

General Market C3F Driver for MPC56x

User's Manual

Embedded Memory Center
Technology and Manufacturing
Semiconductor Products Sector
Motorola

Change History

Version	Date	Author	Comments
0.1	2000.11.21	JiaZheng Shi	First Draft
0.2	2000.12.01	JiaZheng Shi	update
0.3	2000.12.03	JiaZheng Shi	update
0.4	2000.12.06	Chuck Kuecker	Correct return codes, add comments
0.5	2000.12.07	Rick Miller	< www.RickMiller.com >, Update drawings, add comments
1.0	2000.12.08	Rick Miller	Clarify PEGOOD results
1.1	2000.12.15	Jiazheng Shi	Update the check sum output. Update the return code value
1.2	2000.12.21	Jiazheng Shi	Add some tips and troubleshooting
1.3	2000.12.28	Jiazheng Shi	Add notes to arguments to clarify the dest, source and compare result
1.4	2001.01.07	Rick Miller	Updated comments, added IRQ cautions.
1.5	2001.01.08	Rick Miller	Add FlashInit(). Leave protect registers alone. Added table colors.
1.6	2001.01.10	Rick Miller	Add sub module functions
1.7	2001.01.21	Rick Miller	Fixed error list, etc.
1.8	2001.01.28	Rick Miller	Dest parameter is now masked to be relative. User can provide relative or absolute flash address: works either way.
1.9	2001.02.01	Rick Miller	Censor tips, added arrayBase, updated error list.
2.0	2001.02.05	Rick Miller	Updated censor error list and added censor explanations.
2.1	2001.02.11	Rick Miller	Update FlashCheckShadow() and erase error codes.
2.2	2001.02.12	Rick Miller	Update erase error codes for protect status.
2.3	2001.02.14	Rick Miller	Update arrayBase use, and ARRAY_RANGE errors.
2.4	2001.02.14	Rick Miller	Fix FlashErase() bit map for enabledSBlocks[], no block disable .
2.5	2001.02.15	Max Willis	Reformat and update User Manual Content
2.6	2001.02.15	Pamela Wolfe	Reformat and update User Manual Content
2.7	2001.02.15	Max Willis	Final Changes for 2-15-01 Release
3.0	2001.10.08	Jiazheng Shi	Update with new features, suspend/resume, interrupt supports Make it support MPC56x derivatives with C3F List affected registers by this driver Change enabledBDM definition
3.1	2001.11.15	Jiazheng Shi	Clarification the combinations of enabledSBlock[] and enabledBlock[] of FlashErase Change deviceMode definition
3.2	2001.12.06	Jiazheng Shi	Modify enabledBDM range description Modify FlashCheckShadow tip for the input parameter, dest Give description for C3F_ERROR_SRC_DEST_VERIFY
3.3	2001.12.21	Pamela Wolfe & Max Willis	Final changes for 2001-12-21 release
3.4	2002.01.03	Pamela Wolfe	Additional minor changes, primarily to API procedures.

Table of Contents

CHAPTER 1: INTRODUCTION.....	6
1.1 OVERVIEW	6
1.2 DEVELOPMENT AND TEST PROCEDURES	8
1.3 DOCUMENTATION REFERENCE	9
1.4 SYSTEM REQUIREMENTS.....	9
1.5 INSTALLATION	9
CHAPTER 2: OPERATION	10
2.1 SYSTEM AND FLASH MODULE INITIALIZATION	10
2.2 C-ARRAY DRIVER FORMAT FOR EMBEDDED APPLICATIONS	10
2.3 S-RECORD DRIVER FORMAT FOR BDM PROGRAMMING TOOLS	10
2.3.1 Relocating S-Records with SAR.EXE	11
2.3.2 Building a Custom S-Record	11
2.4 OBJECT LIBRARY DRIVER FORMAT FOR CODE COMPRESSION	12
2.5 CONCURRENCY, CALLBACK FUNCTIONS, AND FLASH SUSPEND/RESUME	12
2.5.1 Callback Template A: No Concurrency, No Read From Flash.....	13
2.5.2 Callback Template B: Polled Concurrency, No Read From Flash	13
2.5.3 Callback Template C: Polled Concurrency, Read From Flash.....	13
2.5.4 Callback Template D: Interrupted Concurrency, Read From Flash.....	14
CHAPTER 3: API SPECIFICATION	16
3.1 GENERAL OVERVIEW	16
3.2 FUNCTION RETURN VALUE ERROR CODES	16
3.3 FLASHINIT	18
3.3.1 Description.....	18
3.3.2 Procedure.....	18
3.3.3 Definition.....	19
3.3.4 Arguments	19
3.3.5 Return Values	20
3.3.6 Tips.....	20
3.3.7 Troubleshooting	21
3.3.8 Affected Register	22
3.3.9 Revision String	22
3.4 FLASHCHECKSHADOW	23
3.4.1 Description.....	23
3.4.2 Procedure.....	23
3.4.3 Definition.....	24
3.4.4 Arguments	24
3.4.5 Return Values	25
3.4.6 Tips.....	25
3.4.7 Troubleshooting	25
3.4.8 Affected Register	25
3.4.9 Revision String	25
3.5 FLASHERASE	26
3.5.1 Description.....	26
3.5.2 Procedure.....	26
3.5.3 Definition.....	27
3.5.4 Arguments	28
3.5.5 Return Values	29
3.5.6 Tips.....	29

3.5.7 Troubleshooting	30
3.5.8 Affected Register	30
3.5.9 Revision String	31
3.6 BLANKCHECK.....	32
3.6.1 Description.....	32
3.6.2 Procedure.....	32
3.6.3 Definition.....	33
3.6.4 Arguments	33
3.6.5 Return Values	34
3.6.6 Tips.....	34
3.6.7 Troubleshooting	35
3.6.8 Affected Register	35
3.6.9 Revision String	35
3.7 FLASHPROGRAM.....	36
3.7.1 Description.....	36
3.7.2 Procedure.....	36
3.7.3 Definition.....	38
3.7.4 Arguments	38
3.7.5 Return Values	39
3.7.6 Tips.....	39
3.7.7 Troubleshooting	40
3.7.8 Affected Register	40
3.7.9 Revision String	41
3.8 FLASHVERIFY	42
3.8.1 Description.....	42
3.8.2 Procedure.....	42
3.8.3 Definition.....	43
3.8.4 Arguments	44
3.8.5 Return Values	45
3.8.6 Tips.....	45
3.8.7 Troubleshooting	45
3.8.8 Affected Register	46
3.8.9 Revision String	46
3.9 CHANGECENSOR.....	47
3.9.1 Description.....	47
3.9.2 Procedure.....	47
3.9.3 Definition.....	48
3.9.4 Arguments	49
3.9.5 Return Values	50
3.9.6 Tips.....	50
3.9.7 Troubleshooting	51
3.9.8 Affected Register	51
3.9.9 Revision String	52
3.10 CHECKSUM.....	53
3.10.1 Description.....	53
3.10.2 Procedure.....	53
3.10.3 Definition.....	54
3.10.4 Arguments	54
3.10.5 Return Values	55
3.10.6 Tips.....	55
3.10.7 Troubleshooting	55
3.10.8 Affected Register	56
3.10.9 Revision String	56
APPENDIX A: PERFORMANCE DATA.....	57
A.1 CODE SIZE	57

A.2 STACK SIZE.....	58
A.3 PROGRAM / ERASE TIMES	59
A.4 CALLBACK PERIOD	60

Chapter 1: Introduction

1.1 Overview

The General Market C3F Driver for MPC565/MPC566 and MPC563/MPC564 provides the following driver functions:

- FlashInit
- FlashCheckShadow
- FlashErase
- BlankCheck
- FlashProgram
- FlashVerify
- ChangeCensor
- CheckSum

Each position-independent, ROM-able General Market Driver (GMD) function is provided as an independent binary executable so that the end user is free to choose the function subset that meets their system requirements. Because of the EABI compliant stack frame interface, each GMD function is accessed via a standard C function call. Thus no special interface code is required.

Finally, each GMD function runs on *either* the MPC565 *or* the MPC563. This feature reduces development time for flash programming tools, and it minimizes porting effort for application code that must run on both of these parts.

The following file formats are provided for the GMD function set:

- C-array: This file format can be automatically linked with the user's application code.
- S-Record: This file format can be used as the only target resident code in a BDM programming tool.
- Object Library: This file format is compiled with code compression enabled so the driver functions can be linked to the user's application prior to the compression step.

To support concurrency, each GMD function accepts a user-supplied callback function as an argument. In a polling environment where read while write capability is not required, the driver functions periodically pass control to the callback function so servicing of communication ports, watchdog timers, and other activities can proceed concurrently with flash operations.

The user-supplied callback function can also support *read while write* (RWW). That is if a concurrent activity requires reading the flash during either program or erase, the callback function can use the hardware suspend/resume feature to temporarily suspend the high voltage so reads to the flash return valid data. Callback templates and demos are provided that show how this can be done for two situations:

- A polling environment where data must be read from a flash module that is subject to high voltage program/erase operations. In this case the hardware suspend/resume feature is used to suspend high voltage long enough to perform flash reads.
- An interrupted environment where the interrupt vectors are stored in a flash module subject to high voltage program/erase operations. In this case the hardware suspend/resume is used to suspend high voltage long enough so that interrupt vectors can service pending interrupts.

So that the binary GMD components can be identified, an ASCII revision string is appended to the end of each GMD function so that the CPU platform, flash technology, GMD function name, and GMD function revision can be identified. The revision string format is as follows.

