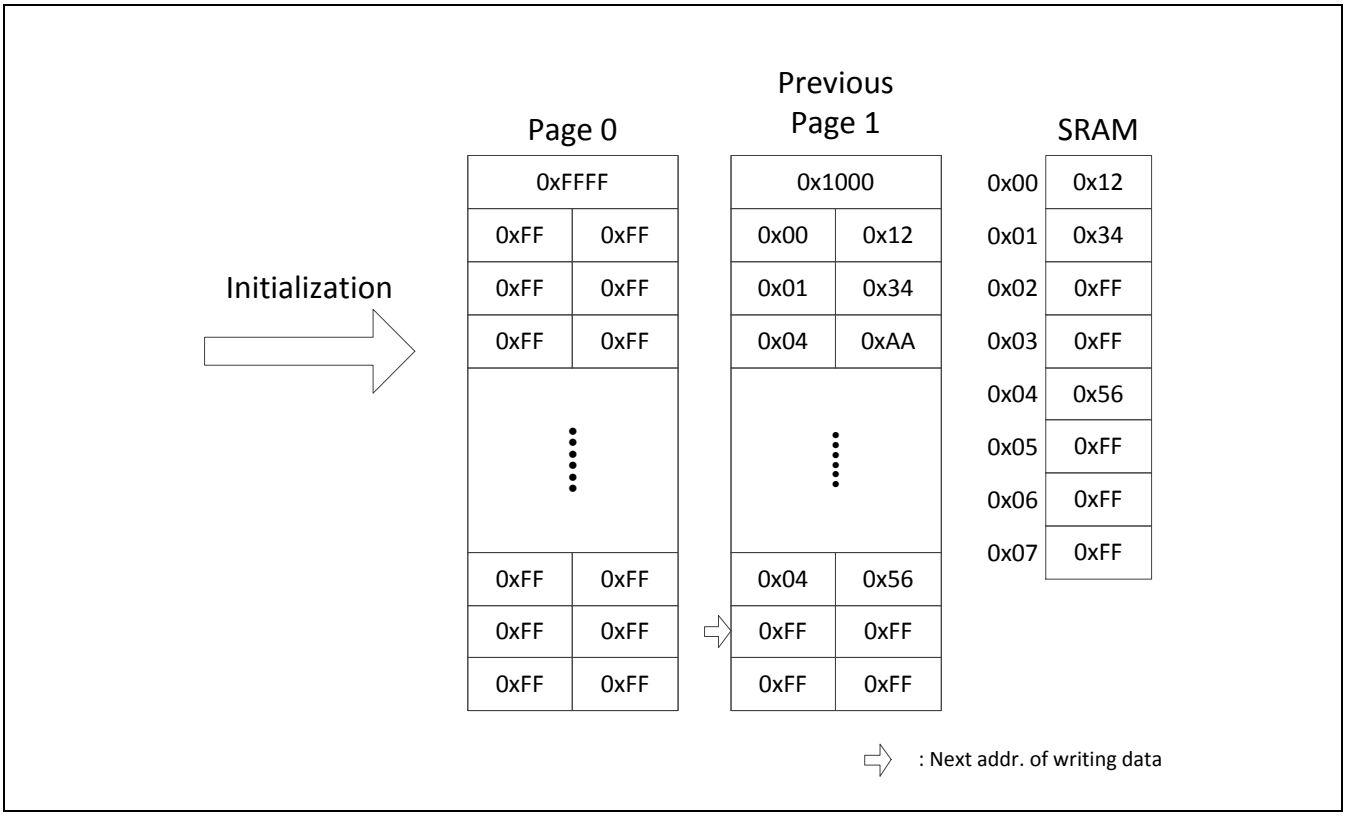

Figure 2-6 Example - Emulated EEPROM Initialization

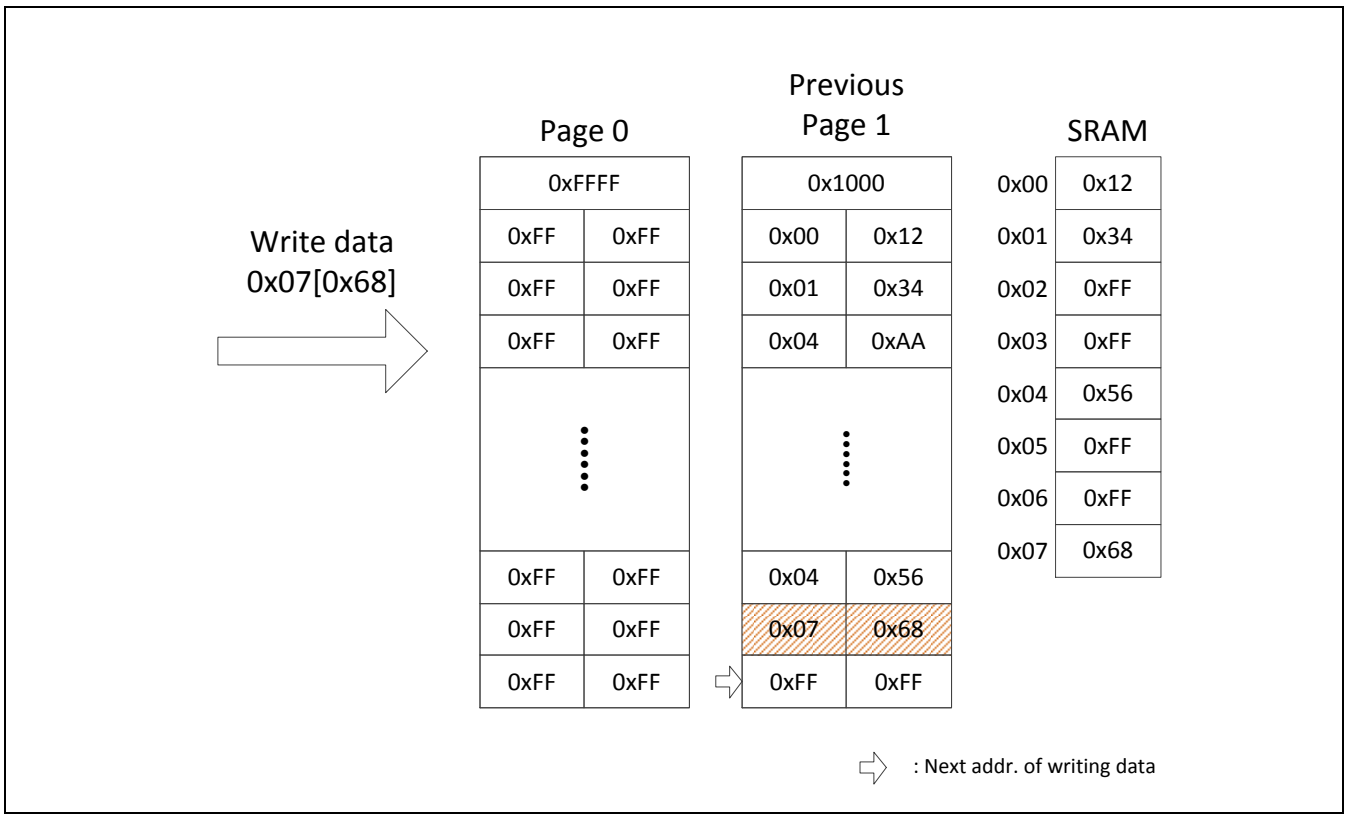

Figure 2-7 Example - Write Data into Emulated EEPROM

If the user wants to write data 0x04[0x56], as shown in Figure 2-8, because the data 0x56 is the same as SRAM data in address 0x04, the write process will be ignored.

