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.

AN205210

FR Family MB91460 Series Flash Write/Erase via CPU

This application note is to collect the steps to make sure that FLASH ROM is correctly handled while during write/erase procedures executed by the CPU.

Contents

1 Introduction....1
2 Basic Steps for FLASH write/erase....1
2.1 Overview....1
2.2 List of Steps....1
A Appendix....7
A.1 Complete Sample Code for FLASH memory erase/write....7
3 Document History....15
Worldwide Sales and Design Support....16
Products....16
PSoC® Solutions....16
Cypress Developer Community....16
Technical Support....16

1 Introduction

The Target of this Application Note is to collect the steps to make sure that FLASH ROM memory (called only FLASH in the remainder of this document) is correctly handled while during write/erase procedures executed by the CPU (e.g. from within an application).

If you are searching for information about FLASH write/erase Tools (for example serial FLASH programming using FME-FR Flash Programmer or using parallel FLASH programming via Galep Parallel Programmer), please refer to the application note AN205153 - FR Family MB91460 Series, Flash Programming.

The scope of this application is:

Using FLASH Auto Algorithms from within an application.

Handling the FLASH Interface of MB91460 Series which supports features like:

64Bit read-only access, 32Bit read/write and 16Bit write modes.

Selection of Wait-Cycles corresponding to the selected CPU speed.

Handing Interrupt requests during FLASH write/erase procedures.

2 Basic Steps for FLASH write/erase

Here the basic steps are listed which have to be met for FLASH write/erase.

2.1 Overview

For the FLASH memory interface on MB91460 Series MCUs wait cycles are adjustable to reach optimum performance required for fast code execution and in the same time assuring safe FLASH memory access depending on CPU and Bus speed. Additionally Prefetch, Instruction Cache and 64Bit FLASH memory access can be supported by MB91460 Series MCUs. Therefore this FLASH memory interface must be handled correctly to assure the trouble-free write/erase access from within an application.

MB91460 Series MCUs only support single bank FLASH memory. For this reason code execution out of the FLASH memory is not possible during write/erase access. This has implications on the placement of the FLASH write/erase routines in the application code and also on the way how interrupts are served during write/erase access.

2.2 List of Steps

- Make sure all the routines necessary to handle FLASH write/erase are placed in RAM memory.
- Decide how to handle interrupts by CPU during the time of write/erase access to FLASH.

- Prepare FLASH Interface of the CPU to be in 32Bit read/write Mode or 16Bit write Mode.
- Start corresponding Auto Algorithm of embedded FLASH macro for write/erase of FLASH memory. The progress of Auto Algorithm for FLASH write/erase can be supervised via Hardware Sequence Flags provided by the embedded FLASH macro. After finishing the Auto Algorithm return the embedded FLASH macro to read mode again (this is done automatically by embedded FLASH macro when write/erase was successful; in case of error while write/erase this has to be done by CPU).
- Prepare CPU FLASH interface for 64Bit read or 32Bit read/write mode again if necessary.

2.2.1 Transfer the FLASH programming routines from FLASH memory to RAM

This is for simplicity done in the Start91460.asm (you may find Start91460.asm in the template project for your MB91460 Series FLASH derivate).

The module FLASH.c containing all necessary routines for handling FLASH write/erase has to be placed in the RAMCODE section. This can be done by adding

```
#pragma section CODE=RAMCODE,attr=CODE
```

to your FLASH.c module.

In the Linker settings of your project there have to be two sections RAMCODE (placed in the RAM of the MCU) and @RAMCODE (placed in the FLASH ROM of the MCU).

In the Start91460.asm you will find a part which is copying the code placed in the @RAMCODE section the RAMCODE section at startup. For Details on this please refer to the Manuals of Softune FR Workbench Linker and to the Softune Workbench Template Project corresponding to the used MB91460 Series FLASH MCU.

