AN205384

F²MC - 8FX Family, MB95200H/210H Series, Flash Operation

This application note describes flash operation in MB95200H/210H Series. This application note describes the programming algorithm and explains a complete project which includes flash.asm, flash erase C code flash write C code and main.c.).

Contents

1 Introduction

This application note describes flash operation in MB95200H/210H Series.

This application note describes the programming algorithm and explains a complete project which includes flash.asm, flash erase C code flash write C code and main.c.

2 Programming Algorithm

This chapter introduces the algorithm of Flash Operation.

2.1 Overview

If a user program is running in the flash memory, the flash memory cannot be erased or programmed simultaneously. The only way for the user is to store flashing routines into the RAM area, so that the CPU executes instructions in this memory area and the flash can be erased and programmed.

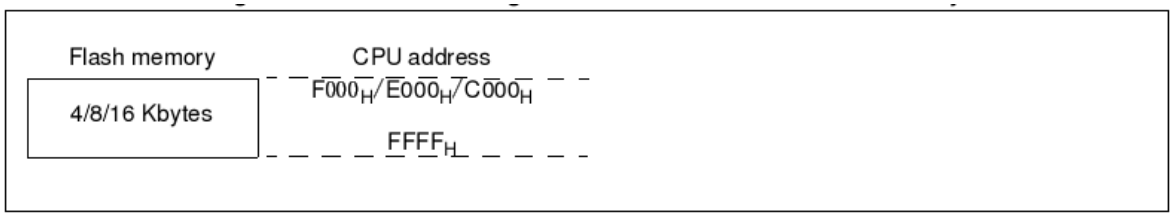
2.2 Features of Flash Memory

- Sector configuration: 4KB x 8 bits / 8KB x 8 bits / 16KB x 8 bits
- Automatic program algorithm (Embedded Algorithm)
- Detecting the completion of programming/erasing using the data polling or toggle bit function
- Detecting the completion of programming/erasing by CPU interrupts
- Compatible with JEDEC standard commands
- Erase/program cycle (minimum): 100,000 times

2.3 Sector Configuration

Figure 1 shows the sector configuration of the 32/64/128-KBIT flash memory. The upper and lower addresses of each sector are given in the figure.

Figure 1. Sector Configuration of 32/64/128-KBIT Flash Memory



2.4 Register of Flash Memory

Figure 2 shows the register of flash memory. For details please refer to Chapter 20 of the MB95200H/210H Series hardware manual.

Figure 2. Register of Flash Memory

Flash memory status register (FSR)

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

R/W: Readable/writable (Read value is the same as write value)
R(RM1), W: Readable/writable (Read value is different from write value, "1" is read by read-modify-write instruction)
R/WX: Read only (Readable, writing has no effect on operation)
R/W0: Reserved bit (Write value is "0", read value is the same as write value)
R0/WX: Undefined bit (Read value is "0", writing has no effect on operation)
X: Indeterminate

2.5 Starting Flash Memory Automatic Algorithm

There are three types of commands that invoke the flash memory automatic algorithm: read/reset, write (program) and chip-erase. Figure 3 lists commands used in programming/erasing flash memory.

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	$F0_H$	-	-	-	-	-	-	-	-	-	-
	4	$UAAA_H$	AA_H	$U554_H$	55_H	$UAAA_H$	$F0_H$	RA	RD	-	-	-	-
Write	4	$UAAA_H$	AA_H	$U554_H$	55_H	$UAAA_H$	$A0_H$	PA	PD	-	-	-	-
Chip erase	6	$XAAA_H$	AA_H	$X554_H$	55_H	$XAAA_H$	80_H	$XAAA_H$	AA_H	$X554_H$	55_H	$XAAA_H$	10_H

- RA : Read address
- PA : Write (program) address
- RD : Read data
- PD : Write (program) data
- U : Upper 4 bits same as RA, PA and SA
- F_X : FF/FE
- X : Arbitrary address

*: Both commands can reset flash memory to read mode.

3 Source Codes

Sample codes for Flash Operation

3.1 How to Compile the RAM Code

3.1.1 Overview

This sample project demonstrates the flash operation. The assembly flash.asm routine is executed in the RAM; it is permanently located in the RAM during run-time. The assembly flash.asm routine is stored in the flash memory, but its addresses need to be compiled and linked to the RAM, so that it can run correctly when it is copied from the flash memory to the RAM. To get permanent RAMCODE working, four steps have to be done.

1. Define code in RAMCODE section
2. Link Code for RAM and ROM
3. Copy code from the ROM to the RAM in start-up file
4. Jump to the RAM to execute the RAMCODE

The four steps will be explained one by one below.

