General Market CMF Driver for MPC555

User's Manual

Non-Volatile Memory Technology Center

Motorola SPS

Change History

Version	Date	Author	Comments
1.0.0	May 18, 2001	Vettey Yan	Initial version
1.0.1	May 24, 2001	Vettey Yan	Updated after review
1.0.2	May 24, 2001	Chen He	Integrate API Spec into user manual
1.0.3	May 24, 2001	Chen He	Add more info for system clock mismatch to ParallelProgram's troubleshooting
1.0.4	May 25, 2001	Chen He	Updated after Beijing team's review

Table of Contents

Chapter 1: Introduction

1.1 Overview

The General Market CMF Driver for MPC555 provides the following driver functions:

- ParallelInit
- ParallelErase
- BlankCheck
- ParallelProgram
- ParallelVerify
- CheckSum
- ChangeCensor

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 for embedded applications. Furthermore, a global data flag can be configured to choose between two function return modes:

- A normal stackframe return for embedded applications, and
- A switch from run mode to BDM mode to signal the host BDM controller that the function is complete

To support concurrency, each GMD function accepts a user-supplied callback function as an argument. In a polling environment, the driver functions periodically pass control to the callback function so servicing of communication ports, watchdog timers, and other periodic activities can proceed concurrently with flash operations. Worst case service periods are no longer than 100 μS for any function (See Appendix).

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

- **Binary:** This file format is tool-independent and can be dynamically loaded at run-time.
- **C-array:** This file format is tool-independent and can be automatically linked with the user's application code at build-time.
- **S-Record:** This file format can be used as the only target resident code for 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.

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 four categories of testing:

- ***API Testing***
  - There are 14 main categories of API and functional testing, with between 5 and 87 test cases per category.
- ***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.
- ***Specification Conformance***
  - The data generator is tested to show that the pulsing scheme defined in the electrical specification is properly customized for the desired system clock speed.
  - Program and erase pulse widths initiated by the driver functions are measured so prove that the pulsing scheme conforms to the electrical specification.
- ***Quality of Programming***
  - Voltage Threshold (Vt) Scans are measured for a physical checkerboard pattern to show that the driver functions properly manage the population of bits in the flash memory array. This test is capable of detecting timing errors, over-programming due to missed margin reads, under-programming and under-erasing due to incorrect margin read completion tests, and other program/erase anomalies.

1.3 Documentation Reference

MPC555 User Manual: MPC555/556UM/AD

SqueeZard MPC5xx Code Compression Tool User's Manual

1.4 System Requirements

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

The C-array demos were developed and tested with the Windriver Diab C compiler v4.3 and SDS Singlestep debugger v7.5. However with minor modifications the standard uncompressed elf files should be capable of source-level debugging with other tool-sets.

The S-Record demos were developed using SDS Singlestep scripts to control the target resident GMD functions.

The binary executable drivers and C language header files are tool-independent so the drivers can be integrated into a wide range of application environments by using a variety of development tools.

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 write only to CMF control registers, CMF main array and CMF shadow array locations. Initialization of system clock speed, the module mapping register, the shadow information words, and other MPC555 system functions is the responsibility of the user. Sample system initializations can be found in the demonstration code provided with this release.

2.2 C-Array Driver Format for Embedded Applications

The c-array GMD function format is intended to simplify automated builds of 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 simple C-language function calls. 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 execute a normal 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 by executing a

subroutine return. At this point error return codes and 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:

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 GMD functions and the data object:

```
set SAR=sar.exe
%SAR% ..\gmd_driver\gmd_ppc_cmf_300_A61_400.s19    temp_io.s19     0x3F9800
%SAR% ..\gmd_driver\gmd_pi.s19                     temp_pi.s19     0x3F9C00
%SAR% ..\gmd_driver\gmd_pe.s19                     temp_pe.s19     0x3FA000
%SAR% ..\gmd_driver\gmd_bc.s19                     temp_bc.s19     0x3FA800
%SAR% ..\gmd_driver\gmd_pp.s19                     temp_pp.s19     0x3FAC00
%SAR% ..\gmd_driver\gmd_pv.s19                     temp_pv.s19     0x3FB400
%SAR% ..\gmd_driver\gmd_cs.s19                     temp_cs.s19     0x3FB800
%SAR% ..\gmd_driver\gmd_cc.s19                     temp_cc.s19     0x3FBC00
```

```
copy /B temp_io.s19+temp_pi.s19+temp_pe.s19+temp_bc.s19+temp_pp.s19+temp_pv.s19+ temp_cs.s19+
temp_cc.s19 ..\gmd_driver\gmd.s19

del temp*.s19 /q
```

The batch file 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 SQUEEZARD.EXE, the ELF file compression application, treats these file formats as data. Since this application compresses code but not data, maximum compression efficiency cannot be achieved with these GMD function formats. However the object file format, precompiled with code compression enabled, can be linked to your application and then compressed for maximum compression efficiency.

Chapter 3: API Specification

3.1 General Overview

This section defines function return codes, global data structures, and function definitions. 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 Values

ID	Value	Description
CMF_OK	0x00	The requested operation was successful.
CMF_ERROR_ARRAY_EMR_FAIL	0x05	The erase margin read for the main array has failed. The maximum erase pulse count was exceeded and the requested blocks were not successfully erased.
CMF_ERROR_SHADOW_EMR_FAIL	0x06	The erase margin read for the shadow row has failed. The maximum erase pulse count was exceeded. The requested block contained a shadow row that was not successfully erased.
CMF_ERROR_ARRAY_PMR_FAIL	0x07	The program margin read of the main array has failed. This error indicates the maximum number of program pulses was exceeded and the selected program pages were not successfully programmed.
CMF_ERROR_SHADOW_PMR_FAIL	0x08	The program margin read for the shadow row has failed. This error indicates the maximum number of program pulses was exceeded.
CMF_ERROR_NOT_BLANK	0x09	The specified memory location is not blank.
CMF_ERROR_SRC_DEST_VERIFY	0x0A	The specified flash data does not match the source data.
CMF_ERROR_PROTECTED_BLOCK	0x0B	This value will be returned when trying to erase, program or verify protected blocks which are specified in the tCMF_PART descriptor or the enabledBlocks doesn't contain relate shadow row when trying to operate shadow row.
CMF_ERROR_CENSOR_DEVICE_MODE	0x0D	Pass in an invalid device censor mode (>1) when changing censor.
CMF_ERROR_CENSOR_MODULE	0x0E	Pass in an invalid module number (>1) when changing censor.

CMF_ERROR_CENSOR_INVALID_REQUEST	0x0F	The request to change censor violates the hardware specification when changing censor.
CMF_ERROR_CENSOR_CHANGE	0x10	The pulse exceeds the max pulse for censor when changing censor.
CMF_ERROR_CENSOR_VALUE	0x11	Pass in an invalid censor value to be changed (>3) when changing censor.
CMF_ERROR_SHADOW_RANGE	0x12	An invalid shadow range was passed.
CMF_ERROR_ALIGNMENT	0x13	The address alignment is different from the required alignment.
CMF_ERROR_ARRAY_RANGE	0x14	An invalid array range was passed.
CMF_ERROR_SYSCLK_MISMATCH	0x15	The current system clock on the target system doesn't match the reference value stored in the descriptor.

3.3 Information Values from ParallelInit

ID	Value	Description
CMF_INFO_CTRL_REGS_INIT_A	0x80	An error occurred during initialization of the CMF module A's control registers.
CMF_INFO_CTRL_REGS_INIT_B	0x40	1.An error occurred during initialization of the CMF module B's control registers. 2.The enabledBlocks in part descriptor exceeds the range. Module B has only 6 blocks and the enabledBlocks[1] in part descriptor should be from 0x00 to 0xFC.
CMF_INFO_NO_VPP_A	0x20	High voltage is not present for the CMF module A. Program and erase operations are not possible for the CMF module A.
CMF_INFO_NO_VPP_B	0x10	High voltage is not present for the CMF module B. Program and erase operations are not possible for the CMF module B.
CMF_INFO_CENSOR_MODULE_A	0x08	Module A is cleared censor or information censor.
CMF_INFO_CENSOR_MODULE_B	0x04	Module B is cleared censor or information censor.
CMF_INFO_INVALID_ARRAY_BASE	0x02	The array base address mismatches between the address specified in the tCMF_PART descriptor and IMMR register setting for the CMF module.
CMF_INFO_BACKUP_CLOCK	0x01	The target system is currently in backup clock mode.