2.2.2 Handling Interrupts

Handling of Interrupts by the CPU is normally done via Vector Table (fetch of address for Interrupt Service Routines) which is placed inside FLASH memory. And of cause code fetch of the ISR is done form the FLASH memory.

Since no fetching of data or code form the FLASH memory is possible while write/erase of FLASH is possible handling of interrupts can be disabled before write/erase operation on the FLASH memory and be re-enabled afterwards. Somewhat more complicated but also possible is to setup an interrupt vector table and the corresponding interrupt service routines in the RAM of the MCU. In this case the interrupt vector table base register has to set to the RAM interrupt vector table for the time the FLASH memory is write/erase accessed.

In the following we chose to disable interrupt handling by the CPU by clearing the interrupt enable flag in the CPU Control Register (CCR). For details on the CPU Control Register please refer to the FR Programming Manual.

Since one would like to restore the original setting of the CPU enable interrupt flag in the CCR the before clearing the flag the status of this flag has to saved. For this reason one is not able to use the Cypress C-Language macros __EI() (set CCR interrupt enable flag) and __DI() (clear CCR interrupt enable flag).

Please find in **Box 1** two simple assembler routines included via `#pragma asm/endasm` in the FLASH.c module which handling Save Disable Global Interrupt and Restore Global Interrupt.

```
FLASH.h:

unsigned int FLASH_SaveDisableInterruptFlag(void);
void FLASH_RestoreInterruptFlag(unsigned int flag);


FLASH.c:

static unsigned int IFlag;

#pragma asm
_FLASH_SaveDisableInterruptFlag:
    STM0 (R0)
    MOV PS,R4
    LDI #0x00000010,R0
    AND R0,R4         ; Store Original Flag
    ANDCCR #0xFFFFFFEF      ; Clear Interrupt Flag
    LDM0 (R0)
    RET
#pragma endasm

#pragma asm
_FLASH_RestoreInterruptFlag:
    STM0 (R0)
    MOV PS,R0         ; Get current PS
    OR R4,R0          ; Set Flag as saved
    MOV R0,PS         ; Write back PS
    LDM0 (R0)
    RET
#pragma endasm
```

Box 1: Routines for Save Disable and Restore of global Interrupt Enable Flag.

2.2.3 Prepare FLASH Interface for Write-Mode

The FLASH Interface of the MB91460 supports different access modes, but only in the 32Bit read/write mode and in the 16Bit read/write mode one can write to the FLASH memory. It is recommended to use routines provided by the Boot-ROM of the MB91460 series MCUs to safely switch between the different access modes of the MB91460 series MCUs (if one prefers to do the switching manually one has to keep in mind that during the access mode switching no code can be executed from FLASH memory).

For the details of the FLASH memory access mode switching please refer to the Hardware Manual of the MB91460 series Chapter 53 “Fixed Mode-Reset Vector / BOOT-ROM” Point 4 “Flash Access Mode Switching”.

Writing/erasing the FLASH memory of the MB91460 series MCUs also requires an access timing (wait cycles) which is different from timing used for normal read/execute asscess.. For details on the needed read and write access timings depending on the used CPU speed please refer to the Hardware Manual for the MB91460 Series Chapter 11 “Memory Controller” Point 9 “FLASH access timing settings”.

In the following you will find in **Box 2** an example for the implementation of calls for the access mode switching routines from C-Code via `#pragma asm/endasm` including assembler code into C-Code. Also included in this sample code is the setting of the approporiate FLASH memory access timing for write access.

```
FLASH.h:

void FLASH_PrepareWriteWordMode();


FLASH.c:

void FLASH_PrepareWriteWordMode()
{
    /* Set FLASH Access Mode via BootROM Routine      */
    /* For details refer to the Hardware Manual or Data Sheet */
#pragma asm
    STM0 (R4,R5)
    STM1 (R12)
    LDI #0x00,R4     ; Set FLASH to 32Bit read/write Mode
    LDI #0x04,R5     ; Go 4 times through delay loop (64MHz CLKB)
    LDI #0xBF60,R12
    CALL @R12
    LDM1 (R12)
    LDM0 (R4,R5)
#pragma endasm

    /* Set the FLASH Interface to Write Timing */
    /* For details refer to the Hardware Manual or Data Sheet */
    /* Setting shown here is for CLKB = 64MHz  */
    FMWT_ATD = 1;
    FMWT_WEXH = 0;
    FMWT_WTC = 8;
}
```

Box 2: Sample Code for using the BOOT-ROM Access Mode Switching routines and setting appropriate access timings for FLASH memory write.