3.1.2 Define Code in RAMCODE Section

RAMCODE is just a section name, which can be named by the user. Use the same name when setting the section in the Softune Workbench project environment. More will be described in 3.1.3

To define code in RAMCODE Section and in Assembly language modules, the following statement can be used:

```
.SECTION RAMCODE, CODE
```

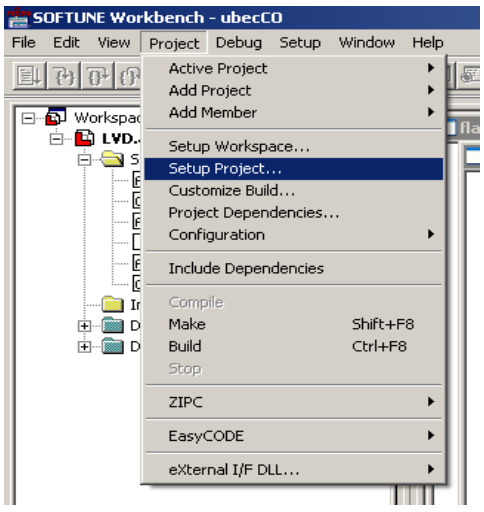
This line forces to locate all Assembly code of flash.asm to RAMCODE. A separate module (file with extension 'asm') should be used for RAMCODE functions only.

3.1.3 Link Code for RAM and ROM

The linker of the Softune Workbench provides a dedicated mechanism to link code for RAM and ROM, which address of every instruction of the program in RAMCODE section are resolved for RAM and the belonging code data are stored in ROM.
The same settings in the Workbench have to be made in project which can be done through 4 steps as shown below.

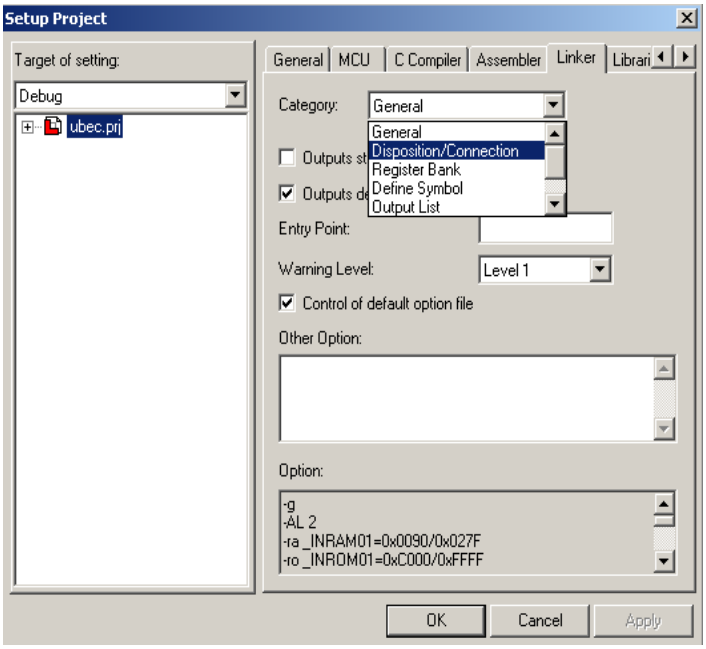
Step1: Select **Setup Project** in **Project** menu.

Figure 4. Select Setup Project in Project



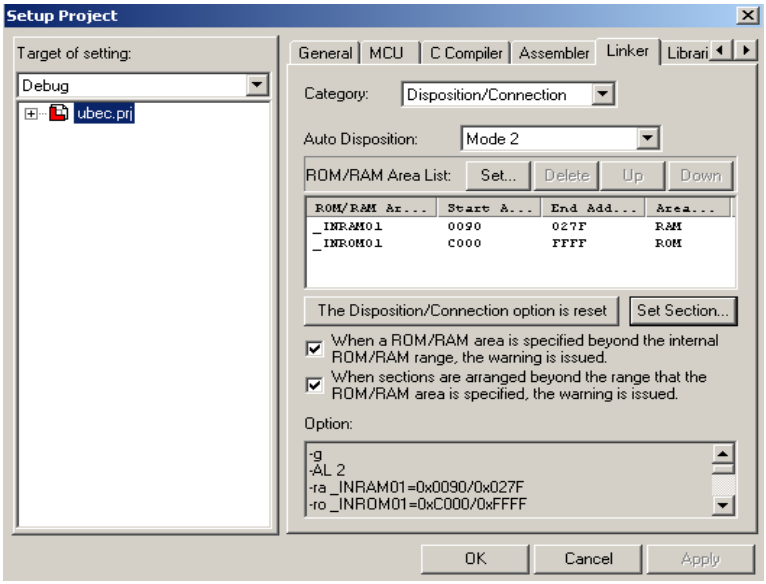
Step2: Select Disposition/Connection from Category in Setup Project window.

