Rev. 0, 9/03

MOTOROLA
intelligence everywhere™

digital dna™

56F83xx Lends Hybrid Applications EEPROM Capability FlashEE

White Paper

Motorola 56F8300 Hybrid Controller Family

MOTOROLA.COM/SEMICONDUCTORS

56F83xx Lends Hybrid Applications EEPROM Capability -- FlashEE

John L. Winters

Contents

1. Introduction

1.1 Overview

Motorola has introduced a new class of hybrid MCU/DSP devices addressing the needs associated with automotive, industrial, consumer, motor control and other system control applications requiring Flash memory *and* EEPROM capability. These devices are the 56F8322, 56F8323, 56F8345, 56F8346, 56F8356, and 56F8357 parts, collectively referred to as the 56F83xx family.

This report presents the architectural advantage offered by the 56F83xx family of devices with respect to using its on-chip Flash as EEPROM. It also recounts the method used to demonstrate this capability and displays some of the source code.

Why use on-chip Flash as EEPROM?

A control system may be required to store a variety of information, including control information, calibration data, phone numbers, data logs, temperature exteme data, maximum velocity data, boundry GPS coordinates, and many other types of data. This data may need to be updated by the control system autonomously, perhaps in the field of deployment, and may be required for the system's self-calibration. The information stored may need to be retained, even when power is not supplied to the device. In many applications, an EEPROM is required to store non-volatile data; it could be eliminated if the 56F83xx's on-board Flash could be used to store **all** of this data. Using Flash to replace EEPROM has been named "FlashEE**"** in a previous Motorola Application Note (please see Reference 1).

1.2 The Keys to FlashEE

The keys to unlock FlashEE reside within the Flash memory subsystem of the 56F83xx family:

- A minimum of up to 10K write/erase cycles across temperature range; up to 100K is typical
- A data Flash Memory Page size of only 256 words
- No special voltages are required for Flash programming

1.3 How the Keys Open the Way

The Motorola 56F83xx family includes all three keys. But what if just one of these keys is omitted?

- Omit the first key and it will wear out the Flash before the normal life cycle of the product
- Omit the second key and the *number of data words used up each time any parameter must be updated* will soon cycle through all available memory, resulting in a linear decrease of product life
  For example, if the page size was 8K and only 256 words needed to be written at once, then most of the 8K words would be wasted and the *product life would decrease by a factor of 16*, assuming equal memory sizes.
- Omit the last key and special voltages would be needed just to update data in FlashEE

There's no need to do without any of these keys, since they are all built into the Flash Memory subsystem of the Motorola 56F83xx family of devices.

2. Using the Keys -- FlashEE on a 56F3xx Device

2.1 A Program Illustrates the Concept

The tested, working C program at the end of this section has been used to train engineers using FlashEE on a 56F83xx controller. **Section 2.2** discusses the operation of this program in detail.

By writing such a simple program, it's easy to use as a seed for more complex applications requiring FlashEE. It can be cut/pasted into an application, tested, then scaled up to the data structure to be saved. Integration would then meld the FlashEE technique into the application. This is a stand-alone example which uses simple subroutines.

This simple example just writes one 16-bit word to the Data FLASH using the FlashEE technique. When a page finally fills up, it is not used again until it is erased. This example then shows how to save the value of a 16-bit word in a non-volatile manner using the FlashEE technique.

Why FlashEE 16 bits and not just one bit at a time? Since both the row size and the word size are 16 bits, each word may be written only once prior to erasing. Thus, if only one bit is be stored, it would have to be stored in a 16-bit word. Repeated writes to a word without an erase can damage the Flash, since the specifications would be exceeded.

Code Example 1 is a part of a larger program, which is available with all source code, including subroutines. See Reference 3.

