MPC55xx H7Fa Standard Software Driver

User's Manual

Document Change History

Version	Date	Author	Comments
0.1	10/18/03	Alan Yu	Draft version
0.2	11/4/03	Chen He	Modifications for preliminary release: Section 2.1, FlashProgram and FlashSuspend.
0.3	11/6/03	Chen He	Preliminary release
0.4	3/5/04	Chen He	Release for MPC5554 Rev0.1 parts
0.5	3/5/04	Chen He	After review with Randy Dees
0.6	3/12/04	Chen He	Add return codes for wrong part ID and wrong password to unlock blocks. Add password as an additional input argument to SetLock function so that the passwords won't appear in the driver codes.
0.7	3/18/04	Morrison Yang	Add description on system registers used for passing parameter and return code. Add description on system initialization in section 2.1. Updated performance data.
0.8	3/30/04	Chen He	Updated section 2.1.
0.9	4/23/04	Alan Yu	Update Table 63 for code size of SSD v3.08
1.0	4/23/04	Chen He	Release for MPC5554 Rev0.3 parts
1.1	7/1/04	Morrison Yang	Add performance data for SSD v3.09
1.2	9/14/04	Alan Yu	Add performance data for SSD v3.10. Add a warning for FlashErase function Update the tips and return code for FlashProgram
1.3	12/02/04	Cloud Li	Update the performance data for SSD v3.11.
1.4	12/7/04	Chen He	Add comment on page-wise programming used in FlashProgram
1.5	03/14/05	Cloud Li	Add return code *H7FA_ERROR_STOP* in FlashInit. And remove the EER checking function.
1.6	03/16/05	Chen He	Add Flash driver version table in Appendix A

Table of Contents

List of Tables

Chapter 1: Introduction

1.1 Overview

The MPC55xx H7Fa Standard Software Driver (SSD) provides the following driver functions:

- *FlashInit*
- *FlashErase*
- *BlankCheck*
- *FlashProgram*
- *ProgramVerify*
- *CheckSum*
- *FlashSuspend*
- *FlashResume*
- *GetLock*
- *SetLock*
- *RWECheck*
- *GetWaitState*
- *SetWaitState*

1.2 Features

The MPC55xx H7Fa Standard Software Driver provides the following features:

- *Small code size and high speed program/erase (P/E) operations*
- *Support Master Mode and Slave Mode for MPC5500 family with H7Fa flash*
- *Position-independent and ROM-able*
- *Concurrency support via callback*
- *Three release formats: library, s-record and binary c-array*
- *Ready-to-use demos illustrates the usage of the driver*

1.3 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 the following tests:

Table 1 System Test Cases

Function Name	Number of Test Cases
FlashInit	7
FlashErase	22
BlankCheck	22
FlashProgram	51
ProgramVerify	21
CheckSum	19
FlashSuspend	14
FlashResume	7
GetLock	3
SetLock	6
RWECheck	3
GetWaitState	2
SetWaitState	2
Total	179

There are test cases to show the drivers work properly when they are:

- integrated into an embedded application;
- operated as stand-alone executables under debug mode;
- support concurrency via callbacks in a polled environment;
- executed from anywhere within internal RAM;
- can be executed from anywhere within internal ROM, i.e. Flash.

1.4 Documentation Reference

- MPC5554 Reference Manual v1.1
- CodeWarrior User Manuals for MPC5500
- MPC5554 GAP TOOL User's Manual

1.5 System Requirements

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

The demos for CodeWarrior debugger included in this release are developed and tested with the Metrowerks CodeWarrior for MPC5500 v1.5.

The demo for GAP debugger tool included in this release is developed with Diab C compiler and tested on Motorola GAP debugger tool v1.0.

1.6 Installation

To install the software on a 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 H7Fa Flash Module Initialization

While the SSD functions may read both H7Fa and non-H7Fa control registers, they only write to H7Fa control registers and the H7Fa arrays. Initializing other MPC55xx system functions, such as setting up MMU entries for memory mapping, is the responsibility of the user.

The followings are restrictions of MMU entry for Flash module when using the SSD:

- The whole flash main array space shall be mapped as a contiguous region;
- All read/write accesses in MAS3 shall be enabled for the flash module; and
- The data cache needs to be disabled or cache-inhibited bit in MAS2 needs to be set for Flash when verifying Flash contents. By default, the data cache is disabled after reset.

Prior to any H7Fa flash module operation using the SSD, it is important to set up the SSD configuration structure and call FlashInit function to initialize/check the flash module. Moreover, it is the user's responsibility to control the hardware locking bits (FLASH_MCR-EPE and FLASH_MCR-BBEPE) and manage software locking registers (FLASH_LMLR, FLASH_HLR and FLASH_SLMLR) with GetLock and SetLock functions to ensure that the blocks to be programmed/erased are unlocked.

2.2 Calling Convention Supported by SSD

MPC55xx H7Fa Standard Software Driver is built with the compiler in CodeWarrior for MPC5500, which supports the standard PowerPC EABI (Embedded Application Binary Interface). The user needs to use an EABI-compatible calling convention when using the binary c-array or s-record format SSD with other compilers. As shown in the demo for GAP tool, the SSD can be directly used with the Diab compiler.

2.2.1 Argument Passing

Up to eight scalar values are passed by using R3 through R10. If there are more arguments than can be passed using the registers, space for the additional arguments is allocated for them in the stack frame's Function Parameters Area.

2.2.2 Returning Results

The scalar return values are passed back in R3 and R4. And F1 is used to return a floating-point value. If there are more return values than can be passed using R3 and R4 (or F1), they will be passed by using the function parameters area.

2.2.3 Register Usage

The PowerPC architecture defines 32 general-purpose registers (GPRs) and 32 floating-point registers (FPRs). The following table lists the PowerPC EABI register usages.

Table 2 PowerPC EABI Register Usage

Register	Type	Used for
R0	Volatile	Language Specific
R1	Dedicated	Stack Pointer (SP)
R2	Dedicated	Read-only small data area anchor
R3 - R4	Volatile	Parameter passing / return values
R5 - R10	Volatile	Parameter passing
R11 - R12	Volatile	
R13	Dedicated	Read-write small data area anchor
R14 - R31	Nonvolatile	
F0	Volatile	Language specific
F1	Volatile	Parameter passing / return values
F2 - F8	Volatile	Parameter passing
F9 - F13	Volatile	
F14 - F31	Nonvolatile	
Fields CR2 - CR4	Nonvolatile	
Other CR fields	Volatile	
Other registers	Volatile	

For each SSD function, special notes for their register usage are listed as below:

- R3 is used to pass 32-bit return value;
- R0 and R12 will be used directly inside SSD function and their values may be changed after function return;
- GPRs which are used for argument passing, except R3, will not be changed inside SSD function;
- Other GPRs which are used for local variables, will be saved on function entry and restored at function return;
- LR and CR will be saved on function entry and restored at function return. There is no other system registers affected.

2.3 SSD Driver Formats

MPC55xx H7Fa Standard Software Driver is delivered in three binary formats: c-array, s-record and library.

2.3.1 C-Array Driver Format for Embedded Applications

The binary c-array file format is intended to simplify automated builds of embedded applications such as boot loaders. Each self-contained binary c-array driver is designed to be tool-independent, i.e. it can be compiled and linked together with embedded applications in tools other than CodeWarrior for MPC5500, provided an EABI-compatible calling convention is used. As illustrated in the c-array demo provided with this release, the hexadecimal coded c-array file can be automatically integrated with an application at link time, and the SSD functions can be used as normal C-language function calls. The SSD functions can also be called from assembly language applications so long as compliant calling conventions and parameter passing options are properly used. Also note that the *BDMEnable* flag in the SSD configuration structure must be set to FALSE so that the SSD functions execute a normal return to the calling application.

2.3.2 S-Record Driver Format for Debug-Mode Programming Tools

The s-record file format is intended to simplify construction of debug-mode programming tools. Since the SSD functions provide all the functionalities that are required for a typical debug-mode programming tool, no other target resident code is required. In this class of applications, the debug port, controlled by a host controller such as CodeWarrior command-line debugger, is used to:

- download SSD functions to the target microprocessor,
- download data buffers to the target,
- set up the stack and pass in the input parameters from registers,
- set the program counter,
- enter the run mode.