3.4 Constants, Type Defines and Structures

3.4.1 General

Name	Value	Description
FALSE	0	Boolean False
TRUE	(! FALSE)	Boolean True

*typedef unsigned char **BOOL**;*

*typedef signed char **INT8**;*

*typedef unsigned char **UINT8**;*

*typedef volatile signed char **VINT8**;*

*typedef volatile unsigned char **VUINT8**;*

*typedef signed short **INT16**;*

*typedef unsigned short **UINT16**;*

*typedef volatile signed short **VINT16**;*

*typedef volatile unsigned short **VUINT16**;*

*typedef signed long **INT32**;*

*typedef unsigned long **UINT32**;*

*typedef volatile signed long **VINT32**;*

*typedef volatile unsigned long **VUINT32**;*

where :

Mnemonic	Size	Description
BOOL	8-bits	unsigned char
INT8	8-bits	signed char
UINT8	8-bits	unsigned char
VUINT8	8-bits	volatile unsigned char
INT16	16-bits	signed short
UNIT16	16-bits	unsigned short
VUINT16	16-bits	volatile unsigned short
INT32	32-bits	signed long
UNIT32	32-bits	unsigned long
VUINT32	32-bits	volatile unsigned long

3.4.2 CMF Module Descriptor

```
typedef struct
{
   VUINT32 *cmfMCR;
   VUINT32 *cmfTST;
   VUINT32 *cmfCTL;
   UINT32 cmfMCRInit;
} tCMF_MODULE;
```

where :

Structure Member	Type	Description
cmfMCR	VUINT32 *	The address of the CMF Module Configuration Register.
cmfTST	VUINT32 *	The address of the CMF Test Register.
cmfCTL	VUINT32 *	The address of the CMF High Voltage Control Register.
cmfMCRInit	UINT32	Value applied to the CMFMCR during module initialization. This variable is applied during execution of the ParallelInit function.

3.4.3 CMF Part Descriptor

Name	Value	Description
CMF_MODULES	2	The number of CMF modules in the part.

```
typedef struct
{
   tCMF_MODULE cmf[CMF_MODULES];
   UINT32 arrayBase;
   UINT32 pulseCnt;
   UINT8 enabledBlocks[CMF_MODULES];
   BOOL enableBDM;
   UINT8 reserved;
} tCMF_PART;
```

where:

Structure Member	Type	Description
cmf [CMF_MODULES]	tCMF_ MODULE	Array of CMF module descriptors. One instance per CMF module.
ArrayBase	UINT32	CMF array base address. This value should be compatible with the IMMR register and only the following values are valid: 0x0000 0000 0x0040 0000 0x0080 0000 0x00C0 0000 0x0100 0000 0x0140 0000 0x0180 0000 0x01C0 0000 Note that it is the user's responsibility to set the IMMR and set the related value to this field before calling any GMD functions.
PulseCnt	UINT32	Total pulse count for the erase or program operation.
EnabledBlocks [CMF_MODULES]	UINT8	Bitmask indicating the block(s) to be exercised. The MSB of each element represents block zero. Block numbers increase moving from left to right. The range is 0x00 to 0xFF for module A and 0x00 to 0xFC for module B. This field can be used to protect individual blocks from program, erase and verification operations. Only enabled blocks will be programmed, erased or verified. If the operations (ParallelErase, ParallelProgram and ParallelVerify) access the protected blocks, an error CMF_ERROR_PROTECTED_BLOCK will be returned. For each bit, 0 = Protect the block 1 = Enable the block for operation
EnableBDM	BOOL	TRUE = Enter BDM after API driver function execution. FALSE = Return to calling program after API driver function execution.
Reserved	UINT8	Reserved space for padding.

3.4.4 CMF Program Algorithm Descriptor

Name	Value	Description
CMF_PPS	9	The number of program Pulse & Amplitude Width Modulation (PAWS) transition steps. While CMF core Release 5 only supports four program PAWS transition steps, nine total elements will be defined. The last element in pawsProgPulses will be set to 0xFFFF to prevent transition to the reset non-existent levels. This value is set to 9 to support P/E algorithm 5.0, 6.0 and 6.1.

```
typedef struct
{
    UINT32 oscClk;
    UINT32 sysClk;
    UINT32 ctlProg[CMF_PPS];
    UINT16 pawsProgData[CMF_PPS];
    UINT16 pawsProgPulses[CMF_PPS];
    UINT8 pawsProgMode[CMF_PPS];
    UINT8 reserved;
    UINT16 maxProgramPulses;
} tCMF_PROGRAM_DATA;
```

where :

Structure Member	Type	Description
oscClk	UINT32	Magnified source clock. The value equals the source clock frequency multiplies 10 and truncates all the numbers after the point. For example: 40 (for 4 MHz oscillator frequency)
sysClk	UINT32	Magnified system clock. The value equals the system clock multiplies 10 and truncates all the numbers after the point. For MPC555, sysClk should be within the range from 80 (i.e. 8 MHz system clock) to 400 (i.e. 40 MHz system clock). Note that GMD functions ParallelProgram will calculate the sysClk according to oscClk and the real settings for the clock module on board. The calculated value should be within the sysClk +/- 1. If out of the range, an error CMF_ERROR_SYSCLK_MISMATCH will be returned.
ctlProg [CMF_PPS]	UINT32	Array for the program CMFCTL timing data applied during each of the nine program PAWS transition steps. The SCLKR, CLKPE, and CLKPM comprise the timing data bit-fields.
pawsProgData [CMF_PPS]	UINT16	Array for the CMFCTL register data used to generate the nine program PAWS transition steps. This determines the value of PAWS, GDB, and NVR bits.
pawsProgPulses [CMF_PPS]	UINT16	Array for the number of program pulses executed before moving to the next program PAWS transition step.
pawsProgMode [CMF_PPS]	UINT8	Array for the program algorithm control switches used during a program PAWS transition step.
Reserved	UINT8	Reserved space for padding.
maxProgramPulses	UINT16	The maximum number of program pulses allowed for page programming.

3.4.5 CMF Erase Algorithm Descriptor

Name	Value	Description
CMF_EPS	9	The number of erase Pulse & Amplitude Width Modulation (PAWS) transition steps. While CMF core Release 5 only supports one erase PAWS transition step, nine total elements will be defined. The last element in pawsErasePulses will be set to 0xFFFF to prevent transition to the reset non-existent levels. This value is set to 9 to support P/E algorithm 5.0, 6.0 and 6.1.
IGNORE_EMR	0x80	This value is defined in pawsEraseMode to indicate to ignore erase margin read during this step of erase.

```
typedef struct
{
    UINT32 oscClk;
    UINT32 sysClk;
    UINT32 ctlErase[CMF_EPS];
    UINT16 pawsEraseData[CMF_EPS];
    UINT16 pawsErasePulses[CMF_EPS];
    UINT8 pawsEraseMode[CMF_EPS];
    UINT8 reserved;
    UINT16 maxErasePulses;
} tCMF_ERASE_DATA;
```

where:

Structure Member	Type	Description
oscClk	UINT32	Magnified source clock. The value equals the source clock frequency multiplies 10 and truncates all the numbers after the point. For example: 40 (for 4 MHz oscillator frequency)
sysClk	UINT32	Magnified system clock. The value equals the system clock multiplies 10 and truncates all the numbers after the point. Note that GMD function ParallelErase will calculate the sysClk according to oscClk and the real settings for the clock module on board. The calculated value should be within the sysClk +/-1. If out of the range, an error CMF_ERROR_SYSCLK_MISMATCH will be returned.
ctlErase [CMF_EPS]	UINT32	Array for the erase CMFCTL timing data applied during each of the nine erase PAWS transition steps. The SCLKR, CLKPE, and CLKPM comprise the timing data bit-fields.
pawsEraseData [CMF_EPS]	UINT16	Array for the CMFCTL register data used to generate the nine erase PAWS transition steps. This determines the value of PAWS, GDB, and NVR bits.
pawsErasePulses [CMF_EPS]	UINT16	Array for the number of erase pulses executed before moving to the next erase PAWS transition step.
pawsEraseMode [CMF_EPS]	UINT8	Array for the erase algorithm control switches used during an erase PAWS transition step. The “ignore erase margin read” option and the “apply extra erase pulse” options are defined here.
reserved	UINT8	Reserved space for padding.
maxErasePulses	UINT16	The maximum number of erase pulses allowed for block erasing.