2.2.4 FLASH Auto-Algorithms and Hardware Sequence Flags

The MB91460 series MCUs support six commands for read/write/erase of FLASH memory:

1. Read/Reset – Recover the FLASH Read Mode after time out of previous write/erase command.
2. Write – Write data to giveen address of FLASH memory.
3. Chip erase – Erase complete FLASH memory.
4. Sector erase – Erase only one or a number of given FLASH memory sectors. For details of the arrangement of the FLASH memory sectors of a give MCU of the MB91460 series please refer to the Hardware Manual of the MB91460 series and to the data sheet of the corresponding MCU.
5. Erase suspend – Suspend currently running sector erase command to be able to read from FLASH memory sector which are not erased by the current sector erase command.
6. Erase resume – Resume a sector erase command which was previously suspended.

The commands are started by writing given sequences to the FLASH memory when FLASH memory write mode is enabled.

Once a FLASH memory command was successfully started the current status of the FLASH memory executing the command can be supervised using Hardware Sequence Flags which can be accessed by reading from the FLASH memory during the execution of the FLASH memory command.

For details on the usage of the FLASH memory Auto-Algorithms (commands) and of Hardware Sequence Flags please refer to the Hardware Manual of the MB91460 series Chapter 54 “FLASH memory” Point 6 “Auto Algorithms”.

In **Box 3** and **Box 4** you will find an example for the usage of the Chip Erase command sequence to erase the complete FLASH memory the Chip erase command sequence. In the Appendix of this document you will in addition find sample code for the other FLASH memory commands (supporting in a simple way the handling of Interrupts during Sector erase command).

```
FLASH.h:

#define wseq_1 ((volatile unsigned int *)0x00041557)
#define wseq_2 ((volatile unsigned int *)0x00040AAF)

#define DPOLL 0x0080
#define TLOVER 0x0020
#define SETIMR 0x0008

unsigned char FLASH_ChipErase(void);


FLASH.c:

unsigned char FLASH_ChipErase()
{
    unsigned char flag = 0;

    /* Disable Interrupts if necessary  */
    IFlag = FLASH_SaveDisableInterruptFlag();

    /* Set FLASH access mode to 32Bit Write Mode      */
    FLASH_PrepareWriteWordMode();

    /* Start FLASH Chip Erase Sequence  */
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x0080;
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x0010;

Will be continued in Box 4 ...
```

Box 3: Sample Code showing the usage of Auto Algorithm Chip Erase Command and Hardware Sequence Flags.

```
Continued from Box 3 ... FLASH_ChipErase()

    /* Wait for the Auto Algorithm to finish   */
    while( flag == 0 )
    {
        /* Feed Hardware Watchdog */
        HWWD_CL = 0;
        if( ( *wseq_1 & DPOLL ) )
        {
flag = 1;
        }
        if( ( *wseq_1 & TLOVER ) )
        {
            if( ( *wseq_1 & DPOLL ) )
            {
flag = 1;
            }
            else
            {
                /* Reset FLASH      */
                FLASH_ReadReset();
                flag = 2;
            }
        }
    }

    /* Set FLASH access mode to 32Bit Read Mode       */
    FLASH_PrepareReadMode();

    /* Restore the original Interrupt Flag     */
    FLASH_RestoreInterruptFlag(IFlag);

    return flag;
}
```

Box 4: Sample Code for usage of Auto Algorithm Chip Erase Command and Hardware Sequence Flags (continued).