- CPU core + Flash Tech. + ff + xyz

where:

- CPU core is a three character abbreviation for the CPU platform, ie PPC for a PowerPC CPU core.
- Flash Tech is a three character abbreviation for the flash technology, ie C3F for CDR3-1T flash technology
- ff is a two character abbreviation for the GMD function, ie:
 - FI = FlashInit
 - FS = FlashCheckShadow
 - FE = FlashErase
 - FP = FlashProgram
 - BC = BlankCheck
 - FV = FlashVerify
 - CC = ChangeCensor
 - CS = CheckSum
- xyz is a three digit version number

1.2 Development and Test Procedures

Each software component is developed and tested to SEI Level 5 standards at Software Center Motorola China. The driver functions and data undergo two categories of testing:

- ***API Testing***

- There are numerous test cases for API and functional testing.

Function name	Number of test cases for MPC565	Number of test cases for MPC563
FlashInit	10	7
FlashCheckShadow	23	23
FlashErase	20	25
BlankCheck	44	35
FlashProgram	60	52
FlashVerify	95	66
Checksum	43	33
ChangeCensor	468	236
Total	758	477

- ***Operating Environment Testing***

- There are test cases to show the drivers work properly when integrated into an embedded application.
- There are test cases to show the drivers work properly when they are:
 - the only target-resident code, and
 - controlled by BDM commands issued by a host computer
- There are test cases to show the drivers work properly when executed from internal RAM.
- There are test cases to show the drivers work properly when executed from internal ROM.
- There are test cases to show the drivers work properly when the flash array(s) is(are) mapped to different locations.

1.3 Documentation Reference

MPC565 / MPC566 Reference Manual, MPC565RM/D

MPC561 /MPC562 / MPC563 / MPC564 Reference Manual, MPC561_3RM/D

SqueeZard MPC56x Code Compression Tool User's Manual

1.4 System Requirements

The product is distributed as an InstallShield setup.exe for Microsoft Windows.

The uncompressed demos were developed and tested with the Windriver Diab C compiler v4.4a and SDS Singlestep debugger v7.6.2.

The compressed demos were developed with the Windriver Diab C compiler v4.4a but with the Lauterbach TRACE32 debugger.

The S-Record demos were developed using SDS Singlestep scripts to control the target resident GMD functions. These scripts show all required register, stack and memory operations required to operate the GMD functions as stand-alone target resident code.

1.5 Installation

To install the software on your Microsoft Windows system:

1. Unzip the distribution file into a temporary directory.
2. Execute the SETUP.EXE file to launch the Installshield installation process.
3. Follow the on-screen instructions to install the driver and demo files. No reboot is necessary.

Chapter 2: Operation

2.1 System and Flash Module Initialization

While the GMD functions may read both flash and non-flash control registers, they only write to C3F control registers, the C3F main array, and C3F shadow array locations. Initialization of system clock speed, the IMMR register, and other MPC565 system functions is the responsibility of the user. The user is also responsible for initializing the following C3F registers for each module prior to calling the GMD functions:

- PROTECT[0:7] UC3FMCR[24:31]
- SBEN[0:1] UC3FMCRE[0:1]
- SBPROTECT[0:1] UC3FMCRE[6:7]

The GMD functions are responsible for managing the following C3F registers:

- BLOCK[0:7] UC3FCTL[16:23]
- SBBLOCK[0:1] UC3FCTL[14:15]

2.2 C-Array Driver Format for Embedded Applications

The c-array GMD function format is intended to simplify automated builds of uncompressed embedded applications such as boot loaders. As illustrated in the c-array demo provided with this release, the hexadecimal coded c-array file can be automatically integrated with your application at link time. The EABI compliant stack frame interface is designed so the GMD functions can be accessed by a simple C-language function call. The GMD functions can also be called from assembly language applications so long as the EABI stack frame interface is properly duplicated. Please refer to the S-Record demo for the exact sequence of operations. Also note that the EnabledBDM flag must be set to FALSE so that the GMD functions will properly return to the calling application.

2.3 S-Record Driver Format for BDM Programming Tools

The S-Record GMD function format is intended to simplify construction of BDM programming tools. Since the supplied GMD functions provide all the functionality that is required for a typical BDM programming tool, no other target resident code is required. In this class of applications, the BDM port is used to:

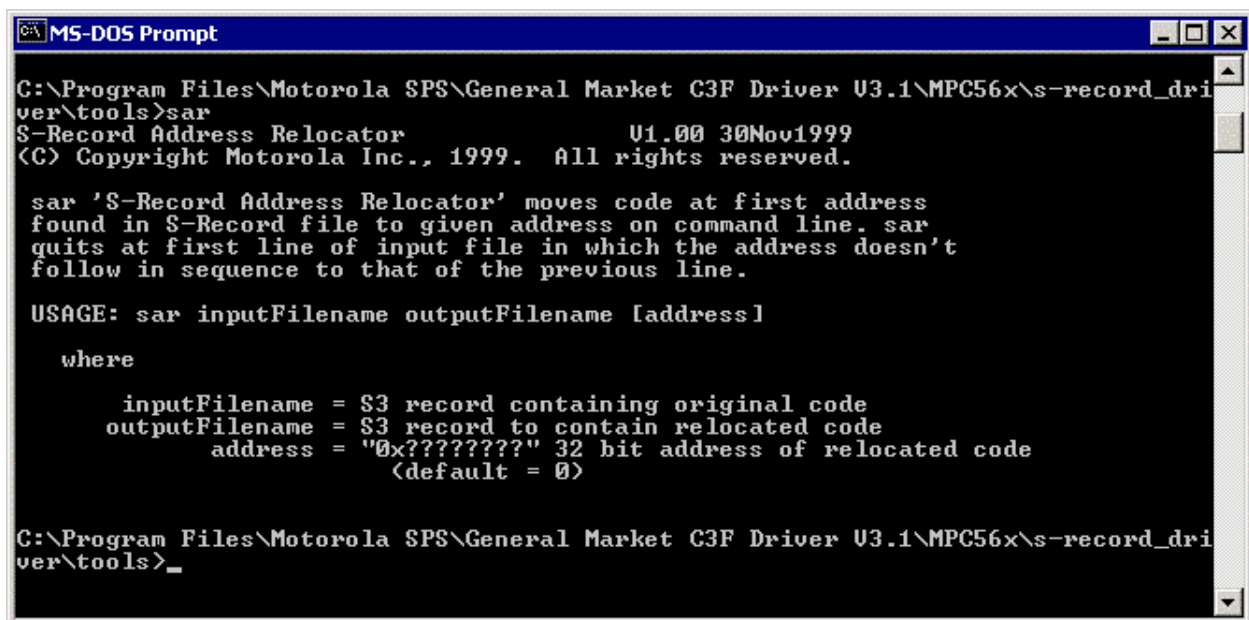
- download GMD functions to the target microprocessor,
- download data buffers to the target,
- set up the stack,
- set the program counter,
- and enter run mode.

By setting the EnabledBDM flag to TRUE, each target resident GMD function signals completion to the host computer by switching to BDM mode rather than executing a subroutine return. At this point error return codes and function return parameters can be retrieved from the target via the BDM port.

Individual S-Record format files are provided for each GMD function. In addition a single S-Record is provided that contains all GMD functions. The user can generate single S-Record files containing custom GMD function subsets by using the following procedure.

2.3.1 Relocating S-Records with SAR.EXE

Since each GMD function is position-independent code, they can be located at any valid memory location. However since S-Record files are mapped to explicit address ranges, the 0x0 based S-Records provided for each GMD function must be explicitly mapped to a particular address range. The SAR.EXE utility is provided for this purpose. The command syntax for SAR.EXE is illustrated below:



```

C:\Program Files\Motorola SPS\General Market C3F Driver U3.1\MPC56x\s-record_driver\tools>sar
S-Record Address Relocator          U1.00 30Nov1999
(C) Copyright Motorola Inc., 1999.  All rights reserved.

sar 'S-Record Address Relocator' moves code at first address
found in S-Record file to given address on command line. sar
quits at first line of input file in which the address doesn't
follow in sequence to that of the previous line.

USAGE: sar inputFilename outputFilename [address]

where

    inputFilename = S3 record containing original code
    outputFilename = S3 record to contain relocated code
    address = "0x???????" 32 bit address of relocated code
              (default = 0)

C:\Program Files\Motorola SPS\General Market C3F Driver U3.1\MPC56x\s-record_driver\tools>_

```

2.3.2 Building a Custom S-Record

The following DOS batch file illustrates how to use SAR.EXE and the copy command to build an S-record with all eight GMD functions:

SAR ..\gmd_driver\FlashInit.s19	fi.s19	0x3F9800
SAR ..\gmd_driver\FlashCheckShadow.s19	fs.s19	0x3F9A00
SAR ..\gmd_driver\FlashErase.s19	fe.s19	0x3F9C00
SAR ..\gmd_driver\BlankCheck.s19	bc.s19	0x3FA000
SAR ..\gmd_driver\FlashProgram.s19	fp.s19	0x3FA200
SAR ..\gmd_driver\FlashVerify.s19	fv.s19	0x3FA600
SAR ..\gmd_driver\ChangeCensor.s19	cc.s19	0x3FA800
SAR ..\gmd_driver\Checksum.s19	cs.s19	0x3FAE00

```
copy fi.s19+fs.s19+fe.s19+bc.s19+fp.s19+fv.s19+cc.s19+cs.s19 ..\gmd_driver\gmd.s19
del *.s19
```

Both the batch file (build.bat) and the S-Record built with it are included with the file distribution for your convenience.

2.4 Object Library Driver Format for Code Compression

The object library GMD function format is intended to simplify automated builds of compressed embedded applications such as boot loaders. As illustrated in the object library demo provided with this release, the object library can be automatically integrated with your application at link time. The EABI compliant stack frame interface is designed so the GMD functions can be accessed by a simple C-language function call. The GMD functions can also be called from assembly language applications so long as the EABI stack frame interface is properly duplicated. Please refer to the S-Record demo for the exact sequence of operations. Also note that the EnabledBDM flag must be set to FALSE so that the GMD functions will properly return to the calling application.

