

Standard Software Driver for C90TFS/FTFx Flash
User's Manual

© *Copyright Freescale Semiconductor 2010-2014.*
All Rights Reserved

REVISION LIST

Version No.	Date	Author	Description
1.0	06-07-2010	FPT Team	Initial Version
1.1	09-22-2010	FPT Team	Update for the FTFx_KX_256K_256K_4K_2K_2K derivative
1.2	09-30-2010	FPT Team	Update performance result for the FTFx_KX_256K_256K_4K_2K_2K derivative
1.3	02-24-2011	FPT Team	Update for the FTFx_KX_512K_0K_0K_2K_0K derivative
1.4	03-16-2011	FPT Team	Update for the FTFx_JX_128K_32K_2K_1K_1K derivative
1.5	05-05-2011	FPT Team	Update for the FTFx_FX_256K_32K_2K_1K_1K derivative Update callback time for FTFx_JX_128K_32K_2K_1K_1K
1.6	10-31-2011	FPT Team	Update for the FTFx_KX_512K_512K_16K_4K_4K
1.7	11-17-2011	FPT Team	Update for the FTFx_KX_1024K_0K_16K_4K_0K
1.8	12-27-2011	FPT Team	Update list of derivative in table 7. Add comment for FlashReadResource() function.
1.9	02-21-2012	FPT Team	Update to support more Kinetis derivatives. Update PFlashSwap() function prototype. Add performance/code size for FTFx_KX_128K_32K_2K_1K_1K derivative (K20).
2.0	06-12-2012	FPT Team	Update to support FTFx_NX_256K_32K_2K_2K_1K FTFx_NX_128K_32K_2K_2K_1K FTFx_NX_96K_32K_2K_2K_1K FTFx_NX_64K_32K_2K_2K_1K derivatives. Update FlashProgramOnce() and FlashReadInce() function prototypes. Add performance/code size for FTFx_NX_256K_32K_2K_2K_1K derivative (Nevis2).
2.1	07-25-2012	FPT Team	Update to support derivatives: FTFx_LX_128K_0K_0K_1K_0K FTFx_LX_64K_0K_0K_1K_0K FTFx_LX_32K_0K_0K_1K_0K FTFx_LX_16K_0K_0K_1K_0K FTFx_LX_8K_0K_0K_1K_0K Add performance/code size for FTFx_LX_128K_0K_0K_1K_0K (L2K) and FTFx_LX_32K_0K_0K_1K_0K (L1PT)
2.2	08-23-2012	FPT Team	Update to support derivatives: FTFx_AX_64K_0K_0K_1K_0K FTFx_AX_48K_0K_0K_1K_0K FTFx_AX_32K_0K_0K_1K_0K FTFx_AX_16K_0K_0K_1K_0K Recollect performance data for FTFx_LX_128K_0K_0K_1K_0K (L2K), FTFx_LX_32K_0K_0K_1K_0K (L1PT), and FTFx_AX_64K_0K_0K_1K_0K (MC56F82748)
3.0	05-28-2013	FPT Team	<ul style="list-style-type: none"> - Customize the content of document by removing unnecessary sections. - Update functional prototype of below APIs to match to latest driver version (v034): FlashProgramLongWord(), FlashProgramPhrase(), EEWrite(), FFlashReadResource(), and

			ssdConfigStructure. - Add important note section. - Update code size section. - Remove write/erase time and callback time.
3.1	07-09-2013	FPT Team	- Update prototype of FlashProgramCheck(). - Add performance data - Add appendix B
3.2	12-19-2013	FPT Team	- Redefine value for return code - Change to use macros for FlashGet/SetInterruptEnable API. - Remove PFlashGetSwapStatus API and add PFlashSwapCtl API - Update performance data for code size optimization - Add to support MK22FN512/MK22FN256 in appendix B
3.3	01-16-2014	FPT Team	Change prototype of PFlashSwapCtl function to provide current swap block status and next block swap status information
3.4	04-08-2014	FPT Team	- Update functional prototypes and redefine basic types to conform SDK (MISRA –C : 2004) - Update PFlashSwap() to support both NULL_SWAP_CALLBACK function and user's Callback function - Update performance data for code size optimization
3.5	14-10-2014	FPT	- Add BLOCK_COMMANDS description into section 2.3 - Add section: Porting to New Derivative Guideline - Update Appendix B to support KV30, KV31, KV40, KV43, KV44, KV45, KV46, K24S, K02 K60F2M0 platforms

TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	Document Overview	1
1.2	System Overview	1
1.3	Features	1
1.4	System Requirements	1
1.5	Documentation References	2
1.6	Terms	2
1.7	Acronyms	2
2	API SPECIFICATION	3
2.1	General Overview	3
2.2	General Type Definitions	3
2.3	Configuration parameter	3
2.4	Configuration Macros	4
2.5	Flash Interrupt Macros	5
2.6	CallBack function	5
2.7	Return Codes	6
2.8	Port to New Derivative Guide Line	6
2.8.1	Add New Derivative Macro to SSD_FTFx_Common.h	6
2.8.2	Add Flash Configuration File for The New Derivative	6
2.8.3	Include The Flash Configuration File on SSD_FTFx_Internal.h	9
2.9	SSD General Functions	9
2.9.1	FlashInit()	9
2.9.2	PFlashGetProtection()	10
2.9.3	PFlashSetProtection()	11
2.9.4	DFlashGetProtection()	11
2.9.5	DFlashSetProtection()	12
2.9.6	EERAMGetProtection()	13
2.9.7	EERAMSetProtection()	13
2.9.8	FlashGetSecurityState()	14
2.9.9	FlashSecurityBypass()	15
2.9.10	FlashEraseAllBlock()	15
2.9.11	FlashEraseBlock()	16
2.9.12	FlashEraseSector()	16
2.9.13	FlashEraseSuspend()	17
2.9.14	FlashEraseResume()	18
2.9.15	FlashProgramSection()	18
2.9.16	FlashProgram ()	19
2.9.17	FlashCheckSum()	20
2.9.18	FlashVerifyAllBlock()	20
2.9.19	FlashVerifyBlock()	21
2.9.20	FlashVerifySection()	22
2.9.21	FlashReadOnce()	23
2.9.22	FlashProgramOnce()	23
2.9.23	FlashProgramCheck()	24
2.9.24	FlashReadResource()	25
2.9.25	DEFlashPartition()	26
2.9.26	SetEEEEEnable()	26
2.9.27	EEEWrite()	27
2.9.28	PFlashSwapCtl ()	28
2.9.29	PFlashSwap()	29
2.9.30	RelocateFunction()	32
2.10	SSD Internal Functions	33
2.10.1	FlashCommandSequence()	33
3	TROUBLESHOOTING	34
4	IMPORTANT NOTE	35