Code Example 1.

```
/* Using Flash as EEPROM on 56F836 -- Training Module */

#include "ssd_types.h"
#include "ssd_hfm.h"
#include "ssd_hfm_config.h"
#include "ssd_demo.h"
#include "ssd_hfm_clk.h"
#include "clock.h"
#include <stdio.h>
#include <stdlib.h>

// Flash configuration stucture
FlashConfig flashConfig =
{
 HFM_BASE,
 HFM_CONFIG_BASE,
 FLASH_CLOCK_DIVIDER,
 false
};
UWord16 buffer[SOURCE_DATA_BUFFER_SIZE];
UWord16 main(void)
{
 UWord16  Flash_bits;            // as read from flash
 UWord16  i;                     // Index
 FLASH_TYPE flashType;           // The type of flash block
 UWord32  flashBaseAddr, // The base address of flash block
          flashSize;             // The size of flash block
 UWord32  eraseBlkStartAddr;     // The start address of flash block
 UWord16  number;                // Flash page number operation applicable
 UWord32  source;                // Source address for program and verify
 UWord32  dest,                  // Flash start address operation applicable
          size;                  // Flash size operation applicable
 UWord32  Flash_p ;              // Pointer to flash
 // Set the vector base address
 REG_WRITE(INTC_BASE_ADDRESS + INTC_VBA, 0x300);
 // Set the PLL
 SetPLL(PLL_PRESCALER, PLL_POSTSCALER, PLL_MULTIPLIER);
 //========================== Initialize HFM Module for Data Flash
==========================
 FlashInit((UWord32)(&flashConfig));
 flashType = FLASH_TYPE_D;
 flashBaseAddr = DATA_FLASH_START_ADDR;
 flashSize = DATA_FLASH_SIZE;
 number = DATA_FLASH_SIZE / FLASH_PAGE_SIZE_D;
 // Clear the all protect bits
 FlashSetProtection((UWord32)(&flashConfig), flashType, 0x0000);
 //========================== Find unused flash word (=1)
==========================
 for (Flash_p =  DATA_FLASH_START_ADDR;
      Flash_p < DATA_FLASH_START_ADDR + DATA_FLASH_SIZE;
      Flash_p ++
     )   // if word is 0xfff it must be errased already:
```

```
  if (*(UWord16*)Flash_p == 0xffff) break; // search for an unused word.
 if (Flash_p == DATA_FLASH_START_ADDR + DATA_FLASH_SIZE) // if flash used up, mass
erase.
 {
  //============================ Mass Erase Flash (if all bits zero)
==============================
  eraseBlkStartAddr = flashBaseAddr;
  FlashMassErase((UWord32)(&flashConfig), flashType, eraseBlkStartAddr);
  Flash_p = DATA_FLASH_START_ADDR;  // flash now empty again!
 }
 // Flash_p now points to first flash location with  unused word.. which must
exist.
 Flash_bits = 0x1234 ;                // change to any value other than 0xffff
 //============================= Program the one word into the Flash
==============================
 source =(UWord32) &Flash_bits;      // source for write to flash
 dest = Flash_p;                     // destination for write to flas
 size = 1;                           // number of 16 bit words to write to flash
 FlashProgram((UWord32)(&flashConfig), flashType, dest, size, source);
 printf ("\n *Flash_p= %x \n", *(UWord16*)Flash_p);    // show word in flash
 printf ("\n Flash_p= %x \n",(UWord16)Flash_p);              // show flash address

// Solution

 if (  Flash_p & 1             // each time program runs, it writes another word.
             // After one run, the program prints odd
             // After two, even.  The pattern continues.
             // Power failures do not affect it.
             // After all words are used, they are all cleared and the
             // pattern continues.  Since erasure is infrequent, the life
             // of the memory is long.

    )
  printf ("\n\n Odd");
  else printf ("\n\n Even");

}
```

2.2 The Program Explained

Code Example 2 brings in both the header files for low-level Flash subroutines that access the Flash, and standard header files so that the debugger can output messages.

Code Example 2.

```
/* Using Flash as EEPROM on 56F836 -- Training Module */

#include "ssd_types.h"
#include "ssd_hfm.h"
#include "ssd_hfm_config.h"
#include "ssd_demo.h"
#include "ssd_hfm_clk.h"
#include "clock.h"
#include <stdio.h>
#include <stdlib.h>
```

The FLASH configuration structure in **Code Example 3** configures the Flash for operation.

Code Example 3.

```
----------
// Flash configuration stucture
FlashConfig flashConfig =
{
 HFM_BASE,
 HFM_CONFIG_BASE,
 FLASH_CLOCK_DIVIDER,
 false
};
UWord16 buffer[SOURCE_DATA_BUFFER_SIZE];
```

Code Example 4 includes the start of the actual code for the main program, the first application-oriented item to gain control of the processor.

Code Example 4.

```
----------
UWord16 main(void)
{
```

Code Example 5 displays data illustrating the technique of FlashEE.