Note that the binary or c-array GMD file formats are *not* appropriate for a code compressed environment since the SQUEEZARD.EXE file compression application treats these file formats as data. Since SQUEEZARD.EXE compresses code but not data, maximum compression efficiency cannot be achieved with the other GMD function formats. However the object file format is precompiled with code compression enabled. This means it can be linked to your software application and then compressed for maximum compression efficiency.

2.5 Concurrency, Callback Functions, and Flash Suspend/Resume

Various degrees of concurrency can be achieved by using the callback mechanism that is designed into the GMD functions. To keep a GMD function from monopolizing the CPU for extended periods a user-supplied callback function is called from hard-coded locations within the GMD functions. While the actual time between successive callbacks varies, the design requirement for the maximum callback period is 100 μ s for a 40 MHz system clock. Actual maximum callback periods for each function are listed in Appendix A.

By customizing the contents of the callback function, a variety of concurrency mechanisms can be accommodated.

Concurrency Environment	Concurrent Read From Flash?	Suspend/Resume of Program/Erase Required?	Callback Template
None	No	No	A
Polling	No	No	B
Polling	Yes	Yes	C
Interrupt	Yes	Yes	D

2.5.1 Callback Template A : No Concurrency, No Read From Flash

This is the simplest callback template in that it contains no user-supplied logic, and it contains no suspend/resume logic.

```
void CallBack (void)
{
}
```

2.5.2 Callback Template B : Polled Concurrency, No Read From Flash

This callback template contains user-supplied logic, but it contains no suspend/resume logic since reads from flash are not required. Note that more user-supplied logic means more total execution time for the GMD functions. Generally speaking, the user-supplied callback logic should be built like an interrupt service routine in that the user-supplied logic should be as short as possible.

```
void CallBack (void)
{
    /******
    /*      user supplied logic      */
    /******
}
```

2.5.3 Callback Template C : Polled Concurrency, Read From Flash

When no high voltage flash operations are in progress, this case works like Callback Template B. However if a program or an erase operation is in progress, this template suspends the flash operation before taking the user-specified action that requires a read from flash. The flash operation is resumed before the callback returns to the GMD function.

Note that censor operations cannot be suspended, so in this case the template simply returns to the GMD function after taking no user-specified action.

Also note that suspend/resume requests should be ‘infrequent’. If suspend/resume requests are not periodic, they should be limited to just a few for each erase operation. If suspend/resume requests are periodic, they should be limited to no more than one every millisecond.

```

void CallBack (void)
{
    if EHV bit in C3F control register is 0, High voltage operation is not in progress.
    {
        // Suspend operation is not required to access C3F flash module.
        /******
        /* Read C3F flash module */
        /******
    }
    else EHV bit in C3F control register is 1. High voltage flash operation is in progress.
    {
        if CSC bit in C3F control register is 0, Erase or Program operation is in progress.
        {
            // if PE bit in C3F control register is 1, Erase operation is in progress.
            // if PE bit in C3F control register is 0, Program operation is in progress.

            // Suspend operation is required to access C3F flash module.
            if time since last suspend/resume > 1ms // Allow at least 1ms between suspend/resume operations.
            {
                write 1 to HSUS bit in C3F control register; // Suspend the program or erase operation.
                while HVS bit in C3F control register is 1; // Wait for the suspend operation to complete.
                /******
                /* Read C3F flash module */
                /******
                write 0 to HSUS bit in C3F control register; // Resume the flash program or erase operation.
            }
        }
        else if CSC bit in C3F control register is 1, Censor operation is in progress.
        {
            // Suspend operation is not allowed.
        }
    }
}

```

2.5.4 Callback Template D: Interrupted Concurrency, Read From Flash

For the MPC56x series of embedded flash parts, an interrupted environment normally implies that reads from flash are required. This is because the interrupt vectors are typically located in low addresses of flash module A. If a module A flash operation is in progress, a flash suspend request is required to make the interrupt vectors visible to the CPU, and hence to make interrupt processing possible.

This case differs from the previous cases in that the user-supplied logic is now located in interrupt service routines (ISRs). The callback function permits access to the ISRs by simply enabling interrupts. If a program or erase is in progress and an interrupt is pending, the flash operation is suspended before enabling interrupts. If a censor operation is in progress, or if no interrupts are pending, the callback simply returns to the calling GMD function without performing either a suspend request and without enabling interrupts.

Again note that suspend/resume requests should be ‘infrequent’. If suspend/resume requests are not periodic, they should be limited to just a few for each erase operation. If suspend/resume requests are periodic, they should be limited to no more than one every millisecond.

```
void CallBack (void)
{
    if interrupt is pending
    {
        if EHV bit in C3F control register is 1, High voltage operation is in progress.
        {
            if CSC bit in C3F register is 0, Erase or Program operation is in progress.
            {
                // if PE bit in C3F control register is 1, Erase operation is in progress.
                // if PE bit in C3F control register is 0, Program operation is in progress.

                // Suspend operation is required to access C3F flash module.
                if time since last suspend/resume > 1 ms // Allow at least 1ms between suspend/resume operations.
                {
                    write 1 to HSUS bit in C3F control register; // Suspend the program or erase operation.
                    while HVS bit in C3F control register is 1; // Wait for the suspend operation to complete.
                    /* Enable external interrupts */
                    /* Disable external interrupts */
                    /* Enable external interrupts */
                    write 0 to HSUS bit in C3F control register; // Resume the flash program or erase operation.
                }
            }
            else if EHV bit in C3F control register is 0, Censor operation is in progress.
            {
                // Suspend operation is not allowed.
            }
        }
        else if EHV bit in C3F control register is 0, High voltage operation is not in progress.
        {
            // Suspend operation is not required to access C3F flash module.
            /* Enable external interrupts */
            /* Disable external interrupts */
            /* Enable external interrupts */
        }
    }
}
```

Chapter 3: API Specification

3.1 General Overview

This section defines function return codes and function parameters. The function definitions include a listing of registers affected by that function as well as a high-level pseudo-code listing the operations performed by that function.

3.2 Function Return Value Error Codes

Name	Value	Description
C3F_OK	0x00000000	The requested operation was successful.
C3F_INFO_LOCK_B	0x00000001	The LOCK bit in C3FMCR register is set. The control registers of C3F module B cannot be changed.
C3F_INFO_LOCK_A	0x00000002	The LOCK bit in C3FMCR register is set. The control registers of C3F module A cannot be changed.
C3F_INFO_EPEE_B	0x00000004	High voltage is not present for the C3F module B. Program and erase operations are not possible.
C3F_INFO_EPEE_A	0x00000008	High voltage is not present for the C3F module A. Program and erase operations are not possible.
C3F_INFO_BOEPEE_B	0x00000010	High voltage is not present for either small block 0 or the lowest numbered block of C3F module B. Program and erase operations are not possible for these blocks.
C3F_INFO_BOEPEE_A	0x00000020	High voltage is not present for either small block 0 or the lowest numbered block of C3F module A. Program and erase operations are not possible for these blocks.
C3F_INFO_HVS_B	0x00000040	High voltage operations are now in progress in C3F module B. Unless suspended, this module (including both Shadow and main array) can not be accessed.
C3F_INFO_HVS_A	0x00000080	High voltage operations are now in progress in C3F module A. Unless suspended, this module (including both Shadow and main array) can not be accessed.
C3F_INFO_CENSOR_B	0x00000100	Module B censor is active.
C3F_INFO_CENSOR_A	0x00000200	Module A censor is active.
C3F_INFO_STOP_B	0x00000400	C3F module B is in low power operation.
C3F_INFO_STOP_A	0x00000800	C3F module A is in low power operation.
C3F_INFO_FLEN	0x00001000	On-chip flash memory is disabled.
C3F_INFO_FLASHID	0x00002000	In early release FLASHID = 0 parts, no censor operation is possible.
C3F_ERROR_INVALID_ENABLED_BLOCK	0x00004000	Specified block must be unprotected to erase or program.
RESERVED	0x00008000	Reserved.
RESERVED	0x00010000	Reserved.
C3F_ERROR_PARTID	0x00020000	Current flash driver cannot support the given part.
C3F_ERROR_ALIGNMENT	0x00040000	A parameter does not meet an alignment requirement.
C3F_ERROR_SHADOW_RANGE	0x00080000	An invalid shadow range was passed.
C3F_ERROR_ARRAY_RANGE	0x00100000	An invalid array range was passed.
C3F_ERROR_ENABLED_SMALL_BLOCK	0x00200000	The enabled small block passed from the caller program is invalid according to the settings of C3F registers or the small blocks setting in SBLK[0:1] is inconsistent with the specified value.

C3F_ERROR_ERASE_PEGOOD	0x00400000	Erase operation failed.
C3F_ERROR_PROGRAM_VERIFY	0x00800000	Verification failed after the operation has passed PEGOOD checking.
C3F_ERROR_PROGRAM_PEGOOD	0x01000000	Program operation failed because this operation cannot pass PEGOOD check.
C3F_ERROR_NOT_BLANK	0x02000000	The specified memory location is not blank.
C3F_ERROR_SRC_DEST_VERIFY	0x04000000	The specified flash data does not match the source data.
C3F_ERROR_CENSOR_MODULE	0x08000000	An invalid module number was passed. The module number depends on the given part.
C3F_ERROR_CENSOR_VALUE	0x10000000	An invalid censor value (>3) was passed.
C3F_ERROR_CENSOR_DEVICE_MODE	0x20000000	An invalid device censor mode (>3) was passed when changing censor.
C3F_ERROR_CENSOR_INVALID_REQUEST	0x40000000	The request to change censor violates the hardware specifications.
C3F_ERROR_CENSOR_PEGOOD	0x80000000	Censor operation failed.