APPENDIX A: PERFORMANCE DATA.....	36
Code Size.....	36
Timing	37
CallBack Time Periods.....	39
APPENDIX B: DERIVATIVE AND DEVICE MAPPING	40

LIST OF TABLES

Table 1: System Requirements	1
Table 2: References	2
Table 3: Terms.....	2
Table 4: Acronyms	2
Table 5: Type Definitions.....	3
Table 6: SSD Configuration Structure Field Definition	4
Table 7: User’s defined macros	4
Table 8: Return Codes	6
Table 9: Flash configuration file content.....	6
Table 10: Arguments for FlashInit().....	9
Table 11: Return Values for FlashInit().....	10
Table 12: Arguments for PFlashGetProtection()	10
Table 13: Return Values for PFlashGetProtection().....	10
Table 14: Arguments for PFlashSetProtection ().....	11
Table 15: Return Values for PFlashSetProtection ()	11
Table 16: Arguments for DFlashGetProtection()	11
Table 17: Return Values for DFlashGetProtection()	12
Table 18: Arguments for DFlashSetProtection()	12
Table 19: Return Values for DFlashSetProtection().....	12
Table 20: Arguments for EERAMGetProtection().....	13
Table 21: Return Values for EERAMGetProtection().....	13
Table 22: Arguments for EERAMSetProtection().....	14
Table 23: Return Values for EERAMSetProtection()	14
Table 24: Arguments for FlashGetSecurityState()	14
Table 25: Return Values for FlashGetSecurityState()	14
Table 26: Arguments for FlashSecurityBypass().....	15
Table 27: Return Values for FlashSecurityBypass()	15
Table 28: Arguments for FlashEraseAllBlock().....	15
Table 29: Return Values for FlashEraseAllBlock().....	16
Table 30: Arguments for FlashEraseBlock()	16
Table 31: Return Values for FlashEraseBlock().....	16
Table 32: Arguments for FlashEraseSector()	17
Table 33: Return Values for FlashEraseSector()	17
Table 34: Arguments for FlashEraseSuspend()	17
Table 35: Return Values for FlashEraseSuspend().....	17
Table 36: Arguments for FlashEraseResume().....	18
Table 37: Return Values for FlashEraseResume()	18
Table 38: Arguments for FlashProgramSection().....	18
Table 39: Return Values for FlashProgramSection()	19
Table 40: Arguments for FlashProgramLongword().....	19
Table 41: Return Values for FlashProgramLongword().....	19
Table 42: Arguments for FlashCheckSum().....	20
Table 43: Return Values for FlashCheckSum().....	20
Table 44: Arguments for FlashVerifyAllBlock()	21
Table 45: Return Values for FlashVerifyAllBlock()	21
Table 46: Arguments for FlashVerifyBlock()	21
Table 47: Return Values for FlashVerifyBlock()	22
Table 48: Arguments for FlashVerifySection()	22
Table 49: Return Values for FlashVerifySection().....	22
Table 50: Arguments for FlashReadOnce().....	23
Table 51: Return Values for FlashReadOnce().....	23
Table 52: Arguments for FlashProgramOnce()	24
Table 53: Return Values for FlashProgramOnce()	24
Table 54: Arguments for FlashProgramCheck()	24

Table 55: Return Values for FlashProgramCheck()	25
Table 56: Arguments for FlashReadResource()	25
Table 57: Return Values for FlashReadResource()	25
Table 58: Arguments for DEFlashPartition()	26
Table 59: Return Values for DEFlashPartition()	26
Table 60: Arguments for SetEEEEEnable()	27
Table 61: Return Values for SetEEEEEnable()	27
Table 62: Arguments for EEEWrite().....	27
Table 63: Return Values for EEEWrite()	28
Table 64: Arguments for PFlashSwapCtl ()	28
Table 65: Return Values for PFlashSwapCtl ()	29
Table 66: Arguments for PFlashSwap()	29
Table 67: Return Values for PFlashSwap()	30
Table 68: Arguments for RelocateFunction ()	32
Table 69: Return Values for RelocateFunction ().....	33
Table 70: Arguments for FlashCommandSequence()	33
Table 71: Return Values for FlashCommandSequence()	33
Table 72: Hardware error troubleshooting	34
Table 73: Software error troubleshooting.....	34
Table 74: Code size configuration.....	36
Table 75: Code size for different configurations	36
Table 76: Timing configuration.....	38
Table 77: Timing for different configurations	38
Table 78: Callback time period for different configurations	39
Table 79: Mapping between derivative and device	40

1 INTRODUCTION

1.1 Document Overview

This document is the user manual of the Standard Software Driver (SSD) for C90TFS/FTFx Flash family. The roadmap of the document is as follows:

[Section 1](#) provides a brief overview of the system for general background knowledge

[Section 2](#) describes the API specifications.

[Section 3](#) provides the troubleshooting for all error codes returned in each API.

[Section 4](#) provides some information that user needs to pay attention when using this driver. It is recommended that user should not skip this section to obtain some important notes regarding to this driver.

[Section 5](#) gives the performance data including code size and read/write time for different compilers and different configurations.

[Section 6](#) provides the mapping between derivative name and corresponding supported devices as well as tested device.

1.2 System Overview

A Standard Software Driver is a set of APIs that enables user's application to access flash memory blocks. The Standard Software Driver (SSD) for C90TFS/FTFx flash will provide driver functions to perform high-speed program/erase operations.

This driver contains the separate APIs for each flash command to do the associate task. In addition, it provides APIs to report flash memory configuration, report security or protection state and some other additional features. [Section 2.8](#) will provide more information of those APIs.

1.3 Features

The C90TFS/FTFx SSD provides the following features:

- Drivers released in source code format to provide compiler-independent supporting for non-debug-mode embedded applications.
- Each driver function is independent to each other. So, the end user can choose the function subset to meet their particular needs.
- Position-independent and ROM-able.
- Concurrency support via callback.

1.4 System Requirements

The Standard Software Driver is to support different derivatives of C90TFS/FTFx flash family on different compilers. Below table provides general information about system requirement of this driver. According to each specific derivative, some of items in below table will be used to run this driver.