Figure 5. Select Disposition/Connection in Setup Project



Step 3: Select Set Section in Disposition/Connection

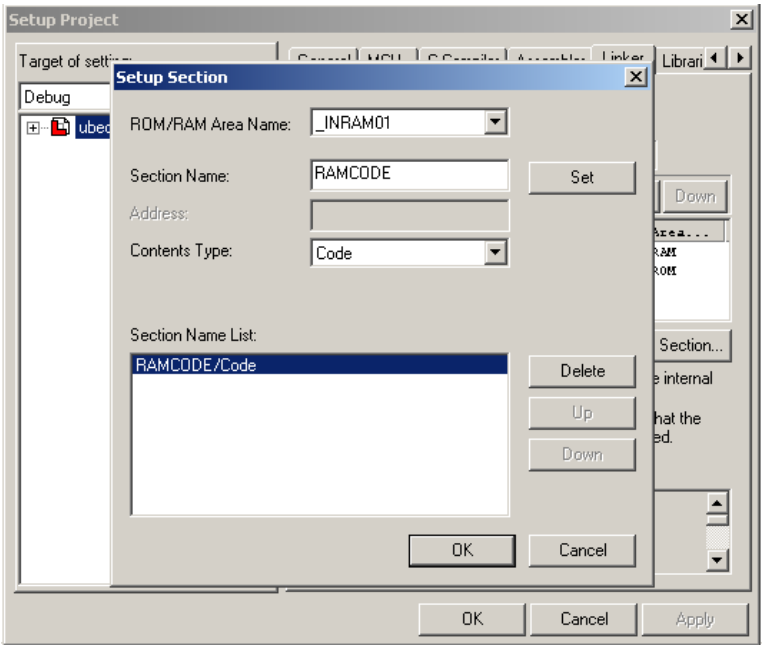
Figure 6. Select Set Section in Disposition/Connection



Step 4: RAM Setting in Setup Section.

The linker is informed of a code section to be located in RAM, after setting RAMCODE/Code in _INRAM01.

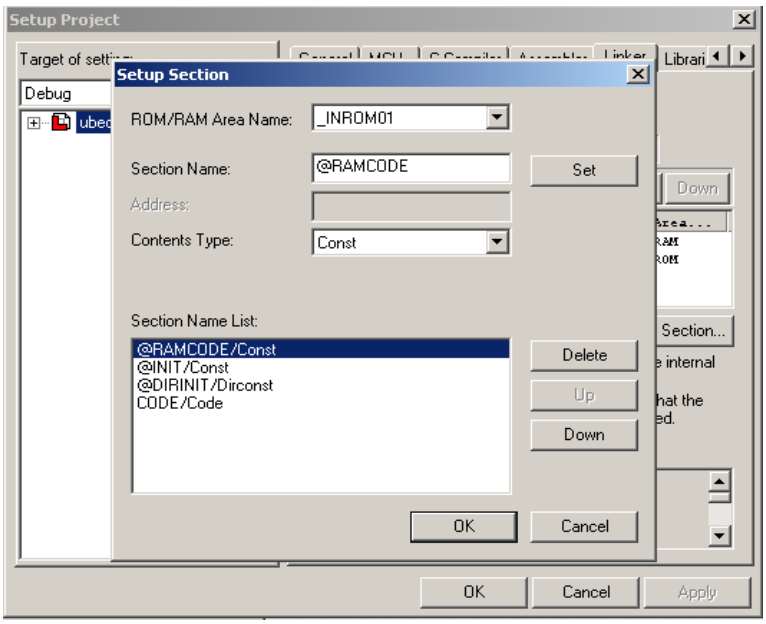
Figure 7. RAM Setting in Setup Section



Step 5: ROM Setting in Setup Section

The linker will recognize the wish to store the initialized data in ROM, after setting a "@" section with same name " RAMCODE" and setting @RAMCODE/Const in _INROM01.

Figure 8. ROM Setting in Setup Section



3.1.4 Copy Code from ROM to RAM

In order to be able to handle these sections, the linker provides generic labels for the start addresses of the sections in the RAM and the ROM.

RAM + section name

ROM + section name

As the section is named "RAMCODE" in this example, labels with RAMCODE appended to are generated:

_RAM_RAMCODE

_ROM_RAMCODE

These labels have to be imported in the startup file, before the code is copied from the ROM to the RAM.

For copying code from ROM to RAM, four points have to be added in current startup file.

These labels are imported by the startup file when point 1and point 2 are added as below.

```
;------------------------------------------------------------------------
; external declaration of symbols
;------------------------------------------------------------------------
     .IMPORT    _RAM_RAMCODE   ; point 1 added here
     .IMPORT    _ROM_RAMCODE   ; point 2 added here
```

The startup file adds a descriptor of the sections in ROM and RAM and their size. Then startup file is prepared to copy a section with name "RAMCODE", when point 3 is added.

```
;------------------------------------------------------------------------
; definition to start address of data, const and code section
;------------------------------------------------------------------------
     .SECTION    RAMCODE, CODE, ALIGN=1  ; point 3 added here
```

The code in RAMCODE section is copied from the ROM to the RAM when instructions are implemented as shown below.

```
;------------------------------------------------------------------------
; copy initial value *CONST(ROM) section to *INIT(RAM) section
;------------------------------------------------------------------------
  ICOPY     _ROM_RAMCODE, _RAM_RAMCODE, RAMCODE; point 4 added here
```

ICOPY src_addr, dest_addr, src_section is a macro, which can copy initial value from _ROM_RAMCODE (ROM) section to _RAM__RAMCODE (RAM) section. The detailed code of ICOPY macro is showed on Page22 on Appendix.- Project

3.1.5 Jump to RAM to Execute the RAMCODE

The code that has been copied from the ROM to the RAM is called by instructions.