3.4.6 CMF Censor Algorithm Descriptor

Name	Value	Description
CMF_CPS	9	The number of erase Pulse & Amplitude Width Modulation (PAWS) transition steps. While CMF core Release 5 only supports one erase PAWS transition step, nine total elements will be defined. The last element in pawsCensorPulses will be set to 0xFFFF to prevent transition to the reset non-existent levels. This value is set to 9 to support P/E algorithm 5.0, 6.0 and 6.1.
IGNORE_EMR	0x80	This value is defined in pawsCensorMode to indicate to ignore erase margin read during this step of erase.

```
typedef struct
{
   UINT32 oscClk;
   UINT32 sysClk;
   UINT32 ctlCensor[CMF_CPS];
   UINT16 pawsCensorData[CMF_CPS];
   UINT16 pawsCensorPulses[CMF_CPS];
   UINT8 pawsCensorMode [CMF_CPS];
   UINT8 reserved;
   UINT16 maxCensorPulses;
} tCMF_CENSOR_DATA;
```

where :

Structure Member	Type	Description
oscClk	UINT32	Magnified source clock. The value equals the source clock frequency multiplies 10 and truncates all the numbers after the point. For example: 40 (for 4 MHz oscillator frequency)
sysClk	UINT32	Magnified system clock. The value equals the system clock multiplies 10 and truncates all the numbers after the point. Note that GMD function ChangeCensor will calculate the sysClk according to oscClk and the real settings for the clock module on board. The calculated value should be within the sysClk +/-1. If out of the range, an error CMF_ERROR_SYSCLK_MISMATCH will be returned.
ctlCensor[CMF_CPS]	UINT32	Array for the erase CMFCTL timing data applied during each of the nine erase PAWS transition steps. The SCLKR, CLKPE, and CLKPM comprise the timing data bit-fields.
pawsCensorData [CMF_CPS]	UINT16	Array for the CMFCTL register data used to generate the nine erase PAWS transition steps. This determines the value of PAWS, GDB, and NVR bits.
pawsCensorPulses [CMF_CPS]	UINT16	Array for the number of erase pulses executed before moving to the next erase PAWS transition step.
pawsCensorMode [CMF_CPS]	UINT8	Array for the erase algorithm control switches used during an erase PAWS transition step. The “ignore erase margin read” option and the “apply extra erase pulse” options are defined here.
reserved	UINT8	Reserved space for padding.
maxCensorPulses	UINT16	The maximum number of erase pulses allowed for censor changing.

3.4.7 CMF Compare Data Descriptor

typedef struct

{

UINT32 failingAddr;

UINT32 failingDestData;

UINT32 failingSourceData;

} ***tCMF_COMP_DATA****;*

where :

Structure Member	Type	Description
failingAddr	UINT32	Address of first failing verify or blank check. This address will reference a 32-bit word.
failingDestData	UINT32	Memory data contained at the failing address. This will be a 32-bit word value.
failingSourceData	UINT32	Source data contained at the related offset of the failing address (only used with ParallelVerify). This will be a 32-bit source buffer word value.

3.5 Functions

3.5.1 ParallelInit

3.5.1.1 Description

This function checks the condition of the CMF flash module, flash memory base address, write-locked registers, VPP, system clock and censor bits status for each CMF flash module.

The result returned is a bit combination of target system condition. If everything is OK, it simply returns CMF_OK.

This function need only be called once prior to multiple flash driver operations.

In some cases this function returns CMF_INFO_CENSOR_MODULE_A and CMF_INFO_CENSOR_MODULE_B when executing on CMF core release 5 parts. Users may ignore this information, because censor function is not functional for CMF core release 5.

3.5.1.2 Definition

UINT8 **ParallelInit** (tCMF_PART *cmfPart);

3.5.1.3 Procedure

1. Check whether the actual CMF flash array base address is matched with the user specified base address stored in the descriptor.
2. Initialize the CMFMCR register with the reset value.
3. Check whether the high voltage operations on CMF flash module are possible.
4. Check whether the CMF write-locked registers are protected or not.
5. Check whether the CMF flash module is censored or not.
6. Perform operations 2 through 5 for another flash module if necessary.
7. Check whether the target system clock has lost lock or been in backup mode.
8. Check whether the enabled blocks in module B are out of its valid range.

9. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

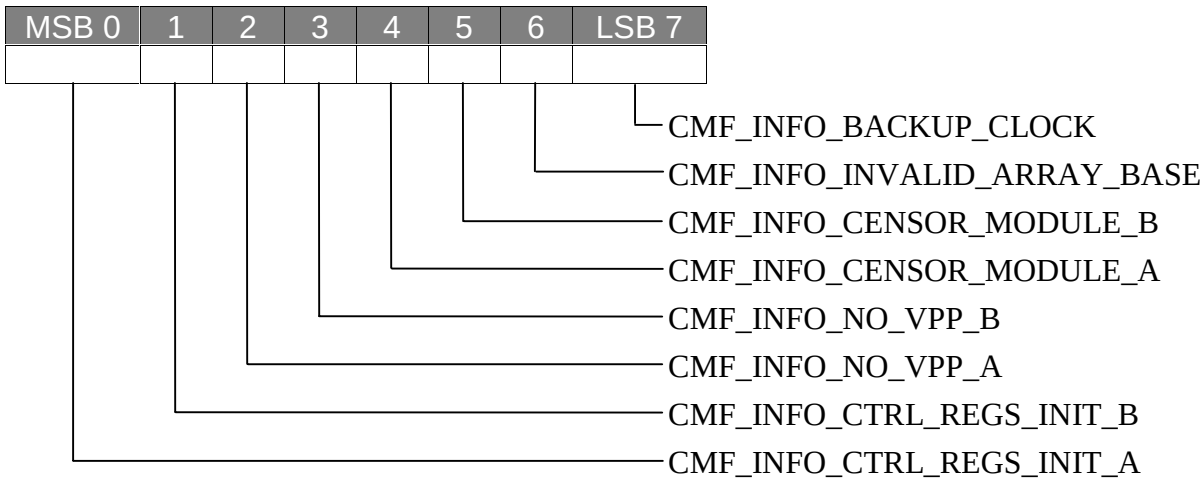
3.5.1.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF part descriptor structure	Address area containing the information of the part to be operated

3.5.1.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or compound information values. This function checks every module. Each bit in the returned value indicates a kind of status of the current operation environment except for CMF_OK.	CMF_OK CMF_INFO_CTRL_REGS_INIT_A CMF_INFO_CTRL_REGS_INIT_B CMF_INFO_NO_VPP_A CMF_INFO_NO_VPP_B CMF_INFO_CENSOR_MODULE_A CMF_INFO_CENSOR_MODULE_B CMF_INFO_INVALID_ARRAY_BASE CMF_INFO_BACKUP_CLOCK

The following diagram explains bits combination for the information values returned from ParallelInit.



3.5.1.6 Tips

This function will clear the EHV and SES bit in CMFCTL.

3.5.1.7 Troubleshooting

Return Value	Description	Solution
CMF_INFO_BACKUP_CLOCK	The current target system is under backup system clock. In backup system clock the following GMD functions can't execute properly: ParallelErase ParallelProgram ChangeCensor	Consult with the MPC555 User Manual to set system clock in normal mode.
CMF_INFO_INVALID_ARRAY_BASE	GMD finds the arrayBase in the part descriptor mismatches the IMMR setting.	Check both IMMR register and the arrayBase in the part descriptor to make sure they are matched.
CMF_INFO_CENSOR_MODULE_B	Module B is in cleared censor or information censor, i.e. censor bit of module B is 0 or 3. Any operation that accesses the main array and shadow row of module B will cause an exception.	Changing censor bits of module B to no censor states by the GMD function ChangeCensor.
CMF_INFO_CENSOR_MODULE_A	Module A is in cleared censor or information censor, i.e. censor bit of module A is 0 or 3. Any operation that accesses the main array and shadow row of module A will cause an exception.	Change censor bits of module A to no censor states by the GMD function ChangeCensor first.
CMF_INFO_NO_VPP_B	High voltage is not present for the CMF module B. Program and erase operations are not possible for all blocks in this module. The following GMD functions can't execute properly when operating module B: ParallelErase ParallelProgram ChangeCensor	Check target board manual to enable VPP.