Table 1: System Requirements

No.	Tool Name	Description	Version No
1	IAR Embedded Workbench for ARM	Development tool	6.4.2
2	CW for MCU	Development tool	10.4 & 10.5
3	JLink ARM	Debugger tool	n/a

3	PE micro Multilink universal	Debugger tool	n/a
3	Mini/micro USB cable	Debugger tool	n/a
4	Hardware (tower/base board,...) for specific device	Hardware	n/a

1.5 Documentation References

Table 2: References

No	Document Name	Version	Document Identifier
1	K40 Sub-Family Reference Manual	3	K40P144M100SF2RM
2	K60 Sub-Family Reference Manual	3	K60P144M100SF2RM
3	MCF51JF128 Reference Manual	1	MCF51JF128RM
4	MCF51FD256 Reference Manual	2	MCF51FD256RM
5	K70 Sub-Family Reference Manual	0	P3_K70P196M150SF3RM
6	K20 Sub-Family Reference Manual	1	K20P64M50SF0RM
7	MC56F847xx Reference Manual	1	MC56F847XXRM
8	KL25 Sub-Family Reference Manual	0	
9	K05L Sub-Family Reference Manual	0	

1.6 Terms

Table 3: Terms

Term	Definition
derivative	A term is used to determine specific memory configuration which is chip-dependent such as block size, sector size, MCU type, ...
P-Flash	Program Flash
D-Flash	Data Flash

1.7 Acronyms

Table 4: Acronyms

Abbreviation	Complete Name
API	Application Programming Interface
SSD	Standard Software Driver
RWW	Read While Write
MCU	Micro Controller Unit

2 API SPECIFICATION

The C90TFS/FTFx SSD provides common APIs to support all features of all C90TFS/FTFx derivatives. However, some APIs are not applicable on some specific derivatives since associate features are not available on them. For instance, the APIs relating to swap feature are not available on derivatives without supporting swap feature. User needs to refer to corresponding reference manual to get detailed information about supported features on the flash module.

2.1 General Overview

The C90TFS/FTFx SSD provides general APIs to handle specific operations on C90TFS/FTFx flash module. User can use those APIs directly in his application. In addition, it provides internal functions called by driver itself and they are not intended to call from user's application directly but user still can use those APIs for his purpose.

2.2 General Type Definitions

Table 5: Type Definitions

Derived type	Size	C language type Description
bool	8-bits	unsigned char
int8_t	8-bits	signed char
vint8_t	8-bits	volatile signed char
uint8_t	8-bits	unsigned char
vuint8_t	8-bits	volatile unsigned char
int16_t	16-bits	signed short
vint16_t	16-bits	volatile signed short
uint16_t	16-bits	unsigned short
vuint16_t	16-bits	volatile unsigned short
int32_t	32-bits	signed long
vint32_t	32-bits	volatile signed long
uint32_t	32-bits	unsigned long
vuint32_t	32-bits	volatile unsigned long

2.3 Configuration parameter

The configuration parameters, used for the SSD are given in this section. They are handled as structure as below :

```
typedef struct _ssd_config
{
    uint32_t          ffxRegBase;
    uint32_t          PFlashBlockBase;
    uint32_t          PFlashBlockSize;
    uint32_t          DFlashBlockBase;
    uint32_t          DFlashBlockSize;
    uint32_t          EERAMBlockBase;
    uint32_t          EEEBlockSize;
    bool              DebugEnable;
    PCALLBACK         CallBack;
} FLASH_SSD_CONFIG, *PFLASH_SSD_CONFIG;
```

The structure includes the static parameters for C90TFS/FTFx which are device-dependent. The user should correctly initialize the fields including *ftfxRegBase*, *PFlashBlockBase*, *PFlashBlockSize*, *DFlashBlockBase*, *EERAMBlockBase*, *DebugEnable* and *CallBack* before passing the structure to SSD functions. The rest of parameters such as *DFlashBlockSize*, and *EEEBlockSize* will be initialized in '*FlashInit()*' automatically. The pointer to *CallBack* has to be initialized either for null callback or a valid call back function.

Table 6: SSD Configuration Structure Field Definition

Parameter Name	Type	Description
ftfxRegBase	uint32_t	The register base address of C90TFS/FTFx module
PFlashBlockBase	uint32_t	The base address of P-Flash memory
PFlashBlockSize	uint32_t	The size in byte of P-Flash memory
DFlashBlockBase	uint32_t	For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory). For non-FlexNVM device, this field is unused.
DFlashBlockSize	uint32_t	For FlexNVM device, this is the size in byte of area which is used as D-Flash from FlexNVM memory. For non-FlexNVM device, this field is unused.
EERAMBlockBase	uint32_t	The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device).
EEEBlockSize	uint32_t	For FlexNVM device, this is the size in byte of EEPROM area which was partitioned from FlexRAM. For non-FlexNVM device, this field is unused.
DebugEnable	bool	Defines the state of background debug mode with below values: <ul style="list-style-type: none"> - TRUE: enable debug mode. - FALSE: disable debug mode. Refer to section 4 for more information.
CallBack	Function pointer	Call back function to service the time critical events.

2.4 Configuration Macros

Since this is a common driver to support all derivatives of C90TFS/FTFx flash memory, user needs to provide specific information in the following table to specify the environment in his application via user's defined macros in *user_cfg.h* file.

Table 7: User's defined macros

No.	Macro	Description
1	FLASH_DERIVATIVE	This macro is to select properly memory configuration in the list of derivative macros provided by the driver
2	BLOCK_COMMANDS	This macro instructs that the platform has erase block and read 1s block commands

Depending upon the macro selected, there will be a change in some internal macros and source codes. The derivative name will be defined based on the following rule:

FTFx_AA_BB_CC_DD_EE_FF

In which:

FTFx: abbreviation for C90TFS/FTFx flash family

AA: MCU type

BB: P-Flash memory size.

CC: FlexNVM memory size.

DD: FlexRam or Acceleration Ram size.

EE: P-Flash sector size.

FF: D-Flash sector size.

Refer to [APPENDIX B](#) for more information.

2.5 Flash Interrupt Macros

Besides the flash configuration macros, the driver also provides macros to disable/ enable interrupt as well as get flash interrupt enable status on flash memory controller:

- **SET_FLASH_INT_BITS(ftfxRegBase, value):** sets the Flash interrupt enable bits in the FCNFG register.
- **GET_FLASH_INT_BITS(ftfxRegBase):** read the FCNFG register and return the interrupt enable states for all flash interrupt types

In these macros:

- Parameter ftxRegBase: specifies register base address of flash module.
- Parameter value: The bit map value (0: disabled, 1 enabled). The numbering is marked from 0 to 7 where bit 0 is the least significant bit. Bit 7 is corresponding to command complete interrupt. Bit 6 is corresponding to read collision error interrupt.