For example, if Flash Erase and Flash Write routine have been copied to the RAM, and _EraseStart is the start address of Flash Erase routine and _WriteStart is the start address of Flash Write routine, they can be called by instructions below:

```
      CALL _EraseStart
```

or

```
      CALL _WriteStart
```

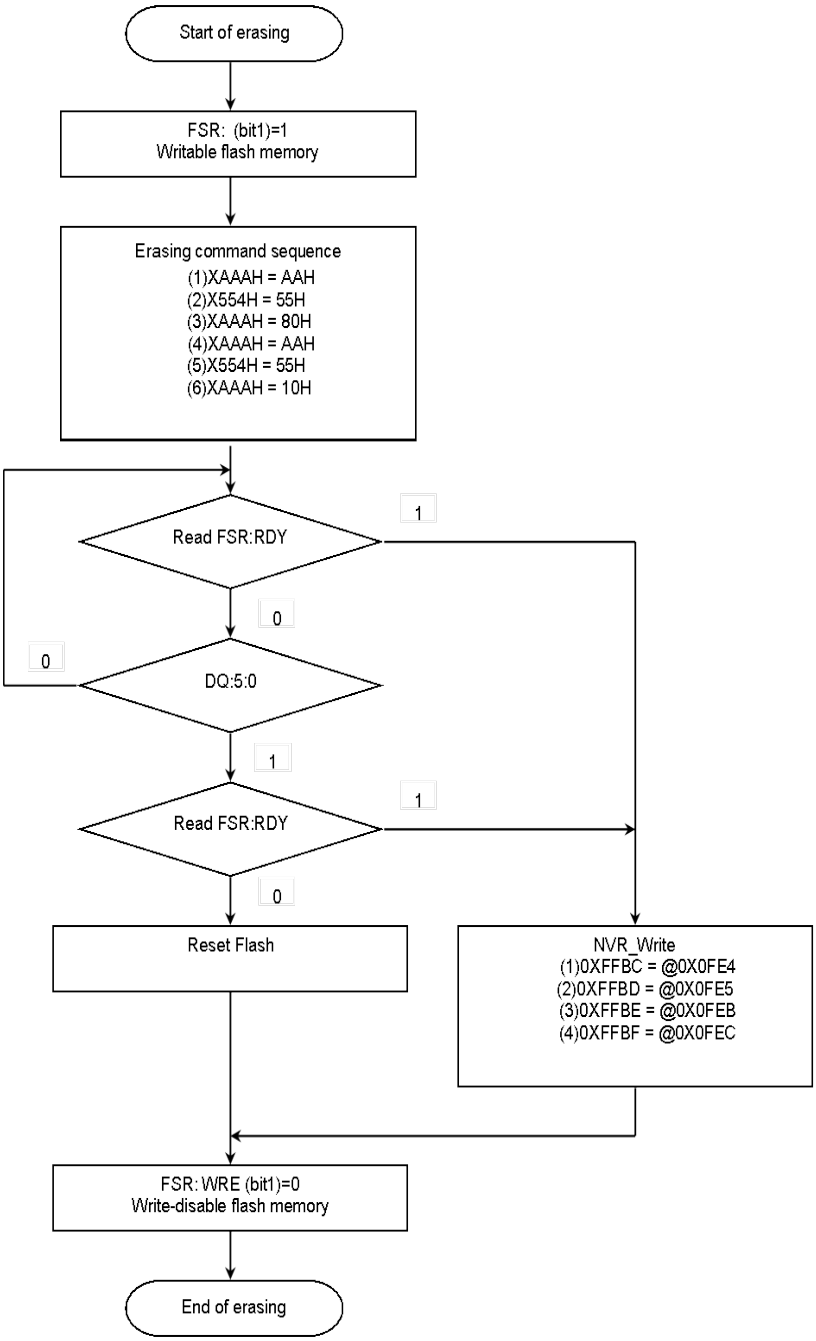
Then the program jump to RAM where RAMCODE is located and RAMCODE is executed within RAM.

3.2 Flashing Routines

The following assembly source code contains the routines to erase and program the flash memory. The following flow chart shows the algorithm of the routines.