CMF_INFO_NO_VPP_A	High voltage is not present for the CMF module A. Program and erase operations are not possible for both the boot block and all the other blocks in this module. The following GMD functions can't execute properly when operating module A: ParallelErase ParallelProgram ChangeCensor	Check target board manual to enable VPP.
CMF_INFO_CTRL_REGS_INIT_B	1. The initialization of the CMF module B control register is not successful. Don't try to use the following GMD functions when operating module B: ParallelErase ParallelProgram ChangeCensor 2. The enabledBlocks in part descriptor exceeds the range. Module B has only 6 blocks and the enabledBlocks[1] in part descriptor should be from 0x00 to 0xFC.	Consult with MPC555 User Manual. Check the value of enabledBlocks[1] in the part descriptor and make sure it should be within 0xFC.
CMF_INFO_CTRL_REGS_INIT_A	An error occurred during initialization of the CMF module A's control registers. Don't try to use the following GMD functions when operating module A: ParallelErase ParallelProgram ChangeCensor	Consult with MPC555 User Manual.

3.5.1.8 Affected Register

Name	Bit	Description
IMMR	ISB	Read
CMFMCR	whole register	Write
	LOCK, CENSOR	Read
CMFCTL	EHV, SES	Write
	EPEE	Read
SCCR	BUCS	Read

3.5.1.9 Revision String

A character ASCII text revision string is appended to the end of the ParallelInit 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 ParallelInit function has been loaded. The ParallelInit revision string is formatted as follows:

PPCCMFPIxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
PI	Driver Routine	ParallelInit acronym
x	Major Revision	0 – 9
yz	Minor Revision	00 - 99

3.5.2 ParallelErase

3.5.2.1 Description

This function will erase the specified enabled blocks. 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. This function will store the erase pulse count value in the part descriptor upon exit.

Parameters are range checked on entry, and an appropriate error code is returned if an error is found.

Note that this function always disables the SIE bits of the two modules on exit. That is, the main array is the default exit state.

3.5.2.2 Procedure

1. Check the system clock. Return an error if the system clock register does not match the reference value stored in the descriptor. Refer to Section 3.5.4.6 for more information.

2. Check the selected blocks to be erased are activated in the cmfPart variable.

3. Write PROTECT[0:7] = 0 to disable protection on the array.

4. Set the initial pulse width bit settings to the CMF test register CMFTST according to specified CMF erase algorithm, which is contained in tCMF_ERASE_DATA structure. Write the pulse width timing control fields for an erase pulse in the CMFCTL register and write the BLOCK[0:7] to select the blocks to be erased.

5. Set PE = 1 to configure for erase operation.

6. Set SES = 1 in the CMFCTL register to start an erase sequence.

7. Execute an erase interlock write to any array location on each CMF module.

8. Apply a high voltage pulse to the CMF flash with setting EHV = 1 in the CMFICTL2 register. Poll the CMFICTL1 register until HVS = 0. Then disable high voltage by setting EHV = 0.

9. Perform erase margin read if it is necessary to verify the erase operation, read all locations that are being erased, including the shadow row if the block containing it is erased.

10. If another pulse is required, update the pulse width bit setting and timing control in CMF registers. Then go to Step 8 to initiate another pulse. Otherwise continue with Step 11.

11. Save the total pulse count for the erase operation to the part descriptor.

12. Set SES = 0 in the CMFCTL register to end erase sequence.

13. Clear the BLOCK[0:7] and write PROTECT[0:7]=0xFF to enable protection on the array.

14. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.2.3 Definition

```
UINT8 ParallelErase (   tCMF_PART *cmfPart,
                        tCMF_ERASE_DATA *eraseData,
                        UINT8 *enabledBlocks,
                        void (*CallBack)(void)      );
```

3.5.2.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF descriptor structure	Address area containing the information of the part to be operated
eraseData	tCMF_ERASE_DATA *	Pointer to the erase algorithm data structure	Address area containing the erase algorithm
enabledBlocks	UINT8 *	Pointer to array of module bitmasks. Used to select the blocks to be erased.	The values in this array should be activated in the active blocks contained in the cmfPart variable.
CallBack	Void (*)(void)	Address of void callback function pointer	Any valid void function address

The parameter enabledBlocks is a pointer to an UINT8 data array and the data contained in this array are sorted by module number. The bit map for enabledBlocks[module] is listed in the following table.

Setting any bit to 1 will tag that block to be erased when this function is called.

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

3.5.2.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or error value.	CMF_OK CMF_ERROR_SYSCLK_MISMATCH CMF_ERROR_PROTECTED_BLOCK CMF_ERROR_ARRAY_EMR_FAIL CMF_ERROR_SHADOW_EMR_FAIL

3.5.2.6 Tips

In order to ensure the GMD is using the correct data object for a given system clock, this function performs a system clock check where the register value is compared with a reference register image stored in the descriptor when the data object was built. Refer to Section 3.5.4.6 for more information.

The function can handle module A and module B at the same time if there are two modules in the part. The parameter enabledBlocks[0] is used for selecting blocks in module A while enabledBlocks[1] is used for module B.

If the blocks to be erased have subsidiary shadow rows, that shadow row will also be erased. The user is responsible for saving and restoring the shadow contents if necessary.

The MSB of parameter enabledBlocks represents block 0 in one module and the LSB for block 7 in the same module. The BLOCK[7:0] in CMFCTL has the reversed bit sequence compared with enabledBlocks. See the MPC555 user manual for more information.

If both of the two elements in the enabledBlocks array are set to 0x00, the GMD function will return CMF_OK even though no erase operation is performed.

If executing from a given flash module, the flash driver will not be able to erase any memory blocks within its own module. High voltage disables normal reads of flash memory. For example, if the driver is executing from module A, then any part of module B may be programmed or erased while no part of module A may be programmed or erased.

Do NOT erase the blocks on modules that are in cleared censorship mode or information censorship mode. It may cause machine-check or check-stop exceptions.

3.5.2.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_SYSCLK_ MISMATCH	The current system clock on the target system doesn't match it in the erase algorithm data.	The erase algorithm data made by GMD Data Generator has a system clock item, which indicates the algorithm should run at the specified system clock. Make sure that the system clock is same as that of in the erase algorithm data. Please refer to remarks in 3.5.4.6 for detailed information of this failure.
CMF_ERROR_PROTECTE D_BLOCK	The parameter enabledBlocks is out of the blocks activated in the part descriptor.	Check the parameter enabledBlocks and make sure that all the enabled bits are within the activated blocks in the part descriptor.
CMF_ERROR_ARRAY_E MR_FAIL	The erase margin read for the main array has failed. The maximum erase pulse count was exceeded and the requested block(s) was not successfully erased.	1) Check whether the pulseCnt in the cmfPart exceeds the maximum erase pulse count in the eraseData; 2) Call ParallelInit to make sure that the operation environment is set properly. 3) Verify that proper programming voltage is applied.
CMF_ERROR_SHADOW_ EMR_FAIL	The erase margin read for the shadow row has failed. The maximum erase pulse count was exceeded. The requested block contained a shadow row that was not successfully erased.	1) Check whether the pulseCnt in the cmfPart exceeds the maximum erase pulse count in the eraseData; 2) Call ParallelInit to make sure that the operation environment is set properly. 3) Verify that proper programming voltage is applied.

3.5.2.8 Affected Register

Name	Bit	Description
PLPRCR	MF, DIVF, CSRC	Read
SCCR	DFNL, DFNH	Read
CMFMCR	PROTECT, SIE	Write
CMFTST	NVR, PAWS, GDB	Write
CMFCTL	SCLKR, CLKPE, CLKPM, BLOCK, PE, SES, EHV	Write
	HVS	Read

3.5.2.9 Revision String

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

PPCCMFPExyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
PE	Driver Routine	ParallelErase acronym
x	Major Revision	0 – 9
yz	Minor Revision	00 – 99

3.5.3 BlankCheck

3.5.3.1 Description

This function reads the memory and compares against all ones (erased bitcell logic state, i.e. 0xFFFFFFFF).

Note that the memory area to be checked can be any place including RAM, main array, shadow row and other accessible memory space. The user should ensure that the memory area to be checked is available to the target system.

If the blank check fails, then the cmfCompare data structure is updated with the first failing address and the failing data.

Note that this function always disables the SIE bits of the two modules on exit. That is, the main array is the default exit state.

3.5.3.2 Procedure

1. Check the data alignment first. If the shadow row is selected for BlankCheck, then the shadow row will be enabled and the parameters will be range checked.

2. Check the memory within the given range and compared with 0xFFFFFFFF. Update tCMF_COMP_DATA structure when the memory checked is not blank.