2.6 Callback function

The Standard Software Driver facilitates the user to supply a pointer to Callback function so that time-critical events can be serviced during Standard Software driver operations. Servicing watchdog timers is one such time critical event. The application passes the pointer to the callback function in the structure discussed in [2.3 SSD Configuration parameters](#). The job processing callback notifications shall have no parameters and no return value. The function pointer is defined as following:

```
typedef void (* PCALLBACK) (void);
```

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 ((PCALLBACK) 0xFFFFFFFF)
```

There is special callback definition for PFlashSwap() function to provide user ability to interfere in a swap progress by letting the user code know about the swap state in each phase of the process and determine which actions will be taken in the next step. Unlike normal callback, the swap callback follows below type definition:

```
typedef bool (* PFLASH_SWAP_CALLBACK) (uint8_t function);
```

and NULL swap call back function:

```
#define NULL_SWAP_CALLBACK ((PFLASH_SWAP_CALLBACK) 0xFFFFFFFF)
```

2.7 Return Codes

The Return Code will be returned to the caller function to notify the success or errors of the API execution. The Return Code consists of following values:

Table 8: Return Codes

Name	Value	Description
FTFx_OK	0x0000	Function executes successfully
FTFx_ERR_MGSTAT0	0x0001	MGSTAT0 bit is set in the FSTAT register
FTFx_ERR_PVIOL	0x0010	Protection violation is set in FSTAT register
FTFx_ERR_ACCERR	0x0020	Access error is set in the FSTAT register
FTFx_ERR_CHANGEPROT	0x0100	Cannot change protection status
FTFx_ERR_NOEEE	0x0200	FlexRAM is not set for EEPROM use
FTFx_ERR_EFLASHONLY	0x0400	FlexNVM is set for full EEPROM backup
FTFx_ERR_RAMRDY	0x0800	Programming acceleration RAM is not available
FTFx_ERR_RANGE	0x1000	Address is out of the valid range
FTFx_ERR_SIZE	0x2000	Misaligned size

2.8 Port to New Derivative Guide Line

This section provides steps to porting the driver to a new derivative. To support new derivative, you need modify and add header files.

2.8.1 Add New Derivative Macro to SSD_FTFx_Common.h

The macro should be named using naming convention described in Section 2.4.

For example:

```
/*! @brief MCU type: Kinetis, P-Flash block size: 512K, FlexNVM block
size: 0K, FlexRAM/AccRam size: 0K, P-Flash sector size: 2K, FlexNVM
sector size: 0K */
#define FTFx_KX_512K_0K_0K_2K_0K 40 /* Senna K22FN512 */
```

2.8.2 Add Flash Configuration File for The New Derivative

The flash configuration file must contain macro definitions listed in the following table:

Table 9: Flash configuration file content

Name	Description	Note
FTFE_M	Specify that this is FTFE module	Only required for platforms utilizes FTFE flash module.
FTFA_M	Specify that this is FTFA module	Only required for platforms utilizes FTFA flash module.
SWAP_M	P-Flash swap feature is available	Only required for platforms having swap feature

BYTE2WORD(x)	Convert from word(2 bytes) address to byte address	Two address types are only different in DSC devices. In Kinestis and Coldfire devices, they are the same
WORD2BYTE(x)	Convert from word(2 bytes) address to byte address	
ENDIANNESS	Possible values: LITTLE_ENDIAN, BIG_ENDIAN	Required for all platforms
CPU_CORE	Possible value: COLDFIRE, ARM_CORTEX_M, DSC_56800EX	Required for all platforms
FTFx_PSECTOR_SIZE	P-Flash sector size	Required for all platforms
FTFx_DSECTOR_SIZE	D-Flash sector size	Required for all platforms
DEBLOCK_SIZE	D-Flash memory size	Required for all platforms
EEESIZE_0000	Emulated eeprom size code 0000 mapping to emulated eeprom size in bytes	Only required for platforms having FlexNVM memory
EEESIZE_0001	Emulated eeprom size code 0001 mapping to emulated eeprom size in bytes	
EEESIZE_0010	Emulated eeprom size code 0010 mapping to emulated eeprom size in bytes	
EEESIZE_0011	Emulated eeprom size code 0011 mapping to emulated eeprom size in bytes	
EEESIZE_0100	Emulated eeprom size code 0100 mapping to emulated eeprom size in bytes	
EEESIZE_0101	Emulated eeprom size code 0101 mapping to emulated eeprom size in bytes	
EEESIZE_0110	Emulated eeprom size code 0110 mapping to emulated eeprom size in bytes	
EEESIZE_0111	Emulated eeprom size code 0111 mapping to emulated eeprom size in bytes	
EEESIZE_1000	Emulated eeprom size code 1000 mapping to emulated eeprom size in bytes	
EEESIZE_1001	Emulated eeprom size code 1001 mapping to emulated eeprom size in bytes	
EEESIZE_1010	Emulated eeprom size code 1010 mapping to emulated eeprom size in bytes	
EEESIZE_1011	Emulated eeprom size code 1011 mapping to emulated eeprom size in bytes	
EEESIZE_1100	Emulated eeprom size code 1100 mapping to emulated eeprom size in bytes	
EEESIZE_1101	Emulated eeprom size code 1101 mapping to emulated eeprom size in bytes	
EEESIZE_1110	Emulated eeprom size code 1110 mapping to emulated eeprom size in bytes	
EEESIZE_1111	Emulated eeprom size code 1111 mapping to emulated eeprom size in bytes	
DEPART_0000	FlexNVM partition code 0000 mapping to data flash size in bytes	
DEPART_0001	FlexNVM partition code 0001 mapping to data flash size in bytes	

DEPART_0010	FlexNVM partition code 0010 mapping to data flash size in bytes	
DEPART_0011	FlexNVM partition code 0011 mapping to data flash size in bytes	
DEPART_0100	FlexNVM partition code 0100 mapping to data flash size in bytes	
DEPART_0101	FlexNVM partition code 0101 mapping to data flash size in bytes	
DEPART_0110	FlexNVM partition code 0110 mapping to data flash size in bytes	
DEPART_0111	FlexNVM partition code 0111 mapping to data flash size in bytes	
DEPART_1000	FlexNVM partition code 1000 mapping to data flash size in bytes	
DEPART_1001	FlexNVM partition code 1001 mapping to data flash size in bytes	
DEPART_1010	FlexNVM partition code 1010 mapping to data flash size in bytes	
DEPART_1011	FlexNVM partition code 1011 mapping to data flash size in bytes	
DEPART_1100	FlexNVM partition code 1100 mapping to data flash size in bytes	
DEPART_1101	FlexNVM partition code 1101 mapping to data flash size in bytes	
DEPART_1110	FlexNVM partition code 1110 mapping to data flash size in bytes	
DEPART_1111	FlexNVM partition code 1111 mapping to data flash size in bytes	
DFLASH_IFR_READRESO URCE_ADDRESS	Data flash IFR map address	Only required for platforms having FlexNVM memory
PFLASH_IFR_OFFSET	Program flash IFR map offset	Not required for the driver but we can put here for user application reference
PFLASH_IFR_SIZE	Program flash IFR map size	
DFLASH_IFR_OFFSET	Data flash IFR map offset	
DFLASH_IFR_SIZE	Data flash IFR map size	
PERSBLK_ALIGN_SIZE	P-Flash Erase block command address alignment	
DERSBLK_ALIGN_SIZE	FlexNVM Erase block command address alignment	
PERSSEC_ALIGN_SIZE	P-Flash Erase sector command address alignment	
DERSSEC_ALIGN_SIZE	FlexNVM Erase sector command address alignment	
PPGMSEC_ALIGN_SIZE	P-Flash Program section command address alignment	
DPGMSEC_ALIGN_SIZE	FlexNVM Program section command address alignment	
PRD1BLK_ALIGN_SIZE	P-Flash Read 1st block command address alignment	

DRD1BLK_ALIGN_SIZE	FlexNVM Read 1st block command address alignment	
PRD1SEC_ALIGN_SIZE	P-Flash read 1s section command address alignment	
DRD1SEC_ALIGN_SIZE	FlexNVM read 1s section command address alignment	
PGMCHK_ALIGN_SIZE	Program check command address alignment	Required for all platforms
RDRSRC_ALIGN_SIZE	Read resource command address alignment	Not required for the driver but we can put here for user application reference
SWAP_ALIGN_SIZE	P-Flash swap command address alignment	
PGM_SIZE_BYTE	Program unit size	Required for all platforms
RESUME_WAIT_CNT	Resume wait count used in FlashResume function	Required for all platforms

2.8.3 Include the Flash Configuration File on SSD_FTFx_Internal.h

Extend the include section for the configuration file that you have just created as the below example:

```
#elif ((FTFx_KX_512K_0K_0K_2K_0K == FLASH_DERIVATIVE) ||
(FTFx_KX_256K_0K_0K_2K_0K == FLASH_DERIVATIVE) \ ||
(FTFx_KX_128K_0K_0K_2K_0K == FLASH_DERIVATIVE) ||
(FTFx_KX_64K_0K_0K_2K_0K == FLASH_DERIVATIVE))
#include "FTFx_KX_(512_256_128_64)K_0K_0K_2K_0K.h"
```

2.9 SSD General Functions

This section provides information about general APIs which can be called directly in user's application.

2.9.1 FlashInit()

2.9.1.1 Overview

This API will initialize flash module by clearing status error bit and reporting the memory configuration via SSD configuration structure.

2.9.1.2 Prototype

uint32_t FlashInit (PFLASH_SSD_CONFIG pSSDConfig)

2.9.1.3 Arguments

Table 10: Arguments for FlashInit()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

2.9.1.4 Return Values

Table 11: Return Values for FlashInit()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK

2.9.1.5 Comments

The flash module cannot be initialized correctly in low power mode.

2.9.2 PFlashGetProtection()

2.9.2.1 Overview

This API retrieves current P-Flash protection status. Considering the time consumption for getting protection is very low and even can be ignored, it is not necessary to utilize the Callback function to support the time-critical events.

2.9.2.2 Prototype

uint32_t PFlashGetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint32_t* protectStatus)

2.9.2.3 Arguments

Table 12: Arguments for PFlashGetProtection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .
protectStatus	uint32_t*	To return the current value of the P-Flash Protection. Each bit is corresponding to protection status of 1/32 of the total P-Flash. The least significant bit is corresponding to the lowest address area of P-Flash. The most significant bit is corresponding to the highest address area of P-Flash and so on. There are two possible cases as below: <ul style="list-style-type: none">- 0: this area is protected.- 1: this area is unprotected.

2.9.2.4 Return Values

Table 13: Return Values for PFlashGetProtection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK

2.9.2.5 Comments

None.

2.9.3 PFlashSetProtection()

2.9.3.1 Overview

This API sets the P-Flash protection to the intended protection status. Setting P-Flash protection status is subject to a protection transition restriction. If there is any setting violation, it will return an error code and the current protection status will not be changed.

2.9.3.2 Prototype

uint32_t PFlashSetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint32_t protectStatus)

2.9.3.3 Arguments

Table 14: Arguments for PFlashSetProtection ()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
protectStatus	uint32_t	The expected protect status user wants to set to P-Flash protection register. Refer to section 2.8.2.3 for more details.

2.9.3.4 Return Values

Table 15: Return Values for PFlashSetProtection ()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_CHANGEPROT

2.9.3.5 Comments

User must never invoke this API while command is running (CCIF = 0).

2.9.4 DFlashGetProtection()

2.9.4.1 Overview

This API retrieves current D-Flash protection status. Considering the time consumption for getting protection is very low and even can be ignored, there is no need to utilize the Callback function to support the time-critical events.

2.9.4.2 Prototype

uint32_t DFlashGetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t* protectStatus)

2.9.4.3 Arguments

Table 16: Arguments for DFlashGetProtection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

protectStatus	uint8_t*	To return the current value of the D-Flash Protection Register. Each bit is corresponding to protection status of 1/8 of the total D-Flash. The least significant bit is corresponding to the lowest address area of D-Flash. The most significant bit is corresponding to the highest address area of D-Flash and so on. There are two possible cases as below: <ul style="list-style-type: none"> - 0 : this area is protected. - 1 : this area is unprotected.
---------------	----------	---

2.9.4.4 Return Values

Table 17: Return Values for DFlashGetProtection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_EFLASHONLY

2.9.4.5 Comments

None

2.9.5 DFlashSetProtection()

2.9.5.1 Overview

This API sets the D-Flash protection to the intended protection status. Setting D-Flash protection status is subject to a protection transition restriction. If there is any setting violation, it will return failed information and the current protection status will not be changed.

2.9.5.2 Prototype

`uint32_t DFlashSetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)`

2.9.5.3 Arguments

Table 18: Arguments for DFlashSetProtection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
protectStatus	uint8_t	The expected protect status user wants to set to D-Flash protection register. Refer to section 2.8.4.3 for more details.

2.9.5.4 Return Values

Table 19: Return Values for DFlashSetProtection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_EFLASHONLY FTFx_ERR_CHANGEPROT

2.9.5.5 Comments

User must never invoke this API while command is running (CCIF = 0).

2.9.6 EERAMGetProtection()

2.9.6.1 Overview

This API retrieves which EEPROM sections of FlexRAM are protected against program and erase operations. Considering the time consumption for getting protection is very low and even can be ignored, it is not necessary to utilize the Callback function to support the time-critical events.

2.9.6.2 Prototype

`uint32_t EERAMGetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t* protectStatus)`

2.9.6.3 Arguments

Table 20: Arguments for EERAMGetProtection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .
protectStatus	uint8_t*	To return the current value of the EEPROM Protection Register. Each bit is corresponding to protection status of 1/8 of the total EEPROM use. The least significant bit is corresponding to the lowest address area of EEPROM. The most significant bit is corresponding to the highest address area of EEPROM and so on. There are two possible cases as below: <ul style="list-style-type: none">- 0: this area is protected.- 1: this area is unprotected.

2.9.6.4 Return Values

Table 21: Return Values for EERAMGetProtection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_NOEEE

2.9.6.5 Comments

None

2.9.7 EERAMSetProtection()

2.9.7.1 Overview

This API sets protection to the intended protection status for EEPROM use area of FlexRam. This is subject to a protection transition restriction. If there is any setting violation, it will return failed information and the current protection status will not be changed.

2.9.7.2 Prototype

`uint32_t EERAMSetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)`

2.9.7.3 Arguments

Table 22: Arguments for EERAMSetProtection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .
protectStatus	uint8_t	The intended protection status value should be written to the EEPROM Protection Register. Refer to section 2.8.6.3 for more details.

2.9.7.4 Return Values

Table 23: Return Values for EERAMSetProtection()

Type	Description	Possible Values
uint32_t	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_NOEEE FTFx_ERR_CHANGEPROT

2.9.7.5 Comments

User must never invoke this API while command is running (CCIF = 0).

2.9.8 FlashGetSecurityState()

2.9.8.1 Overview

This API retrieves the current Flash security status, including the security enabling state and the backdoor key enabling state.

2.9.8.2 Prototype

`uint32_t FlashGetSecurityState (PFLASH_SSD_CONFIG pSSDConfig, uint8_t* securityState)`

2.9.8.3 Arguments

Table 24: Arguments for FlashGetSecurityState()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .
securityState	uint8_t*	To return the current security status code. <ul style="list-style-type: none"> FLASH_NOT_SECURE - 0x01, Flash currently not in secure state; FLASH_SECURE_BACKDOOR_ENABLED - 0x02, Flash is secured and backdoor key access enabled; FLASH_SECURE_BACKDOOR_DISABLED - 0x04, Flash is secured and backdoor key access disabled.

2.9.8.4 Return Values

Table 25: Return Values for FlashGetSecurityState()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK

2.9.8.5 Comments

None

2.9.9 FlashSecurityBypass()

2.9.9.1 Overview

This API will unsecure the device by comparing the user's provided backdoor key with the ones in the Flash Configuration Field. If they are matched each other, then security will be released. Otherwise, an error code will be returned

2.9.9.2 Prototype

