UM1584
User manual

Standard software driver for C90LC Flash in SPC560xx, SPC56APxx and SPC563M64xx

Introduction

This document is the user manual for the Standard Software Driver (SSD) for C90LC Flash in SPC560Bxx, SPC560Cxx, SPC560Dxx, SPC560Pxx, SPC56APxx. SPC563M64xx.

www.BDTIC.com/ST

Contents

1 Introduction 4
1.1 Document overview 4
1.2 System overview 4
1.3 Features 5
1.4 System requirements 5
1.5 Acronyms 5

2 API specification 6
2.1 General overview 6
2.2 General type definitions 6
2.3 Configuration parameters and macros 6
2.4 Callback notification 7
2.5 Return codes 7
2.6 Normal mode functions 9
2.6.1 FlashInit() 9
2.6.2 FlashErase() 10
2.6.3 BlankCheck() 12
2.6.4 FlashProgram() 14
2.6.5 ProgramVerify() 16
2.6.6 CheckSum() 17
2.6.7 FlashSuspend() 18
2.6.8 FlashResume() 21
2.6.9 GetLock() 22
2.6.10 SetLock() 24
2.7 User Test Mode Functions 25
2.7.1 FlashArrayIntegrityCheck() 25
2.7.2 FlashECCLogicCheck() 28
2.7.3 FactoryMarginReadCheck() 29

Appendix A Documentation references 32

Revision history 33

www.BDTIC.com/ST

List of tables

Table 1. System requirements . . . 5
Table 2. Acronyms . . . 5
Table 3. Type definitions . . . 6
Table 4. SSD configuration structure field definition . . . 7
Table 5. Return codes . . . 8
Table 6. Arguments for FlashInit() . . . 9
Table 7. Return values for FlashInit() . . . 9
Table 8. Troubleshooting for FlashInit() . . . 9
Table 9. Arguments for FlashErase() . . . 10
Table 10. Return Values for FlashErase() . . . 11
Table 11. Troubleshooting for FlashErase() . . . 11
Table 12. Arguments for BlankCheck() . . . 12
Table 13. Return values for BlankCheck() . . . 13
Table 14. Troubleshooting for BlankCheck() . . . 13
Table 15. Arguments for FlashProgram() . . . 14
Table 16. Return values for FlashProgram() . . . 14
Table 17. Troubleshooting for FlashProgram() . . . 15
Table 18. Arguments for ProgramVerify() . . . 16
Table 19. Return values for ProgramVerify() . . . 16
Table 20. Troubleshooting for ProgramVerify() . . . 17
Table 21. Arguments for CheckSum() . . . 17
Table 22. Return Values for CheckSum() . . . 18
Table 23. Troubleshooting for CheckSum() . . . 18
Table 24. Arguments for FlashSuspend() . . . 18
Table 25. Return values for FlashSuspend() . . . 19
Table 26. SuspendState definitions . . . 19
Table 27. Suspending state and flag vs. C90LC status . . . 20
Table 28. Arguments for FlashResume() . . . 21
Table 29. Return values for FlashResume() . . . 21
Table 30. ResumeState Definitions . . . 21
Table 31. Arguments for GetLock() . . . 22
Table 32. Return Values for GetLock() . . . 22
Table 33. Troubleshooting for GetLock() . . . 22
Table 34. blkLockIndicator definitions . . . 23
Table 35. Arguments for SetLock() . . . 24
Table 36. Return Values for SetLock() . . . 24
Table 37. Troubleshooting for SetLock() . . . 25
Table 38. Arguments for FlashArrayIntegrityCheck() . . . 25
Table 39. Return values for FlashArrayIntegrityCheck() . . . 26
Table 40. Troubleshooting for FlashArrayIntegrityCheck() . . . 27
Table 41. Arguments for FlashECCLogicCheck() . . . 28
Table 42. Return Values for FlashECCLogicCheck() . . . 28
Table 43. Troubleshooting for FlashECCLogicCheck() . . . 28
Table 44. Arguments for FactoryMarginReadCheck() . . . 29
Table 45. Return Values for FactoryMarginReadCheck() . . . 30
Table 46. Troubleshooting for FactoryMarginReadCheck() . . . 30
Table 47. Document revision history . . . 33

www.BDTIC.com/ST

1 Introduction

1.1 Document overview

This document is the user manual for the Standard Software Driver (SSD) for C90LC Flash in SPC560xx, SPC56APxx and SPC563M64xx. The roadmap for the document is as follows.

Section 1.2 provides a brief overview of the system for general background knowledge of the project. *Section 1.3* shows the features of the driver. *Section 1.4* details the system requirement for the driver development. *Appendix A: Documentation references* lists the documents referred and terms used in making of this document. *Section 1.5* lists the acronyms used.

Chapter 2 describes the API specifications. In this section there are many sub sections, which describe the different aspects of the driver. *Section 2.1* provides a general overview of the driver. *Section 2.2* mentions about the type definitions used for the driver. *Section 2.3* mentions the driver configuration parameters. *Section 2.4* and *Section 2.5* describe the CallBack notifications and return codes for the driver respectively. *Section 2.6* and *Section 2.7* provides the detailed description of normal mode and special mode standard software Flash Driver APIs' respectively.