By setting the BDMEnable flag in the SSD configuration structure to TRUE, each target resident SSD function signals completion to the host controller by forcing the device into debug mode. The host controller will then read the return codes from the system registers.

2.3.3 Library Format Driver for Embedded Applications

The library format drivers are intended to simplify automated builds of embedded applications, such as boot loaders, in CodeWarrior environment. The library file contains all driver functions listed in section 1.1, which is built by the CodeWarrior compiler for MPC5500 and can be used in the same way as other libraries in CodeWarrior. If users want to use the library format drivers with a compiler other than CodeWarrior, the driver library file may need to be recompiled using that compiler.

Chapter 3: API Specification

3.1 General Overview

The MPC55xx H7Fa Standard Software Driver operates the H7Fa memory module embedded in MPC5500 family microcontrollers. It contains the following driver functions:

- *FlashInit*
- *FlashErase*
- *BlankCheck*
- *FlashProgram*
- *ProgramVerify*
- *CheckSum*
- *FlashSuspend*
- *FlashResume*
- *GetLock*
- *SetLock*
- *RWECheck*
- *GetWaitState*
- *SetWaitState*

3.2 Version String

An ASCII version string is appended to the end of each c-array and s-record format SSD function, so that revision of each SSD function can be identified. The format of the version string is defined as follows:

CPU core + Flash Technology + ff (ff) + xyz

Where:

- CPU core is the abbreviation for the CPU platform: MPC
- Flash Technology is the character abbreviation for the flash technology: H7FA
- ff (ff) is the abbreviation for the SSD function:
  - *FI = FlashInit*
  - *FE = FlashErase*
  - *BC = BlankCheck*
  - *FP = FlashProgram*

- *PV = ProgramVerify*
- *CS = CheckSum*
- *FS = FlashSuspend*
- *FR = FlashResume*
- *GL = GetLock*
- *SL = SetLock*
- *RC = RWECheck*
- *GW = GetWaitState*
- *SW = SetWaitState*

- xyz is a three digit version number
  - x is the major revision: 3
  - yz is the minor revision: 12 (e.g.)

3.3 Constant and Type Definition

3.3.1 General Type Definition

Table 3 Type Definitions

Mnemonic	Size	Description
BOOL	8-bits	unsigned char
INT8	8-bits	signed char
VINT8	8-bits	volatile signed char
UINT8	8-bits	unsigned char
VUINT8	8-bits	volatile unsigned char
INT16	16-bits	signed short
VINT16	16-bits	volatile signed short
UINT16	16-bits	unsigned short
VUINT16	16-bits	volatile unsigned short
INT32	32-bits	signed long
VINT32	32-bits	volatile signed long
UINT32	32-bits	unsigned long
VUINT32	32-bits	volatile unsigned long
INT64	64-bits	signed long long
VINT64	64-bits	volatile signed long long
UINT64	64-bits	unsigned long long
VUINT64	64-bits	volatile unsigned long long

Table 4 Values for BOOL Type

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

3.3.2 SSD Configuration Structure Definitions

This structure includes static parameters for H7Fa and debug mode selection flag for flash driver. The static parameters for H7Fa are chip-dependent. The user should correctly initialize the fields including *h7faRegBase*, *mainArrayBase*, *shadowRowBase*, *shadowRowSize* and *BDMEnable* before passing the structure to SSD functions. The rest parameters, *mainArraySize*, *lowBlockNum*, *midBlockNum* and *highBlockNum*, will be initialized in *FlashInit* automatically.

```
typedef struct _ssd_config
{
    UINT32 h7faRegBase;
    UINT32 mainArrayBase;
    UINT32 mainArraySize;
    UINT32 shadowRowBase;
    UINT32 shadowRowSize;
    UINT32 lowBlockNum;
    UINT32 midBlockNum;
    UINT32 highBlockNum;
    UINT32 BDMEnable;
} SSD_CONFIG, *PSSD_CONFIG;
```

Table 5 SSD Configuration Structure Field Definitions

Name	Type	Description
h7faRegBase	UINT32	The base address of H7Fa and FLASH_BIUCR control registers.
mainArrayBase	UINT32	The base address of flash main array.
mainArraySize	UINT32	The size of flash main array.
shadowRowBase	UINT32	The base address of shadow row
shadowRowSize	UINT32	The size of shadow row in byte.
lowBlockNum	UINT32	Block number of the low address space.
midBlockNum	UINT32	Block number of the mid address space.
highBlockNum	UINT32	Block number of the high address space.
BDMEnable	UINT32	Debug mode or non-debug mode selection

3.3.3 NULL Macro for CallBack Function

The driver permits the user to supply a CallBack function address so that time-critical events can be serviced during MPC55xx SS driver operations. In this way, watchdog timers can be serviced.

If it is unnecessary to provide the CallBack service, the user can disable it by a NULL function macro which is defined in "ssd_h7fa.h":

```
#define NULL_CALLBACK          ((void *)0xFFFFFFFF)
```

3.3.4 Function Return Codes

Table 6 Function Return Codes

Name	Value	Description
H7FA_OK	0x00000000	The requested operation is successful.
H7FA_INFO_RWE	0x00000001	RWE bit is set before flash operations.
H7FA_INFO_EER	0x00000002	EER bit is set before flash operations.
H7FA_INFO_EPE	0x00000004	The program/erase for all blocks including shadow row and excluding the boot block is disabled.
H7FA_INFO_BBEPE	0x00000008	The program/erase for boot block is disabled
H7FA_ERROR_PARTID	0x00000010	The SSD cannot operate on the part with the detected part ID
H7FA_ERROR_STOP	0x00000020	The flash module is disabled.
RESERVED	0x00000040	*RESERVED*
RESERVED	0x00000080	*RESERVED*
H7FA_ERROR_ALIGNMENT	0x00000100	Alignment error.
H7FA_ERROR_RANGE	0x00000200	Address range error.
H7FA_ERROR_BUSY	0x00000300	New program/erase cannot be preformed while a high voltage operation is already in progress.
H7FA_ERROR_PGOOD	0x00000400	The program operation is unsuccessful.
H7FA_ERROR_EGOOD	0x00000500	The erase operation is unsuccessful.
H7FA_ERROR_NOT_BLANK	0x00000600	There is a non-blank Flash memory location within the checked Flash memory region.
H7FA_ERROR_VERIFY	0x00000700	There is a mismatch between the source data and the content in the checked flash memory.
H7FA_ERROR_LOCK_INDICATOR	0x00000800	Invalid block lock indicator.
H7FA_ERROR_RWE	0x00000900	Read-while-write error occurred in previous reads.
RESERVED	0x00000A00	*RESERVED*
H7FA_ERROR_PASSWORD	0x00000B00	The password provided cannot unlock the block lock register for register writes

3.4 Functions

3.4.1 FlashInit

3.4.1.1 Description

This function checks and initializes the control registers of the H7Fa module. It will first check the part ID to ensure that correct version of SSD being used on correct revisions of MPC55xx parts. Then it will enable the H7Fa module for flash operation, check the FLASH_MCR-EER, FLASH_MCR-RWE, FLASH_MCR-EPE and FLASH_MCR-BBEPE bits, read the module configuration register to initialize the parameters in the SSD configuration structure. *FlashInit* must be called prior to any other flash operations.

3.4.1.2 Definition

UINT32 FlashInit (PSSD_CONFIG pSSDConfig);

3.4.1.3 Arguments

Table 7 Arguments for FlashInit

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Point to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.

3.4.1.4 Return Values

Table 8 Return Values for FlashInit

Type	Returning Location	Description	Possible Values
UINT32	R3	Indicates either success or failure type. It is a bit mapped return code so that more than one condition can be returned with a single return code. Each bit in the returned value, except for H7FA_OK, indicates a kind of current status of H7Fa module.	H7FA_OK H7FA_INFO_EER H7FA_INFO_RWE H7FA_INFO_EPE H7FA_INFO_BBEPE H7FA_ERROR_PARTID H7FA_ERROR_STOP

3.4.1.5 Tips

FlashInit will first check the part number and mask number to ensure that correct Flash driver release is used on supported silicon revisions.