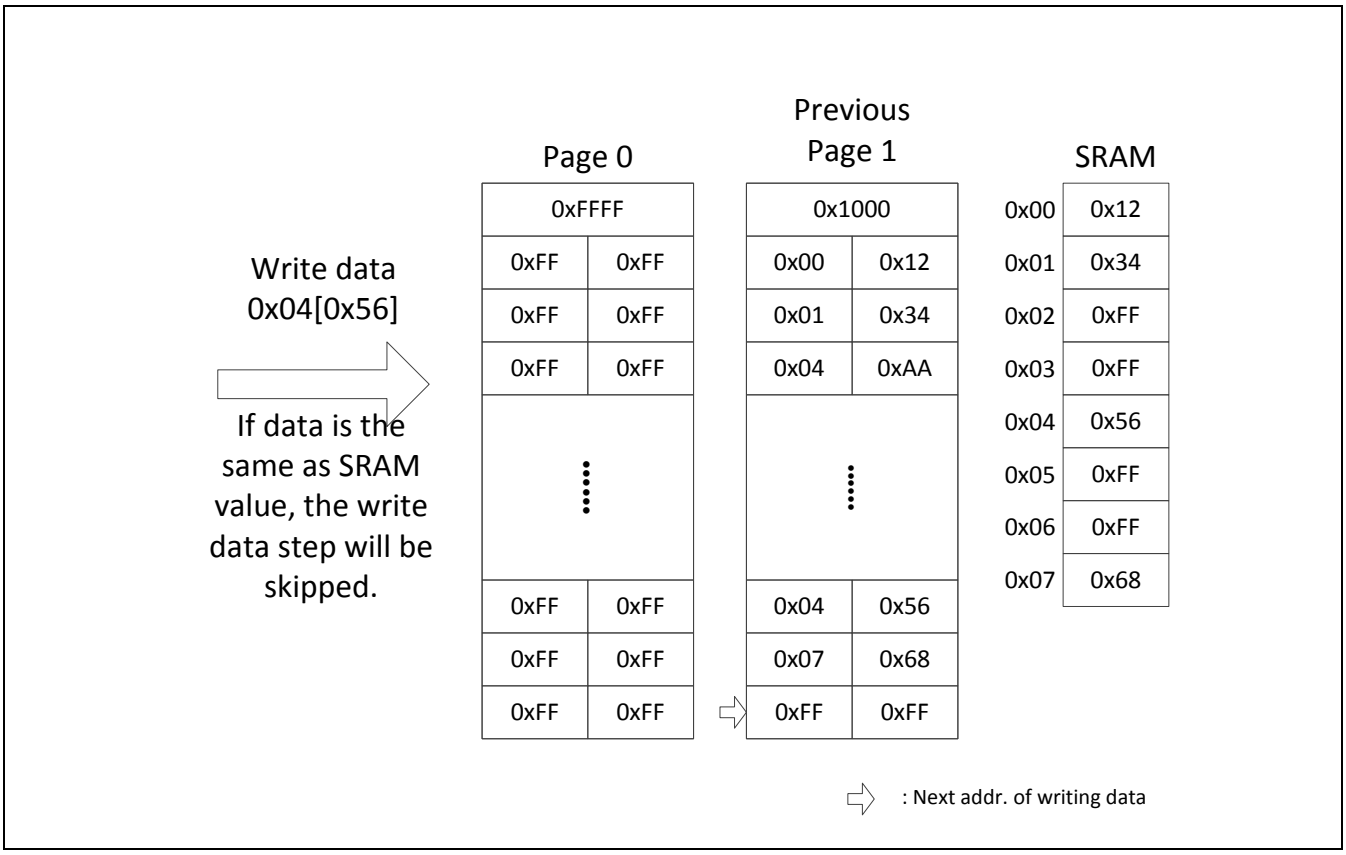

Figure 2-8 Example - Write the Identical Data into Emulated EEPROM

To write the data to the address 0x00[0xFF], as shown in Figure 2-9, whether the data is the same as the SRAM data will be determined. If the data is not the same as SRAM, the emulated EEPROM will write it into Data Flash and corresponding address 0x00 of SRAM.

Once the Data Flash page is full after the data is written, the emulated EEPROM will duplicate the SRAM data into new Data Flash page and add 1 to page counter value. Then, the emulated EEPROM will move the valid address and data (not 0xFF) to the new Data Flash page. In this example, address 0x01, 0x04, 0x06, 0x07 and its corresponding data will be duplicated to new Data Flash page. After the data is duplicated, the old Data Flash page will be erased and the pointer will point to the next address that can write data.