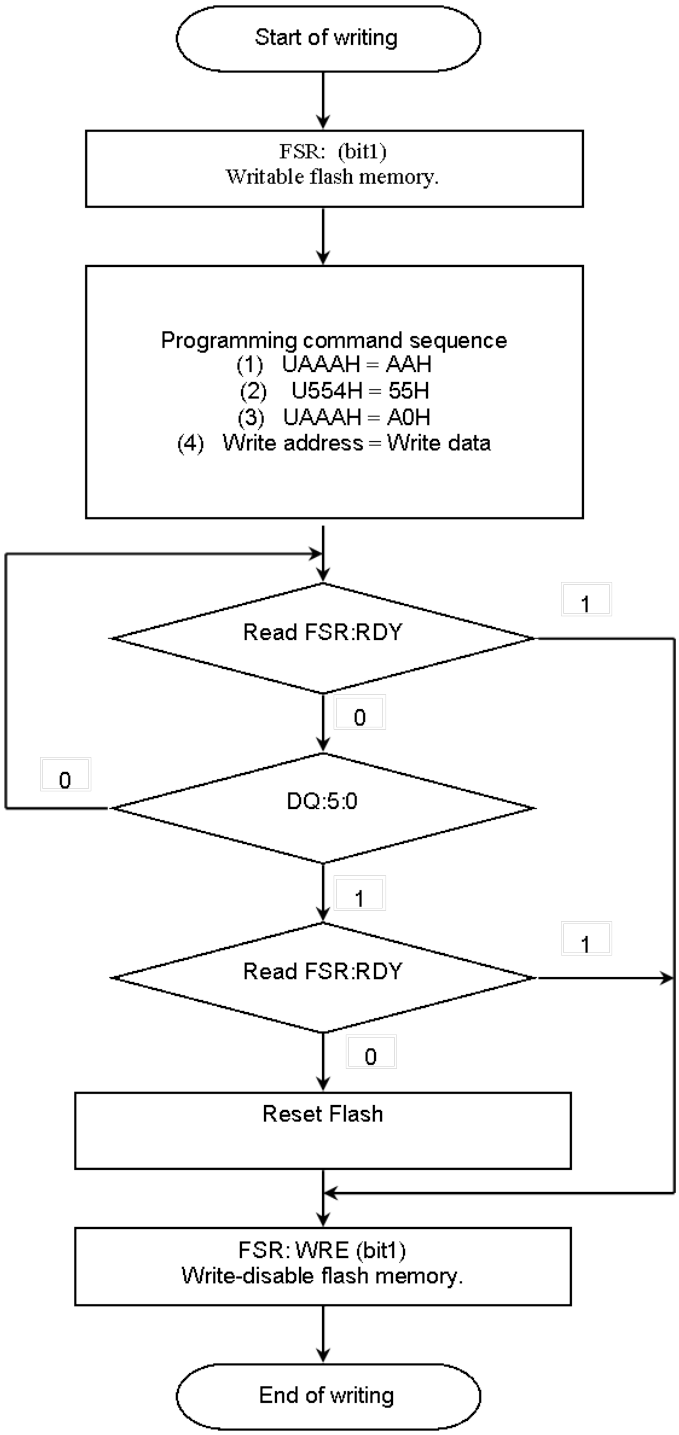
3.2.1 Flash Erase Routine Flow Chart

Figure 9. Flash Erase Routine Flow Chart



3.2.2 Flash Write Routine Flow Chart

Figure 10. Flash Write Routine Flow Chart



3.2.3 Flash.asm

Flash.asm is an assembly code to implement Flash write/erase.

Some key points are explained below. For complete Code, refer to Sector 5.1 Sample Code in Chapter 6 Appendix.

1. _EraseStart is the start address of Flash Erase routine in Flash.asm. _WriteStart is the start address of Flash Write routine in Flash.asm. They're exported so that they can be called by instructions in main ().

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

2. Erase address is saved from EP, which is assigned in main ().

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

Write address is saved from EP, which is assigned in main ().Write data is saved from IX, which is assigned in main ().

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

3. According to erasing/writing Flash automatic algorithm, the U in UAAAH and U5554H is the upper 4bits same as write address. The code below is to calculate the U and get correct UAAAH and U5554H.

```
      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)
```

4. Write Flash enable by setting bit WRE in register FSR.

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

5. Erase Flash memory automatic algorithm.

```
       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
        MOVW   EP,A
        MOV    A,#10H  ; The last data.
        MOV    @EP,A          ; Start Erase
```

Write Flash memory automatic algorithm.

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

6. Erase Loop

```
        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?
```

Write 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?
```

Why do we need add a NOPx2 before checking of RDY bit before Erase Loop and Write Loop, please refer to RDY bit in 20.3.1 in Chapter 20 of the MB95200H/210H Series hardware manual.

7. Erase Flash fails, Reset Flash and Set error Flag in A.

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

Write Flash fails, Reset Flash and Set error Flag in A.

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

8. Write Flash disable by clear bit WRE in register FSR.

```
      CLRB    FSR:WRE       ; write disable
```

9. If erase Flash successes, Set success Flag in A.

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

If write Flash successes, Set success Flag in A.

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

If the flash is successfully erased, the flash memory should be blank, and NVR data can be programmed to the flash memory. For details, please refer to Sector 4.1 Reprogram the NVR after Erasing in Chapter 4 Notes on Flash Operation.

