

FR FAMILY
32-BIT MICROCONTROLLER
MB91460 SERIES

FLASH WRITE/ERASE VIA CPU

APPLICATION NOTE



Revision History

Date	Issue
2006-08-25	RSCHUM V1.0 First Version

This document contains 25 pages.

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment.**

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.
2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH's entire liability and the customer's exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH's sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.
3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.
4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH's and its suppliers' liability is restricted to intention and gross negligence.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES

To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect

Contents

REVISION HISTORY	2
WARRANTY AND DISCLAIMER	3
CONTENTS	4
1 INTRODUCTION.....	5
2 BASIC STEPS FOR FLASH WRITE/ERASE.....	6
2.1 Overview	6
2.2 List of Steps	6
2.2.1 Transfer the FLASH programming routines from FLASH memory to RAM.....	6
2.2.2 Handling Interrupts.....	7
2.2.3 Prepare FLASH Interface for Write-Mode	8
2.2.4 FLASH Auto-Algorithms and Hardware Sequence Flags	9
2.2.5 FLASH Interface Read Mode	12
3 APPENDIX.....	14
3.1 Complete Sample Code for FLASH memory erase/write.....	14

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 mcu-an-300012-e-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.
- Handling 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 course code fetch of the ISR is done from the FLASH memory.

Since no fetching of data or code from 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 be 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 be saved. For this reason one is not able to use the Fujitsu 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 handle saving, disabling, and restoring the 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 #0xFFFFFEEF     ; 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 SaveDisable 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 access.. 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 appropriate 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:

- a) Read/Reset – Recover the FLASH Read Mode after time out of previous write/erase command.
- b) Write – Write data to given address of FLASH memory.

- c) Chip erase – Erase complete FLASH memory.
- d) 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.
- e) 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.
- f) 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.

3 Appendix

3.1 Complete Sample Code for FLASH memory erase/write

In the following you find a complete sample code 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/gsd.htm. The name of the FLASH programming sample is 91460_flash_programming_demo_mb91467d.

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

FLASH.h:

```

/*****
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */
*****/

#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

```

/*****
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS. FUJITSU */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES. */
/* (C) Fujitsu Microelectronics Europe GmbH */
*****/

#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() ) HWWC_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 #0xFFFFFEEF    ; 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;
}
```

-- END --