Code Example 5.

```
----------
UWord16  Flash_bits;            // as read from flash
 UWord16  i;                     // Index
 FLASH_TYPE flashType;           // The type of flash block
 UWord32  flashBaseAddr, // The base address of flash block
          flashSize;             // The size of flash block
 UWord32  eraseBlkStartAddr;     // The start address of flash block
 UWord16  number;                // Flash page number operation applicable
 UWord32  source;                // Source address for program and verify
 UWord32  dest,                  // Flash start address operation applicable
          size;                  // Flash size operation applicable
 UWord32  Flash_p ;              // Pointer to flash
```

Code Example 6 includes initialization code and will vary from application to application. It is the first code run within this *main.c* program.

Code Example 6.

```
----------
// Set the vector base address
 REG_WRITE(INTC_BASE_ADDRESS + INTC_VBA, 0x300);
 // Set the PLL
 SetPLL(PLL_PRESCALER, PLL_POSTSCALER, PLL_MULTIPLIER);
 //========================== Initialize HFM Module for Data Flash
==========================
 FlashInit((UWord32)(&flashConfig));
 flashType = FLASH_TYPE_D;
 flashBaseAddr = DATA_FLASH_START_ADDR;
 flashSize = DATA_FLASH_SIZE;
 number = DATA_FLASH_SIZE / FLASH_PAGE_SIZE_D;
 // Clear the all protect bits
 FlashSetProtection((UWord32)(&flashConfig), flashType, 0x0000);
```

Code Example 7 makes use of the row size of 16 bits. It searches for a word equal to 0xffff, or all 16 bits equal to one when viewed as a binary number: 1111111111111111. This is the value the word takes on when it is erased.

Code Example 7.

```
----------
//========================== Find unused flash word (=1)
==========================
 for (Flash_p =  DATA_FLASH_START_ADDR;
      Flash_p < DATA_FLASH_START_ADDR + DATA_FLASH_SIZE;
      Flash_p ++
     )   // if word is 0xfff it must be errased already:
  if (*(UWord16*)Flash_p == 0xffff) break; // search for an unused word.
 if (Flash_p == DATA_FLASH_START_ADDR + DATA_FLASH_SIZE) // if flash used up, mass
erase.
 {
```

As shown in **Code Example 8**, if the code "falls through" to this point, it means that all Flash is used and a mass erase will take place.

Code Example 8.

```
----------
//=================== Mass Erase Flash (if all bits zero) ======================
  eraseBlkStartAddr = flashBaseAddr;
  FlashMassErase((UWord32)(&flashConfig), flashType, eraseBlkStartAddr);
  Flash_p = DATA_FLASH_START_ADDR;  // flash now empty again!
 }
```

In **Code Example 9**, the meaning of the token 0xffff is erased, so we can store any other number using this technique.

This will keep track of how much memory has been "used" so that it's not necessary to use even more memory (and time) to track this data.

Code Example 9.

```
----------
// Flash_p now points to first flash location with  unused word.. which must exist.
 Flash_bits = 0x1234 ;                // change to any value other than 0xffff
```

The simple toggle function calculates odd or even numbers of program runs; each run uses one word. See **Code Example 10**.

Code Example 10.

```
----------
//============================== Program the one word into the Flash
================================
 source =(UWord32) &Flash_bits;      // source for write to flash
dest = Flash_p;                     // destination for write to flas
 size = 1;                           // number of 16 bit words to write to flash
 FlashProgram((UWord32)(&flashConfig), flashType, dest, size, source);
 printf ("\n *Flash_p= %x \n", *(UWord16*)Flash_p);    // show word in flash
 printf ("\n Flash_p= %x \n",(UWord16)Flash_p);              // show flash address

// Solution

 if (  Flash_p & 1              // each time program runs, it writes another word.
             // After one run, the program prints odd
             // After two, even.  The pattern continues.
             // Power failures do not affect it.
             // After all words are used, they are all cleared and the
             // pattern continues.  Since erasure is infrequent, the life
             // of the memory is long.

    )
  printf ("\n\n Odd");
  else printf ("\n\n Even");

}
```

3. Conclusion

A real application can store real data in these words. Since only one word is written at a time, and the Flash is page erasable, making a useful FlashEE application is simple. The specifications of the Flash make it well-suited for use as EEPROM when the total updates are in the tens to hundreds of thousands. The design of the Motorola 56F83xx Flash unit makes this practical.

4. References

1) *Using FLASH as EEPROM on the MC68HC908CP32*, AN2183/D

2) *Motorola 56F83x Hybrid MCU/DSP Flash Family*

3) *Using FLASH as EEPROM Training Exercise*

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors

EEPROMFLASH/D

Freescale Semiconductor, Inc.