When H7Fa module is in STOP mode, all flash address spaces, registers and register bits are deactivated except for the FLASH_MCR-STOP bit. So *FlashInit* need to check the FLASH_MCR-STOP bit and return *H7FA_ERROR_STOP* if it is set. The user should enable the flash module and call FlashInit again before other flash operations.

FlashInit will check the FLASH_MCR-RWE and FLASH_MCR-EER bits, but will not clear them when any of them is set. Please note that even if RWE or EER bit is set, flash program/erase operations can still be performed. However, users need to be careful not to program the region contains the address with double-bit errors (ECC error).

This function will also check the hardware lock bits, FLASH_MCR-EPE and FLASH_MCR-BBEPE. If FLASH_MCR-EPE is 0, H7FA_INFO_EPE will be set in the return code, which means all blocks exclusive of the boot block are disabled for program/erase. If FLASH_MCR-BBEPE is 0, H7FA_INFO_BBEPE will be set in the return code, which means boot block is not enabled for program/erase. *FlashInit* cannot change FLASH_MCR-EPE and FLASH_MCR-BBEPE bits because they are read-only. It is the user's responsibility to control them at the system level.

3.4.1.6 Troubleshooting

Table 9 Troubleshooting for FlashInit

Return Value	Description	Solution
H7FA_INFO_EER	An ECC Error occurred during a previous read.	Check the ECC error status and clear FLASH_MCR-EER bit.
H7FA_INFO_RWE	A Read While Write Error occurred during a previous read	Call *RWECheck* to check the FLASH_MCR-RWE bit and clear it.
H7FA_INFO_EPE	The FLASH_MCR-EPE bit is 0, which means all blocks exclusive of the boot block are disabled for program/erase.	Enable all blocks excluding the boot block at system level.
H7FA_INFO_BBEPE	The FLASH_MCR-BBEPE bit is 0, which means boot block is not enabled for program/erase.	Enable the boot block at system level.
H7FA_ERROR_PARTID	The SSD doesn't support the detected revision of the part.	Make sure correct Flash driver release is used.
H7FA_ERROR_STOP	The flash module is disabled.	Clear the STOP bit in MCR register to enable the flash module.

3.4.1.7 Affected Register

Table 10 Register Affected in FlashInit

Name	Bit	Description
FLASH_MCR	STOP, SIZE, LAS, MAS, EER, RWE, EPE, BBEPE	Read
SIU_MIDR	PARTNUM, MASKNUM	Read

3.4.2 FlashErase

3.4.2.1 Description

This function will erase the enabled blocks in the main array or the shadow row. Input arguments together with relevant flash module status will be checked, and proper error code will be returned if there is any error.

This function does not check the hardware lock bits (FLASH_MCR-EPE and FLASH_MCR-BBEPE) and the software lock registers (FLASH_LMLR, FLASH_HLR and FLASH_SLMLR). It is up to the user to unlock the blocks to be erased before calling this function.

WARNING: If the shadow block needs to be erased, the contents of 0x00FF_FCC8 (and 0x00FF_FCE8) should be saved and reprogrammed back into these locations. In addition, both the Serial Passcode (address 0x00FF_FDD8, the 64-bit Public Password is 0xCAFE_BEEF_FEED_FACE) and the Flash Control Word (address 0x00FF_FDE0, value of 0x55AA_55AA) must be reprogrammed into the shadow block prior to asserting any reset to the device to prevent being "locked out" of the device by the censorship features of the device.

3.4.2.2 Definition

```
UINT32 FlashErase (PSSD_CONFIG pSSDConfig,
                   BOOL shadowFlag,
                   UINT32 lowEnabledBlocks,
                   UINT32 midEnabledBlocks,
                   UINT32 highEnabledBlocks,
                   void (*CallBack)(void));
```

3.4.2.3 Arguments

Table 11 Arguments for FlashErase

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.

shadowFlag	BOOL	R4	Indicate either the main array or the shadow row to be erased.	TRUE: the shadow row will be erased. The *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* will be ignored; FALSE: the main array will be erased. Which blocks will be erased in low, mid and high address spaces is specified by *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* respectively.
lowEnabledBlocks	UINT32	R5	To select the array blocks in low address space for erasing.	Bit-mapped value. Select the block in the low address space to be erased by setting 1 to the appropriate bit of *lowEnabledBlocks*. If there is not any block to be erased in the low address space, *lowEnabledBlocks* must be set to 0.
midEnabledBlocks	UINT32	R6	To select the array blocks in mid address space for erasing.	Bit-mapped value. Select the block in the middle address space to be erased by setting 1 to the appropriate bit of *midEnabledBlocks*. If there is not any block to be erased in the middle address space, *midEnabledBlocks* must be set to 0.
highEnabledBlocks	UINT32	R7	To select the array blocks in high address space for erasing.	Bit-mapped value. Select the block in the high address space to be erased by setting 1 to the appropriate bit of *highEnabledBlocks*. If there is not any block to be erased in the high address space, *highEnabledBlocks* must be set to 0.
CallBack	Void (*)(void)	R8	Address of void call back function pointer.	Any addressable void function address. To disable it by *NULL_CALLBACK* macro.

3.4.2.4 Return Values

Table 12 Return Values for FlashErase

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_BUSY H7FA_ERROR_EGOOD H7FA_ERROR_PARTID

3.4.2.5 Tips

FlashErase will first check the part number and mask number to ensure that correct Flash driver release is used on supported silicon revisions.

When *shadowFlag* is set to FALSE, this function will erase the blocks in the main array. It is capable of erasing any combination of blocks in the low, mid and high address spaces in one operation. If *shadowFlag* is TRUE, this function will erase the shadow row.

The inputs *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* are bit-mapped arguments that are used to select the blocks to be erased in the Low/Mid/High address spaces of main array. A main array block is selected/deselected by setting/clearing the corresponding bit in *lowEnabledBlocks*, *midEnabledBlocks* or *highEnabledBlocks*.

From right (LSB) to left (MSB), the bit allocations for blocks in one address space are: bit 0 is assigned to block 0, bit 1 to block 1, etc. The following diagrams show the formats of *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* for the H7Fa module embedded in MPC5554.

Bit Allocation for Blocks in Low Address Space:

MSB LSB

bit 31	…	bit 16	bit 15	bit 14	…	bit 1	bit 0
reserved	…	reserved	block 15	block 14	…	block 1	block 0

Bit Allocation for Blocks in Middle Address Space:

MSB LSB

bit 31	…	bit 4	bit 3	bit 2	bit 1	bit 0
reserved	…	reserved	block 3	block 2	block 1	block 0

Bit Allocation for Blocks in High Address Space:

MSB LSB

bit 31	…	bit 28	bit 27	bit 26	…	bit 1	bit 0
reserved	…	reserved	block 27	block 26	…	Block 1	Block 0

Note:

1. For low address space valid bits are from bit 0 to bit 15;
2. For middle address space valid bits are from bit 0 to bit 3;
3. For high address space valid bits are from bit 0 to bit 27;
4. The number of actual flash blocks in each address spaces is determined by *lowBlockNum*, *midBlockNum* and *highBlockNum* of SSD Configuration Structure. Those parameters are initialized in *FlashInit* function by reading the module configuration register. Please refer to Section 3.3.2 for more details. *FlashErase* automatically masks off all the reserved bits in *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks*.

FlashErase does not check the hardware lock bits (FLASH_MCR-EPE and FLASH_MCR-BBEPE) and the software lock registers (FLASH_LMLR, FLASH_HLR and FLASH_SLMLR). It is up to the user to unlock the blocks to be erased before calling this function. User can use the *GetLock* and *SetLock* functions to manage all software locks, but the hardware locks can only be controlled at system level.

If the selected main array blocks or the shadow row is locked for erasing, those blocks or the shadow row will not be erased, but *FlashErase* will still return H7FA_OK. User needs to check the erasing result with the *BlankCheck* function.

It is impossible to erase any flash block or shadow row when a program or erase operation is already in progress on H7Fa module. *FlashErase* will return H7FA_ERROR_BUSY when trying to do so. Similarly, once an erasing operation has started on H7Fa module, it is impossible to run another program or erase operation.