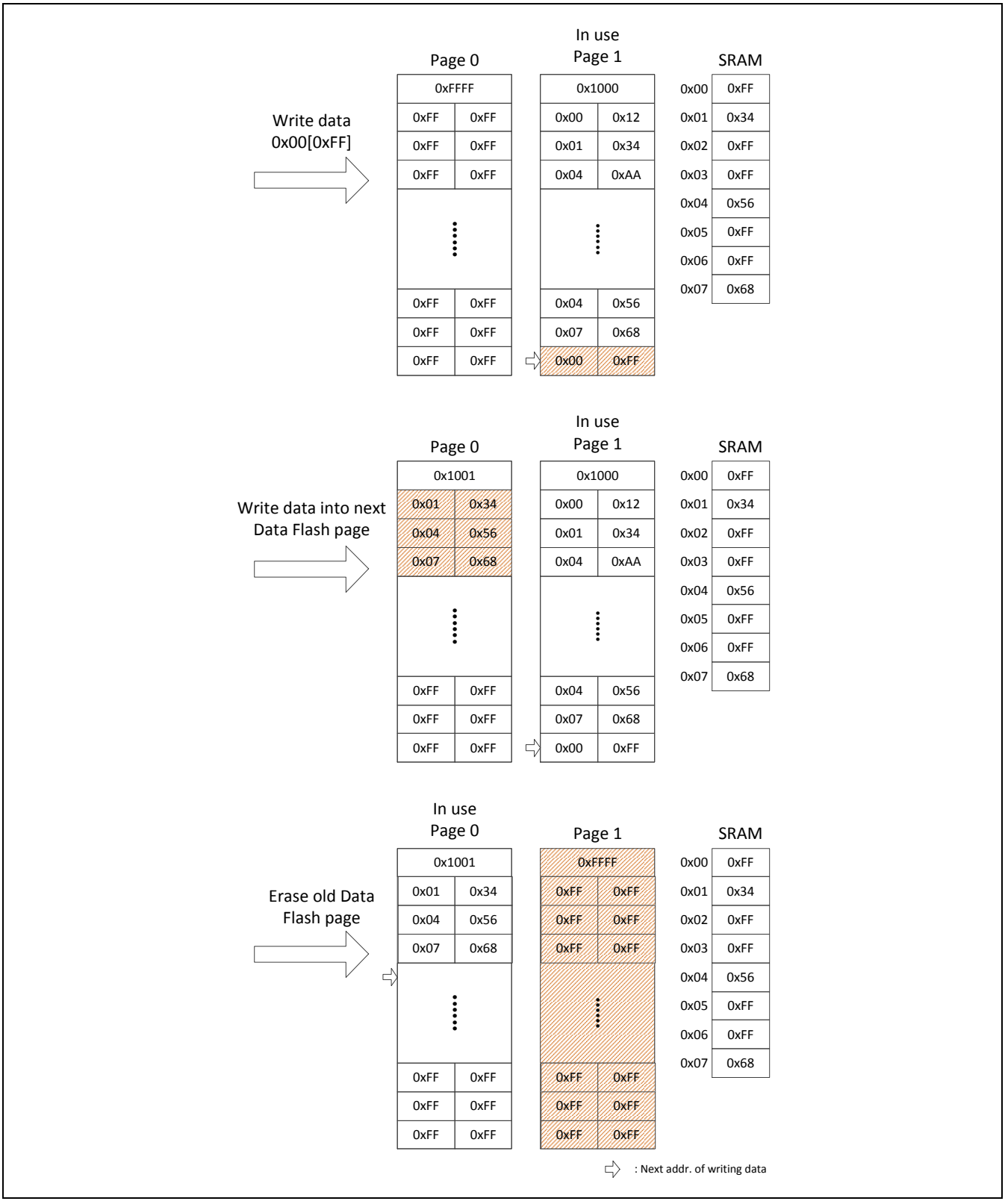

Figure 2-9 Example - Write New Data and Switch to the Next Data Flash Page

To read the data, the user only has to read the SRAM data instead of reading the data from the used Data Flash page.