1.2 System overview

The SSD is a set of API's that enables user application to operate on the Flash module embedded on a microcontroller. The C90LC SSD contains a set of functions to program/erase C90LC Flash modules on STMicroelectronics SPC560xx, SPC56APxx and SPC563M64xx microcontrollers.

The C90LC Standard Software Driver (SSD) will provide the following API's:

- FlashInit
- FlashErase
- BlankCheck
- FlashProgram
- ProgramVerify
- CheckSum
- FlashSuspend
- FlashResume
- GetLock
- SetLock
- FlashArrayIntegrityCheck
- FlashECCLogicCheck
- FactoryMarginReadCheck

www.BDTIC.com/ST

1.3 Features

The C90LC SSD provides the following features:

- Two sets of driver binaries built with Standard PowerPC Book E instruction set and Variable-Length-Encoding (VLE) instruction set.
- Three sets of driver binaries built with three different toolchains.
- Drivers released in binary c-array format to provide compiler-independent support for non-debug-mode embedded applications.
- Drivers released in s-record format to provide compiler-independent support for debug-mode/JTAG programming tools.
- Each driver function is independent of each other so the end user can choose the function subset to meet their particular needs.
- Support page-wise programming for fast programming.
- Position-independent and ROM-able
- Ready-to-use demos illustrating the usage of the driver
- Concurrency support via callback

1.4 System requirements

The C90LC SSD is designed to support C90LC Flash module embedded on different microcontrollers. Before using this SSD on a different derivative microcontroller, user has to provide the information specific to the derivative through a configuration structure.

Table 1. System requirements

Tool name	Description	Version number
CodeWarrior IDE	Development tool	2.5
Diab PowerPC compiler	Compiler	5.7.0.0
Green Hills PowerPC compiler	Compiler	5.1.7
Lauterbach ICD tool	Jtag Debugger	n.a

1.5 Acronyms

Table 2. Acronyms

Abbreviation	Complete name
API	Application Programming Interface
BIU	Bus Interface Unit
ECC	Error Correction Code
EVB	Evaluation Board
RWW	Read While Write
SSD	Standard Software Driver

www.BDTIC.com/ST

2 API specification

2.1 General overview

The C90LC SSD has APIs to handle the erase, program, erase verify and program verify operations on the Flash. Apart from these, it also provides the feature for locking specific blocks and calculating Check sum. This SSD also provides 3 User Test APIs for checking the Array Integrity and the ECC Logic.

2.2 General type definitions

Table 3. Type definitions

Derived type	Size	C language type 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

2.3 Configuration parameters and macros

The configuration parameter which is used for SSD operations is explained in this section. The configuration parameters are handled as structure. The user should correctly initialize the fields including *c90flRegBase*, *mainArrayBase*, *shadowRowBase*, *shadowRowSize*, *pageSize* and *BDMEnable* before passing the structure to SSD functions. The pointer to *CallBack* has to be initialized either to a null pointer or a valid *CallBack* function pointer.

www.BDTIC.com/ST

Table 4. SSD configuration structure field definition

Parameter name	Type	Parameter description
c90flRegBase	UINT32	The base address of C90LC and BIU 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
pageSize	UINT32	The page size of the C90LC Flash
BDMEnable	UINT32	Defines the state of background debug mode (enable /disable)

The type definition for the structure is given below.

```
typedef struct _ssd_config
{
    UINT32 c90flRegBase;
    UINT32 mainArrayBase;
    UINT32 mainArraySize;
    UINT32 shadowRowBase;
    UINT32 shadowRowSize;
    UINT32 lowBlockNum;
    UINT32 midBlockNum;
    UINT32 highBlockNum;
    UINT32 pageSize;
    UINT32 BDMEnable;
} SSD_CONFIG, *PSSD_CONFIG;
```

2.4 Callback notification

The Standard Software Driver facilitates the user to supply a pointer to '*CallBack()*' function so that time-critical events can be serviced during C90LC Standard Software driver operations. Servicing watchdog timers is one such time critical event. If it is not necessary to provide the CallBack service, the user will be able to disable it by a NULL function macro.

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

The job processing callback notifications shall have no parameters and no return value.

2.5 Return codes

The return code will be returned to the caller function to notify the success or errors of the API execution. These are the possible values of return code:

www.BDTIC.com/ST

Table 5. Return codes

Name	Value	Description
C90FL_OK	0x00000000	The requested operation is successful.
C90FL_INFO_RWE	0x00000001	RWE bit is set before Flash operations.
C90FL_INFO_EER	0x00000002	EER bit is set before Flash operations.
C90FL_ERROR_ALIGNMENT	0x00000100	Alignment error.
C90FL_ERROR_RANGE	0x00000200	Address range error.
C90FL_ERROR_BUSY	0x00000300	New program/erase cannot be preformed while a high voltage operation is already in progress.
C90FL_ERROR_PGOOD	0x00000400	The program operation is unsuccessful.
C90FL_ERROR_EGOOD	0x00000500	The erase operation is unsuccessful.
C90FL_ERROR_NOT_BLANK	0x00000600	There is a non-blank Flash memory location within the checked Flash memory region.
C90FL_ERROR_VERIFY	0x00000700	There is a mismatch between the source data and the content in the checked Flash memory.
C90FL_ERROR_LOCK_INDICATOR	0x00000800	Invalid block lock indicator.
C90FL_ERROR_RWE	0x00000900	Read-while-write error occurred in previous reads.
C90FL_ERROR_PASSWORD	0x00000A00	The password provided cannot unlock the block lock register for register writes
C90FL_ERROR_AIC_MISMATCH	0x00000B00	In '*FlashArrayIntegrityCheck()*' the MISR values generated by the hardware do not match the values passed by the user.
C90FL_ERROR_AIC_NO_BLOCK	0x00000C00	In '*FlashArrayIntegrityCheck()*' no blocks have been enabled for Array Integrity check
C90FL_ERROR_FMR_MISMATCH	0x00000D00	In '*FactoryMarginReadCheck()*' the MISR values generated by the hardware do not match the values passed by the user.
C90FL_ERROR_FMR_NO_BLOCK	0x00000E00	In '*FactoryMarginReadCheck()*' no blocks have been enabled for Array Integrity check
C90FL_ERROR_ECC_LOGIC	0x00000F00	In '*FlashECCLogicCheck()*' the simulated ECC error has not occurred.
C90FL_ERROR_SUSP	0x00001000	On going high voltage operation cannot be suspended.

www.BDTIC.com/ST

2.6 Normal mode functions

2.6.1 FlashInit()

Description: This function will read the Flash configuration information from the Flash control registers and initialize parameters in SSD configuration structure. '*FlashIni()*' must be called prior to any other Flash operations.

Definition: `UINT32 FlashInit (PSSD_CONFIG pSSDConfig);`

Arguments:

Table 6. Arguments for FlashInit()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.

Returns:

Table 7. Return values for FlashInit()

Type	Description	Possible values
UINT32	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 C90FL_OK, indicates a kind of current status of C90LC module.	C90FL_OK C90FL_INFO_EER C90FL_INFO_RWE

Troubleshooting:

Table 8. Troubleshooting for FlashInit()

Error Codes	Possible Causes	Solution
C90FL_INFO_EER	An ECC Error occurred during a previous read.	Clear FLASH_MCR-EER bit.
C90FL_INFO_RWE	A Read While Write Error occurred during a previous read	Clear the FLASH_MCR-RWE bit.

Comments: '*FlashIni()*' will check the C90FL_MCR_RWE and C90FL_MCR_EER bit, but does not clear them when any of them is set. If RWE bit is set, Flash program/erase operations can still be performed.

Assumptions: None.

www.BDTIC.com/ST

2.6.2 FlashErase()

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 relevant error code will be returned if there is any error.

Definition:

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

Arguments:

Table 9. Arguments for FlashErase()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
shadowFlag	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 are specified by *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* respectively.
lowEnabledBlocks	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	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	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	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

www.BDTIC.com/ST

Returns:

Table 10. Return Values for FlashErase()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_BUSY C90FL_ERROR_EGOOD

Troubleshooting:

Table 11. Troubleshooting for FlashErase()

Error codes	Possible causes	Solution
C90FL_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: erase in progress (FLASH_MCR-ERS is high); program in progress (FLASH_MCR-PGM is high);
C90FL_ERROR_ EGOOD	Erase operation failed.	Check if the C90LC is available and high voltage is applied to C90LC. Then try to do the erase operation again.

Comments: When *shadowFlag* is set to FALSE, the *'FlashErase()'* 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. The selection of the blocks of the main array is determined by setting/clearing the corresponding bit in *lowEnabledBlocks*, midEnabledBlocks or *highEnabledBlocks*.

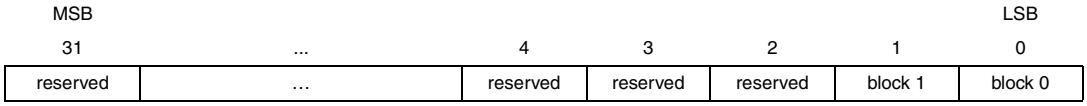
The bit allocations for blocks in one address space are: bit 0 is assigned to block 0, bit 1 to block 1, etc. The following diagrams give an example of the formats of *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* for the C90LC module. Refer to *Appendix A: Documentation references* for the valid bits configuration depending from the specific microcontroller Flash module sectors.

Bit Allocation for Blocks in Low Address Space:

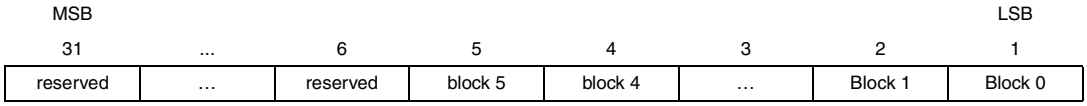
MSB							LSB
31	…	10	9	8	…	1	0
reserved	…	reserved	block 9	block 8	…	block 1	block 0

www.BDTIC.com/ST

Bit Allocation for Blocks in Middle Address Space:



Bit Allocation for Blocks in High Address Space:



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 C90FL_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 C90LC module. '*FlashErase()*' will return C90FL_ERROR_BUSY when trying to do so. Similarly, once an erasing operation has started on C90LC 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 module having one or more blocks being erased. Otherwise, it will cause a Read-While-Write error.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

2.6.3 BlankCheck()

Description: This function will check on the specified Flash range in the main array or shadow row for blank state. If the blank checking fails, the first failing address and the failing data in Flash block will be saved.