2.2.5 FLASH Interface Read Mode

After successful write/erase of the FLASH memory the FLASH memory Interface has be set for the needed Read Access Mode to again allow code execution from the FLASH memory. Here it is again recommended to use the routines provided by the BOOT-ROM for safe FLASH Access Mode Switching.

In addition the FLASH access timing can be set back to the values allowed for a given CPU speed. Please see the information under Point 2.2.3 “Prepare FLASH Interface for Write-Mode” of this document and refer to the Hardware Manual of the MB91460 series for details on the allowed FLASH access timings for a given CPU speed.

A Appendix

A.1 Complete Sample Code for FLASH memory erase/write

In the following you find a complete sample code example necessary for gaining experience in the application of FLASH memory write/erase commands to manipulate the FLASH memory content from within your own MCU application.

To include this code into your test application you have to make configure the SWB to place the code in FLASH but linked for execution form RAM. Make sure that the RAMCODE copy option is enabled in the Start91460.asm file.

You may also find a FLASH programming sample project for the Softune Workbench on the web-site www.fme.gsdc.de\gsdc.htm. The name of the FLASH programming sample is 91460_flash_programming_demo_mb91467d.

Note: Please keep in mind that this sample code software is for test purposes only and warranty will be given.

FLASH.h:

```
#ifndef __FLASH_H__
#define __FLASH_H__

#include "MB91467D.H"

#define hseq_1 ((volatile unsigned short int *)0x00041557)
#define hseq_2 ((volatile unsigned short int *)0x00040AAF)
#define wseq_1 ((volatile unsigned int *)0x00041557)
#define wseq_2 ((volatile unsigned int *)0x00040AAF)


#define DPOLL 0x0080
#define TLOVER 0x0020
#define SETIMR 0x0008

void FLASH_PrepareWriteWordMode();
void FLASH_PrepareWriteHalfWordMode();
void FLASH_PrepareReadMode();
unsigned char FLASH_WriteHalfWord(unsigned int adr, unsigned short int data);
unsigned char FLASH_WriteWord(unsigned int adr, unsigned int data);
unsigned char FLASH_SectorErase(unsigned int sec_adr);
unsigned char FLASH_ChipErase(void);
unsigned char FLASH_SectorBlankCheck(unsigned int secaddr, unsigned int size);
unsigned char FLASH_ReadReset(void);
unsigned char FLASH_SuspendSectorErase(unsigned int secaddr);
unsigned char FLASH_ResumeSectorErase(unsigned int secaddr);
unsigned int FLASH_SaveDisableInterruptFlag(void);
void FLASH_RestoreInterruptFlag(unsigned int flag);
unsigned char FLASH_CheckPendingInterrupt(void);

#endif /* __FLASH_H__       */
```