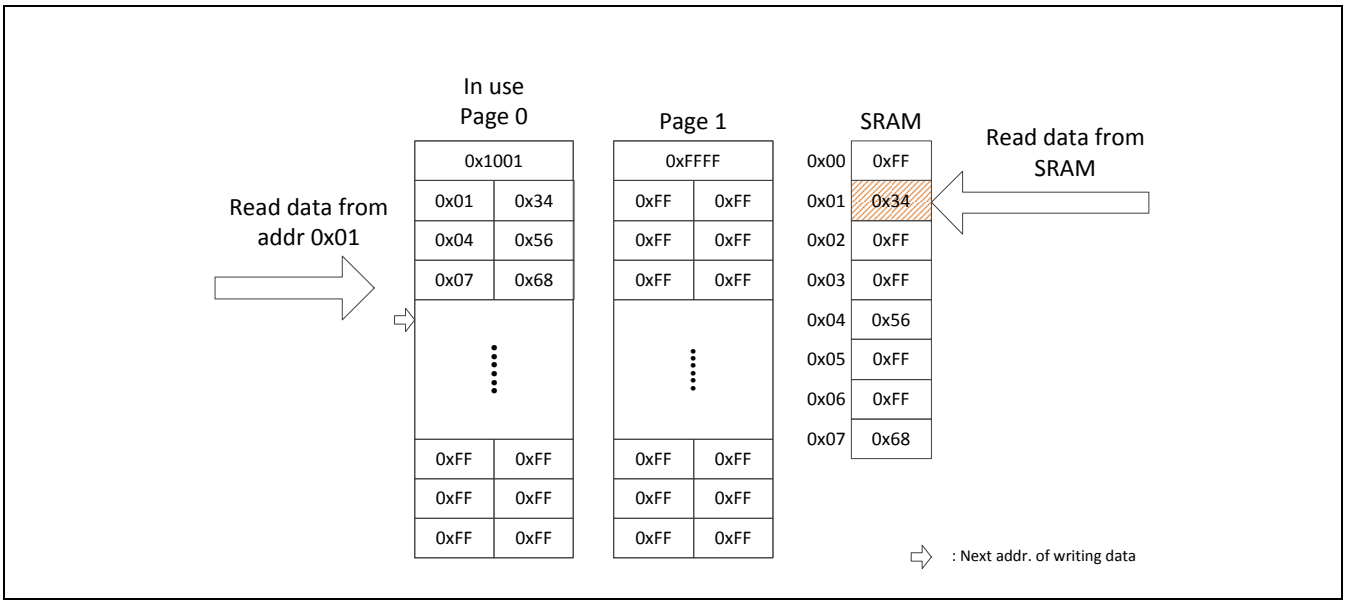

Figure 2-10 Example - Read Data from SRAM Instead of Data Flash

2.7 Reliability Calculation

Evaluating the reliability and endurance of emulated EEPROM is based on average reliable write/erase cycles of each block (2 bytes including data and address). The average reliable write/erase cycles of each data needs more than 1 million cycles and the following is the formula.

$$\text{Avg. reliable write/erase cycles of each data} = \frac{(S - C - D)}{D} \times P \times E$$

S: Size of Data Flash page. The NuMicro® Cortex®-M0 series is 512 bytes. The NuMicro® Cortex®-M4 series is 2048 bytes.

C: Size of counter. It is fixed to be 2 bytes.

D: Block size × number of data. Block size is fixed to be 2 bytes.

P: Number of Data Flash pages. User needs to estimate how much Data Flash page will be used to emulate the EEPROM.

E: Reliable write/erase cycles of Data Flash page.

Taking the NuMicro® M051 DE as an example, if 4 pages of Data Flash (2 Kbytes) are used to emulated EEPROM for storing 8 records of data, the average reliable write/erase cycles of each data is listed as follows.

$$\text{Avg. reliable write/erase cycles of each data} = \frac{(512 - 2 - 2 \times 8)}{2 \times 8} \times 4 \times 20{,}000$$

$$= 2{,}470{,}000 \text{ cycles}$$

2.8 Read/Write Speed

Table 2-1 shows the differential data corresponding to Data Flash pages. It only needs less than 1 us to read data from SRAM. For switching to a new Data Flash page, the time that puts the data into new Data Flash page from SRAM is less than 100 us.

HCLK = 50 MHz	**8 records of data needs 4 pages of Data Flash**	**20 records of data needs 4 pages of Data Flash**	**20 records of data needs 6 pages of Data Flash**
Init_EEPROM()	8 us	13.28 us	13.28 us
Search_Valid_Page()	61.2 us	61.2 us	65.6 us
Write_Data()	52 us	62.8 us	62.8 us
Write_Data() and Manage_Next_Page()	20400 us	20753 us	20753 us
Read_Data()	0.88 us	0.88 us	0.88 us

Table 2-1 Read/Write Speed Table

2.9 Reduce the Number of Data Flash Pages

2.9.1 NuMicro® Cortex®-M0 Series

For the NuMicro® Cortex®- M0 series, as shown in Figure 1-1, if the data that user wants to store is increased, to achieve the target write/erase cycles, the number of Data Flash page will be increased. For example, if there is 100 records of data that needs to be stored, to achieve 2 million times of write/erase cycles, the user needs to use 65 pages of Data Flash (32,500 Bytes) for emulating EEPROM. If there is 50 records of data, the user needs 50 pages of Data Flash (25,000 Bytes) for emulating EEPROM. Please refer to Table 2-2.

Data Size	Data Flash Page
50	25
100	65
50 + 50	50

Table 2-2 Size vs. Page Comparison Table - NuMicro® Cortex®-M0 Series

Hence, it is suggested that user can divide the data into smaller sets because the smaller Data Flash page can achieve the target write/erase cycles.

2.9.2 NuMicro® Cortex®-M4 Series

For the NuMicro® Cortex®-M4 series, as shown in Figure 1-2, if the data that user wants to store is increased, to achieve the target write/erase cycles, the number of Data Flash page will not be increased. For example, if there is 100 records of data that needs to be stored, to achieve 2 million times of write/erase cycles, the user needs to use 11 pages of Data Flash (22,000 Bytes) for emulating EEPROM. If there is 50 records of data, the user needs 6 pages of Data Flash (12,000 Bytes) for emulating EEPROM. Please refer to Table 2-3.

Data Size	Data Flash Page
50	6
100	11
50 + 50	12

Table 2-3 Size vs. Page Comparison Table - NuMicro® Cortex®-M4 Series

Hence, the user does not need to divide the data into smaller sets.