3. Enable main array.

4. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.3.3 Definition

```
UINT8 BlankCheck (      tCMF_PART *cmfPart,
                        UINT32 dest,
                        UINT32 size,
                        BOOL shadow,
                        tCMF_COMP_DATA *cmfCompare,
                        void (*CallBack)(void)      );
```

3.5.3.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF descriptor structure	Address area containing the information defining the array structure of the part to be operated
dest	UINT32	Destination memory address to be checked	Any accessible address aligned with 4-byte boundary in memory
size	UINT32	Memory size (in bytes) to be blank checked	Must align with 4-byte boundary.
shadow	BOOL	Shadow row enable flag	TRUE = Select shadow row FALSE = Select main array, RAM or other accessible memory address
cmfCompare	tCMF_COMP_DATA *	Pointer to the compare data structure. Used to return information.	Address area containing a tCMF_COMP_DATA structure variable for saving test result. The pointed area must be in RAM. The data in this structure is valid only when the function returns CMF_ERROR_NOT_BLANK.
CallBack	void (*)(void)	Address of void call back function pointer	Any valid void function address

3.5.3.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or error value.	CMF_OK CMF_ERROR_ALIGNMENT CMF_ERROR_SHADOW_RANGE CMF_ERROR_NOT_BLANK

3.5.3.6 Tips

If parameter size equals to 0, the function returns CMF_OK if the other parameters are all valid.

The tCMF_COMP_DATA structure will be cleared first before performing the blank check on the specified memory range.

When executing the driver from flash memory, do NOT check the shadow row in the same module that the function is executing from. Reading the shadow row will temporarily make the main flash array unreadable.

Do NOT check the blocks on modules that are in cleared censorship mode or information censorship mode. It may cause machine-check or check-stop exceptions.

3.5.3.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_ALIGNMENT	This error indicates that either the dest or the size does not follow the alignment requirement.	1) Check if the parameter dest is aligned with the word (32-bit or 4-byte) boundary, i.e. the dest is multiple of 4. 2) Check if the parameter size is aligned with 4-byte address boundary.
CMF_ERROR_SHADOW_RANGE	The area specified by dest and size is out of the valid shadow row range when blank checking shadow row.	1) Check if dest is out of shadow range. 2) Check if dest+size is out of shadow range. Note the size of shadow row is 256 bytes.
CMF_ERROR_NOT_BLANK	There is a non-blank word (i.e. not 0xFFFFFFFF) within the area to be checked.	The user can go to the cmfCompare to find both the first non-blank address as well as the actual failing data.

3.5.3.8 Affected Register

Name	Bit	Description
CMFMCR	SIE	Write

3.5.3.9 Revision String

A 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:

PPCCMFBCxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
BC	Driver Routine	BlankCheck acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.5.4 ParallelProgram

3.5.4.1 Description

This function supports both serial (ie single page) programming and parallel (ie multi-page) programming. That is ParallelProgram will program individual pagesets consisting of one or more pages per pageset. In this context a pageset is defined by the set of pages that is taken from the unprogrammed state to the programmed state by pulsing until margin is passed. By definition, each page within a pageset lies at the same offset from the beginning of its block.

In addition, this function will also automatically program buffers containing multiple pagesets. With unlimited buffer space, the maximum number of pagesets that can be programmed in one function call is 512. In most systems the maximum number of pagesets that can be programmed in one function call is limited by the amount of available RAM.

The source data cannot reside in the same module as the destination data because the flash cannot be read during programming. 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. These hardware limitations can be bypassed by copying the code to RAM for execution.

ParallelProgram will capture the total pulse count applied to all the pagesets in the data buffer. That is for two pagesets per buffer, the total pulse count captured in the descriptor equals the total pulse count for pageset number one plus the total pulse count for pageset number two.

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

If it is desired to program less than a full 64 byte program page, the user application must fill the remaining bytes of the given page with 0xFF, so that the original data for those bytes in the flash memory will be retained.

The standard exit state for ParallelProgram, as for all GMD functions, is main array active.

3.5.4.2 Procedure

1. Check the system clock. Return an error if the system clock register does not match the reference value stored in the descriptor. Refer to Section 3.5.4.6 for more information.

2. Check the alignment and check the selected blocks to be programmed are activated in the cmfPart variable.

3. Enable the shadow row or main array according to the shadow parameter. Perform range checking on the parameters.

4. Write PROTECT = 0 to disable protection on the array.

5. Set the initial pulse width bit settings to the CMF test register CMFTST according to the data contained in the structure tCMF_PROGRAM_DATA. Write the pulse width timing control fields for a program pulse in the CMFCTL register and write the BLOCK[7:0] to select the blocks to be programmed.

6. Set PE = 0 to configure for program operation.

7. Set SES = 1 in the CMFCTL register to start a program sequence.

8. Write the source data to the 64-byte array locations to be programmed. The first write defines the block offset used for all pages in a pageset.

9. Apply high voltage on the CMF flash with setting EHV = 1 in the CMFCTL register. Poll the CMFCTL register until HVS = 0. Then disable high voltage by setting EHV = 0.

10. Read at least one word from each of the read pages in all active program pages. That is do program margin reads to test completion of the programming operation. Programming is complete when all sixteen words in each active program page pass the margin read test.

11. If another pulse is required, update the pulse width bit setting and timing control in CMF registers and then go to Step 9 to start another pulse. If programming is complete, continue to Step 12.

12. Set SES = 0 in the CMFCTL register to end program sequence for the pageset.

13. Program next pageset if necessary by going to Step 5.

14. Save the total program pulse count in the descriptor.

15. Clear the BLOCK[7:0] and write PROTECT = 1 to enable protection on the array.

16. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.4.3 Definition

```
UINT8 ParallelProgram (   tCMF_PART *cmfPart,
                          tCMF_PROGRAM_DATA *programData,
                          UINT8 *enabledBlocks,
                          UINT32 source,
                          UINT32 offset,
                          UINT16 pagesetNum,
                          BOOL shadow,
                          void (*CallBack)(void)        );
```

3.5.4.4 Arguments

Argument	Type	Description	Range
CmfPart	tCMF_PART *	Pointer to the CMF descriptor structure	Address area defining the configuration of the part to be operated
programData	tCMF_PROGRAM_DATA *	Pointer to the program algorithm data structure	Address area containing the program algorithm
enabledBlocks	UINT8 *	Pointer to array of module bitmask. Used to select the blocks to be programmed.	The values in this array should be activated in the cmfPart variable..
source	UINT32	Source buffer address to be programmed	This address must align with 4-byte boundary.
offset	UINT32	The offset from the base address of the block	Range: 0x00000000 - 0x00007FC0 This is a zero-based value and must align with 64-byte boundary. The same offset address is used for all blocks selected by enableBlocks.

pagesetNum	UINT16	The number of pagesets, where a pageset is a group of 64 byte data pieces to be programmed.	Range: 0 – 512 (0 – 4 for shadow rows)
shadow	BOOL	Shadow Row enable flag	TRUE = Select shadow row FALSE = Select only Main Array
CallBack	void (*)(void)	Address of void call back function pointer	Any valid void function address

3.5.4.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or error value.	CMF_OK CMF_ERROR_SYSCLK_MISMATCH CMF_ERROR_ALIGNMENT CMF_ERROR_PROTECTED_BLOCK CMF_ERROR_SHADOW_RANGE CMF_ERROR_ARRAY_RANGE CMF_ERROR_SHADOW_PMR_FAIL CMF_ERROR_ARRAY_PMR_FAIL

3.5.4.6 Tips

Programming can only change bits from logic “1” to logic “0”. The area to be programmed should be erased first. Note that the function will NOT check this issue. To make sure the data are programmed correctly, users should use ParallelVerify to verify between the source data and programmed data.

In order to ensure the GMD is using the correct data object for a given system clock, this function performs a system clock check where the register value is compared with a reference register image stored in the descriptor when the data object was built.