```
uint32_t FlashSecurityBypass (PFLASH_SSD_CONFIG pSSDConfig, \
                              uint8_t* keyBuffer, \
                              pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.9.3 Arguments

Table 26: Arguments for FlashSecurityBypass()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
keyBuffer	uint8_t*	Point to the user buffer containing the backdoor key
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.9.4 Return Values

Table 27: Return Values for FlashSecurityBypass()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.9.9.5 Comments

None

2.9.10 FlashEraseAllBlock()

2.9.10.1 Overview

This API will erase all Flash memory, initialize the FlexRAM, verify all memory contents, and then release MCU security.

2.9.10.2 Prototype

```
uint32_t FlashEraseAllBlock (PFLASH_SSD_CONFIG pSSDConfig, \
                              pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.10.3 Arguments

Table 28: Arguments for FlashEraseAllBlock()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function
-----------------------	-----------------------	--

2.9.10.4 Return Values

Table 29: Return Values for FlashEraseAllBlock()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0 FTFx_ERR_ACCERR

2.9.10.5 Comments

None.

2.9.11 FlashEraseBlock()

2.9.11.1 Overview

This API will erase all addresses in an individual P-Flash or D-Flash block.

2.9.11.2 Prototype

```
uint32_t FlashEraseBlock (PFLASH_SSD_CONFIG pSSDConfig, \
                          uint32_t dest, \
                          pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.11.3 Arguments

Table 30: Arguments for FlashEraseBlock()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended erase operation.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.11.4 Return Values

Table 31: Return Values for FlashEraseBlock()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.9.11.5 Comments

For the derivatives including multiple logical P-Flash or D-Flash blocks, this API just erases a single block in a single call.

2.9.12 FlashEraseSector()

2.9.12.1 Overview

This API will erase one or more sectors in P-Flash or D-Flash memory.

2.9.12.2 Prototype

```
uint32_t FlashEraseSector(PFLASH_SSD_CONFIG pSSDConfig, \
                          uint32_t dest, \
                          uint32_t size, \
                          pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.12.3 Arguments

Table 32: Arguments for FlashEraseSector()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Address in the first sector to be erased
size	uint32_t	Size to be erased in bytes. It is used to determine number of sectors to be erased.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.12.4 Return Values

Table 33: Return Values for FlashEraseSector()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_SIZE FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.9.12.5 Comments

This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

2.9.13 FlashEraseSuspend()

2.9.13.1 Overview

This API is used to suspend a current operation of flash erase sector command.

2.9.13.2 Prototype

```
uint32_t FlashEraseSuspend (PFLASH_SSD_CONFIG pSSDConfig)
```

2.9.13.3 Arguments

Table 34: Arguments for FlashEraseSuspend()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

2.9.13.4 Return Values

Table 35: Return Values for FlashEraseSuspend()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK

2.9.13.5 Comments

None.

2.9.14 FlashEraseResume()

2.9.14.1 Overview

This API is used to resume a previous suspended operation of flash erase sector command.

2.9.14.2 Prototype

```
uint32_t FlashEraseResume (PFLASH_SSD_CONFIG pSSDConfig)
```

2.9.14.3 Arguments

Table 36: Arguments for FlashEraseResume()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

2.9.14.4 Return Values

Table 37: Return Values for FlashEraseResume()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK

2.9.14.5 Comments

None.

2.9.15 FlashProgramSection()

2.9.15.1 Overview

This API will program the data found in the Section Program Buffer to previously erased locations in the Flash memory. Data is preloaded into the Section Program Buffer by writing to the acceleration Ram and FlexRam while it is set to function as a RAM. The Section Program Buffer is limited to the value of FlexRam divides by a ratio. Refer to the associate reference manual to get correct value of this ratio.

2.9.15.2 Prototype

```
uint32_t FlashProgramSection (PFLASH_SSD_CONFIG pSSDConfig, \
                               uint32_t dest, \
                               uint16_t number, \
                               pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.15.3 Arguments

Table 38: Arguments for FlashProgramSection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended program operation.
number	uint16_t	Number of alignment unit to be programmed. Refer to associate reference manual to get correct value of this alignment constrain.

pFlashComm- andSequence	pFLASHCOMMANDS- EQUENCE	Pointer to the flash command sequence function
----------------------------	----------------------------	--

2.9.15.4 Return Values

Table 39: Return Values for FlashProgramSection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0 FTFx_ERR_RAMRDY

2.9.15.5 Comments

For derivatives including swap feature, the swap indicator address is encountered during 'FlashProgramSection', it is bypassed without setting FPVIOL but the content are not be programmed. In addition, the content of source data used to program to swap indicator will be re-initialized to 0xFF after completion of this command.