3 Library

3.1 Init_EEPROM()

```
/**
* @brief     Initial Data Flash as EEPROM.
* @param[in]  data_size: The amount of user's data, unit in byte. The maximun amount is
                         128.
* @param[in] use_pages: The amount of page which user want to use.
* @retval    Err_OverPageSize: The amount of user's data is over than the maximun amount.
* @retval    Err_OverPageAmount: The amount of page which user want to use is over than
                                 the maximun amount.
* @retval    0: Success
*/
uint32_t Init_EEPROM(uint32_t data_amount, uint32_t use_pages)
{
    uint32_t i;

    /* The amount of data includes 1 byte address and 1 byte data */
    Amount_of_Data = data_amount;
    /* The amount of page which user want to use */
    Amount_Pages = use_pages;

    /* Check setting is valid or not */
    /* The amount of user's data is more than the maximum amount or not */
    if(Amount_of_Data > Max_Amount_of_Data)
        return   Err_OverAmountData;
    /* For M051 Series, the max. amount of Data Flash pages is 8 */
    if(Amount_Pages > 8)
        return   Err_OverPageAmount;

    /* Init SRAM for data */
    Written_Data = (uint8_t *)malloc(sizeof(uint8_t) * Amount_of_Data);
    /* Fill initial data 0xFF*/
    for(i = 0; i < Amount_of_Data; i++)
    {
        Written_Data[i] = 0xFF;
    }
    return 0;
}
```

3.2 Search_Valid_Page ()

```
/**
* @brief     Search which page has valid data and where is current cursor for the next
             data to write.
*/
void Search_Valid_Page(void)
{
    uint32_t i, temp;
    uint8_t  addr, data;
    uint16_t *Page_Status_ptr;

    /* Enable FMC ISP function */
    FMC_Enable();

    /* Set information of each pages to Page_Status */
    Page_Status_ptr = (uint16_t *)malloc(sizeof(uint16_t) * Amount_Pages);
    for(i = 0; i < Amount_Pages; i++)
    {
        Page_Status_ptr[i] = (uint16_t)FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE
* i));
    }

    /* Search which page has valid data */
    for(i = 0; i < Amount_Pages; i++)
    {
        if(Page_Status_ptr[i] != Status_Unwritten)
            Current_Valid_Page = i;
    }
    /* If Data Flash is used for first time, set counter = 0 */
    if(Page_Status_ptr[Current_Valid_Page] == Status_Unwritten)
    {
        /* Set counter = 0 */
        FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page),
0xFFFF0000);
        /* Set cursor to current Data Flash address */
        Current_Cursor = 2;
    }
    else
    {
        /* Search where is current cursor for the next data to write and get the data has
been written */
```

```
        /* Check even value */
        temp = FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page));
        addr = (temp & Even_Addr_Mask) >> Even_Addr_Pos;
        data = (temp & Even_Data_Mask) >> Even_Data_Pos;
        /* Check Address is 0xFF (un-written) of not */
        if(addr == 0xFF)
        {
            /* If Address is 0xFF, then set cursor to current Data Flash address */
            Current_Cursor = 2;
        }
        else
        {
            /* Copy the address and data to SRAM */
            Written_Data[addr] = data;
            /* Check the whole Data Flash */
            for(i = 4; i < FMC_FLASH_PAGE_SIZE; i += 4)
            {
                /* Check odd value */
                temp = FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE *
Current_Valid_Page) + i);
                addr = (temp & Odd_Addr_Mask) >> Odd_Addr_Pos;
                data = (temp & Odd_Data_Mask) >> Odd_Data_Pos;
                /* Check Address is 0xFF (un-written) of not */
                if(addr == 0xFF)
                {
                /* If Address is 0xFF, then set cursor to current Data Flash address */
                    Current_Cursor = i;
                    break;
                }
                else
                {
                    /* Copy the address and data to SRAM */
                    Written_Data[addr] = data;
                }

                /* Check even value */
                addr = (temp & Even_Addr_Mask) >> Even_Addr_Pos;
                data = (temp & Even_Data_Mask) >> Even_Data_Pos;
                /* Check Address is 0xFF (un-written) of not */
                if(addr == 0xFF)
                {
```

```
                /* If Address is 0xFF, then set cursor to current Data Flash address */
                    Current_Cursor = i + 2;
                    break;
                }
                else
                {
                    /* Copy the address and data to SRAM */
                    Written_Data[addr] = data;
                }
            }
        }
    }
}
```