Description:

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

Arguments:

Table 12. Arguments for BlankCheck()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
dest	Destination address to be checked.	Any accessible address aligned on double word boundary in main array or shadow row
size	Size, in bytes, of the Flash region to check.	If size = 0, the return value is C90FL_OK. It should be multiple of 8 and its combination with *dest* should fall in either main array or shadow row.

www.BDTIC.com/ST

Table 12. Arguments for BlankCheck() (continued)

Argument	Description	Range
pFailAddress	Return the address of the first non-blank Flash location in the checking region	Only valid when this function returns C90FL_ERROR_NOT_BLANK.
pFailData	Return the content of the first non-blank Flash location in the checking region.	Only valid when this function returns C90FL_ERROR_NOT_BLANK.
CallBack	Address of void callback function	Any addressable void function address. To disable it use NULL_CALLBACK macro.

Returns:

Table 13. Return values for BlankCheck()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_ALIGNMENT C90FL_ERROR_RANGE C90FL_ERROR_NOT_BLANK

Troubleshooting:

Table 14. Troubleshooting for BlankCheck()

Returned error bits	Description	Solution
C90FL_ERROR_ALIGNMENT	The *dest*/*size* are not properly aligned.	Check if *dest* and size are aligned on double word (64-bit) boundary.
C90FL_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid C90LC array ranges.	Check *dest* and *dest*+*size*. The area to be checked must be within main array space or shadow space.
C90FL_ERROR_NOT_BLANK	There is a non-blank double word within the area to be checked.	Erase the relevant blocks and check again.

Comments: 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.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

www.BDTIC.com/ST

2.6.4 FlashProgram()

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 relevant error code will be returned if there is any error.

Description:

```
UINT32 FlashProgram (PSSD_CONFIG pSSDConfig,
UINT32 dest,
UINT32 size,
UINT32 source,
void (*CallBack)(void));
```

Arguments:

Table 15. Arguments for FlashProgram()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
Dest	Destination address to be programmed in Flash memory.	Any accessible address aligned on double word boundary in main array or shadow row.
Size	Size, in bytes, of the Flash region to be programmed.	If size = 0, C90FL_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	Source program buffer address.	This address must reside on word boundary.
CallBack	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

Returns:

Table 16. Return values for FlashProgram()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_BUSY C90FL_ERROR_ALIGNMENT C90FL_ERROR_RANGE C90FL_ERROR_PGOOD

www.BDTIC.com/ST

Troubleshooting:

Table 17. Troubleshooting for FlashProgram()

Returned error bits	Description	Solution
C90FL_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 (MCR-PGM high); 2. program not in progress (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.
C90FL_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.
C90FL_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid C90LC address range.	Check *dest* and *dest*+*size*. Both should fall in the same C90LC address ranges, i.e. both in main array or both in shadow row
C90FL_ERROR_PGOOD	Program operation failed because this operation cannot pass PEG check.	Repeat the program operation. Check if the C90LC is invalid or high voltage applied to C90LC is unsuitable.

Comments: 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 C90FL_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 C90LC module. '*FlashProgram()*' will return C90FL_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.

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.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

www.BDTIC.com/ST

2.6.5 ProgramVerify()

Description: This function will check if a programmed Flash range matches the corresponding source data buffer. In case of mismatch, the failed address, destination value and source value will be saved and relevant error code will be returned.

Definition:

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

Arguments:

Table 18. Arguments for ProgramVerify()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
Dest	Destination address to be verified in Flash memory.	Any accessible address aligned on double word boundary in main array or shadow row.
Size	Size, in byte, of the Flash region to verify.	If size = 0, C90FL_OK will be returned. Its combination with *dest* should fall within either main array or shadow row.
Source	Verify source buffer address.	This address must reside on word boundary.
pFailAddress	Return first failing address in Flash.	Only valid when the function returns C90FL_ERROR_VERIFY.
pFailData	Returns first mismatch data in Flash.	Only valid when this function returns C90FL_ERROR_VERIFY.
pFailSource	Returns first mismatch data in buffer.	Only valid when this function returns C90FL_ERROR_VERIFY.
CallBack	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

Returns:

Table 19. Return values for ProgramVerify()

Type	Description	Possible Values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_ALIGNMENT C90FL_ERROR_RANGE C90FL_ERROR_VERIFY

www.BDTIC.com/ST

Troubleshooting:

Table 20. Troubleshooting for ProgramVerify()

Returned error bits	Description	Solution
C90FL_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
C90FL_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid C90LC address range.	Check *dest* and *dest*+*size*, both should fall in the same C90LC address ranges, i.e. both in main array or both in shadow row
C90FL_ERROR_VERIFY	The content in C90LC and source data mismatch.	Check the correct source and destination addresses, erase the block and reprogram data into Flash.

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

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

2.6.6 CheckSum()

Description: This function will perform a 32-bit sum over the specified Flash memory range without carry, which provides a rapid method for checking data integrity.