The following listing shows the source code that contributes to the error code of CMF_ERROR_SYSCLK_MISMATCH:

```
//macro definitions used for system clock checking
#define MACRO_PLPRCR(ARRAY_BASE)                    \
        (*(UINT32*)(ARRAY_BASE + MPC555_PLPRCR))
#define MACRO_SCCR(ARRAY_BASE)                      \
        (*(UINT32*)(ARRAY_BASE + MPC555_SCCR))
```

```
#define MACRO_CALCULATE_SYSTEM_CLOCK(ARRAY_BASE, MAGNIFIED_OSCCLK)\
(                                                                   \
        (                                                           \
                (                                                   \
                        ( MAGNIFIED_OSCCLK << 5)                    \
                        *                                           \
                        ( ( (MACRO_PLPRCR(ARRAY_BASE) & MPC555_PLPRCR_MF) \
                        >> 20 ) + 1 )                               \
                        /                                           \
                        ( ( MACRO_PLPRCR(ARRAY_BASE) & MPC555_PLPRCR_DIVF )\
                        + 1 )                                       \
                )                                                   \
                >>                                                  \
                (                                                   \
                        ( MACRO_PLPRCR(ARRAY_BASE) & MPC555_PLPRCR_CSRC )\
                        ?                                           \
                        ( ( ( MACRO_SCCR(ARRAY_BASE) & MPC555_SCCR_DFNL )   \
                        >> 4 ) + 1 )                                \
                        :                                           \
                        ( MACRO_SCCR(ARRAY_BASE) & MPC555_SCCR_DFNH )       \
                )                                                   \
        ) >>5                                                       \
)

#define SYSCLK_DELTA            1
.
.
.
// check if the system clock is consistent with the intended clock for program algorithm
temp = MACRO_CALCULATE_SYSTEM_CLOCK(cmfPart->arrayBase, programData->oscClk);
temp -= programData->sysClk;
if ((temp & 0x80000000))
        temp = ~temp + 1;
if (temp > SYSCLK_DELTA)
{
        returnCode=CMF_ERROR_SYSCLK_MISMATCH;
        goto BACK;
}
```

The inputs to the system clock checking are ARRAY_BASE, PLPRCR, SCCR, oscClk, and sysClk. If the 'actual' system clock, obtained by calculation based on these inputs, and the 'intended' system clock, obtained from the program descriptor in the GMD data object, differ by more than 1 MHz, then the CMF_ERROR_SYSCLK_MISMATCH error code is returned.

If all the elements in the enableBlocks array are 0x00 and the other parameters are all valid, the function will return CMF_OK even though no programming operation is performed.

If the parameter pagesetNum equals to 0, the function returns CMF_OK if the other parameters are all valid.

The MSB of parameter enabledBlocks represents block 0 in one module and the LSB for block 7 in the same module. The BLOCK[7:0] in CMFCTL has the reversed bit sequence compared with enabledBlocks. See MPC555 user manual for more information.

The function CANNOT 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. The flash driver will also not be able to program data in the same module that the driver is executing from.

Do NOT program the blocks on modules that are in cleared censorship mode or information censorship mode. It may cause machine-check or check-stop exceptions.

3.5.4.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_SYSCLK_ MISMATCH	The current system clock on the target system doesn't match the reference value stored in the descriptor.	The program algorithm data made by GMD Data Generator contains a user-specified clock register image. Make sure that the system clock is same as the reference value stored in the erase algorithm data. Please refer to Section 3.5.4.6 for more information.
CMF_ERROR_ALIGNME NT	This error indicates that either the source or the offset does not follow the alignment requirement.	1) Check if the parameter source is aligned with the word (32-bit or 4-byte) boundary, i.e. the source is multiple of 4. 2) Check if the parameter offset is aligned with 64-byte boundary. Note that the parameter offset independents from array base. It is zero-based.
CMF_ERROR_PROTECT ED_BLOCK	The parameter enabledBlocks is set to access blocks that are out of the allowable block range as defined in the part descriptor.	1) Check the parameter enabledBlocks and make sure that all the enabled bits are within the block range defined in the part descriptor. 2) When programming the shadow row, check if the enabledBlocks has a shadow row.

CMF_ERROR_SHADOW_RANGE	The area specified by offset and pagesetNum is out of the valid shadow row range.	1) Check if offset is out of shadow range. 2) Check if pagesetNum exceed 4. 3) Check if offset+pagesetNum*64 is less than 256, the size of shadow row.
CMF_ERROR_ARRAY_RANGE	The area specified by offset and pagesetNum is out of the valid block address range.	1) Check if offset is out of block address range. 2) Check if pagesetNum exceed 512. 3) Check if offset+pagesetNum*64 is less than 32*1024. Note that the block size is 32*1024 bytes and one block has 512 pages.
CMF_ERROR_SHADOW_PMR_FAIL	The program margin read for the shadow row has failed. The program pulse has exceeded the maximum value and the selected program pages were not successfully programmed.	1) Check whether the pulse count in the cmfPart exceeds the maximum program pulse count in the programData; 2) Call ParallelInit to make sure that the operation environment is set properly. 3) Verify that proper programming voltage is applied.
CMF_ERROR_ARRAY_PMR_FAIL	The program margin read of the main array has failed. The program pulse has exceeded the maximum value and the selected program pages were not successfully programmed.	1) Check whether the pulse count in the cmfPart exceeds the maximum program pulse count in the programData; 2) Call ParallelInit to make sure that the operation environment is set properly. 3) Verify that proper programming voltage is applied.

3.5.4.8 Affected Register

Name	Bit	Description
PLPRCR	MF, DIVF, CSRC	Read
SCCR	DNFL, DFNH	Read
CMFMCR	PROTECT, SIE	Write
CMFTST	NVR, PAWS, GDB	Write
CMFCTL	SCLKR, CLKPE, CLKPM, BLOCK, PE, SES, EHV	Write
	HVS	Read

3.5.4.9 Revision String

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

PPCCMFPPxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
PP	Driver Routine	ParallelProgram acronym
x	Major Revision	0 – 9
yz	Minor Revision	00 – 99

3.5.5 ParallelVerify

3.5.5.1 Description

This function performs a comparison of interleaved source data and interleaved flash data.

While this function does not verify multiple contiguous pages, it does use the same offset and pageset scheme as the ParallelProgram function. The source buffer will contain the interleaved data that will be compared against a corresponding flash region.

If the verification fails, the tCMF_COMPARE data structure is updated with the mismatch information.

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

Note that this function always exits with the main array active.

Note that the source memory area to be verified against main array or shadow row can be main array, shadow row, RAM and other accessible memory address. The user should ensure that the source memory area to be verified is available to the target system.

3.5.5.2 Procedure

1. Check the alignment and examine whether the selected blocks are enabled or not.

2. Enable the shadow row or main array, according to the shadow parameter. Perform range checking on the parameters.

3. Verify the flash data against the source data. Update the tCMF_COMP_DATA structure when the data are not identical.

4. Enable the main array.

5. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.5.3 Definition

```
UINT8 ParallelVerify (      tCMF_PART *cmfPart,
                            UINT8 *enabledBlocks,
                            UINT32 source,
                            UINT32 offset,
                            UINT16 pagesetNum,
                            BOOL shadow,
                            tCMF_COMP_DATA *cmfCompare,
                            void (*CallBack)(void)      );
```

3.5.5.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF descriptor structure	Address area containing the information of the part to be operated
enabledBlocks	UINT8 *	Pointer to array of module bitmask. Used to select the blocks to be verified	The values in this array should be activated in the active blocks contained in the cmfPart variable.
source	UINT32	Source buffer address to be verified	This address must align with 4-byte boundary.
offset	UINT32	The offset from the start address of the block	Range: 0x00000000 - 0x00007FC0 This is a zero-based value and must align with 64-byte boundary. The same offset address is used for all blocks selected by enabledBlocks.

pagesetNum	UINT16	The number of pagesets, where a pageset is a group of 64-byte program pages to be verified.	Range: 0 – 512 (0 – 4 for shadow rows)
shadow	BOOL	Shadow row enable flag	TRUE = Select shadow row FALSE = Select main array, RAM or other accessible memory address
cmfCompare	tCMF_COMP_DATA *	Pointer to the compare data structure. Used to return verify result information.	Address area containing a tCMF_COMP_DATA structure variable for saving compare result. The pointed area must be in RAM. Note only when the whole function returns CMF_ERROR_SRC_DEST_VERIFY, the result in this structure is valid.
CallBack	void (*)(void)	Address of void call back function pointer	Any valid void function address

3.5.5.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or error value.	CMF_OK CMF_ERROR_ALIGNMENT CMF_ERROR_PROTECTED_BLOCK CMF_ERROR_SHADOW_RANGE CMF_ERROR_ARRAY_RANGE CMF_ERROR_SRC_DEST_VERIFY

3.5.5.6 Tips

DO NOT verify the data between the shadow row and the main array of the same module, because at any time only one of them is visible.

If parameter size equals to 0, the function returns CMF_OK if the other parameters are all valid.

If all the elements in the enableBlocks array are 0x00 and the other parameters are all valid, the function will return CMF_OK.

If parameter pagesetNum equals to 0, the function returns CMF_OK if the other parameters are all valid.

The MSB of parameter enabledBlocks represent block 0 in one module and the LSB for block 7 in the same module. The BLOCK[7:0] in CMFCTL has the reversed bit sequence compared with enabledBlocks. See MPC555 user manual for more information.

The tCMF_COMP_DATA structure will be cleared first before verification is performed.