FLASH.c

```
#include "Flash.h"

static unsigned int IFlag;

#pragma section CODE=RAMCODE,attr=CODE

void FLASH_PrepareWriteWordMode()
{
    /* Set FLASH Access Mode via BootROM Routine */
    /* For details refer to the Hardware Manual or Data Sheet */
#pragma asm
    STM0 (R4,R5)
    STM1 (R12)
    LDI #0x00,R4     ; Set FLASH to 32Bit read/write Mode
    LDI #0x04,R5     ; Go 4 times through delay loop (64MHz CLKB)
    LDI #0xBF60,R12
    CALL @R12
    LDM1 (R12)
    LDM0 (R4,R5)
#pragma endasm

    /* Set the FLASH Interface to Write Timing */
    /* For details refer to the Hardware Manual or Data Sheet */
    /* Setting shown here is for CLKB = 64MHz */
    FMWT_ATD = 1;
    FMWT_WEXH = 0;
    FMWT_WTC = 8;
}

void FLASH_PrepareWriteHalfWordMode()
{
    /* Set FLASH Access Mode via BootROM Routine */
    /* For details refer to the Hardware Manual or Data Sheet */
#pragma asm
    STM0 (R4,R5)
    STM1 (R12)
    LDI #0x01,R4     ; Set FLASH to 32Bit read/write Mode
    LDI #0x04,R5     ; Go 4 times through delay loop (64MHz CLKB)
    LDI #0xBF60,R12
    CALL @R12
    LDM1 (R12)
    LDM0 (R4,R5)
#pragma endasm

    /* Set the FLASH Interface to Write Timing */
    /* For details refer to the Hardware Manual or Data Sheet */
    /* Setting shown here is for CLKB = 64MHz */
    FMWT_ATD = 1;
    FMWT_WEXH = 0;
    FMWT_WTC = 8;
}

void FLASH_PrepareReadMode()
{
    /* Set FLASH Access Mode via BootROM Routine */
    /* For details refer to the Hardware Manual or Data Sheet */
#pragma asm
    STM0 (R4,R5)
    STM1 (R12)
```

```
    LDI #0x00,R4     ; Set FLASH to 32Bit read/write Mode
    LDI #0x04,R5     ; Go 4 times through delay loop (64MHz CLKB)
    LDI #0xBF60,R12
    CALL @R12
    LDM1 (R12)
    LDM0 (R4,R5)
#pragma endasm

    /* Set the FLASH Interface to Read Timing */
    /* For details refer to the Hardware Manual or Data Sheet */
    /* Setting shown here is for CLKB = 64MHz */
    FMWT_ATD = 1;
    FMWT_EQ = 3;
    FMWT_WTC = 4;
}

unsigned char FLASH_ChipErase()
{
    unsigned char flag = 0;

    /* Disable Interrupts if necessary */
    IFlag = FLASH_SaveDisableInterruptFlag();

    /* Set FLASH access mode to 32Bit Write Mode */
    FLASH_PrepareWriteWordMode();

    /* Start FLASH Chip Erase Sequence */
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x0080;
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x0010;

    /* Wait for the Auto Algorithm to finish */
    while( flag == 0 )
    {
        /* Feed Hardware Watchdog */
        HWWD_CL = 0;

        if( ( *wseq_1 & DPOLL ) )
        {
            flag = 1;
        }
        if( ( *wseq_1 & TLOVER ) )
        {
            if( ( *wseq_1 & DPOLL ) )
            {
                flag = 1;
            }
            else
            {
                /* Reset FLASH */
                FLASH_ReadReset();

                flag = 2;
            }
        }
    }

    /* Set FLASH access mode to 32Bit Read Mode */
```

```
    FLASH_PrepareReadMode();

    /* Restore the original Interrupt Flag */
    FLASH_RestoreInterruptFlag(IFlag);

    return flag;
}

unsigned char FLASH_SectorErase(unsigned int secadr)
{
    unsigned char flag = 0;
    volatile unsigned int value = 0;

    /* Disable Interrupts if necessary */
    IFlag = FLASH_SaveDisableInterruptFlag();

    /* Set FLASH access mode to 32Bit Write Mode */
    FLASH_PrepareWriteWordMode();

    secadr |= 0x0003;

    /* Start FLASH Sector Erase Sequence */
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x0080;
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *(unsigned  int *)secadr = 0x0030;

    /* Wait for the Auto Algorithm to start */
    while( !( *(unsigned  int *)secadr & SETIMR ) )
    {
        /* Feed the Hardware Watchdog */