In addition, when *FlashErase* is running, it is unsafe to read the data from the flash partitions having one or more blocks being erased. Otherwise, it will cause a Read-While-Write error. However, user can use *FlashSuspend* to suspend the on-going erase operation before such reading. Please refer to *FlashSuspend* function for more details.

After the erase sequence stopped, the BIU line read buffer will be invalidated and then re-enabled to ensure data coherency.

3.4.2.6 Troubleshooting

Table 13 Troubleshooting for FlashErase

Return Value	Description	Solution
H7FA_ERROR_BUSY	New erase operation cannot be performed because there is program/erase sequence in progress on the flash module.	Wait until all previous program/erase operations on the flash module finish. Possible cases that erase cannot start are: 1. erase in progress (FLASH_MCR-ERS is high); 2. program in progress (FLASH_MCR-PGM is high);
H7FA_ERROR_EGOOD	Erase operation failed.	Check if the H7Fa is available and high voltage is applied to H7Fa. Then try to do the erase operation again.
H7FA_ERROR_PARTID	The SSD doesn't support the detected revision of the part.	Make sure correct Flash driver release is used.

3.4.2.7 Affected Register

Table 14 Register Affected in FlashErase

Name	Bit	Description
FLASH_MCR	DONE, PEG, PGM	Read
	ERS, EHV	Read/Write
FLASH_LMSR	Bits in accordance with selected blocks.	Read/Write
FLASH_HSR	Bits in accordance with selected blocks.	Read/Write
FLASH_BIUCR	BFEN	Write
SIU_MIDR	PARTNUM, MASKNUM	Read

3.4.3 BlankCheck

3.4.3.1 Description

This function will check on the specified flash range in the main array or shadow row for blank state.

3.4.3.2 Definition

```
UINT32 BlankCheck (PSSD_CONFIG pSSDConfig,
                   UINT32 dest,
                   UINT32 size,
                   UINT32 * pFailAddress,
                   UINT64 *pFailData,
                   void  (*CallBack) (void ));
```

3.4.3.3 Arguments

Table 15 Arguments for BlankCheck

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
dest	UINT32	R4	Destination address to be checked.	Any accessible address aligned on double word boundary in main array or shadow row
size	UINT32	R5	Size, in bytes, of the flash region to check.	If *size* = 0, the return value is H7FA_OK. It should be multiple of 8 and its combination with *dest* should fall in either main array or shadow row.
pFailAddress	UINT32 *	R6	Return the address of the first non-blank Flash location in the checking region	Only valid when this function returns H7FA_ERROR_NOT_BLANK.
pFailData	UINT64 *	R7	Return the content of the first non-blank Flash location in the checking region.	Only valid when this function returns H7FA_ERROR_NOT_BLANK.
CallBack	void (*)(void)	R8	Address of void callback function	Any addressable void function address. To disable it by *NULL_CALLBACK* macro.

3.4.3.4 Return Values

Table 16 Return Values for BlankCheck

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_ALIGNMENT H7FA_ERROR_RANGE H7FA_ERROR_NOT_BLANK

3.4.3.5 Tips

If the blank checking fails, the first failing address will be saved to **pFailAddress*, and the failing data in flash will be saved to **pFailData*. The contents pointed by *pFailAddress* and *pFailData* are updated only when there is a non-blank location in the checked flash range.

This function will not check the FLASH_MCR-EER bit and FLASH_MCR-RWE bit while checking the flash. If users want to know whether any Read While Write error occurred or not during the blank checking. please call *RWECheck* function to check the MCR-RWE bit after *BlankCheck* function.

3.4.3.6 Troubleshooting

Table 17 Troubleshooting for BlankCheck

Return Value	Description	Solution
H7FA_ERROR_ALIGNMENT	The *dest*/*size* are not properly aligned.	Check if *dest* and *size* are aligned on double word (64-bit) boundary.
H7FA_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid H7Fa array ranges.	Check both *dest* and *dest*+*size*. The area to be checked must be within main array space or shadow space.
H7FA_ERROR_NOT_BLANK	There is a non-blank double word within the area to be checked.	Erase the relevant blocks and check again.

3.4.3.7 Affected Register

None.

3.4.4 FlashProgram

3.4.4.1 Description

This function will program the specified flash areas with the provided source data. Input arguments together with relevant flash module status will be checked, and proper error code will be returned if there is any error.

For fastest programming speed, FlashProgram programs flash in pages (256-bit). It only requires the source data to be double-word (64-bit) aligned, i.e. it handles any incomplete page at the beginning or end of the source data.

This function does not check the hardware lock bits (FLASH_MCR-EPE and FLASH_MCR-BBEPE) and the software lock registers (FLASH_LMLR, FLASH_HLR and FLASH_SLMLR). It is up to the user to unlock the blocks to be programmed before calling this function.

3.4.4.2 Definition

UINT32 FlashProgram (PSSD_CONFIG pSSDConfig,

UINT32 dest,

UINT32 size,

UINT32 source,

void (*CallBack)(void));

3.4.4.3 Arguments

Table 18 Arguments for FlashProgram

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
dest	UINT32	R4	Destination address to be programmed in flash memory.	Any accessible address aligned on double word boundary in main array or shadow row.
size	UINT32	R5	Size, in bytes, of the flash region to be programmed.	If size = 0, H7FA_OK will be returned. It should be multiple of 8 and its combination with dest should fall in either main array or shadow row.
source	UINT32	R6	Source program buffer address.	This address must reside on word boundary.

Argument	Type	Passing Location	Description	Range
CallBack	void (*)(void)	R7	Address of void call back function pointer.	Any addressable void function address. To disable it by *NULL_CALLBACK* macro.

3.4.4.4 Return Values

Table 19 Return Values for FlashProgram

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_BUSY H7FA_ERROR_ALIGNMENT H7FA_ERROR_RANGE H7FA_ERROR_PGOOD

3.4.4.5 Tips

FlashProgram does not check the hardware lock bits (FLASH_MCR-EPE and FLASH_MCR-BBEPE) and the software lock registers (FLASH_LMLR, FLASH_HLR and FLASH_SLMLR). It is up to the user to unlock the blocks to be programmed before calling this function. User can use the *GetLock* and *SetLock* functions to control all software locks, but the hardware locks can only be controlled at system level.

If the selected main array blocks or the shadow row is locked for programming, those blocks or the shadow row will not be programmed, and *FlashProgram* will still return H7FA_OK. User needs to verify the programmed data with *ProgramVerify* function.

It is impossible to program any flash block or shadow row when a program or erase operation is already in progress on H7Fa module. *FlashProgram* will return H7FA_ERROR_BUSY when doing so. However, user can use the *FlashSuspend* function to suspend an on-going erase operation on one block to perform a program operation on another block. An exception is that once the user has begun an erase operation on the shadow row, it may not be suspended to program the main array and vice-versa. Please refer to *FlashSuspend* for more details.

It is unsafe to read the data from the flash partitions having one or more blocks being programmed when *FlashProgram* is running. Otherwise, it will cause a Read-While-Write error. However, user can use *FlashSuspend* to suspend the programming operation before such reading. Please refer to *FlashSuspend* for more details.

After the program sequence completes, the BIU line read buffer will be invalidated and then re-enabled to ensure data coherency.

In addition, this function will not check the FLASH_MCR-EER bit and FLASH_MCR-RWE bit while programming data from a flash area to another flash area. If users want to know whether any Read While Write error occurred or not during flash-to-flash programming, please call *RWECheck* function to check the FLASH_MCR-RWE bit after *FlashProgram* function.

3.4.4.6 Troubleshooting

Table 20 Troubleshooting for FlashProgram

Return Value	Description	Solution
H7FA_ERROR_BUSY	New program operation cannot be performed because the flash module is busy with some operation and cannot meet the condition for starting a program operation.	Wait until the current operations finish. Conditions that program cannot start are: 1. program in progress (FLASH_MCR-PGM high); 2. program not in progress (FLASH_MCR-PGM low), but: a). erase in progress but not suspended; b). erase on main array is suspended but program is targeted to shadow row; c). erase on shadow row is suspended.
H7FA_ERROR_ALIGNMENT	This error indicates that *dest*/*size*/*source* isn't properly aligned	Check if *dest* and *size* are aligned on double word (64-bit) boundary. Check if *source* is aligned on word boundary.
H7FA_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid H7Fa address range.	Check both *dest* and *dest*+*size*. Both should fall in the same H7FA address ranges, i.e. both in main array or both in shadow row
H7FA_ERROR_PGOOD	Program operation failed because this operation cannot pass PEG check.	Repeat the program operation. Check if the H7Fa is invalid or high voltage applied to H7Fa is unsuitable.

