The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

How to Check the Ordering Part Number

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

For More Information

Please contact your local sales office for additional information about Cypress products and solutions.

About Cypress

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204370

8FX Family, Example of Flash operation for Starter Kit

Associated Part Family:	Series Name	Product Number
	MB95560H	MB95F564H/K, MB95F563H/K, MB95F562K/H
	MB95570H	MB95F574H/K, MB95F573H/K, MB95F572K/H
	MB95580H	MB95F584H/K, MB95F583H/K, MB95F582H/K

This application note explains the method of the Flash Memory operation for the “New 8FX” family. MB95560H/570H/580H series is used in the explanation of control methods.

Contents

1 Introduction....1
2 Algorithm of the programming....1
2.1 Overview....1
2.2 Overview of Dual Operation Flash Memory....1
2.3 Sector/Bank Configuration of Flash Memory....2
2.4 Register for Flash Memory....2
2.5 Invoking Flash Memory Automatic Algorithm....3
3 Source code....3
3.1 flash.asm....3
3.2 Main.c....7
4 Notes on Flash Operation....10
5 Document History....11

1 Introduction

This application note explains the method of the Flash Memory operation for the “New 8FX” family. MB95560H/570H/580H series is used in the explanation of control methods. This method is applicable for all “New 8FX” family. This example includes the programming algorithm, flash.asm, erasing of a flash and Main.c source code.

2 Algorithm of the programming

Algorithm of the flash programming

2.1 Overview

The dual operation Flash memory consists of an upper bank and lower bank. Unlike conventional Flash products, programming / erasing data to / from one bank and reading data from another bank can be executed simultaneously. This example is sector programming/sector erase from upper bank to lower bank.

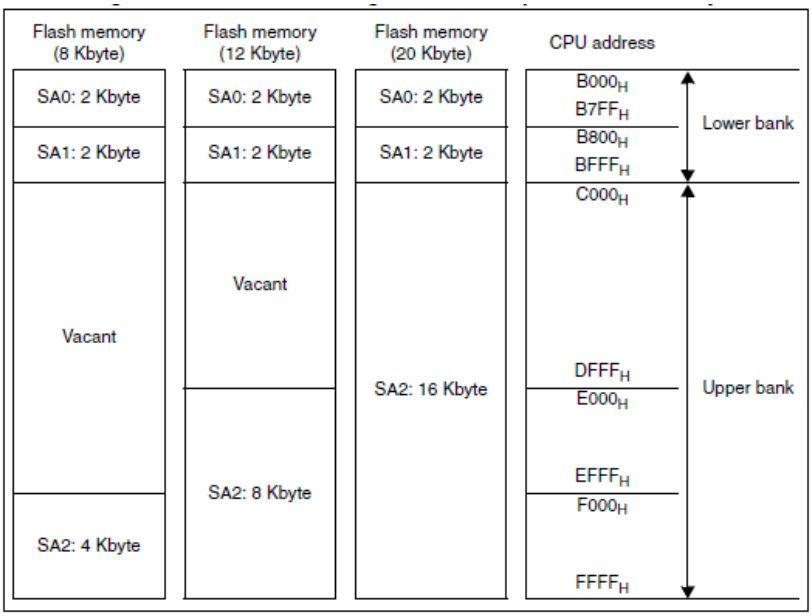
2.2 Overview of Dual Operation Flash Memory

- Sector configuration: Upper bank 4KB / 8KB / 16KB + Lower bank 2KB x2
- Automatic program algorithm (Embedded Algorithm)
- Detecting the completion of programming/erasing using the data poling flag or the toggle bit
- Detecting the completion of programming/erasing by CPU interrupt
- Compatible with JEDEC standard commands
- Programming/erase 100000 times

2.3 Sector/Bank Configuration of Flash Memory

Figure 1 shows the sector configuration of the Dual operation Flash Memory. The upper and lower addresses of each sector are shown in the figure.

Figure 1. 8/12/20 KB Sector configuration of Flash Memory



2.4 Register for Flash Memory

Figure 2 shows the Flash Memory status register.