3.3 FlashInit

3.3.1 Description

This function initializes the control registers for the C3F modules. For each module, the function checks the condition of the control registers, EPEE pin and Block 0 EPEE pin status, sensor status and other environment configurations. The result will be returned to the user. If everything is OK, it simply returns C3F_OK. Otherwise subsequent functions will not work. This function need only be called once prior to multiple flash driver operations.

The input argument &arrayBase is simply 4 bytes of RAM that will become a pointer to the array at exit. This is used by other functions to point to the absolute start of the array. It is only necessary to call this function again if the flash memory is re-mapped to another absolute address.

3.3.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 11.
3. Check the FLEN bit in the IMMR register to verify that the on-chip flash is enabled. If not, set the return value to C3F_INFO_FLEN and go to step 11.
4. Calculate the array base of the on-chip flash from the ISB bits in the IMMR register and store in &arrayBase.
5. Determine the number of on-chip C3F modules from the part number.
6. Check the STOP bit in the C3FMCR register to verify that the C3F flash array is enabled. If not, set the return value to C3F_INFO_STOP_A(B) and go to step 11.
7. Check the LOCK bit in the C3FMCR register to verify that the write-locked register bits are unlocked. If not, set the return value to C3F_INFO_LOCK_A(B) and go to step 11.
8. Verify that high voltage operations on the C3F flash module are possible. If not, set the return value to C3F_INFO_EPEE_A(B), C3F_INFO_BOEPEE_A(B), or C3F_INFO_HVS_A(B) accordingly and go to step 11.
9. Check the SENSOR bits in the C3FMCR register to verify that C3F array accesses are allowed. If not, set the return value to C3F_INFO_CENSOR_A(B) and go to step 11.

10. If another C3F module exists, go to step 6.

11. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise, return to the calling program with external interrupts disabled.

3.3.3 Definition

UINT32 FlashInit (UINT8 enabledBDM,
 UINT32 &arrayBase);

3.3.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled; If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled; If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.
&arrayBase	UINT32	Raw RAM address to store a pointer to flash memory start.	Any RAM allocated to hold a pointer. It becomes a pointer to flash at return.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.3.5 Return Values

Type	Description	Values
UINT32	Indicates either success or failure type. The function checks each embedded C3F module. Each bit in the returned value indicates a kind of status of the current operation environment except for C3F_OK.	C3F_OK C3F_INFO_LOCK_A C3F_INFO_LOCK_B C3F_INFO_EPEE_A C3F_INFO_EPEE_B C3F_INFO_BOEPEE_A C3F_INFO_BOEPEE_B C3F_INFO_HVS_A C3F_INFO_HVS_B C3F_INFO_CENSOR_A C3F_INFO_CENSOR_B C3F_INFO_STOP_A C3F_INFO_STOP_B C3F_INFO_FLEN C3F_ERROR_PARTID

3.3.6 Tips

At return, the function will disable EHV for each C3F module.

When returning C3F_ERROR_PARTID, all the other checking will not be performed because the given part is not supported by the current flash driver.

When returning C3F_INFO_FLEN, all but its previous checkpoint C3F_ERROR_PARTID checking will not be performed because a wrong array base may cause exceptions.

Under slave/master environment, the input argument, &arrayBase will point to the internal array base of the master microcontroller.

3.3.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_INFO _FLEN	Flash memory in the part is disabled because of the invalid IMMR setting. No flash driver function can run properly.	Set the IMMR register according to the intended arrayBase.
C3F_INFO _SENSOR_MODULE_B	Module B is in cleared sensor or information sensor, i.e. module B sensor bit is 0 or 3. Any operation that accesses the main array and shadow row of module B will cause an exception.	Change sensor bits of module B to the no sensor state with ChangeSensor().
C3F_INFO _SENSOR_MODULE_A	Module A is in cleared sensor or information sensor, i.e. module A sensor bit is 0 or 3. Any operation that accesses the main array and shadow row of module A will cause an exception.	Change sensor bits of module A to the no sensor state with ChangeSensor().
C3F_INFO _EPEE_B	High voltage is not present for module B. Flash program and erase operations are not possible except for array block 0 or the lowest numbered block. Refer to C3F_INFO_B0EPEE_B	Check target board manual to enable EPEE.
C3F_INFO _EPEE_A	High voltage is not present for the C3F module A. Program and erase operations are not possible except for array block 0 or the lowest numbered block. Refer to C3F_INFO_B0EPEE_A	Check target board manual to enable EPEE.
C3F_INFO _B0EPEE_B	High voltage is not present for module B. Program and erase operations are not possible for block 0 or the lowest numbered block.	Check target board manual to enable B0EPEE.
C3F_INFO _B0EPEE_A	High voltage is not present for module A. Program and erase operations are not possible for block 0 or the lowest numbered block.	Check target board manual to enable B0EPEE.
C3F_INFO _HVS_B	High voltage operations are now in progress in module B. Unless suspended, shadow and main array can not be accessed.	Check if other operations are pending.
C3F_INFO _HVS_A	High voltage operations are now in progress in module A. Unless suspended, shadow and main array can not be accessed.	Check if other operations are pending.
C3F_INFO _LOCK_B	The LOCK bit in C3FMCR register is set. The control registers of C3F module B cannot be changed. All flash driver functions cannot run properly.	Disable the lock by a reset, but from background debug mode with CSC=0, this bit can be written from 0 to 1.
C3F_INFO _LOCK_A	The LOCK bit in C3FMCR register is set. The control registers of C3F module A cannot be changed. All flash driver functions cannot run properly.	Disable the lock by a reset, but from background debug mode with CSC=0, this bit can be written from 0 to 1.
C3F_INFO _STOP_B	Module B is in low power operation. Flash memory is not available, and driver functions cannot run properly. Can still read and write C3FMCR.	Check STOP bit in C3FMCR and the input pin, c3f_stopin.
C3F_INFO _STOP_A	Module A is in low power operation. Flash memory is not available, and driver functions cannot run properly. Can still read and write C3FMCR.	Check STOP bit in C3FMCR and the input pin, c3f_stopin.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

Note: For some parts owning only one C3F module, return value for module B checking will be reserved.

3.3.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM, FLEN, ISB	Read
MSR	EE	Write
C3FMCR	LOCK, STOP, SENSOR	Read
C3FCTL	EHV	Write
	EPEE, BOEM, HVS	Read
C3FMCRE	FLASHID	Read

3.3.9 Revision String

An 11-character ASCII text revision string is appended to the end of the FlashInit function executable. Use a hex viewer utility to view this revision string in the binary image. Or use the ASCII option in the debugger memory window dump once the FlashInit function has been loaded. The FlashInit revision string is formatted as follows:

PPCC3FFIxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
FI	Driver Routine	FlashInit acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.4 FlashCheckShadow

3.4.1 Description

This sub module function sets up control registers and range checks shadow row parameters. The result will be returned to the user. If everything is OK, it simply returns C3F_OK. This function should be called before FlashProgram, BlankCheck, FlashVerify or CheckSum operation, even if shadow memory will not be used. This is important to prevent unintended shadow row operations. If multiple operations are done to the same memory type, either shadow or main flash memory, this function need only be called once. It is used to select between main or shadow flash memory.

3.4.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 9.
3. Determine the number of on-chip C3F modules, the relative C3F array address range, and the relative C3F shadow address range from the part number.
4. Calculate the relative destination address from the destination input and the relative C3F array address range.
5. Calculate the absolute address of the C3FMCR register from the arrayBase argument.
6. If the shadow input argument is FALSE, clear the SIE bit in the C3FMCR register of each on-chip C3F module and go to step 9.
7. Verify that the relative destination range (relative destination address + size) and the size fall within the relative C3F shadow address range. If not, set the return value to C3F_ERROR_SHADOW_RANGE and go to step 9.
8. Set the SIE bit in the C3FMCR register of the C3F module in which the shadow resides.
9. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.4.3 Definition

UINT32 FlashCheckShadow (UINT8 enabledBDM,
 BOOL Shadow,
 UINT32 dest,
 UINT32 size,
 UINT32 arrayBase);

3.4.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	<p>If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled;</p> <p>If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled;</p> <p>If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.</p>
shadow	BOOL	Shadow row select flag.	<p>TRUE = Select Shadow Row.</p> <p>FALSE = Select Main Array.</p>
dest	UINT32	Starting virtual address in flash or other memory.	Any addressable 32-bit word in main array or shadow or other memory. It should fall into C3F shadow.
size	UINT32	Size, in bytes, of the flash region needing FlashShadowCheck.	Must be multiples of a 32-bit word. If size equals to 0, C3F_OK will be returned. Its combination with dest should fall into C3F shadow.
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.4.5 Return Values

Type	Description	Values
UINT32	Indicates either success or failure type. The function checks each embedded C3F module.	C3F_OK C3F_ERROR_SHADOW_RANGE C3F_ERROR_PARTID

3.4.6 Tips

The dest argument will be modified from the Internal Space Base (ISB) in the Internal Memory Map Register (IMMR). For 1MB flash or 512KB flash, this input will first be ANDed with 0xFFFFF or 0x7FFFF respectively, then offset with the base address calculated from the the ISB to produce the absolute flash address.