        HWWD_CL = 0;

        /*  Check for Pending Interrupts  */
        if( FLASH_CheckPendingInterrupt() )
        {
            /* Wait for Sector Erase Suspend */
            FLASH_SuspendSectorErase(secadr);

            /* Restore the original Interrupt Flag */
            FLASH_RestoreInterruptFlag(IFlag);

            /* Keep on checking for pending Interrupts */
            while( FLASH_CheckPendingInterrupt() ) HWWD_CL = 0;

            /* Disable Interrupts if necessary */
            IFlag = FLASH_SaveDisableInterruptFlag();

            /* Sector Erase Resume */
            FLASH_ResumeSectorErase(secadr);
        }
    }

    /* Wait for the Auto Algorithm to finish */
    while( flag == 0 )
    {
        /* Feed Hardware Watchdog */
        HWWD_CL = 0;
```

```
        /* Check for Pending Interrupts */
        if( FLASH_CheckPendingInterrupt() )
        {
            /* Sector Erase Suspend */
            FLASH_SuspendSectorErase(secadr);

            /* Restore the original Interrupt Flag */
            FLASH_RestoreInterruptFlag(IFlag);

            /* Keep on checking for pending Interrupts */
            while( FLASH_CheckPendingInterrupt() ) HWWD_CL = 0;

            /* Disable Interrupts if necessary */
            IFlag = FLASH_SaveDisableInterruptFlag();

            /* Sector Erase Resume */
            FLASH_ResumeSectorErase(secadr);
        }

        /* Check the Hardware Sequence Flags */
        if( ( *(unsigned int *)secadr /* value */ & DPOLL ) )
        {
            flag = 1;
        }
        if( ( *(unsigned int *)secadr /* value */ & TLOVER ) )
        {
            if( ( *(unsigned int *)secadr /* value */ & DPOLL ) )
            {
                flag = 1;
            }
            else
            {
                /*     Reset FLASH   */
                FLASH_ReadReset();

                flag = 2;
            }
        }
    }

    /* Restore the original Interrupt Flag */
    FLASH_RestoreInterruptFlag(IFlag);

    /* Set FLASH access mode to 32Bit Read Mode */
    FLASH_PrepareReadMode();

    return flag;
}

unsigned char FLASH_SectorBlankCheck(unsigned int secaddr, unsigned int size)
{
    unsigned int count;
    unsigned char empty_flag = 0;
    unsigned int addr = secaddr;

    /* Clear FIXE bit to see FLASH memory content instead of fixed reset vector */
    FMCS_FIXE = 0;

    for(count = 0; count < size; count ++)
    {
        /* Clear Hardware Watchdog */
```

```
        HWWD_CL = 0;
        if( *(unsigned int *)addr != 0xFFFFFFFF ) empty_flag = 1;
        addr += 4;
    }

    /* Set FIXE bit to see fixed reset vector */
    FMCS_FIXE = 0;

    if( empty_flag != 0 )
    {
        return 2;
    }

    return 1;
}

unsigned char FLASH_WriteHalfWord(unsigned int adr, unsigned short int data)
{
    unsigned char flag = 0;

    /* Disable Interrupts if necessary */
    IFlag = FLASH_SaveDisableInterruptFlag();

    /* Set FLASH access mode to 32Bit Write Mode */
    FLASH_PrepareWriteHalfWordMode();

    /* Start Write FLASH Sequence */
    *hseq_1 = 0x00AA;
    *hseq_2 = 0x0055;
    *hseq_1 = 0x00A0;
    *((volatile unsigned short int *)adr) = data;

    /* Wait for the Auto Algorithm to finish */
    while( flag == 0 )
    {
        /* Feed Hardware Watchdog */
        HWWD_CL = 0;

        if( ( *(volatile unsigned short int *)adr & DPOLL ) == (data & DPOLL) )
        {
            flag = 1;
        }
        if( ( *(volatile unsigned short int *)adr & TLOVER ) == TLOVER )
        {
            if( ( *(volatile unsigned short int *)adr & DPOLL ) == (data & DPOLL) )
            {
                flag = 1;
            }
            else
            {
                /* Reset FLASH (keep in mind 16Bit access to FLASH) */