3.4.4.7 Affected Register

Table 21 Register Affected in FlashProgram

Name	Bit	Description
FLASH_MCR	DONE, PEG, ERS, PEAS, ESUS	Read
	PGM, EHV	Read/Write
FLASH_BIUCR	BFEN	Write

3.4.5 ProgramVerify

3.4.5.1 Description

This function checks if a programmed flash range matches the corresponding source data buffer.

3.4.5.2 Definition

```
UINT32 ProgramVerify (PSSD_CONFIG pSSDConfig,
                      UINT32 dest,
                      UINT32 size,
                      UINT32 source,
                      UINT32 *pFailAddress,
                      UINT64 *pFailData,
                      UINT64 *pFailSource,
                      void (*CallBack)(void));
```

3.4.5.3 Arguments

Table 22 Arguments for ProgramVerify

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
dest	UINT32	R4	Destination address to be verified in flash memory.	Any accessible address aligned on double word boundary in main array or shadow row.
size	UINT32	R5	Size, in byte, of the flash region to verify.	If *size* = 0, H7FA_OK will be returned. Its combination with *dest* should fall within either main array or shadow row.
source	UINT32	R6	Verify source buffer address.	This address must reside on word boundary.
pFailAddress	UINT32 *	R7	Return first failing address in flash.	Only valid when the function returns H7FA_ERROR_VERIFY.
pFailData	UINT64 *	R8	Returns first mismatch data in flash.	Only valid when this function returns H7FA_ERROR_VERIFY.
pFailSource	UINT64 *	R9	Returns first mismatch data in buffer.	Only valid when this function returns H7FA_ERROR_VERIFY.

Argument	Type	Passing Location	Description	Range
CallBack	void (*)(void)	R10	Address of void call back function pointer.	Any addressable void function address. To disable it by *NULL_CALLBACK* macro.

3.4.5.4 Return Values

Table 23 Return Values for ProgramVerify

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_ALIGNMENT H7FA_ERROR_RANGE H7FA_ERROR_VERIFY

3.4.5.5 Tips

The contents pointed by *pFailLoc*, *pFailData* and *pFailSource* are updated only when there is a mismatch between the source and destination regions.

This function will not check the FLASH_MCR-EER bit and FLASH_MCR-RWE bit while verifying the flash. If users want to know whether any Read While Write error occurred or not during the program verifying, please call *RWECheck* function to check the FLASH_MCR-RWE bit after *ProgramVerify* function.

3.4.5.6 Troubleshooting

Table 24 Troubleshooting for ProgramVerify

Return Value	Description	Solution
H7FA_ERROR_ALIGNMENT	This error indicates that *dest*/*size*/*source* isn't properly aligned	Check if *dest* and size are aligned on double word (64-bit) boundary. Check if *source* is aligned on word boundary
H7FA_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid H7Fa address range.	Check both *dest* and *dest*+*size*, both should fall in the same H7FA address ranges, i.e. both in main array or both in shadow row
H7FA_ERROR_VERIFY	The content in H7Fa and source data mismatch.	Check the correct source and destination addresses, erase the block and reprogram data into flash.

3.4.5.7 Affected Register

None.

3.4.6 CheckSum

3.4.6.1 Description

This function performs a 32-bit sum over the specified flash memory range without carry, which provides a rapid method for checking data integrity.

3.4.6.2 Definition

UINT32 CheckSum (PSSD_CONFIG pSSDConfig,

UINT32 dest,

UINT32 size,

UINT32 *pSum,

void (*CallBack)(void));

3.4.6.3 Arguments

Table 25 Arguments for CheckSum

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
dest	UINT32	R4	Destination address to be summed in flash memory.	Any accessible address aligned on double word boundary in either main array or shadow row.
size	UINT32	R5	Size, in bytes, of the flash region to check sum.	If size is 0 and the other parameters are all valid, H7FA_OK will be returned. Its combination with dest should fall within either main array or shadow row.
pSum	UINT32 *	R6	Returns the sum value.	0x00000000 - 0xFFFFFFFF. Note that this value is only valid when the function returns H7FA_OK.
CallBack	void (*)(void)	R7	Address of void call back function pointer.	Any addressable void function address. To disable it by *NULL_CALLBACK* macro.

3.4.6.4 Return Values

Table 26 Return Values for CheckSum

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error	H7FA_OK

		value.	H7FA_ERROR_ALIGNMENT H7FA_ERROR_RANGE

3.4.6.5 Tips

This function will not check the FLASH_MCR-EER bit and FLASH_MCR-RWE bit while reading the flash. If users want to know whether any Read While Write error occurred or not during reading the flash, please call *RWECheck* function to check the FLASH_MCR-RWE bit after *CheckSum* function.

3.4.6.6 Troubleshooting

Table 27 Troubleshooting for CheckSum

Return Value	Description	Solution
H7FA_ERROR_ALIGNMENT	This error indicates that dest/size isn't aligned on double word boundary.	Check if dest and size are aligned on double word (64-bit) boundary.
H7FA_ERROR_RANGE	The area specified by dest and size is out of the valid H7Fa address range.	Check both dest and dest+size, both should fall in the same H7FA address ranges, i.e. both in main array or both in shadow row.

3.4.6.7 Affected Register

None.

3.4.7 FlashSuspend

3.4.7.1 Description

This function checks if there is any high voltage operation, erase or program, in progress on the H7Fa module and if the operation can be suspended. This function will suspend the ongoing operation if it can be suspended.

3.4.7.2 Definition

UINT32 FlashSuspend (PSSD_CONFIG pSSDConfig,

UINT8 *suspendState,

BOOL *suspendFlag);

3.4.7.3 Arguments

Table 28 Arguments for FlashSuspend

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
suspendState	UINT8 *	R4	Indicate the suspend state of H7Fa module after the function being called.	All return values are enumerated in Table 29.
suspendFlag	BOOL *	R5	Return whether the suspended operation, if there is any, is suspended by this call.	TRUE: the operation is suspended by this call; FALSE: either no operation to be suspended or the operation is suspended not by this call.

Table 29 Definitions of suspendState

Argument	Code	Description	Valid Operation after Suspend
NO_OPERTION	0	There is no program/erase operation.	Erasing operation, programming operation and read are valid on both main array space and shadow row.
PGM_WRITE	1	There is a program sequence in interlock write stage.	Only read is valid on both main array space and shadow row.
ERS_WRITE	2	There is an erase sequence in interlock write stage.	Only read is valid on both main array space and shadow row.
ERS_SUS_PGM_WRITE	3	There is an erase-suspend program sequence in interlock write stage.	Only read is valid on both main array space and shadow row.
PGM_SUS	4	The program operation is in suspended state.	Only read is valid on both main array space and shadow row.

ERS_SUS	5	The erase operation on main array is in suspended state.	Programming operation is valid only on main array space. Read is valid on both main array space and shadow row.
SHADOW_ERS_SUS	6	The erase operation on shadow row is in suspended state.	Read is valid on both main array space and shadow space.
ERS_SUS_PGM_SUS	7	The erase-suspended program operation is in suspended state.	Only read is valid on both main array space and shadow row.