Please refer to the hardware manual of the MB95560H/570H/580H series to more detailed information.

Figure 2. 8/12/20 KB Registers for Flash memory

Flash memory status register 2 (FSR2)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0071_H	PEIEN	PGMEND	PTIEN	PGMTO	EEIEN	ERSEND	ETIEN	ERSTO	00000000_B
	R/W	R(RM1),W	R/W	R(RM1),W	R/W	R(RM1),W	R/W	R(RM1),W	

Flash memory status register (FSR)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0072_H	-	-	RDYIRQ	RDY	Reserved	IRQEN	WRE	SSEN	$000X0000_B$
	R0/WX	R0/WX	R(RM1),W	R/WX	R/W0	R/W	R/W	R/W	

Flash memory sector write control register 0 (SWRE0)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0073_H	Reserved	Reserved	Reserved	Reserved	Reserved	SA2E	SA1E	SA0E	00000000_B
	R/W0	R/W0	R/W0	R/W0	R/W0	R/W	R/W	R/W	

Flash memory status register 3 (FSR3)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0074_H	-	-	-	CERS	ESPS	SERS	PGMS	HANG	$000XXXXX_B$
	R0/WX	R0/WX	R0/WX	R/WX	R/WX	R/WX	R/WX	R/WX	

Flash memory status register 4 (FSR4)

Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Initial value
0075_H	-	CEREND	CTIEN	CERTO	-	-	-	-	00000000_B
	R0/WX	R(RM1),W	R/W	R(RM1),W	R0/WX	R0/WX	R0/WX	R0/WX	

2.5 Invoking Flash Memory Automatic Algorithm

There are four commands that invoke the Flash memory automatic algorithm: read/reset, program, chip erase and sector erase. Figure 3 shows the commands.

Figure 3. Command Sequence

Command sequence	Bus write cycle	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Read/reset*	1	F_XXX_H	$F0_H$	-	-	-	-	-	-	-	-	-	-
	1	RA	RD	-	-	-	-	-	-	-	-	-	-
Program	4	$UAA8_H$	AA_H	$U554_H$	55_H	$UAA8_H$	$A0_H$	PA	PD	-	-	-	-
Chip erase	6	$UAA8_H$	AA_H	$X554_H$	55_H	$UAA8_H$	80_H	$UAA8_H$	AA_H	$X554_H$	55_H	$UAA8_H$	10_H
Sector erase	6	$UAA8_H$	AA_H	$X554_H$	55_H	$UAA8_H$	80_H	$UAAA_H$	AA_H	$U554_H$	55_H	SA	30_H
Unlock bypass entry	3	$UAA8_H$	AA_H	$U554_H$	55_H	$UAA8_H$	20_H	-	-	-	-	-	-
Unlock bypass program	2	$UXXX_H$	$A0_H$	PA	PD	-	-	-	-	-	-	-	-
Unlock bypass reset	2	$UXXX_H$	90_H	$UXXX_H$	any	-	-	-	-	-	-	-	
Sector erase suspend		Programming data "$B0_H$" to the address "$UXXX_H$" suspends erasing during sector erase.											
Sector erase resume		Programming data "30_H" to the address "$UXXX_H$" resumes suspended sector erase.											
Erase sector add		Programming data "30_H" to the SA adds a new sector to be erased.											

RA : Read address
PA : Program address
SA : Sector address (specify arbitrary one address in sector)
RD : Read data
PD : Program data
U : Upper 4 bits same as RA, PA, and SA
F_X : FF/FE
X : Any address value
any : Any program data
*: Both types of read/reset command can reset the Flash memory to read mode.

The hardware manual of MB95560H series has a sector command address of the 0xUAA8, which also can use the 0xUAAA instead of that. This application note uses a command address of 0xUAAA.

3 Source code

Explain of the sample code

3.1 flash.asm

The flash.asm is assembly code for the Flash programming and Sector erase.

The _EraseStart is start address of the Sector erase. Code is on the flash.asm.