2.9.16 FlashProgram ()

2.9.16.1 Overview

This API is used to program four consecutive bytes (for program long word command) and eight consecutive bytes (for program phrase command) on P-flash or D-Flash memory.

2.9.16.2 Prototype

```
uint32_t FlashProgram(PFLASH_SSD_CONFIG pSSDConfig, \
                    uint32_t dest, \
                    uint32_t size, \
                    uint8_t* pData, \
                    pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.16.3 Arguments

Table 40: Arguments for FlashProgramLongword()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended program operation.
size	uint32_t	Size in byte to be programmed
pData	uint8_t*	Pointer of source address from which data has to be taken for program operation.
pFlashComm- andSequence	pFLASHCOMMANDS- EQUENCE	Pointer to the flash command sequence function

2.9.16.4 Return Values

Table 41: Return Values for FlashProgramLongword()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

		FTFx_ERR_PVIOL FTFx_ERR_SIZE FTFx_ERR_MGSTAT0
--	--	---

2.9.16.5 Comments

This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

2.9.17 FlashChecksum()

2.9.17.1 Overview

This API will perform 32 bit sum of each byte data over specified flash memory range without carry, which provides rapid method for checking data integrity.

2.9.17.2 Prototype

```
uint32_t FlashChecksum (PFLASH_SSD_CONFIG pSSDConfig, \
                        uint32_t dest, \
                        uint32_t size, \
                        uint32_t* pSum)
```

2.9.17.3 Arguments

Table 42: Arguments for FlashChecksum()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address of the Flash range to be summed
size	uint32_t	Size in byte of the flash range to be summed
pSum	uint32_t *	To return the sum value

2.9.17.4 Return Values

Table 43: Return Values for FlashChecksum()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_RANGE

2.9.17.5 Comments

The callback period of this API is determined via FLASH_CALLBACK_CS macro in SSD_FTFx_Common.h which is used as a counter value for the 'Callback()' function calling in this API. This value can be changed as per the user requirement. User can change this value to obtain the maximum permissible callback period.

This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

2.9.18 FlashVerifyAllBlock()

2.9.18.1 Overview

This function will check to see if the P-Flash and/or D-Flash, EEPROM backup area, and D-Flash IFR have been erased to the specified read margin level, if applicable, and will release security if the readout passes.

2.9.18.2 Prototype

```
uint32_t FlashVerifyAllBlock (PFLASH_SSD_CONFIG pSSDConfig, \
                             uint8_t marginLevel, \
                             pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.18.3 Arguments

Table 44: Arguments for FlashVerifyAllBlock()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
marginLevel	uint8_t	Read Margin Choice as follows: <ul style="list-style-type: none">marginLevel = 0x0: use 'Normal' read levelmarginLevel = 0x1: use the 'User' readmarginLevel = 0x2: use the 'Factory' read
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.18.4 Return Values

Table 45: Return Values for FlashVerifyAllBlock()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.18.5 Comments

None

2.9.19 FlashVerifyBlock()

2.9.19.1 Overview

This API will check to see if an entire P-Flash or D-Flash memory has been erased to the specified margin level.

2.9.19.2 Prototype

```
uint32_t FlashVerifyBlock(PFLASH_SSD_CONFIG pSSDConfig, \
                          uint32_t dest, \
                          uint8_t marginLevel, \
                          pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.19.3 Arguments

Table 46: Arguments for FlashVerifyBlock()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended verify operation.
marginLevel	uint8_t	Read Margin Choice as follows: <ul style="list-style-type: none">marginLevel = 0x0: use 'Normal' read levelmarginLevel = 0x1: use the 'User' readmarginLevel = 0x2: use the 'Factory' read

pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function
-----------------------	-----------------------	--

2.9.19.4 Return Values

Table 47: Return Values for FlashVerifyBlock()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.19.5 Comments

For the derivative including multiple logical P-Flash or D-Flash blocks, this API just verifies a single block in a single call.

2.9.20 FlashVerifySection()

2.9.20.1 Overview

This API will check to see if a section of P-Flash or D-Flash memory is erased to the specified read margin level.

2.9.20.2 Prototype

```
uint32_t FlashVerifySection (PFLASH_SSD_CONFIG pSSDConfig, \
                             uint32_t dest, \
                             uint16_t number, \
                             uint8_t marginLevel, \
                             pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.20.3 Arguments

Table 48: Arguments for FlashVerifySection()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended verify operation.
number	uint16_t	Number of alignment unit to be verified. Refer to corresponding reference manual to get correct information of alignment constrain.
marginLevel	uint8_t	Read Margin Choice as follows: <ul style="list-style-type: none"> marginLevel = 0x0: use 'Normal' read level marginLevel = 0x1: use the 'User' read marginLevel = 0x2: use the 'Factory' read
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.20.4 Return Values

Table 49: Return Values for FlashVerifySection()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.20.5 Comments

None.

2.9.21 FlashReadOnce()

2.9.21.1 Overview

This API is used to read out a reserved 64 byte field located in the P-Flash IFR via given number of record. Refer to corresponding reference manual to get correct value of this number.

2.9.21.2 Prototype

```
uint32_t FlashReadOnce (PFLASH_SSD_CONFIG pSSDConfig, \
                        uint8_t recordIndex, \
                        uint8_t* pDataArray, \
                        pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.21.3 Arguments

Table 50: Arguments for FlashReadOnce()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
recordIndex	uint8_t	The record index will be read. It can be from 0x0 to 0x7 or from 0x0 to 0xF according to specific derivative.
pdataArray	uint8_t*	Pointer to the array to return the data read by the read once command
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.21.4 Return Values

Table 51: Return Values for FlashReadOnce()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.9.21.5 Comments

None

2.9.22 FlashProgramOnce()

2.9.22.1 Overview

This API is used to program to a reserved 64 byte field located in the P-Flash IFR via given number of record. Refer to corresponding reference manual to get correct value of this number.

2.9.22.2 Prototype

```
uint32_t FlashProgramOnce (PFLASH_SSD_CONFIG pSSDConfig, \
                           uint8_t recordIndex, \
                           uint8_t* pDataArray, \
                           pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.22.3 Arguments

Table 52: Arguments for FlashProgramOnce()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
recordIndex	uint8_t	Refer to section 2.8.21.3
pdataArray	uint8_t*	Pointer to the array from which data will be taken for program once command.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.22.4 Return Values

Table 53: Return Values for FlashProgramOnce()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.22.5 Comments

None.