3.3 Write_Data()

```
/**
* @brief     Write one byte data to SRAM and current valid page.
             If this index has the same data, it will not changed in SRAM and Data Flash.
             If current valid page is full, execute Manage_Next_Page() to copy valid data
             to next page.
* @param[in] index: The index of data address.
* @param[in] data: The data that will be writeen.
* @retval    Err_ErrorIndex: The input index is not valid.
* @retval    0: Success
*/
uint32_t Write_Data(uint8_t index, uint8_t data)
{
    uint32_t temp = 0;

    /* Check the index is valid or not */
    if(index > Amount_of_Data)
    {
        return Err_ErrorIndex;
    }
    /* If the writing data equals to current data, the skip the write process */
    if(Written_Data[index] == data)
    {
        return 0;
    }
```

```
    /* Enable FMC ISP function */
    FMC_Enable();

    /* Current cursor points to odd position*/
    if((Current_Cursor & 0x3) == 0)
    {
        /* Write data to Data Flash */
        temp = 0xFFFF0000 | (index << Odd_Addr_Pos) | (data << Odd_Data_Pos);
        FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page) +
Current_Cursor, temp);
        /* Write data to SRAM */
        Written_Data[index] = data;
    }
    /* Current cursor points to even position*/
    else
    {
        /* Read the odd position data */
        temp = FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page) +
(Current_Cursor - 2));
        /* Combine odd position data and even position data */
        temp &= ~(Even_Addr_Mask | Even_Data_Mask);
        temp |= (index << Even_Addr_Pos) | (data << Even_Data_Pos);
        /* Write data to Data Flash */
        FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page) +
(Current_Cursor - 2), temp);
        /* Write data to SRAM */
        Written_Data[index] = data;
    }

    /* If current cursor points to the last position, then execute Manage_Next_Page() */
    if(Current_Cursor == (FMC_FLASH_PAGE_SIZE - 2))
    {
        /* Copy valid data to next page */
        Manage_Next_Page();
    }
    /* Add current cursor */
    else
    {
        /* Set current cursor to next position */
        Current_Cursor += 2;
    }
    return 0;
```

```
}
```

3.4 Manage_Next_Page()

```
/**
* @brief     Manage the valid data from SRAM to new page.
*/
void Manage_Next_Page(void)
{
    uint32_t i = 0, j, counter, temp = 0, data_flag = 0, new_page;

    /* Copy the valid data (not 0xFF) from SRAM to new valid page */
    /* Get counter from the first two bytes */
    counter = FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page));

    /* If current valid page is the last page, choose the first page as valid page */
    if((Current_Valid_Page + 1) == Amount_Pages)
    {
        new_page = 0;
        /* Add counter to record 1 E/W cycle finished for all pages */
        counter++;
    }
    else
    {
        new_page = Current_Valid_Page + 1;
    }

    /* Enable FMC ISP function */
    FMC_Enable();

    /* Copy first valid data */
    while(1)
    {
        /* Not a valid data, skip */
        if(Written_Data[i] == 0xFF)
        {
            i++;
        }
        /* Combine counter and first valid data, and write to new page */
        else
        {
```

```
                counter &= ~(Even_Addr_Mask | Even_Data_Mask);
                counter |= (i << Even_Addr_Pos) | (Written_Data[i] << Even_Data_Pos);
                FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * new_page), counter);
                i++;
                break;
            }
        }
        /* Copy the rest of data */
        for(j = 4; i < Amount_of_Data; i++)
        {
            /* Not a valid data, skip */
            if(Written_Data[i] == 0xFF)
            {
                continue;
            }
            /* Write to new page */
            else
            {
                /* Collect two valid data and write to Data Flash */
                /* First data, won't write to Data Flash immediately */
                if(data_flag == 0)
                {
                    temp |= (i << Odd_Addr_Pos) | (Written_Data[i] << Odd_Data_Pos);
                    data_flag = 1;
                }
                /* Second data, write to Data Flash after combine with first data */
                else
                {
                    temp |= (i << Even_Addr_Pos) | (Written_Data[i] << Even_Data_Pos);
                    FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * new_page) + j,
temp);
                    temp = 0;
                    data_flag = 0;
                    j += 4;
                }
            }
        }

        /* Set cursor to new page */
        Current_Cursor = j;
```

```
    /* If there is one valid data left, write to Data Flash */
    if(data_flag == 1)
    {
        temp |= 0xFFFF0000;
        FMC_Write(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * new_page) + j, temp);
        Current_Cursor += 2;
    }

    /* Erase the old page */
    FMC_Erase(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE * Current_Valid_Page));
    /* Point to new valid page */
    Current_Valid_Page = new_page;
}
```