Definition:

```
UINT32 CheckSum (PSSD_CONFIG pSSDConfig,
UINT32 dest,
UINT32 size,
UINT32 *pSum,
void (*CallBack)(void));
```

Arguments:

Table 21. Arguments for CheckSum()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
Dest	Destination address to be summed in Flash memory.	Any accessible address aligned on double word boundary in either main array or shadow row.
Size	Size, in bytes, of the Flash region to check sum.	If size is 0 and the other parameters are all valid, C90FL_OK will be returned. Its combination with *dest* should fall within either main array or shadow row.
pSum	Returns the sum value.	0x00000000 - 0xFFFFFFFF. Note that this value is only valid when the function returns C90FL_OK.
CallBack	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

www.BDTIC.com/ST

Returns:

Table 22. Return Values for CheckSum()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_ALIGNMENT C90FL_ERROR_RANGE

Troubleshooting:

Table 23. Troubleshooting for CheckSum()

Returned error bits	Description	Solution
C90FL_ERROR_ALIGNMENT	This error indicates that *dest*/*size* 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
C90FL_ERROR_RANGE	The area specified by *dest* and *size* is out of the valid C90LC address range.	Check *dest* and *dest+size*, both should fall in the same C90LC address ranges, i.e. both in main array or both in shadow row

Comments: None.

Assumptions: It assumes that the Flash block is initialized using a 'FlashInit()' API.

2.6.7 FlashSuspend()

Description: This function will check if there is any high voltage operation, erase or program, in progress on the C90LC module and if the operation can be suspended. This function will suspend the ongoing operation if it can be suspended (for C90LC modules it applies only to erase operation).

Definition:

```
UINT32 FlashSuspend (PSSD_CONFIG pSSDConfig,
UINT8 *suspendState,
BOOL *suspendFlag);
```

Arguments:

Table 24. Arguments for FlashSuspend()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
suspendState	Indicate the suspend state of C90LC module after the function being called.	All return values are enumerated in *Section 2.3*.
suspendFlag	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.

www.BDTIC.com/ST

Returns:

Table 25. Return values for FlashSuspend()

Type	Description	Possible Values
UINT32	Successful completion.	C90FL_OK
UINT32	Ongoing operation cannot be suspended.	C90FL_ERROR_SUSP

Troubleshooting: None.

Comments: After calling '*FlashSuspend()*', if no error is returned, 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.

Following table defines and describes various suspend states and associated suspend codes.

Table 26. SuspendState definitions

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.

www.BDTIC.com/ST

Table 26. SuspendState definitions (continued)

Argument	Code	Description	Valid operation after suspend
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.
NO_SUS	8	The ongoing high voltage operation has not been suspended.	No operation is valid on both main array space and shadow row.

The table below lists the Suspend Flag values returned against the Suspend State and the Flash block status.

Table 27. Suspending state and flag vs. C90LC status

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

The values of EHV, ERS, ESUS, PGM, PSUS and PEAS represent the C90LC status at the entry of FlashSuspend;

0: Logic zero; 1: Logic one; X: Do-not-care.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

www.BDTIC.com/ST

2.6.8 FlashResume()

Description: This function will check if there is any suspended erase or program operation on the C90LC module, and will resume the suspended operation if there is any.

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

Arguments:

Table 28. Arguments for FlashResume()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
resumeState	Indicate the resume state of C90LC module after the function being called.	All return values are listed in *Table 30*.

Returns:

Table 29. Return values for FlashResume()

Type	Description	Possible values
UINT32	Successful completion.	C90FL_OK

Troubleshooting: None.

Comments: 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()*'.

Following table defines and describes various resume states and associated resume codes.

Table 30. ResumeState Definitions

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

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

www.BDTIC.com/ST

2.6.9 GetLock()

Description: This function will check the block locking status of Shadow/Low/Middle/High address spaces in the C90LC module.

Definition:

```
UINT32 GetLock (PSSD_CONFIG pSSDConfig,
UINT8 blkLockIndicator,
BOOL *blkLockEnabled,
UINT32 *blkLockState);
```

Arguments:

Table 31. Arguments for GetLock()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
blkLockIndicator	Indicating the address space and the block locking level, which determines the address space block locking register to be checked.	Refer to *Table 34* for valid values for this parameter.
blkLockEnabled	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	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

Returns:

Table 32. Return Values for GetLock()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_LOCK_INDICATOR

Troubleshooting:

Table 33. Troubleshooting for GetLock()

Returned error bits	Possible causes	Solution
C90FL_ERROR_LOCK_INDICATOR	The input blkLockIndicator is invalid.	Set this argument to correct value listed in *Table 34*

Comments: Following table defines and describes various *blkLockIndicator* values.

www.BDTIC.com/ST

Table 34. blkLockIndicator definitions

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

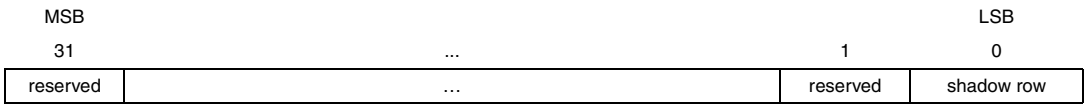
For Shadow/Low/Mid address spaces, there are two block lock levels. The secondary level of block locking provides an alternative means to protect blocks from being modified. A logical “OR” of the corresponding bits in the primary and secondary lock registers for a block determines the final lock status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