```
NVR_Write:

       MOVW   EP,#0xFFBC
       MOV    A,0x0FE4
       MOVW   IX,A
       CALL   _WriteStart
       MOVW   EP,#0xFFBD
       MOV    A,0x0FE5
       MOVW   IX,A
       CALL   _WriteStart
       MOVW   EP,#0xFFBE
       MOV    A,0x0FEB
       MOVW   IX,A
       CALL   _WriteStart
       MOVW   EP,#0xFFBF
       MOV    A,0x0FEC
       MOVW   IX,A
       CALL   _WriteStart
       RET
```

3.3 Sub Functions

The next level in the hierarchy is the flash erase and flash write functions, which transfers flash operation address and data to the assembly flash routine in the RAM area and then calls them. For complete Code, refer to Sector 6.2 Sample code in Chapter 6 APPENDIX.

3.3.1 Flash Erase C Code

Define global variables.

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

According to assembly and c language compile rules, the variable "_address" in assembly language is the same as variable "address" in c language. The code below is to transfer erase address assigned in main () to EP, which will be used in flash.asm.

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

The assembly Flash Erase routine in the RAM area is called by

```
          CALL _EraseStart
```

_EraseStart is the start address of Flash Erase routine in Flash.asm. Then the program jumps to RAM, and will run in RAM.

3.3.2 Flash Write C Code

The code below is to transfer write address assigned in main () to EP, and 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 Flash write routine in the RAM area is called by

```
      CALL _WriteStart
```

_WriteStart is the start address of Flash write routine in Flash.asm. Then the program jumps to RAM, and will run in RAM.

3.4 How to Use the Programming Functions

Below is a short demonstration C main code, which shows how to use the flash programming functions. For complete Code, refer to Sector 6.2 Sample code in Chapter 6 APPENDIX.

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 so that they can be called by instructions in main (). According to assembly and c language compile rules, "_EraseStart" in assembly language is the same as "EraseStart" in c language, and "_WriteStart" is the same as "WriteStart".

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

The user can add program here, according to the result of flash operation.

```
void error(void)
{
                    // do something here
}
void success(void)
{
                    // do something here
}
```

The write address "0xF800" and data "0xA0" are transferred to Flash.asm by global variables address and data. And the write address "0xF800" and data "0xA0" could be assigned by user.

```
 address = 0xF800;//write address
 data = 0xA0;     //write data
```

The flash_write () is called, and result of write flash is return to Flag.

```
Flag = flash_write();
```

Flag ==1, write flash fails, do something in error (). Flag == 0, write flash successes, do something in success ().

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

4 Notes on Flash Operation

This chapter provides notes on flash operation.

4.1 Reprogram the NVR after Erasing

1. A flash erase operation will erase all NVR data.

The flash writer carries out the following procedure to keep original system settings.

- Make a backup of data in CRTH: CRTH4-CRTH0 and CRTL: CRTL4-CRTL0.
- Erase the flash.
- Restore all data in CRTH: CRTH4-CRTH0 and CRTL: CRTL4-CRTL0 to the NVR flash area.

If there is new data in CRTH: CRTH4-CRTH0 and CRTL:CRTL4-CRTL0, the flash writer will program the new data to the NVR flash area.

2. Programming NVR data is the same as programming bytes to flash. For details please refer to Chapter 20 and 23 of the MB95200H/210H Series hardware manual.

4.2 High Voltage Supply on RST PIN

Apply a high DC voltage (+10V) to the RST pin while writing data to or erasing all data from the flash memory. After applying the high voltage, wait for 10ms before writing data to or erasing all data from the flash memory. Keeps the voltage at the RST pin until data writing or erasing is completed.

5 Appendix

5.1 Sample Code

5.1.1 Project

Name: Flash operation
Function: demonstrates the flash operation.

main.c

```
/*                              SAMPLE CODE                                    */
/*****************************************************************************
Main.C:   flash.asm located in RAM.
               See README.TXT for project description and disclaimer.
*****************************************************************************/
#include "mb95200.h"

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

extern EraseStart;
extern WriteStart;

/*****************************************************************************
Name:         error ();
Function:     flash operation error handle process
*****************************************************************************/
void error(void)
{
                                  // do something here
}

/*****************************************************************************
Name:         success ();
Function:     flash operation success handle process
*****************************************************************************/
void success(void)
{
                                  // do something here
}
/*****************************************************************************
Name:         flash_erase ();
Function:     flash erase operation entrance
*****************************************************************************/
unsigned char flash_erase(void)
{
      #pragma asm
            MOVW A, _address    // set erase address
            MOVW EP,A
            CALL   _EraseStart  // Start erase subrountin
            MOV     _result,A   // Reture the result for flash erase
      #pragma endasm

      return result;
}
```

```
/*****************************************************************************
Name:         flash_write ();
Function:     flash write operation entrance
*****************************************************************************/
unsigned char flash_write(void)
{
      #pragma asm
            MOV  A,_data        // set write data
            MOVW IX,A
            MOVW A, _address    // set write address
            MOVW EP,A
            CALL _WriteStart    // Start Flash write subroutine
            MOV     _result,A   // Reture the result for flash write
      #pragma endasm