3.5 Read_Data()

```
/**
* @brief      Read one byte data from SRAM.
* @param[in]  index: The index of data address.
* @param[in] data: The data in the index of data address from SRAM.
* @retval    Err_ErrorIndex: The input index is now valid.
* @retval    0: Success
*/
uint32_t Read_Data(uint8_t index, uint8_t *data)
{
    /* Check the index is valid or not */
    if(index >= Max_Amount_of_Data)
    {
        return Err_ErrorIndex;
    }

    /* Get the data from SRAM */
    *data = Written_Data[index];

    return 0;
}
```

3.6 Get_Cycle_Counter()

```
/**
* @brief     Get the cycle counter for how many cycles has page been erased/programmed.
* @retval    Cycle_Conter: The cycles that page has been erased/programmed.
*/
uint16_t Get_Cycle_Counter(void)
{
    uint16_t Cycle_Counter;

    /* Get the cycle counter from first two bytes in current Data Flash page */
    Cycle_Counter = (uint16_t)FMC_Read(DataFlash_BaseAddr + (FMC_FLASH_PAGE_SIZE *
Current_Valid_Page));

    return Cycle_Counter;
}
```

4 Sample Code

This chapter provides the sample code to implement, and takes the NuMicro® M051 DE series as an example, which uses 4 pages of Data Flash to emulate EEPROM and store 8-byte data. In the main loop, it will write data into the emulated EEPROM (Data Flash) continuously until the counter counts to 20,000.

```
#include <stdio.h>
#include "M051Series.h"
#include "EEPROM_Emulate.h"

#define PLLCON_SETTING      CLK_PLLCON_50MHz_HXT
#define PLL_CLOCK           50000000

#define Test_data_size        8
#define Test_page_amount      4

void SYS_Init(void)
{
    /*---------------------------------------------------------------------------------------------*/
    /* Init System Clock                                                                           */
    /*---------------------------------------------------------------------------------------------*/

    /* Enable External XTAL (4~24 MHz) */
    CLK->PWRCON |= CLK_PWRCON_XTL12M_EN_Msk;

    CLK->PLLCON = PLLCON_SETTING;

    /* Waiting for clock ready */
    CLK_WaitClockReady(CLK_CLKSTATUS_PLL_STB_Msk | CLK_CLKSTATUS_XTL12M_STB_Msk);

    /* Switch HCLK clock source to PLL */
    CLK->CLKSEL0 = CLK_CLKSEL0_HCLK_S_PLL;

    /* Update System Core Clock */
    /* User can use SystemCoreClockUpdate() to calculate PllClock, SystemCoreClock and
CycylesPerUs automatically. */
    SystemCoreClockUpdate();
    PllClock        = PLL_CLOCK;            // PLL
    SystemCoreClock = PLL_CLOCK / 1;        // HCLK
    CyclesPerUs     = PLL_CLOCK / 1000000;  // For SYS_SysTickDelay()
```

```
}

int main()
{
    uint32_t i;
    uint8_t u8Data;

    /* Unlock protected registers */
    SYS_UnlockReg();

    SYS_Init();

    /* Test Init_EEPROM() */
    Init_EEPROM(Test_data_size, Test_page_amount);

    /* Test Search_Valid_Page() */
    Search_Valid_Page();

    /* Test Write_Data() */
    for(i = 0; i < 254; i++)
    {
        Write_Data(i%Test_data_size, i%256);
    }

    /* Test Write_Data() contain Manage_Next_Page() */
    Write_Data(i%Test_data_size, 0xFF);

    /* Test Read_Data() */
    Read_Data(0x7, &u8Data);

    /* Test Write over 20000 times */
    while(Get_Cycle_Counter() < 20000)
    {
        for(i = 0; i < 247; i++)
        {
           Write_Data(i%Test_data_size, i%256);
            }
    }
    while(1);
}
```

5 Conclusion

This application note issues a new mechanism to use Data Flash to emulate EEPROM by combining more than 2 pages of Data Flash. This mechanism can increase the read/write speed by SRAM and the endurance of write/erase emulated EEPROM is up to 1 million times of reliable write/erase cycles.

User can get the written/erased cycles of Data Flash page by reading the counter value. Based on this information, user can estimate the product life.

For the NuMicro® Cortex®-M0 series, it is recommended to divide the target data into several data sets and use less Data Flash to achieve the target endurance of write/erase cycles. For the NuMicro® Cortex®-M4 series, user can use Data Flash directly without dividing the data into several sets.

REVISION HISTORY

Date	Revision	Description
2019. 09. 18	1.00	Initial version

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.