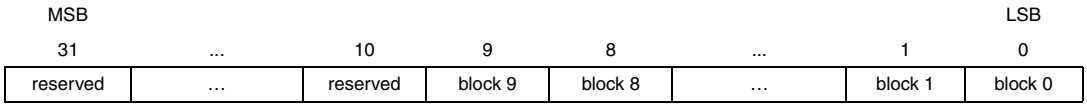
The indicated address space determines the valid bits of *blkLockState*. For either Low/Mid/High address spaces, if blocks corresponding to valid block lock state bits are not present (due to configuration or total memory size), values for these block lock state bits will be always 1 because such blocks are locked by hardware on reset. These blocks cannot be unlocked by software with ‘*SetLock()*’ function.

The following diagrams give an example of the block bitmap definitions of *blkLockState* for shadow/Low/Mid/High address spaces. Refer to *Appendix A: Documentation references* for the valid bits configuration depending from the specific microcontroller Flash module sectors.

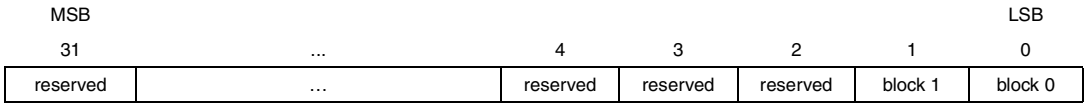
blkLockState Bit Allocation for Shadow Address Space:



blkLockState Bit Allocation for Low Address Space:



blkLockState Bit Allocation for Mid Address Space:



www.BDTIC.com/ST

blkLockState Bit Allocation for High Address Space:

MSB							LSB
31	...	6	5	4	...	1	0
reserved	...	reserved	block 5	block 4	...	block 1	block 0

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

2.6.10 SetLock()

Description: This function will set the block lock state for Shadow/Low/Middle/High address space on the C90LC module to protect them from program/erase. The API will provide password to enable block lock register writes when needed and write the block lock value to block lock register for the requested address space.

Definition:

```
UINT32 SetLock (PSSD_CONFIG pSSDConfig,
UINT8 blkLockIndicator,
UINT32 blkLockState,
UINT32 password);
```

Arguments:

Table 35. Arguments for SetLock()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
blkLockIndicator	Indicating the address space and the protection level of the block lock register to be read.	Refer to *Table 34* for valid codes for this parameter.
blkLockState	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	A password is required to enable the block lock register for register write.	Correct passwords for block lock registers are 0xA1A1_1111 for Low/Mid Address Space Block Locking Register, 0xC3C3_3333 for Secondary Low/Mid Address Space Block Locking Register, and 0xB2B2_2222 for High Address Space Block Select Register.

Returns:

Table 36. Return Values for SetLock()

Type	Description	Possible Values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_LOCK_INDICATOR C90FL_ERROR_PASSWORD

www.BDTIC.com/ST

Troubleshooting: The troubleshooting mentioned below comprises of hardware errors due to both P Flash block erase verify and P Flash section erase verify command. Apart from these the input based error handling is also mentioned.

Table 37. Troubleshooting for SetLock()

Returned error bits	Possible causes	Solution
C90FL_ERROR_LOCK_INDICATOR	The input blkLockIndicator is invalid.	Set this argument to correct value listed in *Table 34*.
C90FL_ERROR_PASSWORD	The given password cannot enable the block lock register for register writes.	Pass in a correct password.

Comments: The bit field allocation for *blkLockState* is same as that in '*GetLock()*' function.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

2.7 User Test Mode Functions

2.7.1 FlashArrayIntegrityCheck()

Description: This function will check the array integrity of the Flash. The user specified address sequence is used for array integrity reads and the operation is done on the specified blocks. The MISR values calculated by the hardware is compared to the values passed by the user, if they are not the same, then an error code is returned.

Definition:

```
UINT32 FlashArrayIntegrityCheck (PSSD_CONFIG pSSDConfig,
UINT32 lowEnabledBlocks,
UINT32 midEnabledBlocks,
UINT32 highEnabledBlocks,
UINT8 addrSeq,
MISR misrValue,
void (*CallBack)(void));
```

Arguments:

Table 38. Arguments for FlashArrayIntegrityCheck()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
lowEnabledBlocks	To select the array blocks in low address space for erasing.	Bit-mapped value. Select the block in the low address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *lowEnabledBlocks*. If there is not any block to be evaluated in the low address space, *lowEnabledBlocks* must be set to 0.

www.BDTIC.com/ST

Table 38. Arguments for FlashArrayIntegrityCheck() (continued)

Argument	Description	Range
midEnabledBlocks	To select the array blocks in mid address space for erasing.	Bit-mapped value. Select the block in the middle address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *midEnabledBlocks*. If there is not any block to be evaluated in the middle address space, *midEnabledBlocks* must be set to 0.
highEnabledBlocks	To select the array blocks in high address space for erasing.	Bit-mapped value. Select the block in the high address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *highEnabledBlocks*. If there is not any block to be evaluated in the high address space, *highEnabledBlocks* must be set to 0.
addrSeq	To determine the address sequence to be used during array integrity checks.	The default sequence (*addrSeq* = 0) is meant to replicate sequences normal “user” code follows, and thoroughly check the read propagation paths. This sequence is proprietary. The alternative sequence (*addrSeq* = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence.
misrValue	A structure variable containing the MISR values calculated by the user using the offline MISR generation tool.	The individual MISR words can range from 0x00000000 - 0xFFFFFFFF
CallBack	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