      return result;
}
/*****************************************************************************
Name:         main ();
Function:     main loop, a sample for flash write function usage
*****************************************************************************/
void main (void)
{
// Please follow procedure below to write flash
// During writing data or erasing all data in flash memory,
// a typical +10VDC voltage should be applied at the RST pin.
// After applying the high voltage,
// wait for 10ms before writing data or erasing all data in flash memory.
// And this applied voltage should be kept at the RST pin until data writing or
// erasing has completed.
      address = 0xF800;          // set write address
      data = 0xA0;               // set write data
      Flag = flash_write();            // flash write routine

      if (Flag == 1)
            error();
      else
            success();
// Please follow procedure below to erase flash
// (1) Copy out all NVR contents from register: 0x0FE4, 0x0FE5, 0X0FEB, 0x0FEC.
// (2) Erase flash
// (3) Restore all NVR contents to flash area: 0xFFBC, 0xFFBD, 0XFFBE, 0xFFBF.
// (1) Copy out,add your routine
// (2) Erase flash
//     address = 0xF800;
//     Flag = flash_erase();
//     if (Flag == 1)
//           error();
//     else
//           success();
// (3) Restore, add your routine
// In this demonstration, restore is implemented in _EraseStart, if flash erase
// successes.And because flash has been erased, the routine can't return to
// main routine which is located in Flash Memory. So, all NVR contents have to be
// restored to flash area  in _EraseStart.
// NOTE: when flash_erase() routine is used, pay more attention please!
}
```

flash.asm

```
;                               SAMPLE CODE
;-------------------------------------------------------------------------
;  flash.asm
;  - description and disclaimer see readme.txt
;  flash.asm is a code module to demonstrate how to use Flash memory write/erase.
;  flash.asm is located in RAM area and all functions within flash.asm are also
;  located in RAM.
;  flash.asm is called by main(), which is located in Flash Memory.

;  A is used to indicate the status of the programmimg:
;  A  : Reture Flag
;                          0 : Successed
;                          1 : Fail
;  ---------------------------------------------------------------------
        .SECTION RAMCODE, CODE
       .EXPORT _EraseStart
       .EXPORT _WriteStart
; FlashROM control register
WRE    .equ   1            ; Flash enable flag
RDY    .equ   4            ; Flash finish flag
FSR    .equ   0x0072 ; Flash control register
; EP   :      Operation Address
; IX   :      Operation Data

;++++++++++++++++++++++++++++++++++++++
; Command_Process
;  Flash Erase
;      IN     :      EP : Erase Sector Address
;      OUT    :      A  : Return Flag
;                          0 : Successed
;                          1 : Fail
;++++++++++++++++++++++++++++++++++++++
_EraseStart:
       MOVW         A,EP
       PUSHW  A                         ; Save write address

       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)

EraseGo:

       SETB         FSR:WRE                     ; Write Enable

       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
        MOVW        EP,A
        MOV         A,#10H                  ; The last data.
        mov         @EP,A                   ; Start Erase

        NOP
        NOP

EraseLoop:
        BBS         FSR:RDY,EraseEnd
        MOV         A,@EP
        AND         A,#0x20                 ; Check Time Out?
        BZ          EraseLoop
        BBS         FSR:RDY,EraseEnd
        NOP
        BBS         FSR:RDY,EraseEnd
EraseError:
        MOV         A,#0xF0
        MOV         0xFF00,A                ; Reset Flash
        MOV         A,#01H                  ; Set error Flag
        CLRB        FSR:WRE                 ; write disable
        RET

EraseEnd:
        MOV         A,#00H                  ; normal ack
        CLRB        FSR:WRE                 ; write disable
;In this demonstration, restore is implemented in _EraseStart, if flash erase
;successes. Because flash has been erased, the routine can't return to main
;routine which is located in Flash Memory. So, all NVR contents have to be
;restored to flash area here.
NVR_Write:
        MOVW    EP,#0xFFBC
        MOV     A,0x0FE4
        MOVW    IX,A
        CALL    _WriteStart

        MOVW    EP,#0xFFBD
        MOV     A,0x0FE5
        MOVW    IX,A
        CALL    _WriteStart

        MOVW    EP,#0xFFBE
        MOV     A,0x0FEB
        MOVW    IX,A
        CALL    _WriteStart

        MOVW    EP,#0xFFBF
        MOV     -A,0x0FEC
        MOVW    IX,A
        CALL    _WriteStart
        RET
```

```
;=====================================================================
;+++++++++++++++++++++++++++++++++++++
; Command_Process
;  Flash Write
;      IN     :      EP : Write Address
;                    IX : Write Data
;      OUT    :      A  : Return Flag
;                          0 : Successed
;                          1 : Fail
;+++++++++++++++++++++++++++++++++++++
_WriteStart:
       PUSHW  IX                       ; Save write data
       MOVW          A,EP
       PUSHW  A                        ; Save write address

       SETB          FSR:WRE                   ; Write Enable
       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)
       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
       POPW          IX
       MOVW          A,IX                      ; write data
       MOV           @EP,A                     ; to write flash
       NOP
       NOP
WriteLoop:
       BBS           FSR:RDY,WriteEnd