The _WriteStart is start address of the Flash programming. Code is on the flash.asm.

These are exported that can call it from main().

```
.EXPORT _EraseStart
.EXPORT _WriteStart
```

Enable the sector programming by SWRE0.

Set the both SA0E and SA1E in the SWRE0

```
MOV    A, #03H       ; SA0E/SA1E write enable
MOV    SWRE0,A
```

The erase address was pushed to keep the address. The address is assigned by main ().

```
_EraseStart:
                          ;
      MOVW   A,EP
      PUSHW  A             ; Save erase address
```

The programming address was pushed to keep the address. The writing data was pushed to keep the data. The address is assigned by main ().

```
_WriteStart:
                    ;
_write_address:
       PUSHW  IX           ; Save write data
       MOVW   A,EP
       PUSHW  A            ; Save write address
```

The U in an address represents the upper four bits of an address.

This code is calculation method of the U for the UAAAH and U554H.

```
MOVW   A,#0xF000
ANDW   A
MOVW   A,#0x0AAA
ORW    A
MOVW   EP,A          ; 0x0AAA | (address & 0xf000)
XCHW   A,T
MOVW   A,#0x0554
ORW    A
MOVW   IX,A          ; 0x0554 | (address & 0xf000)
```

Enables programming by a WRE bit of FSR resistor.

```
SETB   FSR:WRE       ; Write Enable
```

Automatic algorithm for erasing a sector.

```
MOV    A,#0xAA       ; 0x*AAAH <= 0xAA
MOV    @EP,A

MOV    A,#0x55       ; 0x*554 <= 0x55
MOV    @IX,A
MOV    A,#0x80        ; 0x*AAAH <= 0x80
MOV    @EP,A

MOV    A,#0xAA        ; 0x*AAAH <= 0xAA
MOV    @EP,A

MOV    A,#0x55        ; 0x*554 <= 0x55
MOV    @IX,A

POPW   A              ; Restore Erase address SA
MOVW   EP,A
MOV    A,#30H        ; The last data.  Sector erase
MOV    @EP,A         ; Start Erase
```

Automatic algorithm for Flash programming.

```
MOV    A,#0xAA         ; 0xUAAAH <= 0xAA
MOV    @EP,A

MOV    A,#0x55         ; 0xU554 <= 0x55
MOV    @IX,A

MOV    A,#0xA0         ; 0xUAAAH <= 0xA0
MOV    @EP,A

POPW   A
MOVW   EP,A            ; write address PA
POPW   IX

MOVW   A,IX            ; write data PD
MOV    @EP,A           ; to write flash
```

Erasing loop

After a programming/erase command is issued, there is a delay of two machine clock cycles before the RDY bit becomes “0”. After the issue of a programming/erase command, wait for those two machine clock cycles to elapse “inserting NOP twice” before reading this bit.

More detailed information is on the hardware manual.

```
       NOP
       NOP
EraseLoop:
       BBS    FSR:RDY,EraseEnd ; Erase Flash successes?

       MOV    A,@EP
       AND    A,#0x20           ; Check Time Out?
       BZ     EraseLoop

       BBS    FSR:RDY,EraseEnd ; Erase Flash successes?
       NOP
       BBS    FSR:RDY,EraseEnd ; Erase Flash successes?
```

Programming loop

```
       NOP
       NOP
WriteLoop:
       BBS    FSR:RDY,WriteEnd ; write Flash successes?

       MOV    A,@EP
       AND    A,#0x20          ; to check time out?
       BZ     WriteLoop

       BBS    FSR:RDY,WriteEnd ; write Flash successes?
       NOP
       BBS    FSR:RDY,WriteEnd ; write Flash successes?
```

Flash erase was failed. Issue the Flash reset and set the error flag on the A resistor.

```
EraseError:
       MOV    A,#0xF0
       MOV    0xFF00,A      ; Reset Flash
       MOV    A,#01H        ; Set error Flag
```

A flash programming failed. Issue the Flash reset and set the error flag on the A resistor.

```
WriteError:
      MOV   A,#0xF0
      MOV   0xFF00,A     ; Reset Flash
      MOV   A,#01H       ; Set error Flag
```