When executing from flash memory, do NOT verify the shadow row of the same module in that the function is resided. Switching to shadow row will temporarily make the main flash array unreadable.

Do NOT verify the blocks on modules that are in cleared censorship mode or information censorship mode. It may cause machine-check or check-stop exceptions.

3.5.5.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_ALIGNMENT	This error indicates that either the source or the offset does not follow the alignment requirement.	1) Check if the parameter source is aligned with the word (32-bit or 4-byte) boundary, i.e. the source is multiple of 4. 2) Check if the parameter offset is aligned with 64-byte boundary. Note that the parameter offset independents from array base. It is zero-based.
CMF_ERROR_PROTECTED_BLOCK	The parameter enabledBlocks is out of the blocks activated in the part descriptor.	1) Check the parameter enabledBlocks and make sure that all the enabled bits are within the activated blocks in the part descriptor. 2) Check if the enabledBlocks has the related shadow row when verifying shadow row.
CMF_ERROR_SHADOW_RANGE	The area specified by offset and pagesetNum is out of the valid shadow row range when verifying shadow row.	1) Check if offset is out of shadow range. 2) Check if pagesetNum exceed 4. 3) Check if offset+pagesetNum*64 is less than 256, the size of shadow row.

CMF_ERROR_ARRAY_RANGE	The area specified by offset and pagesetNum is out of the valid block address range when verifying main array.	1) Check if offset is out of block address range. 2) Check if pagesetNum exceed 512. 3) Check if offset+pagesetNum*64 is less than 32*1024. Note block size is 32*1024 bytes and one block can be divided into 512 pages.
CMF_ERROR_SRC_DEST_VERIFY	There is a mismatched word (i.e. the source data is different from the data in the flash memory.) within the area to be checked.	The user can go to the cmfCompare to find both the first failing address as well as the actual failing data.

3.5.5.8 Affected Register

Name	Bit	Description
CMFMCR	SIE	Write

3.5.5.9 Revision String

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

PPCCMFPVxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
PV	Driver Routine	ParallelVerify acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.5.6 CheckSum

3.5.6.1 Description

This function performs a simple, byte-wise sum without carry over a range of memory. The calculated checksum is stored in the 8 bit variable pointed to by “sum”.

Note that the memory area to be summed up can be anyplace including RAM, main array, shadow row and other accessible memory address. The user should ensure that the memory area to be summed is available to the target system.

As with all other GMD functions, the default exit state for this function is main array active.

This function can be used for the fast data checking. That is the user can use this function to calculate the sum values of the source data and programmed data respectively and compare them to make sure if the programmed data is correct or not.

3.5.6.2 Procedure

1. If the shadow row is selected for summing, the shadow row will be enabled and the parameters will be range checked.

2. Calculate the byte-wise sum for each byte unit within the given memory range.

3. Enable the main array.

4. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.6.3 Definition

```
UINT8 CheckSum (    tCMF_PART *cmfPart,
                    UINT32 dest,
                    UINT32 size,
                    BOOL shadow,
                    UINT8 *sum,
                    void (*CallBack)(void)    );
```

3.5.6.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF descriptor structure	Address area containing the information of the part to be operated
dest	UINT32	Destination memory address to be summed up	Any accessible address in memory.
size	UINT32	Memory size (in byte) to be summed up	
shadow	BOOL	Shadow row enable flag	TRUE = Select shadow row FALSE = Select main array, RAM or other accessible memory address
sum	UINT8 *	Returned sum value will be stored here.	0x00 - 0xff Note only when the function returns CMF_OK, this value is valid.
CallBack	Void (*)(void)	Address of void call back function pointer	Any valid void function address

3.5.6.5 Return Values

Type	Description	Possible Values
UINT8	Successful completion or error value.	CMF_OK CMF_ERROR_SHADOW_RANGE

3.5.6.6 Tips

If size equals to 0, the sum will be 0x00 with returned value CMF_OK. The content of sum is valid only when CMF_OK returned. This function is byte, not word oriented - it operates only one byte at a time.

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

Do NOT check the blocks on modules that are in cleared censorship mode or information censorship mode. It may cause machine-check or check-stop exceptions.

3.5.6.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_SHADOW_RANGE	The area specified by dest and size is out of the valid shadow row range when shadow row is selected.	1) Check if the dest is out of shadow range. 2) Check if dest+size is out of shadow range (must less than 256). Note the size of shadow row is 256 bytes

3.5.6.8 Affected Register

Name	Bit	Description
CMFMCR	SIE	Write

3.5.6.9 Revision String

A 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:

PPCCMFCSxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
CS	Driver Routine	CheckSum acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

3.5.7 ChangeCensor

3.5.7.1 Description

This function will change the state of the specified module's censor 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 censor bit of one module at a time.

NOTE censor function is not functional for CMF core release 5. This function will get CMF_ERROR_CENSOR_CHANGE when executing on CMF core release 5 parts.

3.5.7.2 Procedure

1. Check the system clock. Return an error if the system clock register does not match the reference value stored in the descriptor. Refer to Section 3.5.4.6 for more information.

2. Check the parameters of censorValue, deviceMode and module.

3. Write PROTECT[0:7] = 0 to disable protection on the array.

4. Check if the censor state transition to be started is permitted.

5. If the operation is to clear censor bits, set PE and BLOCK[0:7] in CMFCTL and clear PROTECT[0:7] in CMFMCR. Otherwise for set censor bits operation, only need to clear PE in CMFCTL register.

6. Set the initial pulse width bit settings to the CMF test register CMFTST according to specified CMF censor algorithm, which is contained in tCMF_CENSOR_DATA structure. Write the pulse width timing control fields in the CMFCTL register for changing censor operation.

7. Set CSC=1 in the CMFCTL register to configure for censor operation.

8. Set SES=1 in the CMFCTL register to start a sequence.

9. If the operation is censor clear, execute an erase interlock write to any CMF array location within the given module. Otherwise for set censor operation, assign the censor value to CENSOR[0:1] bits.

10. Apply high voltage on the NVM bits with setting EHV = 1 in the CMFCTL register and read the CMFCTL register until HVS = 0. Disable high voltage with setting EHV = 0.

11. If the operation is censor clear, censor margin read is required, i.e. read the entire CMF module and the shadow row contained in this module.

12. Compare CENSOR[0:1] bits in the CMFMCR with censor bits value expected. If they are not equal, then update the pulse width bit setting and timing control in CMF registers and go to Step 10 to start another pulse. Otherwise continue to Step 13.

13. Save the total pulse count for the censor operation to the part descriptor.

14. Clear SES in the CMFCTL register. Clear CSC in the CMFCTL register to configure for normal operation.

15. Clear the BLOCK[0:7] and write PROTECT[0:7]=0xFF to enable protection on the array.

16. Enter BDM mode if the enableBDM flag is enabled. Otherwise do a normal return.

3.5.7.3 Definition

```
UINT8 ChangeCensor (    tCMF_PART *cmfPart,
                        tCMF_CENSOR_DATA *censorData,
                        UINT8 module,
                        UINT8 censorValue,
                        UINT8 deviceMode,
                        void (*CallBack)(void)      );
```

3.5.7.4 Arguments

Argument	Type	Description	Range
cmfPart	tCMF_PART *	Pointer to the CMF descriptor structure.	Address area containing the information of the part to be operated.
censorData	tCMF_CENSOR_DATA *	Pointer to the censor algorithm data structure.	Address area containing the erase algorithm. Note that this algorithm is same as the algorithm for erase in this version.
module	UINT8	Module selected for a censor bit change.	0 = module A 1 = module B

censorValue	UINT8	New Censor value to be programmed.	0x00 = cleared censor 0x01 = no censor 0x02 = no censor 0x03 = information censor
deviceMode	UINT8	Specify the device censor mode when this function is executed.	0 = censored mode of the device 1 = un-censored mode of the device only when booting from internal flash
CallBack	void (*)(void)	Address of void call back function pointer.	Any addressable void function address.

3.5.7.5 Return Values

Type	Description	Possible Values
UINT8	Indicates either success or failure type.	CMF_OK CMF_ERROR_SYSCLK_MISMATCH CMF_ERROR_CENSOR_MODULE CMF_ERROR_CENSOR_VALUE CMF_ERROR_CENSOR_DEVICE_MODE CMF_ERROR_CENSOR_INVALID_REQUEST CMF_ERROR_CENSOR_CHANGE

3.5.7.6 Tips

DeviceMode refers to the way memory is executed at reset. If execution begins from flash, then deviceMode will be un-censored = 1. If execution begins from external memory, BDM, or as a slave processor, then the deviceMode is censored = 0.