                *hseq_1 = 0x00F0;   // Keep in Mind (16Bit access)

                flag = 2;
            }
        }
    }

    /* Set FLASH access mode to 32Bit Read Mode */
    FLASH_PrepareReadMode();
```

```
    /* Restore the original Interrupt Flag */
    FLASH_RestoreInterruptFlag(IFlag);

    return flag;
}

unsigned char FLASH_WriteWord(unsigned int adr, unsigned int data)
{
    unsigned char flag = 0;

    /* Disable Interrupts if necessary  */
    IFlag = FLASH_SaveDisableInterruptFlag();

    /* Set FLASH access mode to 32Bit Write Mode      */
    FLASH_PrepareWriteWordMode();

    /* Start FLASH Write Sequence */
    *wseq_1 = 0x00AA;
    *wseq_2 = 0x0055;
    *wseq_1 = 0x00A0;
    *((volatile unsigned int *)adr) = data;

    /* Wait for the Auto Algorithm to finish   */
    while( flag == 0 )
    {
        /* Feed Hardware Watchdog */
        HWWD_CL = 0;

        if( ( *(volatile unsigned int *)adr & DPOLL ) == (data & DPOLL) )
        {
            flag = 1;
        }
        if( ( *(volatile unsigned int *)adr & TLOVER ) == TLOVER )
        {
            if( ( *(volatile unsigned int *)adr & DPOLL ) == (data & DPOLL) )
            {
                flag = 1;
            }
            else
            {
                /*     Reset FLASH   */
                FLASH_ReadReset();

                flag = 2;
            }
        }
    }

    /* Set FLASH access mode to 32Bit Read Mode       */
    FLASH_PrepareReadMode();

    /* Restore the original Interrupt Flag     */
    FLASH_RestoreInterruptFlag(IFlag);

    return flag;
}

unsigned char FLASH_ReadReset()
{
    *wseq_1 = 0x00F0;
```

```
    return 1;
}

#pragma asm
_FLASH_SaveDisableInterruptFlag:
    STM0 (R0)
    MOV PS,R4
    LDI #0x00000010,R0
    AND R0,R4               ; Store Original Flag
    ANDCCR #0xFFFFFFEF      ; Clear Interrupt Flag
    LDM0 (R0)
    RET
#pragma endasm

#pragma asm
_FLASH_RestoreInterruptFlag:
    STM0 (R0)
    MOV PS,R0               ; Get current PS
    OR R4,R0                ; Set Flag as saved
    MOV R0,PS               ; Write back PS
    LDM0 (R0)
    RET
#pragma endasm

unsigned char FLASH_SuspendSectorErase(unsigned int secaddr)
{
    /* Write Sector Erase Suspend Command */
    *(volatile unsigned short int *)secaddr = 0x00B0;

    /* Wait for the FLASH macro to suspend sector erase */
    while(!(*(unsigned int *)secaddr /* value */ & DPOLL) && (*(unsigned int *)secaddr
/* value */ & SETIMR))
    {
        HWWD_CL=0;
    }

    return 1;
}

unsigned char FLASH_ResumeSectorErase(unsigned int secaddr)
{
    /* Write the Sector Erase Resume Command */
    *(volatile unsigned short int *)secaddr = 0x0030;

    /* Wait for the FLASH Macro to resume sector erase */
    while((*(unsigned int *)secaddr /*value */ & DPOLL) && !(*(unsigned int *)secaddr
/*value */ & SETIMR))
    {
        HWWD_CL=0;
    }

    return 1;
}

unsigned char FLASH_CheckPendingInterrupt()
{
    /* Poll for Pending Interrupts which are needed here */

    /* and return 1 when an Interrupt is pending */
    return 0;
}
```

3 Document History

Document Title: AN205210 - FR Family MB91460 Series Flash Write/Erase via CPU

Document Number: 002-05210

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	08/25/2006	Initial release
*A	5123804	NOFL	22/03/2016	Migrated Spansion Application Note from MCU-AN-300034-E-v10 to Cypress format
*B	5872413	AESATMP9	09/04/2017	Updated logo and copyright.
*C	6054520	NOFL	02/05/2018	Updated links. Updated Sales page.

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
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/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

© Cypress Semiconductor Corporation, 2006-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.