3.4.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_SHADOW_RANGE	The area specified by dest and size is out of the valid shadow range.	1) Check if dest is out of shadow range. 2) Check if dest+size is out of shadow range. Note the size of shadow is 512 bytes.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.4.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM,	Read
MSR	EE	Write
C3FMCR	SIE	Read

3.4.9 Revision String

An 11-character ASCII text revision string is appended to the end of the FlashCheckShadow function executable. Use a hex viewer utility to view this revision string in the binary image. Or use the ASCII option in the debugger memory window dump once the FlashCheckShadow function has been loaded. The FlashCheckShadow revision string is formatted as follows:

PPCC3FFSxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
FS	Driver Routine	FlashCheckShadow acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.5 FlashErase

3.5.1 Description

This function will erase the specified enabled array blocks or array small blocks or array residual blocks. Parameters are range checked on entry, and an appropriate error code is returned if an error is found. If a block contains a shadow row, then this row is erased with its parent block. Shadow row contents are not preserved. The user is responsible for preserving and restoring shadow row contents during an erase.

The flash module block protect bits are not changed by this driver, even if required to perform an erase or program. It is up to the user to unprotect blocks in C3FMCR to allow this function to work. If this driver unprotected blocks, it would negate the very purpose of the protect bits.

3.5.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 18.
3. Determine the number of on-chip C3F modules from the part number.
4. Check the validity of the input arguments, enabledBlocks and enabledSBlocks. If either argument is invalid, set the return value to C3F_ERROR_INVALID_ENABLED_BLOCK or C3F_ERROR_ENABLED_SMALL_BLOCK and go to step 18.
5. Calculate the absolute address of the C3F control registers from the arrayBase input.
6. Disable interrupt requests.
7. Write BLOCK[0:7] and SBBLOCK[0:1] to select the blocks to be erased.
8. Set PE = 1 and SES = 1 in the C3FCTL register.
9. Execute an erase interlock write with 0xFFFFFFFF to any C3F array location within each module to be erased.
10. Apply high voltage on the C3F flash by setting EHV = 1 in the C3FCTL register.
11. If another C3F module exists, go to step 5.

12. If EHV bit has been set to 1, read the C3FCTL register until HVS = 0. (While HVS is high, the Callback function will be invoked repeatedly to respond to time-critical events.)
13. Read the C3FCTL register to confirm that PEGOOD = 1. If PEGOOD = 0, set the return value to C3F_ERROR_ERASE_PEGOOD and go to step 18.
14. Disable high voltage by writing EHV = 0 in the UC3FCTL register.
15. Write SES = 0 in the C3FCTL register to end the erase sequence.
16. Clear the BLOCK[0:7] and SBBLOCK[0:1].
17. If another C3F module exists, go to step 12.
18. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.5.3 Definition

```

UINT32 FlashErase ( UINT8 enabledBDM,
                    void (*Callback)(void),
                    UINT8 enabledBlocks[module],
                    UINT8 enabledSBlocks[module],
                    UINT32 arrayBase);

```

3.5.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	<p>If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled;</p> <p>If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled;</p> <p>If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.</p>
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
enabledBlocks[module]	UINT8 *	Pointer to array of bits used to select the array blocks to erase.	The values in this array should be one of the valid layouts of array blocks for the given MPC56x derivative part. If no blocks are selected in a module, set the related array element to 0x00
enabledSBlocks[module]	UINT8 *	Pointer to array of bits used to select the small blocks to erase.	The values in this array should be one of the valid layouts of small block arrays. If no blocks are selected in a module, set the related array element to 0x00
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Note: EnabledSBlocks[module] will determine which array small blocks need erasing. Only two bits will be meaningful and the position of these two bits will be determined by the array small block location code SBLK[0:1].

Bit map for enabledBlocks [module]

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
block 0	block 1	block 2	block 3	block 4	block 5	block 6	block 7

Bit map for enabledSBlocks[module] Note the unexpected bit order.

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
small block 0	small block 1	reserved	reserved	reserved	reserved	reserved	reserved

The C3F array erase is subject to both enabledBlocks[module] and enabledSBlocks[module]. The SBEN[0:1] bits in C3FMCRE will be set according to the enabledSBlocks selected by the user for erase.

For example, consider array small block 0 and array block 0 in module A, based on the above small block layout. The following combinations are illustrated:

enabledSBlock[0] Bit 0	enabledBlock[0] Bit 0	Array locations erased
0	0	None
0	1	Only residual block 0 will be erased
1	0	Only small block 0 will be erased
1	1	All locations in block 0 will be erased

Note: The above table is only valid if SBEN=1, namely small block is enabled. Otherwise, input argument enabledBlock[] will determine array location to be erased. When the given block bit of enabledBlock[] is set, all locations within this block will be erase unless this block is protected.

Bit map for the input parameter enabledBDM

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.5.5 Return Values

Type	Description	Possible Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_ERASE_PEGOOD C3F_ERROR_INVALID_ENABLED_BLOCK C3F_ERROR_ENABLED_SMALL_BLOCK C3F_ERROR_PARTID

3.5.6 Tips

If the blocks to be erased have subsidiary shadow rows, they will also be erased. The user needs to save the shadow contents if needed.

The function will handle module A and module B at the same time if the given part owns two C3F modules. So please always set the enabledBlock and enabledSBlock arrays correctly, e.g. enabledBlock[0] and enabledSBlock[0] for selecting blocks in module A; enabledBlock[1], enabledSBlock[1] for module B. If no blocks are selected in a module, set the related array element to 0x00. More detailed information, refer to above Notes.

Do not erase blocks in a module having cleared censor or information censor.

If both the two elements in the enabledBlock[0], enabledSBlock[0] array and the two elements in the enabledBlock[1], enabledSBlock[1] array are 0x00, the GMD function will return C3F_OK unless the Flash memory is not available.

The flash driver will not be able to erase or program memory within its own module. The erase or program function disables normal reads of flash memory. For example, if the driver is in module A, then any part of module B may be programmed or erased, provided module B reads are not necessary. If an interrupt or polling routine requires reads from a module being erased or programmed, then suspend and resume operations are required to read valid data. Any module containing interrupt vectors should not be erased or programmed while interrupts are possible. This is because the interrupt pointers in flash will not read valid data. Since module A contains interrupt vectors, it is suggested to use module B for all erase and program operations, while keeping the flash driver in module A. If module A erase and program operations are required, one can copy the driver into RAM for execution, and disable all interrupts.

3.5.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_INVALID_ENABLED_BLOCK	Could not erase	This block must be unprotected.
C3F_ERROR_ENABLED_SMALL_BLOCK	Could not erase	This small block must be unprotected.
C3F_ERROR_ERASE_PEGOOD	Erase operation failed	Repeat the erase operation or the C3F is invalid or high voltage applied to C3F is unsuitable.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.5.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write
C3FMCR	PROTECT	Read
C3FCTL	EHV, PE, SES, BLOCK, SBLOCK	Write
	HVS, PEGOOD	Read
C3FMCRE	SBEN	Write
	SBPROTECT	Read

3.5.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the FlashErase function executable. Use a hex viewer utility or the ASCII option in the debugger memory window dump once the FlashErase function has been loaded, to view this revision string in the binary image. The FlashErase revision string is formatted as follows:

PPCC3FFExyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
FE	Driver Routine	FlashErase acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.6 BlankCheck

3.6.1 Description

This function will perform a blank check on the specified memory array or shadow row.

A check is first performed to verify that the selected array is accessible. If this check should fail, the appropriate error code will be returned.

If the blank check fails, then *compareAddress is updated with the first failing address, and *compareData shows the failing data.

3.6.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 10.
3. Determine the number of on-chip C3F modules and the relative C3F array address range from the partnumber.
4. Clear the compareAddress and compareData arguments.
5. Calculate the relative destination address from the destination input and the relative C3F array address range.
6. Verify that the relative destination address and the size input are each word (4-byte) aligned. If not, set the return value to C3F_ERROR_ALIGNMENT and go to step 10.
7. Verify that the relative destination range (relative destination address + size) and the size fall within the relative C3F array address range. If not, set the return value to C3F_ERROR_ARRAY_RANGE and go to step 10.
8. Calculate the absolute destination address from the relative destination address and the arrayBase input.
9. Blank check each word (4 bytes). If there is a word not equal to 0xFFFFFFFF, set the return value to C3F_ERROR_NOT_BLANK, set the compareAddress argument to the failing address, set the compareData argument to the corresponding data, and stop the blank checking operation. (During the blank checking operation, the Callback function will be invoked periodically to respond to time-critical events.)
10. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling

program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.6.3 Definition

```

UINT32 BlankCheck ( UINT8 enabledBDM,

                    void (*CallBack)(void),

                    UINT32 dest,

                    UINT32 size,

                    UINT32 *compareAddress,

                    UINT32 *compareData,

                    UINT32 arrayBase );

```

3.6.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	<p>If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled;</p> <p>If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled;</p> <p>If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.</p>
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
dest	UINT32	Destination address to be checked in flash memory.	You may provide an absolute or relative address, since this input is masked to the size of internal flash address space. It should fall into C3F flash module.
size	UINT32	Size, in bytes, of the flash region to blank check.	Must be multiples of a 32-bit word. If size equals to 0, C3F_OK will be returned. Its combination with dest should fall into C3F flash module.
*compareAddress	UINT32	First non-blank address	Only valid when the whole function returns C3F_ERROR_NOT_BLANK.
*compareData	UINT32	First non-blank data	Only valid when the whole function returns C3F_ERROR_NOT_BLANK.
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Notes:

The dest argument will be modified according to IMMR (Internal Memory Map Register) register. In the cases of a 1MB flash or a 512KB flash, this input will first be ANDed with 0xFFFFF or 0x7FFFF respectively, then offset in memory space according to IMMR to produce the absolute flash address.