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

       BBS           FSR:RDY,WriteEnd
       NOP
       BBS           FSR:RDY,WriteEnd
WriteError:
       MOV           A,#0xF0
       MOV           0xFF00,A                  ; Reset Flash
       MOV           A,#01H                    ; Set error Flag
       CLRB          FSR:WRE                   ; write disable
       RET
WriteEnd:
       MOV           A,#00H                    ; Set normal Flag
       CLRB          FSR:WRE                   ; write disable
       RET
;=====================================================================
```

Startup.asm

```
;  Sample program for initialization
;------------------------------------------------------------------------
             .PROGRAM     start
             .TITLE       start

;------------------------------------------------------------------------
; variable define declaration
;------------------------------------------------------------------------
 #define HWD_DISABLE                    ; if define this, Hard Watchdog will disable.
;------------------------------------------------------------------------
; external declaration of symbols
;------------------------------------------------------------------------
             .EXPORT             __start
             .IMPORT             _main
             .IMPORT             LMEMTOMEM
             .IMPORT             LMEMCLEAR
             .IMPORT             _RAM_INIT
             .IMPORT             _ROM_INIT
             .IMPORT             _RAM_DIRINIT
             .IMPORT             _ROM_DIRINIT
                        .IMPORT                          _RAM_RAMCODE
             .IMPORT             _ROM_RAMCODE
;------------------------------------------------------------------------
; definition to stack area
;------------------------------------------------------------------------
             .SECTION     STACK,    STACK,    ALIGN=1
             .RES.B       128-2
STACK_TOP:
             .RES.B       2

;------------------------------------------------------------------------
; definition to start address of data, const and code section
;------------------------------------------------------------------------
             .SECTION     RAMCODE, CODE, ALIGN=1
             .SECTION     DIRDATA,    DIR,     ALIGN=1
             .SECTION     DIRINIT,         DIR,     ALIGN=1
             .SECTION     DATA,            DATA, ALIGN=1
             .SECTION     INIT,           DATA, ALIGN=1

;------------------------------------------------------------------------
; code area
;------------------------------------------------------------------------
             .SECTION     CODE,     CODE,     ALIGN=1
__start:
;------------------------------------------------------------------------
; set stack pointer
;------------------------------------------------------------------------
             MOVW   A, #STACK_TOP
             MOVW   SP, A
;
```

```
;-------------------------------------------------------------------------
; set register bank is 0
;-------------------------------------------------------------------------
             MOVW   A, PS
             MOVW   A, #0x07FF
             ANDW   A
             MOVW   PS, A
;-------------------------------------------------------------------------
; set ILM to the lowest level(3)
;-------------------------------------------------------------------------
             MOVW   A, PS
             MOVW   A, #0x0030
             ORW    A
             MOVW   PS, A
;-------------------------------------------------------------------------
; *copy initial value *CONST(ROM) section to *INIT(RAM) section
;-------------------------------------------------------------------------
#macro       ICOPY  src_addr, dest_addr, src_section
             MOVW   EP, #\src_addr
             MOVW   A,  #\dest_addr
             MOVW   A,  #SIZEOF (\src_section)
             CALL   LMEMTOMEM
#endm

             ICOPY  _ROM_INIT,                _RAM_INIT,                INIT
             ICOPY  _ROM_DIRINIT, _RAM_DIRINIT, DIRINIT
             ICOPY  _ROM_RAMCODE, _RAM_RAMCODE, RAMCODE
;-------------------------------------------------------------------------
; zero clear of *VAR section
;-------------------------------------------------------------------------
#macro       FILL0  src_section
             MOVW   A, #\src_section
             MOVW   A, #SIZEOF (\src_section)
             CALL   LMEMCLEAR
#endm

             FILL0  DIRDATA
             FILL0  DATA
;-------------------------------------------------------------------------
; call main routine
;-------------------------------------------------------------------------
             CALL   _main
end:         JMP    end
;-------------------------------------------------------------------------
; Hard Watchdog
;-------------------------------------------------------------------------
#ifdef HWD_DISABLE
             .SECTION  WDT, CONST, LOCATE=H'FFBE
             .DATA.W   0xA596
#endif
;-------------------------------------------------------------------------
; reset vector
;-------------------------------------------------------------------------
             .SECTION     RESET,    CONST,    LOCATE=0xFFFC
             .DATA.B      0xFF
             .DATA.B      0
             .DATA.H      __start

             .END   __start
```

6 Additional Information

For more information on Cypress Microcontrollers Products, please visit the following websites:

http://www.cypress.com/cypress-microcontrollers

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205384 – F²MC - 8FX Family, MB95200H/210H Series, Flash Operation

Document Number: 002-05384

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	Jacky, Zhou	20/03/2008	V1.0, First Draft
			21/07/2008	V1.1, Add Explanation of ICOPY command in 3.1.4 Add Explanation of why need NOPx2 command in 3.2.3
			02/11/2009	V1.2, correct an error about NVR_Write in page 19, 29
*A	5264357	HUAL	05/09/2016	Migrated Spansion Application Note “MCU-AN-500015-E-12” 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

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

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

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

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