Returns:

Table 39. Return values for FlashArrayIntegrityCheck()

Type	Description	Possible vlues
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_AIC_MISMATCH C90FL_ERROR_AIC_NO_BLOCK

Troubleshooting: The trouble shooting given here comprises of hardware errors and input parameter error.

www.BDTIC.com/ST

Table 40. Troubleshooting for FlashArrayIntegrityCheck()

Returned error bits	Possible causes	Solution
C90FL_ERROR_AIC_MISMATCH	The MISR value calculated by the user is incorrect.	Re-calculate the MISR values using the correct Data and addrSeq.
	The MISR calculated by the Hardware is incorrect.	Hardware Error.
C90FL_ERROR_AIC_NO_BLOCK	None of the Blocks are enabled for Array Integrity Check	Enable any of the blocks using variables *lowEnabledBlocks, midEnabledBlocks* and *highEnabledBlock.*

Comments: The inputs *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* are bit-mapped arguments that are used to select the blocks to be evaluated in the Low/Mid/High address spaces of main array. The selection of the blocks of the main array is determined by setting/clearing the corresponding bit in *lowEnabledBlocks*, *midEnabledBlocks* or *highEnabledBlocks*.

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 give an example of the formats of *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* for the C90LC module.

Refer to *Appendix A: Documentation references* for the valid bits configuration depending from the specific microcontroller Flash module sectors.

Bit Allocation for Blocks in Low Address Space:

MSB							LSB
31	...	10	9	8	...	1	0
reserved	...	reserved	block 9	block 8	...	block 1	block 0

Bit Allocation for Blocks in Middle Address Space:

MSB						LSB
31	...	4	3	2	1	0
reserved	...	reserved	reserved	reserved	block 1	block 0

Bit Allocation for Blocks in High Address Space:

MSB							LSB
31	...	6	5	4	...	1	0
reserved	...	reserved	block 5	block 4	...	Block 1	Block 0

If no blocks are enabled the C90FL_ERROR_AIC_NO_BLOCK error code is returned.

Depending on the address sequence specified the MISR values are calculated for the enabled blocks using the corresponding sequence. If the MISR values calculated by the hardware is not the same as the values passed to this API by the user then the API returns the error code C90FL_ERROR_AIC_MISMATCH.

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

www.BDTIC.com/ST

2.7.2 FlashECCLogicCheck()

Description: This function will check the ECC logic of the Flash. The API will simulate a single or double bit fault depending on the user input. If the simulated ECC error is not detected, then the error code C90FL_ERROR_ECC_LOGIC is returned.

Definition:

```
UINT32 FlashECCCLogicCheck (PSSD_CONFIG pSSDConfig,
UINT64 dataVal,
UINT64 errBits,
UINT32 eccValue)
```

Arguments:

Table 41. Arguments for FlashECCLogicCheck()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
dataValue	The 64 bits of data for which the ECC is calculated. The bits of *dataValue* are flipped to generate single or double bit faults.	Any 64-bit value.
errBits	Is a 64-bit mask of the bits at which the user intends to inject error.	Any 64-bit value, except zero.
eccValue	It's a 32 bit value which has to be passed by user. This is calculated ny using an offline ECC Calculator.	This is a corresponding ECC value for the data value passed by the user. Note: Same data words should be used in offline ECC calculator and Flash ECC logic check API.

Returns:

Table 42. Return Values for FlashECCLogicCheck()

Type	Description	Possible Values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_ECC_LOGIC

Troubleshooting: The trouble shooting given here comprises of hardware errors and input parameter error.

Table 43. Troubleshooting for FlashECCLogicCheck()

Returned error bits	Possible causes	Solution
C90FL_ERROR_ECC_LOGIC	Hardware Failure.	Hardware error.

Comments: Depending on the *errBits* value, a single or double bit faults are simulated. When a Flash read is done, if the simulated error has not occurred, then the API returns the error code C90FL_ERROR_ECC_LOGIC.

www.BDTIC.com/ST

Assumptions: It assumes that the Flash block is initialized using a '*FlashInit()*' API.

2.7.3 FactoryMarginReadCheck()

Description: This function will check the Factory Margin reads of the Flash. The user specified margin level is used for reads and the operation is done on the specified blocks. The MISR values calculated by the hardware is compared to the values passed by the user, if they are not the same, then an error code is returned.