Disable a programming by a WRE bit of FSR resistor.

```
   CLRBFSR:WRE       ; write disable
```

Flash erase was success. Set the success flag on the A resistor.

```
EraseEnd:
      MOV    A,#00H        ; normal ack
      MOVW   EP,A
```

A flash programming success. Set the success flag on the A resistor.

```
WriteEnd:
      MOV    A,#00H        ; normal ack
      MOVW   EP,A
```

3.2 Main.c

Main.c is the C source code of an executing a programming of flash and erasing sector.

Definition of the global variables

```
unsigned char  result, Flag;
unsigned short address;
unsigned char  data;
```

3.2.1 Definition of the erasing address

The variable number of “_address” in assembly language is the same as variable number of “address” in c language. Below is the transfer to the erase address to EP, which will be used in flash.asm.

```
MOVW A, _address
MOVW EP,A
```

The assembly code of the Flash Erase routine is called.

```
CALL _EraseStart
```

_EraseStart is start address of the Flash.asm.

3.2.2 Definition of the programing address

Below is the transfer to programming address to the EP and transfer to the data to IX, which will be used in flash.asm.

```
MOV  A,_data      ;write data
MOVW IX,A
MOVW A, _address  ;write address
MOVW EP,A
```

The assembly code of the Flash programming routine is called

```
CALL _WriteStart
```

_WriteStart is start address of the Flash.asm.

3.2.3 How to use the programming functions

Below is a sample C main code, which shows how to use the flash programming functions. Refer to the whole sample code.

EraseStart is the start address of Flash Erase routine in flash.asm. WriteStart is the start address of Flash Write routine in flash.asm. EraseStart and WriteStart are imported for instructions in main ().

The variable number of "_EraseStart" in assembly language is the same as variable number of "EraseStart" in c language, and "_WriteStart" is the same as "WriteStart".

```
extern EraseStart;
extern WriteStart;
```

The user can add program, according to the result of flash operation. This example set the GPIO.

```
void error(void)
{
       IO_PDR0.bit.P05=0;         // do something here LED2
}
void success(void)
{
        IO_PDR6.bit.P64=0;        // do something here LED3
}
```

The programming address "0xB000" and data "0xA0" are transferred to flash.asm by global variables address and data. And the programming address "0xB000" and data "0xA0" could be assigned by user. This sample code used a fixed address and data.

3.2.4 Call of sector erase

The flash_erase () is called, and result of sector erase is return to Flag.

```
//-------------------------------------------------------
// Sector Erase flash
//-------------------------------------------------------
       address = 0xB000;          // set write address (SA0)
       Flag = flash_erase();      // flash sector erase routine
```

3.2.5 Call of flash programming

The flash_write () is called, and result of the programming to the flash is set to Flag.

```
//-------------------------------------------------------
// Write
//-------------------------------------------------------
       address = 0xB000;          // set write address (SA0)
       data = 0xA0;               // set write data
       Flag = flash_write();      // flash write routine
```

3.2.6 Check the Flag

Flag ==1 is fails. Do something in error ().

Flag == 0 is success. Do something in success ().

```
if (Flag == 1)
       error();
else
       success();
```

4 Notes on Flash Operation

This section provides notes on flash operation.

This sample code uses a sector-erase command. The commands of the chip-erase do erase a NVR area. That also erases a trimming value of an internal CR data.

Please refer to the chapter NVR of the hardware manual.

5 Document History

Document Title: AN204370 - 8FX Family, Example of Flash operation for Starter Kit

Document Number:002-04370

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	HUAL	03/12/2013	Initial release
			01/31/2014	Company name and layout design changed
*A	5100354	HUAL	01/25/2016	Migrated Spansion Application Note MB95F564K-AN702-00033-1v1-E to Cypress format

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless
Spansion Products	cypress.com/spansion products

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/go/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2013-2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.
This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.
Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.
Use may be limited by and subject to the applicable Cypress software license agreement.