The compareAddress will be the relative address. It should be modified according the IMMR setting before checking this address.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.6.5 Return Values

Type	Description	Possible Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_NOT_BLANK C3F_ERROR_ALIGNMENT C3F_ERROR_ARRAY_RANGE C3F_ERROR_PARTID

3.6.6 Tips

The function can do the blank check only in flash memory space. Do NOT check blocks in cleared sensor or information sensor modules. It may cause machine-check or check-stop exceptions. If size = 0, the function returns C3F_OK if the other parameters are all valid.

When executing the driver from flash memory, do NOT check the shadow row in the module that the function is resident in. Reading the shadow row will temporarily make the main flash memory unreadable. This may also cause interrupts to be improperly vectored.

3.6.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_NOT_BLANK	This is not an error. It just indicates that there is a non-blank word (i.e. not 0xFFFFFFFF) within the area to be checked.	Must erase it first
C3F_ERROR_ARRAY_RANGE	The area specified by dest and size is out of the valid C3F array range.	1) Check if dest is out of C3F array range. 2) Check if dest+size is out of C3F array range. Note the size of C3F array is 1M bytes or 512K bytes depending on the given part.
C3F_ERROR_ALIGNMENT	This error indicates that either the dest or the size isn't valid.	1) Check if the parameter dest is on the word (32-bit or 4-byte) boundary, i.e. the dest is multiple of 4. 2) Check if the parameter size is multiple of 4.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.6.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write

3.6.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the BlankCheck function executable. Use a hex viewer utility or the ASCII option in the debugger memory window dump once the BlankCheck function has been loaded, to view this revision string in the binary image. The BlankCheck revision string is formatted as follows:

PPCC3FBCxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
BC	Driver Routine	BlankCheck acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.7 FlashProgram

3.7.1 Description

This function will program the specified memory areas with the source data provided. A check is first performed to verify that the selected array is accessible. If this check should fail, the appropriate error code will be returned.

If it is desired to program less than a full 32 bit word, the user application should fill the remaining bytes of the given word of source data with 0xFF, so that the original data in the flash memory will be retained.

Unlike the MPC555 flash, the source data can reside in the same block as the destination data, because the flash can be read during programming provided the high voltage applied to the same C3F flash module is suspended. This is also true for the shadow row and main array combinations. When the shadow row is visible, the main array locations are not visible. Each C3F module contains only one shadow row.

Multiple modules must be mapped in one contiguous region. The source buffer contains data that will be loaded into flash memory.

The flash module block protect bits are not changed by this driver, even if required to perform an erase or program. It is up to the user to unprotect blocks in C3FMCR to allow this function to work. If this driver unprotected blocks, it would negate the very purpose of the protect bits.

3.7.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 22.
3. Determine the number of on-chip C3F modules and the relative C3F array address range from the part number.
4. Calculate the relative destination address from the destination input and the relative C3F array address range.
5. Verify that the relative destination address, the size input, and the source input are each word (4-byte) aligned. If not, set the return value to C3F_ERROR_ALIGNMENT and go to step 22.

6. Verify that the relative destination range (relative destination address + size) and the size fall within the relative C3F array address range. If not, set the return value to C3F_ERROR_ARRAY_RANGE and go to step 22.
7. Determine the end destination index for each C3F module to be programmed.
8. Calculate the absolute destination address from the arrayBase input.
9. Disable interrupt requests.
10. Calculate the absolute address of the C3F control registers from the arrayBase input.
11. Write BLOCK[0:7] = 0xFF and SBBLOCK[0:1] = 0x3 to select the array blocks to be programmed,
12. Set SES = 1 and PE = 0 in the C3FCTL register.
13. Write data word (4 bytes) from source to destination.
14. Apply high voltage on the C3F flash with writing EHV = 1 in the C3FCTL register.
15. Read the C3FCTL register until HVS = 0. (While HVS is high, the CallBack function will be invoked repeatedly to respond to time-critical events.)
16. Read the C3FCTL register to confirm PEGOOD = 1. If PEGOOD = 0, set the return value to C3F_ERROR_PROGRAM_PEGOOD and go to step 22.
17. Disable high voltage by writing EHV = 0 in the C3FCTL register.
18. If more data needs to be programmed go to step 13.
19. Write SES = 0 in the C3FCTL register to end program sequence.
20. Clear the BLOCK[0:7] and SBBLOCK[0:1].
21. If another C3F module exists, go to step 10.
22. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.7.3 Definition

```
UINT32 FlashProgram ( UINT8 enabledBDM,  
  
                      void(*CallBack)(void),  
  
                      UINT32 dest,  
  
                      UINT32 size,  
  
                      UINT32 source,  
  
                      UINT32 arrayBase );
```

3.7.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled; If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled; If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
dest	UINT32	Destination address to be programmed in flash memory.	You may provide an absolute or relative address, since this input is masked to the size of internal flash address space. It should fall into C3F flash module.
size	UINT32	Size, in bytes, of the flash region to be programmed.	Must be multiples of a 32-bit word. If size equals 0, C3F_OK will be returned. Its combination with dest should fall into C3F flash module.
source	UINT32	Source program buffer address.	This address must reside on a 4-byte boundary.
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Note:

The dest argument will be modified according to IMMR (Internal Memory Map Register) register. In the cases of a 1MB flash or a 512KB flash, this input will first be ANDed with 0xFFFFF or 0x7FFFF respectively, then offset in memory space according to IMMR to produce the absolute flash address.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.7.5 Return Values

Type	Description	Possible Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_PROGRAM_PEGOOD C3F_ERROR_ALIGNMENT C3F_ERROR_ARRAY_RANGE C3F_ERROR_PARTID

Note:

The user is responsible for initializing the block protect. This flash driver does not alter the existing block protection register. This means one can attempt to program protected flash blocks, and though it will pass PEGOOD status, the Verification status will show a failure. This will be seen after calling FlashVerify(). While SES=1 the PEGOOD status is not a good indicator of success during a block protected condition. When SES=0 the flash data can be verified, and errors will be detected only then. See the following table:

Result	C3F_ERROR_PROGRAM_PEGOOD	C3F_ERROR_PROGRAM_VERIFICATION
Pass	0 = Pass	0 = Pass
Failure	0 = Pass	1 = Failure due to block protected
Failure	1 = Failure	no care since PEGOOD = 1

3.7.6 Tips

When returning the function will clear SIE bits for all modules (in C3FMCR), clear all the BLOCK bits of C3FCTL, and disable small blocks. The small block is disabled during program operation. The two C3F modules are supposed to be a consecutive memory space.

If size = 0, the function returns C3F_OK if the other parameters are all valid.

Do NOT program modules having cleared censor or information censor status. It may cause machine-check or check-stop exceptions.

The function can NOT run properly if the source data is in a flash module and the destination data is in the shadow row of the same flash module. However, the source data and destination can be located both in the array of shadow with the same module.

The flash driver will not be able to erase or program memory within its own module. The erase or program function disables normal reads of flash memory. For example, if the driver is in module A, then any part of module B may be programmed or erased, provided module B reads are not necessary. If an interrupt or polling routine requires reads from a module being erased or programmed, then suspend and resume operations are required to

read valid data. Any module containing interrupt vectors should not be erased or programmed while interrupts are possible. This is because the interrupt pointers in flash will not read valid data. Since module A contains interrupt vectors, it is suggested to use module B for all erase and program operations, while keeping the flash driver in module A. If module A erase and program operations are required, one can copy the driver into RAM for execution, and disable all interrupts.

3.7.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_ALIGNMENT	This error indicates that either the dest or the size isn't valid.	1)Check if the parameter dest is on the word (32-bit or 4-byte) boundary, i.e. the dest is multiple of 4. 2)Check if the parameter size is multiple of 4. 3)Check if the parameter source is multiple of 4.
C3F_ERROR_ARRAY_RANGE	The area specified by dest and size is out of the valid C3F array range.	1) Check if dest is out of C3F array range. 2) Check if dest+size is out of C3F array range. Note the size of C3F array is 1M bytes or 512K bytes depending on the given part.
C3F_ERROR_PROGRAM_PEGOOD	Program operation failed because this operation cannot pass PEGOOD check.	Repeat the program operation or the C3F is invalid or high voltage applied to C3F is unsuitable.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.7.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write
C3FCTL	EHV, PE, SES, BLOCK, SBLOCK	Write
	HVS, PEGOOD	Read

3.7.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the FlashProgram function executable. Use a hex viewer utility or the ASCII option in the debugger memory window dump once the FlashProgram function has been loaded, to view this revision string in the binary image. The FlashProgram revision string is formatted as follows:

PPCC3FFPxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
FP	Driver Routine	FlashProgram acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.8 FlashVerify

3.8.1 Description

This function will verify the contents of the specified flash memory array against a comparison array.

A check is first performed to verify that the selected array is accessible. If this check should fail, the appropriate error code will be returned.

When the shadow row is visible, the main array locations are not visible. Each C3F module contains only one shadow row.

Multiple modules must be mapped in one contiguous region. The source buffer contains data that will be loaded into flash memory. The final data in the given C3F flash location is one copy of the source data.

3.8.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 10.
3. Determine the number of on-chip C3F modules and the relative C3F array address range from the part number.
4. Calculate the relative destination address from the destination input and the relative C3F array address range.
5. Verify that the relative destination address, the size input, and the source input are each word (4-byte) aligned. If not, set the return value to C3F_ERROR_ALIGNMENT and go to step 10.
6. Verify that the relative destination range (relative destination address + size) and the size fall within the relative C3F array address range. If not, set the return value to C3F_ERROR_ARRAY_RANGE and go to step 10.
7. Calculate the absolute destination address from the arrayBase input.
8. Clear the compareAddress, compareData, and compareSourceData arguments.
9. Verify that each word (4 bytes) of the destination data equals the source data. If a destination word is not equal to a source word, set the return value to C3F_ERROR_SRC_DEST_VERIFY, write the failing destination address to compareAddress, write the corresponding destination word to compareData, write the

corresponding source word to compareSourceData, and stop the verifying operation. (During the verifying operation, the CallBack function will be invoked periodically to respond to time-critical events.)

10. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.8.3 Definition

```
UINT32 FlashVerify ( UINT8 enabledBDM,  
                     void (*CallBack)(void),  
                     UINT32 dest,  
                     UINT32 size,  
                     UINT32 source,  
                     UINT32 *compareAddress,  
                     UINT32 *compareData,  
                     UINT32 *compareSourceData,  
                     UINT32 arrayBase );
```

3.8.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	<p>If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled;</p> <p>If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled;</p> <p>If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.</p>
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
shadow	BOOL	Shadow row select flag.	TRUE = Select Shadow Row. FALSE = Select Main Array.
dest	UINT32	Destination address to be verified in flash memory.	You may provide an absolute or relative address, since this input is masked to the size of internal flash address space. It should fall into C3F flash module.
size	UINT32	Size, in bytes, of the flash region to flash verify.	Must be multiples of a 32-bit word. If size equals to 0, C3F_OK will be returned. Its combination with dest should fall into C3F flash module.
source	UINT32	Source program buffer address.	This address must reside on a 4-byte boundary.
*compareAddress	UINT32	First failing address	Only valid when the whole function returns C3F_ERROR_SRC_DEST_VERIFY.
*compareData	UINT32	First failing data	Only valid when the whole function returns C3F_ERROR_SRC_DEST_VERIFY.
*compareSource Data	UINT32	First correct data	Only valid when the whole function returns C3F_ERROR_SRC_DEST_VERIFY.
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Notes:

The dest argument will be modified according to IMMR (Internal Memory Map Register) register. In the cases of a 1MB flash or a 512KB flash, this input will first be ANDed with 0xFFFFF or 0x7FFFF respectively, then offset in memory space according to IMMR to produce the absolute flash address. The compareAddress will be the relative address. It should be modified according the IMMR setting before checking this address.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.8.5 Return Values

Type	Description	Possible Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_ALIGNMENT C3F_ERROR_ARRAY_RANGE C3F_ERROR_SRC_DEST_VERIFY C3F_ERROR_PARTID

3.8.6 Tips

The source data must NOT be in the shadow rows. When running in flash memory, do NOT verify the shadow row of the module that the function is resident in.

The function can NOT run properly if the source data is in a flash module and the destination data is just in the shadow row of the same flash module.

Do NOT verify the blocks having cleared censor or information censor status. It may cause machine-check or check-stop exceptions.

If size = 0, the function returns C3F_OK if the other parameters are all valid.

3.8.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_ARRAY_RANGE	The area specified by dest and size is out of the valid C3F array range.	1) Check if dest is out of C3F array range. 2) Check if dest+size is out of C3F array range. Note the size of C3F array is 1M bytes or 512K bytes depending on the given part.
C3F_ERROR_ALIGNMENT	This error indicates that either the dest or the size isn't valid.	1) Check if the parameter dest is on the word (32-bit or 4-byte) boundary, i.e. the dest is multiple of 4. 2) Check if the parameter size is multiple of 4. 3) Check if the parameter source is multiple of 4.
C3F_ERROR_SRC_DEST_VERIFY	This is not an error. It just indicates that destination data in C3F and source can not match.	1) Check the correct source and destination addresses 2) Reprogram data into flash memory
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.8.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write

3.8.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the FlashVerify function executable. Use a hex viewer utility or the ASCII option in the debugger memory window dump once the FlashVerify function has been loaded, to view this revision string in the binary image. The FlashVerify revision string is formatted as follows:

PPCC3FFV_{xyz}

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
FV	Driver Routine	FlashVerify acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.9 ChangeCensor

3.9.1 Description

This function will change the state of the specified module's sensor bit.

A check is performed to determine if the selected module is accessible. If this check should fail, the appropriate error code will be returned.

This function can only change the sensor bit of one module at a time.

3.9.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 21.
3. Determine the number of on-chip C3F modules from the part number.
4. Check input argument, sensor value to ensure its value is valid. If the value is invalid set return value to C3F_ERROR_CENSOR_VALUE and go to 21.
5. Check input argument, device mode to ensure its value is valid. If the value is invalid set return value to C3F_ERROR_CENSOR_DEVICE_MODE and go to step 21.
6. Check input argument, module number to ensure its value is valid. If the value is invalid set return value to C3F_ERROR_CENSOR_MODULE and go to step 21.
7. Calculate the absolute address of the C3F control registers from the arrayBase input.
8. Check FLASHID to ensure the given part owns sensor feature. If sensor feature is unavailable, set return value with C3F_INFO_FLASHID and go to step 21
9. If the desired sensor value equals the current sensor value of CENSOR[0:1], set return value with C3F_OK and go to step 21
10. Confirm that the sensor state transition is valid. If not, set the return value to C3F_ERROR_CENSOR_INVALID_REQUEST and go to step 21.
11. Disable interrupt requests.
12. Save block protect bits, small block protects bits and small block enable bits for restoration.

13. If clear sensor, unprotect entire module, disable small blocks, enable all module blocks and set PE=1. Otherwise, clear PE=0.
14. Set CSC = 1 and SES = 1 in the C3FCTL register.
15. If clear sensor, do an erase interlock write and if IWS = 0 this write is to array and if IWS=1 this write is to CENSOR bit(s) directly. Otherwise, write a 1 to the CENSOR bit(s) to be set
16. Read the C3FCTL register until HVS = 0. (While HVS is high, the CallBack function will be invoked repeatedly to respond to time-critical events.)
17. Read the C3FCTL register to confirm PEGOOD =1. If PEGOOD = 0, set the return value to C3F_ERROR_CENSOR_PEGOOD and go to step 21.
18. Write EHV = 0 in the C3FCTL register.
19. Write SES = 0 and CSC = 0 in the C3FCTL register.
20. Deselect blocks and restore block protect bits, small block protects bits and small block enable bits.
21. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.9.3 Definition

```

UINT32 ChangeCensor ( UINT8 enabledBDM,
                      void(*CallBack)(void),
                      UINT8 module,
                      UINT8 censorValue ,
                      UINT8 deviceMode,
                      UINT32 arrayBase );

```


3.9.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	<p>If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled;</p> <p>If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled;</p> <p>If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.</p>
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
module	UINT8	Module selected for a censor bit change.	<p>0 = module A</p> <p>1 = module B</p>
censorValue	UINT8	New Censor value to be programmed.	<p>0 = cleared censor</p> <p>1 = no censor</p> <p>2 = no censor</p> <p>3 = information censor</p>
deviceMode	UINT8	Specify the device censor mode when this function is executed and also determine interlock write select.	<p>0 = censored device mode and IWS = 0</p> <p>1 = uncensored device mode and IWS = 0</p> <p>2 = censored device mode and IWS = 1</p> <p>3 = uncensored device mode and IWS = 1</p> <p>Other value is invalid.</p>
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Bit map for the input parameter enabledBDM

Bit	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

Bit map for the input parameter deviceMode

Bit	bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	IWS flag	Mode flag

Notes:

- 1) It is imperative that the user supply the correct value for this input parameter, otherwise, the results will be unpredictable.
- 2) The Mode flag should be cleared for uncensored mode (boot from flash only) and set for censored mode (BDM, boot from external ROM, slave).
- 3) The IWS flag should match the IWS bit (bit 15) in the first shadow row of the module selected for the censor change.

3.9.5 Return Values

Type	Description	Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_CENSOR_MODULE C3F_ERROR_CENSOR_VALUE C3F_ERROR_CENSOR_DEVICE_MODE C3F_ERROR_CENSOR_INVALID_REQUEST C3F_ERROR_CENSOR_PEGOOD C3F_INFO_FLASHID C3F_ERROR_PARTID

3.9.6 Tips

This function depends on correct values in bit 7 and bit 6 of the deviceMode input argument to determine the censor state transition and interlock write. Therefore, it is the user's responsibility to provide the correct settings. The mode (bit 7) refers to the way memory is executed at reset. If execution begins from flash, then the mode is uncensored. If execution begins from external memory, BDM, or as a slave processor, then the mode is censored. The IWS (bit 6) refers to the Interlock Write Select setting, bit 15 in the first shadow word of the relevant module.

When returning, for both modules the function will clear all the BLOCK bits of C3FCTL. The protect bits will be preserved. A device reset may be necessary to allow a 11 to 00 transition.

To clear censor bits, all the contents in the relevant module will be erased, including the main array and the shadow row.

This function only makes one pass, either erasing BOTH bits to zero, or programming one or two bits to a one. The table below shows the allowed censor transitions:

Allowed ChangeCensor() Transitions:

Current censor value	Want CensorValue = 00	Want CensorValue = 01	Want CensorValue = 10	Want CensorValue = 11
00	Same	Allowed	Allowed	Allowed
01	Allowed	Same	NO	Allowed
10	Allowed	NO	Same	Allowed
11	Allowed	NO	NO	Same

If the transition is to the same state, ChangeCensor() does nothing, but does return C3F_OK. If the desired state is not allowed, then first go to state 00, then choose the desired state.

When running in flash memory, do NOT change the censor bits of the module that the function is resident in.