Definition:

```
UINT32 FactoryMarginReadCheck (PSSD_CONFIG pSSDConfig,
UINT32 lowEnabledBlocks,
UINT32 midEnabledBlocks,
UINT32 highEnabledBlocks,
UINT8 marginLevel,
MISR misrValue,
void (*CallBack)(void));
```

Arguments:

Table 44. Arguments for FactoryMarginReadCheck()

Argument	Description	Range
pSSDConfig	Pointer to the SSD Configuration Structure.	The values in this structure are chip-dependent. Please refer to *Section 2.3* for more details.
lowEnabledBlocks	To select the array blocks in low address space for erasing.	Bit-mapped value. Select the block in the low address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *lowEnabledBlocks*. If there is not any block to be evaluated in the low address space, *lowEnabledBlocks* must be set to 0.
midEnabledBlocks	To select the array blocks in mid address space for erasing.	Bit-mapped value. Select the block in the middle address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *midEnabledBlocks*. If there is not any block to be evaluated in the middle address space, *midEnabledBlocks* must be set to 0.
highEnabledBlocks	To select the array blocks in high address space for erasing.	Bit-mapped value. Select the block in the high address space whose array integrity is to be evaluated by setting 1 to the appropriate bit of *highEnabledBlocks*. If there is not any block to be evaluated in the high address space, *highEnabledBlocks* must be set to 0.
marginLevel	To determine the margin level to be used during factory margin read checks.	Selects the margin level that is being checked. Margin can be checked to an erased level (marginLevel = 1) or to a programmed level (marginLevel = 0).
misrValue	A structure variable containing the MISR values calculated by the user using the offline MISR generation tool.	The individual MISR words can range from 0x00000000 - 0xFFFFFFFF
CallBack	Address of void call back function pointer.	Any addressable void function address. To disable it use NULL_CALLBACK macro.

www.BDTIC.com/ST

Returns:

Table 45. Return Values for FactoryMarginReadCheck()

Type	Description	Possible values
UINT32	Successful completion or error value.	C90FL_OK C90FL_ERROR_FMR_MISMATCH C90FL_ERROR_FMR_NO_BLOCK

Troubleshooting: The trouble shooting given here comprises of hardware errors and input parameter error.

Table 46. Troubleshooting for FactoryMarginReadCheck()

Returned error bits	Possible causes	Solution
C90FL_ERROR_FMR_MISMATCH	The MISR value calculated by the user is incorrect.	Re-calculate the MISR values using the correct Data and address.
	The MISR calculated by the Hardware is incorrect.	Hardware Error.
C90FL_ERROR_FMR_NO_BLOCK	None of the Blocks are enabled for Factory Margin Read Check	Enable any of the blocks using variables *lowEnabledBlocks, midEnabledBlocks* and *highEnabledBlock.*

Comments: The inputs *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* are bit-mapped arguments that are used to select the blocks to be evaluated in the Low/Mid/High address spaces of main array. The selection of the blocks of the main array is determined by setting/clearing the corresponding bit in *lowEnabledBlocks*, *midEnabledBlocks* or *highEnabledBlocks*.

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 give an example of the formats of *lowEnabledBlocks*, *midEnabledBlocks* and *highEnabledBlocks* for the C90LC module.

Refer to *Appendix A: Documentation references* for the valid bits configuration depending from the specific microcontroller Flash module sectors.

Bit Allocation for Blocks in Low Address Space:

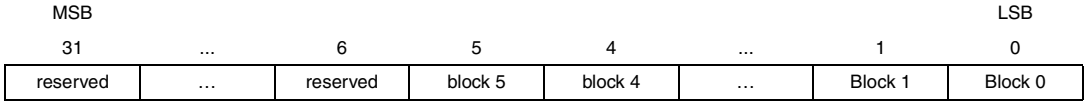
MSB							LSB
31	...	10	90	8	...	1	0
reserved	…	reserved	block 9	block 8	…	block 1	block 0

Bit Allocation for Blocks in Middle Address Space:

MSB						LSB
31	...	4	3	2	1	0
reserved	…	reserved	reserved	reserved	block 1	block 0

Bit Allocation for Blocks in High Address Space:

www.BDTIC.com/ST



If no blocks are enabled the C90FL_ERROR_FMR_NO_BLOCK error code is returned.

The MISR values are calculated for the enabled blocks using the logical sequence. If the MISR values calculated by the hardware is not the same as the values passed to this API by the user then the API returns the error code C90FL_ERROR_FMR_MISMATCH.

Assumptions: It assumes that the Flash block is initialized using a ‘FlashInit()’ API.

www.BDTIC.com/ST

Appendix A Documentation references

1. SPC560D30L1, SPC560D30L3, SPC560D40L1, SPC560D40L3 32-bit MCU family built on the embedded Power Architecture® (RM0045, DocID 16886)
2. SPC560B4x, SPC560B50, SPC560C4x, SPC560C50 32-bit MCU family built on the embedded Power Architecture® (RM0017, DocID 14629)
3. Support microcontrollers SPC560B54x, SPC560B60x and SPC560B64x (RM0037, DocID 15700)
4. SPC560P34/SPC560P40 32-bit MCU family built on the embedded Power Architecture® (RM0046, Doc ID 16912)
5. 32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications (RM0022, DocID 14891)
6. 32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications (RM0083, DocID 018714)
7. SPC563M64xx - 32-bit Power Architecture® based MCU with up to 1.5 Mbyte Flash and 111 Kbyte RAM memories (RM0015, DocID 14499)

www.BDTIC.com/ST

Revision history

Table 47. Document revision history

Date	Revision	Changes
12-Nov-2012	1	Initial release.

www.BDTIC.com/ST

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2012 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com

www.BDTIC.com/ST