Table 30 Suspending State and Flag vs. H7Fa Status

suspendState	*suspendFlag*	EHV	ERS	ESUS	PGM	PSUS	PEAS
NO_OPERATION	FALSE	X	0	X	0	X	X
PGM_WRITE	FALSE	0	0	X	1	0	X
ERS_WRITE	FALSE	0	1	0	0	X	X
ESUS_PGM_WRITE	FALSE	0	1	1	1	0	X
PGM_SUS	TRUE	1	0	X	1	0	X
	FALSE	X	0	X	1	1	X
ERS_SUS	TRUE	1	1	0	0	X	0
	FALSE	X	1	1	0	X	0
SHADOW_ERS_SUS	TRUE	1	1	0	0	X	1
	FALSE	X	1	1	0	X	1
ERS_SUS_PGM_SUS	TRUE	1	1	1	1	0	X
	FALSE	X	1	1	1	1	X

Notes:
1. The values of EHV, ERS, ESUS, PGM, PSUS and PEAS represent the H7Fa status at the entry of *FlashSuspend*;
2. 0: Logic zero; 1: Logic one; X: Do-not-care.

3.4.7.4 Return Values

Table 31 Return Values for FlashSuspend

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion.	H7FA_OK

3.4.7.5 Tips

After calling *FlashSuspend*, read is allowed on both main array space and shadow row without any Read-While-Write error. But data read from the blocks targeted for programming or erasing will be indeterminate even if the operation is suspended.

This function should be used together with *FlashResume*. The *suspendFlag* returned by *FlashSuspend* determine whether *FlashResume* needs to be called or not. If *suspendFlag* is TRUE, *FlashResume* must be called symmetrically to resume the suspended operation.

3.4.7.6 Troubleshooting

This section is intentionally blank.

3.4.7.7 Affected Register

Table 32 Register Affected in FlashSuspend

Name	Bit	Description
FLASH_MCR	PEAS, DONE, EHV, PGM, ERS	Read
	PSUS, ESUS	Read/Write

3.4.8 FlashResume

3.4.8.1 Description

This function checks if there is any suspended erase or program operation on the H7Fa module, and will resume the suspended operation if there is any.

3.4.8.2 Definition

```
UINT32 FlashResume (PSSD_CONFIG pSSDConfig,
                    UINT8 *resumeState);
```

3.4.8.3 Arguments

Table 33 Arguments for FlashResume

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
resumeState	UINT8 *	R4	Indicate the resume state of H7Fa module after the function being called.	All return values are listed in table 34.

Table 34 Definitions of resumeState

Code Name	Value	Description
RES_NOTHING	0	No program/erase operation to be resumed
RES_PGM	1	A program operation is resumed
RES_ERS	2	A erase operation is resumed
RES_ERS_PGM	3	A suspended erase-suspended program operation is resumed

3.4.8.4 Return Values

Table 35 Return Values for FlashResume

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion.	H7FA_OK

3.4.8.5 Tips

This function will resume one operation if there is any operation is suspended. For instance, if a program operation is in suspended state, it will be resumed. If an erase operation is in suspended state, it will be resumed too. If an erase-suspended program operation is in suspended state, the program operation will be resumed prior to resuming the erase operation.

It is better to call this function based on *suspendFlag* returned from *FlashSupend*. Please refer to section 3.4.7.5 for more details.

3.4.8.6 Troubleshooting

This section is intentionally blank.

3.4.8.7 Affected Register

Table 36 Register Affected in FlashResume

Name	Bit	Description
FLASH_MCR	DONE, PSUS, ESUS, PGM	Read
	EHV, PSUS, ESUS	Write

3.4.9 GetLock

3.4.9.1 Description

This function will check the block locking status of Shadow/Low/Middle/High address spaces in the H7Fa module.

3.4.9.2 Definition

UINT32 GetLock (PSSD_CONFIG pSSDConfig,

UINT8 blkLockIndicator,

BOOL *blkLockEnabled,

UINT32 *blkLockState);

3.4.9.3 Arguments

Table 37 Arguments for GetLock

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
blkLockIndicator	UINT8	R4	Indicating the address space and the block locking level, which determines the address space block locking register to be checked.	Refer to Table 38 for valid values for this parameter.
blkLockEnabled	BOOL *	R5	Indicate whether the address space block locking register is enabled for register writes	TRUE – The address space block locking register is enabled for register writes. FALSE – The address space block locking register is disabled for register writes.
blkLockState	UINT32 *	R6	Returns the blocks' locking status of indicated locking level in the given address space	Bit mapped value indicating the locking status of the specified locking level and address space. 1: The block is locked from program/erase. 0: The block is ready for program/erase

Table 38 Definitions of blkLockIndicator

Code Name	Value	Description
LOCK_SHADOW_PRIMARY	0	Primary block lock protection of shadow address space
LOCK_SHADOW_SECONDARY	1	Secondary block lock protection of shadow address space
LOCK_LOW_PRIMARY	2	Primary block lock protection of low address space
LOCK_LOW_SECONDARY	3	Secondary block lock protection of low address space
LOCK_MID_PRIMARY	4	Primary block lock protection of mid address space
LOCK_MID_SECONDARY	5	Secondary block lock protection of mid address space
LOCK_HIGH	6	Block lock protection of high address space

3.4.9.4 Return Values

Table 39 Return Values for GetLock

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_LOCK_INDICATOR

3.4.9.5 Tips

For Shadow/Low/Mid address spaces, there are two block lock levels. The secondary level of block locking provides an alternative means to protect blocks from being modified. A logical "OR" of the corresponding bits in the primary and secondary lock registers for a block determines the final lock status for that block. For high address space there is only one block lock level.

The output parameter *blkLockState* will return a bit-mapped value indicating the block lock status of the specified locking level and address space. A main array block or shadow row is locked from program/erase if its corresponding bit is set.

The indicated address space determines the valid bits of *blkLockState*. From right (LSB) to left (MSB), the bit allocation for blocks in one address space is: bit 0 is assigned to block 0, bits 1 to block 1, etc.

The following diagrams show the block bitmap definitions of *blkLockState* for shadow/Low/Mid/High address spaces.

blkLockState Bit Allocation for Shadow Address Space:

MSB **LSB**

bit 31	…	bit 1	bit 0
reserved	…	reserved	shadow row

blkLockState Bit Allocation for Low Address Space:

MSB **LSB**

bit 31	…	bit 16	bit 15	bit 14	…	bit 1	bit 0
reserved	…	reserved	block 15	block 14	…	block 1	block 0

blkLockState Bit Allocation for Mid Address Space:

MSB						LSB
bit 31	**…**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
reserved	…	reserved	block 3	block 2	block 1	block 0

blkLockState Bit Allocation for High Address Space:

MSB							LSB
bit 31	**…**	**bit 28**	**bit 27**	**bit 26**	**…**	**bit 1**	**bit 0**
reserved	…	reserved	block 27	block 26	…	block 1	block 0

Note:

1. For shadow address space only bit 0 is valid;
2. For low address space valid bits are from bit 0 to bit 15;
3. For mid address space valid bits are from bit 0 to bit 3
4. For high address space valid bits are from bit 0 to bit 27
5. For either Low/Mid/High address spaces, if blocks corresponding to valid block lock state bits are not present (due to configuration or total memory size), values for these block lock state bits will be always 1 because such blocks are locked by hardware on reset. These blocks cannot be unlocked by software with SetLock function.

3.4.9.6 Troubleshooting

Table 40 Troubleshooting for GetLock

Return Value	Description	Solution
H7FA_ERROR_LOCK_INDICATOR	The input *blkLockIndicator* is invalid.	Set this argument to correct value listed in Table 38.

3.4.9.7 Affected Register

Table 41 Register Affected in GetLock

Name	Bit	Description
FLASH_LMLR	SLOCK, MLOCK, LLOCK, LME	Read
FLASH_HLR	HBLOCK, HBE	Read
FLASH_SLMLR	SSLOCK, SMLOCK, SLLOCK, SLE	Read

3.4.10 SetLock

3.4.10.1 Description

This function will set the block lock state for Shadow/Low/Middle/High address space on the H7Fa module to protect them from program/erase.

3.4.10.2 Definition

UINT32 SetLock (PSSD_CONFIG pSSDConfig,

UINT8 blkLockIndicator,

UINT32 blkLockState,

UINT32 password);

3.4.10.3 Arguments

Table 42 Arguments for SetLock

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
blkLockIndicator	UINT8	R4	Indicating the address space and the protection level of the block lock register to be read.	Refer to Table 38 for valid codes for this parameter.
blkLockState	UINT32	R5	The block locks to be set to the specified address space and protection level.	Bit mapped value indicating the lock status of the specified protection level and address space. 1: The block is locked from program/erase. 0: The block is ready for program/erase
password	UINT32	R6	A password is required to enable the block lock register for register write.	Correct passwords for block lock registers are 0xA1A11111 for Low/Mid Address Space Block Locking Register, 0xC3C33333 for Secondary Low/Mid Address Space Block Locking Register, and 0xB2B22222 for High Address Space Block Select Register.