3.9.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_INFO_FLASHID	Module FLASHID = 0	In early release FLASHID = 0 parts, no censor operation is possible. Get a new part.
C3F_ERROR_CENSOR_MODULE	The specified flash Module does not exist	Choose a valid module number from the above Argument list.
C3F_ERROR_CENSOR_VALUE	The specified censorValue is over 3, and does not exist	Choose a valid censorValue from the above Argument list.
C3F_ERROR_CENSOR_DEVICE_MODE	The specified deviceMode is over 3, and does not exist	Choose a valid deviceMode from the above Argument list.
C3F_ERROR_CENSOR_INVALID_REQUEST	User requested an invalid censor transition, or device is in censored deviceMode and cannot do it.	May boot from flash to be in deviceMode 1. ACCESS and FIC in C3FMCR also affect allowed transitions. For more detailed, may refer to the latest C3F specification about Censorship and Non-Censorship Accesses table.
C3F_ERROR_CENSOR_PEGOOD	Censor transition failed.	DeviceMode may be incorrect, or ACCESS and FIC may interfere. A reset may be required before the part can change states.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.9.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write
C3FMCR	PROTECT	Read, Write
	ACCESS, FIC	Read
	CENSOR	Read, Write
C3FMCRE	SBEN, SBPROTECT	Read, Write
	FLASHID	Read
C3FCTL	EHV, PE, SES, CSC BLOCK, SBLOCK	Write
	HVS, PEGOOD	Read

3.9.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the ChangeCensor function executable. Use a hex viewer utility, or use the ASCII option in the debugger memory window dump once the ChangeCensor function has been loaded, to view this revision string in the binary image. The ChangeCensor revision string is formatted as follows:

PPCC3FCCxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
CC	Driver Routine	ChangeCensor acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.10 CheckSum

3.10.1 Description

This function will simply calculate the checksum for the selected block of flash memory by summing 32-bit words. If an overflow occurs, during checksum, the carry bit will be ignored. A check is performed to determine whether the selected memory array is accessible. If it is not, an appropriate error code is returned. The finally calculated checksum is stored in a 32-bit word variable pointed to by “sum”.

3.10.2 Procedure

1. Set the return value to C3F_OK.
2. Get the part number from the PARTNUM bits in the IMMR register to verify that the on-chip flash is supported. If not, set the return value to C3F_INFO_PARTID and go to step 9.
3. Determine the number of on-chip C3F modules and the relative C3F array address range from the part number.
4. Calculate the relative destination address from the destination input and the relative C3F array address range
5. Verify that the relative destination address and the size input are each word (4-byte) aligned. If not, set the return value to C3F_ERROR_ALIGNMENT and go to step 9.
6. Verify that the relative destination range (relative destination address + size) and the size fall within the relative C3F array address range. If not, set the return value to C3F_ERROR_ARRAY_RANGE and go to step 9.
7. Calculate the absolute destination address from the relative destination address and the arrayBase input.
8. Accumulate the sum of each word (4 bytes) into the sum argument. (During the summing operation, the CallBack function will be invoked periodically to respond to time-critical events.)
9. If the BDM flag bit in the enabledBDM input argument is set, enter BDM mode by executing the “sc” instruction with external interrupts disabled. Otherwise, if the Interrupt flag bit in the enabledBDM input argument is set, return to the calling program with external interrupts enabled. Otherwise return to the calling program with external interrupts disabled.

3.10.3 Definition

```
UINT32 CheckSum ( UINT8 enabledBDM,  
  
                  void (*CallBack)(void)  
  
                  UINT32 dest,  
  
                  UINT32 size,  
  
                  UINT32 *sum,  
  
                  UINT32 arrayBase );
```

3.10.4 Arguments

Argument	Type	Description	Range
enabledBDM	UINT8	BDM select flag and interrupt option	If enabledBDM = XXXX-XXX1b, enter BDM mode with external interrupt disabled; If enabledBDM = XXXX-XX10b, return to calling program with external interrupts enabled; If enabledBDM = XXXX-XX00b, return to calling program with external interrupts disabled. Refer to below bitmap.
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.
dest	UINT32	Destination address to be summed in flash memory.	You may provide an absolute or relative address, since this input is masked to the size of internal flash address space. It should fall into C3F flash module.
size	UINT32	Size, in bytes, of the flash region to check sum.	Must be multiples of a 32-bit word. If size equals to 0, C3F_OK will be returned.
sum	UINT32 *	Returned sum value will be stored here.	0x000000000 - 0xFFFFFFFF. Note that this value is only valid when the function returns C3F_OK. Its combination with dest should fall into C3F flash module.
arrayBase	UINT32	Points to beginning flash absolute address	This is arrayBase after FlashInit() returns with &arrayBase initialized.

Notes:

The dest argument is the relative address. It will be modified according to IMMR (Internal Memory Map Register) register. In the cases of a 1MB flash or a 512KB flash, this input will first be ANDed with 0xFFFFF or 0x7FFFF respectively, then offset in memory space according to IMMR to produce the absolute flash address.

Bit map for the input parameter enabledBDM

Bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Interrupt flag	BDM flag

Note: If the BDM flag is set, the Interrupt flag will be ignored and external interrupts will be disabled.

3.10.5 Return Values

Type	Description	Possible Values
UINT32	Successful completion or error value.	C3F_OK C3F_ERROR_ALIGNMENT C3F_ERROR_ARRAY_RANGE C3F_ERROR_PARTID

3.10.6 Tips

The function is designed to only checksum flash memory. The input dest parameter is forced to reside in flash memory range. Do NOT check blocks in cleared sensor or information sensor modules. It may cause machine-check or check-stop exceptions.

If size = 0, the function returns C3F_OK and sum value = 0x00000000. The content of sum is valid only when C3F_OK returned. This function is word, not byte oriented. It operates on 4 bytes at a time.

When executing from flash memory, do NOT check the sum of the shadow row of the module that the function is resident in. Switching to shadow row will temporarily hide the main flash array.

3.10.7 Troubleshooting

Returned Error Bits	Description	Solution
C3F_ERROR_ARRAY_RANGE	The area specified by dest and size is out of the valid C3F array range.	1) Check if dest is out of C3F array range. 2) Check if dest+size is out of C3F array range. Note the size of C3F array is 1M bytes or 512K bytes depending on the given part.
C3F_ERROR_ALIGNMENT	This error indicates that either the dest or the size isn't valid.	1) Check if the parameter dest is on the word (32-bit or 4-byte) boundary, i.e. the dest is a multiple of 4. 2) Check if the parameter size is a multiple of 4. 3) Check if the parameter source is a multiple of 4.
C3F_ERROR_PARTID	Current flash driver cannot support the given part.	Check the part number.

3.10.8 Affected Register

Name	Bit	Description
IMMR	PARTNUM	Read
MSR	EE	Write

3.10.9 Revision String

An eleven-character ASCII text revision string is appended to the end of the C hex array or S-record format of the CheckSum function executable. Use a hex viewer, or the ASCII option in the debugger memory window dump once the CheckSum function has been loaded, to view this revision string in the binary image. The CheckSum revision string is formatted as follows:

PPCC3FCSxyz

where:

Item	Use	Description
PPC	Platform	Power PC CPU
C3F	Flash	Flash technology acronym
CS	Driver Routine	CheckSum acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

Appendix A: Performance Data

A.1 Code Size

	Real Size (bytes)
FlashInit	420
FlashCheckShadow	360
FlashErase	560
BlankCheck	376
FlashProgram	620
FlashVerify	412
Checksum	348
ChangeCensor	824
CallBack function	0
Total	3920

Notes:

1. The code size is determined using the Diab C/C++ 4.4a compiler.
2. The General Market C3F Driver follows EABI convention.

A.2 Stack Size

	Real Size (bytes)
FlashInit	24
FlashCheckShadow	32
FlashErase	48
BlankCheck	56
FlashProgram	80
FlashVerify	64
Checksum	48
ChangeCensor	64
CallBack function	N.A.

Notes:

1. The stack usage is determined using the Diab C/C++ 4.4a compiler.
2. The General Market C3F Driver follows EABI convention.

A.3 Program / Erase Times

	MPC565			MPC563
	Revision 0	Revision A	Revision B	Revision 0A
Program Time	2441 mS	2451 mS	2374 mS	1188 mS
Erase Time	1003 mS	15157 mS	1791 mS	1682 mS
Clear Censor Time (0xFF->0xFF)	1515 mS	14584 mS	2280 mS	2212 mS
Clear Censor Time (0x00->0xFF)	992 ms	14042 ms	1759 ms	1719 ms
Set Censor Time	112 uS	112 uS	112 uS	112 uS

Notes:

1. Program time is measured while programming all the blocks from 0xFF to 0x00, in sets of 128 words.
2. Erase time is measured while erasing all the blocks from 0x00 to 0xFF.
3. The time for the preprogramming step during an erase operation is less when the original data is all zeros.
4. The clear censor operation erases all the blocks in a flash array. Clear censor time is measured for two cases: (1) the original data in the array is all ones, and (2) the original data in the array is all zeros.
5. The timer code uses the C-array General Market Driver.
6. Environment: temperature = 21.0 degrees, Vflash = 5.00V.
7. RCPU is in non-serialized mode, with no show cycles.

A.4 Callback Period

Each driver component has been tested to verify that the maximum callback period is no longer than 100 μ S at 40 MHz system clock speed.

Function	Description	Maximum Callback Period (uS)	
		MPC563	MPC565
FlashInit	N/A	11	17
FlashErase	All blocks	10	15
BlankCheck	All blocks	92	94
FlashCheckShadow	Module A shadow row	7	7
FlashProgram	All blocks	11	11
FlashVerify	All blocks	92	93
Checksum	All blocks	91	93
ChangeCensor	Clear operation	14	14
ChangeCensor	Set operation	12	12

Notes:

1. Callback period for FlashProgram is measured while programming all the blocks in sets of 128 words.
2. Callback period for FlashVerify is measured while verifying all the blocks in sets of 128 words.
3. The only timer code for measuring the CallBack period is in the CallBack function.
4. The timer code uses the C-array General Market Driver.
5. Environment: temperature = 21.0 degrees, Vflash = 5.00V.
6. RCPU is in non-serialized mode, with no show cycles.