When clear censor bits, all the contents in the relevant module will be erased, including main array and 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:

Current censor value	New CensorValue = 00	New CensorValue = 01	New CensorValue = 10	New 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 returns CMF_OK.

If the desired state transition is not allowed, users can first clear both censor bits to go to the cleared censorship, and then change the censor bits to the desired state.

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

3.5.7.7 Troubleshooting

Return Value	Description	Solution
CMF_ERROR_SYSCLK_ MISMATCH	The current system clock on the target system doesn't match the reference value stored in the descriptor.	Make sure that the system clock is same as the reference value stored in the censor algorithm data. Please refer to remarks in 3.5.4.6 for detail information of this failure.
CMF_ERROR_CENSOR_ MODULE	The parameter module is invalid.	Check if the module is invalid (>= 2).
CMF_ERROR_CENSOR_ VALUE	The parameter censorValue is invalid.	Check if the censorValue is invalid (>= 4).
CMF_ERROR_CENSOR_ DEVICE_MODE	The parameter deviceMode is invalid.	Check if the deviceMode is invalid (>=2).
CMF_ERROR_CENSOR_I NVALID_REQUEST	The censor state requested to change is invalid.	Refer to the table shows all allowed transitions..
CMF_ERROR_CENSOR_ CHANGE	Censor transition failed.	DeviceMode may be incorrect, or ACCESS and FIC may interfere. A reset may be required before the part can change states.

3.5.7.8 Affected Register

Name	Bit	Description
PLPRCR	MF, DIVF, CSRC	Read
SCCR	DFNL, DFNH	Read
CMFMCR	PROTECT, SIE, CENSOR	Write
	FIC, SCCESS, CENSOR	Read
CMFTST	NVR, PAWS, GDB	Write
CMFCTL	SCLKR, CLKPE, CLKPM, BLOCK, PE, SES, EHV, CSC	Write
	HVS	Read

3.5.7.9 Revision String

A 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, or 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:

PPCCMFCCxyz

where:

Item	Use	Description
PPC	Platform	PowerPC
CMF	Flash	Flash technology acronym
CC	Driver Routine	ChangeCensor acronym
x	Major Revision	0 - 9
yz	Minor Revision	00 - 99

Appendix A :Program and Erase Algorithms

A.1 Algorithm 5.0

Algorithm 5.0 applies to CMF 5 flash modules only.

No. of Pulses	Program Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
900	25.6 µS	N/A	000	N/A	Automatic	Firmware drain ramp pulsing scheme.

No. of Pulses	Erase Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
1	1 S	N/A	000	N/A	Automatic	Perform margin reads after each pulse until all bits pass margin.

A.2 Algorithm 6.0

Algorithm 6.0 applies to CMF 6 and later flash modules only. This is the original algorithm recommended by Motorola for this range of flash module revisions.

No. of Pulses	Program Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
4	256 µS	1	100	1	Mode 4 NL	Negative gate ramp, low range. Perform margin reads after each pulse until all bits pass margin.
4	256 µS	1	101	1	Mode 5NL	
4	256 µS	1	110	1	Mode 6NL	
4	256 µS	1	111	1	Mode 7NL	
20	50 µS	0	100	1	Mode 4NH	Negative gate ramp, high range. Perform margin reads after each pulse until all bits pass margin.
20	50 µS	0	101	1	Mode 5NH	
20	50 µS	0	110	1	Mode 6NH	
Max 48,000	50 µS	0	111	1	Mode 7NH	

No. of Pulses	Erase Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
1	100 mS	1	100	N/A	Mode 4NL	No margin read following this pulse.
1	100 mS	1	101	N/A	Mode 5NL	No margin read following this pulse.
1	100 mS	1	110	N/A	Mode 6NL	No margin read following this pulse.
1	100 mS	1	111	N/A	Mode 7NL	No margin read following this pulse.
1	100 mS	0	100	N/A	Mode 4NH	No margin read following this pulse.
1	100 mS	0	101	N/A	Mode 5NH	No margin read following this pulse.
1	100 mS	0	110	N/A	Mode 6NH	No margin read following this pulse.
20	100 mS	0	111	N/A	Mode 7NH	Perform post-pulse margin reads until all bits pass margin.

A.3 Algorithm 6.1

Algorithm 6.1 applies to CMF 6 and later flash modules only. This algorithm is optimized to improve programming time without affecting initial programming quality.

No. of Pulses	Program Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
4	256 µS	1	100	1	Mode 4 NL	Negative gate ramp, low range. Perform margin reads after each pulse until all bits pass margin.
4	256 µS	1	101	1	Mode 5NL	
4	256 µS	1	110	1	Mode 6NL	
4	256 µS	1	111	1	Mode 7NL	
20	50 µS	0	100	1	Mode 4NH	Negative gate ramp, high range. Perform margin reads after each pulse until all bits pass margin.
20	50 µS	0	101	1	Mode 5NH	
20	50 µS	0	110	1	Mode 6NH	
20	50 µS	0	111	1	Mode 7NH	
Max 24,000	100 µS	0	111	1	Mode 7NH	

No. of Pulses	Erase Pulse Width	NVR	PAWs	GDB	PAWS Mode	Description
1	100 mS	1	100	N/A	Mode 4NL	No margin read following this pulse.
1	100 mS	1	101	N/A	Mode 5NL	No margin read following this pulse.
1	100 mS	1	110	N/A	Mode 6NL	No margin read following this pulse.
1	100 mS	1	111	N/A	Mode 7NL	No margin read following this pulse.
1	100 mS	0	100	N/A	Mode 4NH	No margin read following this pulse.
1	100 mS	0	101	N/A	Mode 5NH	No margin read following this pulse.
1	100 mS	0	110	N/A	Mode 6NH	No margin read following this pulse.
20	100 mS	0	111	N/A	Mode 7NH	Perform post-pulse margin reads until all bits pass margin.

Appendix B : MPC555

B.1 Part Revision vs Algorithm Matrix

Part Revision	CMF Module Revision	Program/Erase Algorithm
G, K	CMF 5	5.0
K1, K2	CMF 6	6.0, 6.1
K3	CMF 7	6.0, 6.1
M	CMF 9	6.0, 6.1

B.2 Code and Stack Size

Component	Component Size (bytes)	Stack Size (bytes)
Global Data (Algorithm 6.1)	264	0
ParallelInit	284	16
ParallelErase	1412	104
ParallelProgram	1824	136
BlankCheck	352	56
ParallelVerify	584	72
CheckSum	312	52
ChangeCensor	1412	120 for clear operation
		88 for set operation
Total	**6444**	N/A
Maximum	N/A	**136**

Note: Compare with CMF Parallel Driver v2.3 sizes – 13824 bytes code size and 184 bytes stack.

B.3 Callback Period

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

Function	Maximum Callback Period (µS)	Description
ParallelInit	10.4	N/A
ParallelErase	13.6	14 blocks
ParallelProgram	62.8	14 pages/pageset 4 pagesets data all 0x0
BlankCheck	17.6	14 blocks
ParallelVerify	18.8	14 pages/pageset 4 pagesets data all 0x0
CheckSum	17.2	14 blocks
ChangeCensor	34.8	Clear operation
	31.2	Set operation

Note: MPC555 Rev K3. Global Data for Algorithm 6.1. System clock speed 40 MHz with 5.0 volts VPP. Callback function contained only real-time clock stopwatch routines. Callback period varies with function parameters, so performance for worst case function parameters is shown.

B.4 Program/Erase Times

Function	Part Revision	P/E Algorithm	Time (s) V3.0	Time (s) V2.3	Δ%	Description
ParallelErase	K	5.0	1.097	1.159	5%	Erase all blocks starting from 0x0.
ParallelProgram	K	5.0	27.44	41.39	34%	14 pages/pageset, all 0x0.
ParallelErase	K3	6.0/6.1	0.913	0.976	6%	Erase all blocks starting from 0x0.
ParallelProgram	K3	6.0	19.614	23.961	18%	14 pages/pageset, all 0x0.
ParallelProgram	K3	6.1	15.714	18.262	14%	14 pages/pageset, all 0x0.
ChangeCensor	K3	6.1	0.856	1.219	30%	Module B Clear
			0.812	1.049	23%	Module B Set

NOTE: Room temperature, nominal Vdd, Vpp=5 V, 40 MHz system clock. Empty callback function for v3.0, no concurrent activities for v2.3. Listed times may not be typical due to part-to-part variation and variation in operating conditions.