3.4.10.4 Return Values

Table 43 Return Values for SetLock

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_LOCK_INDICATOR H7FA_ERROR_PASSWORD

3.4.10.5 Tips

The bit field allocation for *blkLockState* is same as that in *GetLock* function.

3.4.10.6 Troubleshooting

Table 44 Troubleshooting for SetLock

Return Value	Description	Solution
H7FA_ERROR_LOCK_INDICATOR	The input *blkLockIndicator* is invalid.	Set this argument to correct value listed in Table 38.
H7FA_ERROR_PASSWORD	The given password cannot enable the block lock register for register writes.	Pass in a correct password.

3.4.10.7 Affected Register

Table 45 Register Affected in SetLock

Name	Bit	Description
FLASH_LMLR	LME	Read/ Write
	SLOCK, MLOCK, LLOCK	Write
FLASH_HLR	HBE	Read/ Write
	HBLOCK	Write
FLASH_SLMLR	SLE	Read/ Write
	SSLOCK, SMLOCK, SLLOCK	Write

3.4.11 RWECheck

3.4.11.1 Description

This function will check if a Read-While-Write error occurred on previous reads. If yes, H7FA_ERROR_RWE will be returned and FLASH_MCR-RWE bit will be cleared.

3.4.11.2 Definition

UINT32 RWECheck (PSSD_CONFIG pSSDConfig);

3.4.11.3 Arguments

Table 46 Arguments for RWECheck

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	The pointer to SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.

3.4.11.4 Return Values

Table 47 Return Values for RWECheck

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion or error value.	H7FA_OK H7FA_ERROR_RWE

3.4.11.5 Tips

This section is intentionally blank.

3.4.11.6 Troubleshooting

Table 48 Troubleshooting for RWECheck

Return Value	Description	Solution
H7FA_ERROR_RWE	A Read-While-Write error occurred on previous reads.	*RWECheck* has cleared the FLASH_MCR-RWE bit.

3.4.11.7 Affected Register

Table 49 Register Affected in RWECheck

Name	Bit	Description
FLASH_MCR	RWE	Read/Write

3.4.12 GetWaitState

3.4.12.1 Description

This function will read the FLASH_BIUCR register and return the address pipelining control state, write wait state and read wait state of the flash array.

3.4.12.2 Definition

UINT32 GetWaitState (PSSD_CONFIG pSSDConfig,

UINT32 *pAPCValue,

UINT32 *pWWSCValue,

UINT32 *pRWSCValue);

3.4.12.3 Arguments

Table 50 Arguments for GetWaitState

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to the SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
pAPCValue	UINT32 *	R4	Return number of cycles between pipelined access requests. Read from APC field of FLASH_BIUCR register.	Refer to table 52
pWWSCValue	UINT32 *	R5	Return the number of wait-states to be added to the best-case Flash array access time for Flash array writes. Read from WWSC field of FLASH_BIUCR register.	Refer to table 53
pRWSCValue	UINT32 *	R6	Return the number of wait-states to be added to the best-case Flash array access time for Flash array reads. Read from RWSC field of FLASH_BIUCR register.	Refer to table 54

3.4.12.4 Return Values

Table 51 Return Values for GetWaitState

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion	H7FA_OK

3.4.12.5 Tips

The APC, WWSC and RWSC fields of FLASH_BIUCR register must be set to a value corresponding to the operating frequency of the FLASH_BIUCR by *SetWaitState* function. Possible settings for these fields are listed as below:

Table 52 Settings for APC Field

Value	Additional Hold Cycles Required for Access Requests
b000	0 (Accesses may be pipelined back-to-back)
b001	1
b010	2
b011	3
b100	4
b101	5
b110	6
b111	No address pipelining

Note: This field is set to b111 by hardware reset.

Table 53 Settings for WWSC Field

Value	Additional Wait-states Added for Flash Array Writes
b00	0
b01	1
b10	2
b11	3

Note: This field is set to b11 by hardware reset.

Table 54 Settings for RWSC Field

Value	Additional Wait-states Added for Flash Array Reads
b000	0
b001	1
b010	2
b011	3
b100	4
b101	5
b110	6
b111	7

Note: This field is set to b111 by hardware reset.

3.4.12.6 Troubleshooting

This section is intentionally blank.

3.4.12.7 Affected Register

Table 55 Register Affected in GetWaitState

Name	Bit	Description
FLASH_BIUCR	APC, WWSC, RWSC	Read

3.4.13 SetWaitState

3.4.13.1 Description

This function will set the address pipelining control state, write wait state and read wait state of the flash array to the FLASH_BIUCR register.

3.4.13.2 Definition

UINT32 SetWaitState (PSSD_CONFIG pSSDConfig,

UINT32 APCValue,

UINT32 WWSCValue,

UINT32 RWSCValue);

3.4.13.3 Arguments

Table 56 Arguments for SetWaitState

Argument	Type	Passing Location	Description	Range
pSSDConfig	PSSD_CONFIG	R3	Pointer to the SSD_CONFIG structure.	The values in this structure are chip-dependent. Please refer to Section 3.3.2 for more details.
APCValue	UINT32	R4	The number of cycles between pipelined access requests. This value will be set to APC field of FLASH_BIUCR register	Refer to table 52
WWSCValue	UINT32	R5	The number of wait-states to be added to the best-case Flash array access time for Flash array writes. This value will be set to WWSC field of FLASH_BIUCR register.	Refer to table 53
RWSCValue	UINT32	R6	The number of wait-states to be added to the best-case Flash array access time for Flash array reads. This value will be set to RWSC field of FLASH_BIUCR register	Refer to table 54

3.4.13.4 Return Values

Table 57 Return Values for SetWaitState

Type	Returning Location	Description	Possible Values
UINT32	R3	Successful completion	H7FA_OK

3.4.13.5 Tips

Refer to the same section of *GetWaitState* function.

3.4.13.6 Troubleshooting

This section is intentionally blank.

3.4.13.7 Affected Register

Table 58 Register Affected in SetWaitState

Name	Bit	Description
FLASH_BIUCR	APC, WWSC, RWSC	Write

Appendix A: MPC5500 Flash Driver Version

MPC5500 Silicon Version	Flash Driver Version	Special Program Algorithm Required	Special Erase Algorithm Required
MPC5554 Rev 0.0	None[1]	N/A	N/A
MPC5554 Rev 0.1	3.0.7	Yes	Yes
MPC5554 Rev 0.3	3.0.8, 3.0.9	No	Yes[2]
MPC5554 Rev 0.4	3.0.9	No	No
MPC5554 Rev A	3.1.0, 3.1.1, 3.1.2	No	No
Other MPC5500 Derivatives (e.g. MPC5553)	3.1.1, 3.1.2	No	No

Notes:

1. The internal flash on MPC5554 Rev 0.0 silicon is not usable by customers.
2. The majority of MPC5554 Rev 0.3 devices require the special erase algorithm, but some parts may be able to use the standard erase algorithm. These parts will have 0x4C4A_4F4E programmed into the Shadow block at address 0x00FF_FCC8 (and 0x00FF_FCE8).
3. The standard documented program/erase algorithm as published in the MPC5554 Reference Manual is used when no special algorithm is required.
4. For silicon revisions for which multiple Flash driver versions can be used, it is recommended to use the latest driver version for that silicon.

WARNING

Using the standard, published (in the MPC5554 Reference Manual) programming algorithm on an MPC5554 Rev 0.1 device or the standard erase algorithm on an MPC5554 Rev 0.1 or 0.3 device could render the internal flash inoperative. Use of the internal flash cannot be restored in the field. Devices may be recoverable by returning the parts to the Freescale factory.

APPENDIX B. Performance Data

A.1 Code Size and Stack Usage

Table 59 Code Size and Stack Usage of SSD v3.0.7

Function Name	Code Size (Words)	Stack Usage (Words)
FlashInit	87	8
FlashErase	162	20
BlankCheck	93	16
FlashProgram	182	16
ProgramVerify	105	20
CheckSum	84	16
FlashSuspend	103	8
FlashResume	64	12
GetLock	72	8
SetLock	81	12
EERCheck	36	8
RWECheck	34	8
GetWaitState	31	8
SetWaitState	38	8
Total Code Size / Max Stack Usage	1172	20

Table 60 Code Size and Stack Usage of SSD v3.0.8

Function Name	Code Size (Words)	Stack Usage (Words)
FlashInit	85	8
FlashErase	265	20
BlankCheck	93	16
FlashProgram	160	16
ProgramVerify	105	20
CheckSum	84	16
FlashSuspend	103	8
FlashResume	64	12
GetLock	72	8
SetLock	81	12
EERCheck	36	8
RWECheck	34	8
GetWaitState	31	8
SetWaitState	38	8
Total Code Size / Max Stack Usage	1251	20

Table 61 Code Size and Stack Usage of SSD v3.1.0

Function Name	Code Size (Words)	Stack Usage (Words)
FlashInit	95	8
FlashErase	119	12
BlankCheck	93	16
FlashProgram	153	16

ProgramVerify	105	20
CheckSum	84	16
FlashSuspend	103	8
FlashResume	64	12
GetLock	72	8
SetLock	81	12
EERCheck	36	8
RWECheck	34	8
GetWaitState	31	8
SetWaitState	38	8
Total Code Size / Max Stack Usage	1108	20

Table 62 Code Size and Stack Usage of SSD v3.1.1

Function Name	Code Size (Words)	Stack Usage (Words)
FlashInit	98	8
FlashErase	122	12
BlankCheck	93	16
FlashProgram	153	16
ProgramVerify	105	20
CheckSum	84	16
FlashSuspend	103	8
FlashResume	64	12
GetLock	72	8
SetLock	81	12
EERCheck	36	8
RWECheck	34	8
GetWaitState	31	8
SetWaitState	38	8
Total Code Size / Max Stack Usage	1114	20

Table 63 Code Size and Stack Usage of SSD v3.1.2

Function Name	Code Size (Words)	Stack Usage (Words)
FlashInit	97	8
FlashErase	122	12
BlankCheck	93	16
FlashProgram	153	16
ProgramVerify	105	20
CheckSum	84	16
FlashSuspend	103	8
FlashResume	64	12
GetLock	72	8
SetLock	81	12
RWECheck	34	8
GetWaitState	31	8
SetWaitState	38	8
Total Code Size / Max Stack Usage	1077	20

A.2 Maximum CallBack Period / Executing Time

Table 64 Maximum CallBack Period / Executing Time at 80MHz System Clock of SSD v3.07

Function Name	Time (us)	Remark
FlashInit	3	Executing Time
FlashErase	25	CallBack Period
BlankCheck	44	CallBack Period
FlashProgram	6	CallBack Period
ProgramVerify	46	CallBack Period
CheckSum	46	CallBack Period
FlashSuspend	2	Executing Time
FlashResume	2	Executing Time
GetLock	2	Executing Time
SetLock	3	Executing Time
EERCheck	1	Executing Time
RWECheck	1	Executing Time
GetWaitState	2	Executing Time
SetWaitState	2	Executing Time

Table 65 Maximum CallBack Period / Executing Time at 80MHz System Clock of SSD v3.08

Function Name	Time (us)	Remark
FlashInit	2	Executing Time
FlashErase	20	CallBack Period
BlankCheck	44	CallBack Period
FlashProgram	5	CallBack Period
ProgramVerify	46	CallBack Period
CheckSum	47	CallBack Period
FlashSuspend	2	Executing Time
FlashResume	2	Executing Time
GetLock	2	Executing Time
SetLock	2	Executing Time
EERCheck	1	Executing Time
RWECheck	1	Executing Time
GetWaitState	2	Executing Time
SetWaitState	2	Executing Time

Table 66 Maximum CallBack Period / Executing Time at 80MHz System Clock of SSD v3.09

Function Name	Time (us)	Remark
FlashInit	2.5	Executing Time
FlashErase	19.7	CallBack Period
BlankCheck	44.7	CallBack Period
FlashProgram	5.2	CallBack Period
ProgramVerify	46.8	CallBack Period
CheckSum	47.1	CallBack Period
FlashSuspend	2.7	Executing Time

FlashResume	2.2	Executing Time
GetLock	1.3	Executing Time
SetLock	1.4	Executing Time
EERCheck	2	Executing Time
RWECheck	2.3	Executing Time
GetWaitState	1.4	Executing Time
SetWaitState	1.7	Executing Time

Table 67 Maximum CallBack Period / Executing Time at 80MHz System Clock of SSD v3.1.0

Function Name	Time (us)	Remark
FlashInit	2.8	Executing Time
FlashErase	3.2	CallBack Period
BlankCheck	44.7	CallBack Period
FlashProgram	5.4	CallBack Period
ProgramVerify	46.8	CallBack Period
CheckSum	47.1	CallBack Period
FlashSuspend	2.7	Executing Time
FlashResume	2.2	Executing Time
GetLock	1.3	Executing Time
SetLock	1.4	Executing Time
EERCheck	2	Executing Time
RWECheck	2.3	Executing Time
GetWaitState	1.4	Executing Time
SetWaitState	1.7	Executing Time

Table 68 Maximum CallBack Period / Executing Time at 80MHz System Clock of SSD v3.1.1 and v3.1.2

Function Name	Time (us)	Remark
FlashInit	2.5	Executing Time
FlashErase	2.9	CallBack Period
BlankCheck	44.8	CallBack Period
FlashProgram	5.0	CallBack Period
ProgramVerify	46.8	CallBack Period
CheckSum	47.1	CallBack Period
FlashSuspend	2.9	Executing Time
FlashResume	2.1	Executing Time
GetLock	2.1	Executing Time
SetLock	2.2	Executing Time
EERCheck[1]	1.3	Executing Time
RWECheck	1.4	Executing Time
GetWaitState	1.4	Executing Time
SetWaitState	1.8	Executing Time

Notes:

1. EERCheck is removed in v3.1.2.

A.3 Program / Erase Time

Table 69 Typical Erase Time at 80MHz System Clock of SSD v3.1.0

Block Selected	Block Size (Kbytes)	Erase Time (us)
LAS0	16	1053014
LAS1	48	2300041
LAS4	64	3228351
HAS0	128	9155377
HAS1	128	7092809
HAS2	128	10621299
HAS3	128	7011826
MAS0	128	9370554

Table 70 Typical Program Time at 80MHz System Clock of SSD v3.1.0

Program Size	Program Time (us)
8 Byte	22
16 Byte	35
24 Byte	48
1 Page	60
2 Page	117
4 Page	231
8 Page	460
LAS0 16K	28901
LAS1 48K	87046
LAS4 64K	116288
HAS0 128K	234605
HAS1 128K	234343
HAS2 128K	234235
HAS3 128K	234235
MAS0 128K	234234

Notes: The typical program/erase time is tested on an MPC5554 Rev0.4 part.

Table 71 Typical Erase Time at 80MHz System Clock of SSD v3.1.1 and v3.1.2

Block Selected	Block Size (Kbytes)	Erase Time (us)
LAS0	16	474614
LAS1	48	834795

LAS4	64	1332665
HAS0	128	3416971
HAS1	128	3626669
HAS2	128	3168466
HAS3	128	3414529
MAS0	128	3067599

Table 72 Typical Program Time at 80MHz System Clock of SSD v3.1.1 and v3.1.2

Program Size	Program Time (us)
8 Byte	17
16 Byte	22
24 Byte	27
1 Page 32B	33
2 Page 64B	63
4 Page 128B	123
8 Page 256B	244
LAS0 16K	14611
LAS1 48K	44678
LAS4 64K	60134
HAS0 128K	123888
HAS1 128K	123826
HAS2 128K	123875
HAS3 128K	123888
MAS0 128K	123853

Note: The typical program/erase time is tested on an MPC5554 RevA part.