2.9.23 FlashProgramCheck()

2.9.23.1 Overview

This API tests a previously programmed P-Flash or D-Flash long word to see if it reads correctly at the specified margin level.

2.9.23.2 Prototype

```
uint32_t FlashProgramCheck (PFLASH_SSD_CONFIG pSSDConfig, \
                             uint32_t dest, \
                             uint32_t size, \
                             uint8_t* pExpectedData, \
                             uint32_t* pFailAddr, \
                             uint8_t marginLevel, \
                             pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.23.3 Arguments

Table 54: Arguments for FlashProgramCheck()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended program check operation.
size	uint32_t	Size in byte to check accuracy of program operation
pExpectedData	uint8_t*	The pointer to the expected data
pFailAddr	uint32_t*	Returned the first aligned failing address

marginLevel	uint8_t	Read margin choice as follows: <ul style="list-style-type: none"> marginLevel = 0x1: read at 'User' margin 1/0 level. marginLevel = 0x2: read at 'Factory' margin 1/0 level.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.23.4 Return Values

Table 55: Return Values for FlashProgramCheck()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.23.5 Comments

This API always returns FTFx_OK if size provided by user is zero regardless of the input validation.

2.9.24 FlashReadResource()

2.9.24.1 Overview

This API is used to read data from special purpose memory in flash memory module including P-Flash IFR, swap IFR, D-Flash IFR space and version ID.

2.9.24.2 Prototype

```
uint32_t FlashReadResource (PFLASH_SSD_CONFIG pSSDConfig, \
    uint32_t dest, \
    uint8_t* pDataArray, \
    uint8_t resourceSelectCode, \
    pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.24.3 Arguments

Table 56: Arguments for FlashReadResource()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended read operation.
pDataArray	uint8_t*	Pointer to the data returned by the read resource command.
resourceSelectCode	uint8_t	Read resource select code: <ul style="list-style-type: none"> 0: Flash IFR 1: Version ID
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.24.4 Return Values

Table 57: Return Values for FlashReadResource()

Type	Description	Possible Values
------	-------------	-----------------

uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.9.24.5 Comments

None

2.9.25 DEFlashPartition()

2.9.25.1 Overview

This API prepares the FlexNVM memory for use as D-Flash, EEPROM backup or a combination of both and initializes the FlexRAM.

2.9.25.2 Prototype

```
uint32_t DEFlashPartition (PFLASH_SSD_CONFIG pSSDConfig, \
                          uint8_t EEEDataSizeCode, \
                          uint8_t DEPartitionCode, \
                          pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.25.3 Arguments

Table 58: Arguments for DEFlashPartition()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
EEEDataSizeCode	uint8_t	EEPROM Data Size Code. Refer to associate reference manual for more details.
DEPartitionCode	uint8_t	FlexNVM Partition Code. Refer to associate reference manual for more details.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function

2.9.25.4 Return Values

Table 59: Return Values for DEFlashPartition()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.25.5 Comments

The single partition choice should be used through entire life time of a given application to guarantee the flash endurance and data retention of flash module.

2.9.26 SetEEEnable()

2.9.26.1 Overview

This function is used to change the function of the FlexRAM. When not partitioned for EEPROM backup, the FlexRam is typically used as traditional RAM. Otherwise, the FlexRam is typically used to store EEPROM data and user can use this API to change its functionality according to his application requirement. For example, after partitioning to have EEPROM backup, FlexRAM is used for EEPROM use accordingly. And this API will be used to set FlexRAM is available for traditional RAM for FlashProgramSection() use.

2.9.26.2 Prototype

uint32_t SetEEERenable (PFLASH_SSD_CONFIG pSSDConfig, uint8_t EEERenable)

2.9.26.3 Arguments

Table 60: Arguments for SetEEERenable()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
EEERenable	uint8_t	FlexRam function control code. It can be: <ul style="list-style-type: none">- 0xFF: make FlexRam available for RAM.- 0x00: make FlexRam available for EEPROM.

2.9.26.4 Return Values

Table 61: Return Values for SetEEERenable()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.9.26.5 Comments

None.

2.9.27 EEERWrite()

2.9.27.1 Overview

This API is used to write data to FlexRAM section which is partitioned as EEPROM use for EEPROM operation. Once data has been written to EEPROM use section of FlexRAM, the EEPROM file system will create new data record in EEPROM back-up area of FlexNVM in round-robin fashion.

2.9.27.2 Prototype

**uint32_t EEERWrite (PFLASH_SSD_CONFIG pSSDConfig, **
**uint32_t dest, **
**uint32_t size, **
uint8_t* pData)

2.9.27.3 Arguments

Table 62: Arguments for EEERWrite()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3
dest	uint32_t	Start address for the intended write operation.
size	uint32_t	Size in byte to be written
pData	uint8_t*	Pointer to source address from which data has to be taken for writing operation.

2.9.27.4 Return Values

Table 63: Return Values for EEEWrite()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_RANGE FTFx_ERR_NOEEE FTFx_ERR_PVIOL

2.9.27.5 Comments

There is no alignment constraint for destination and size parameters provided by user. However, according to user's input provided, this API will set priority to write to FlexRAM with following rules:

- 32-bit writing will be invoked if destination is 32 bit aligned and size is not less than 32 bits.
- 16-bit writing will be invoked if destination is 16 bit aligned and size is not less than 16 bits.
- 8-bit writing will be invoked if destination is 8 bit aligned and size is not less than 8 bits.

2.9.28 PFlashSwapCtl ()

2.9.28.1 Overview

This API implements swap control command corresponding with swap control code provided via swapcmd parameter.

2.9.28.2 Prototype

```
uint32_t PFlashSwapCtl(PFLASH_SSD_CONFIG pSSDConfig,  
                      uint32_t addr,  
                      uint8_t swapcmd,  
                      uint8_t* pCurrentSwapMode,  
                      uint8_t* pCurrentSwapBlockStatus,  
                      uint8_t* pNextSwapBlockStatus,  
                      PFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.28.3 Arguments

Table 64: Arguments for PFlashSwapCtl ()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .
addr	uint32_t	Address of swap indicator.
swapcmd	uint8_t	Swap Control Code: 0x01 - Initialize Swap System 0x02 - Set Swap in Update State 0x04 - Set Swap in Complete Stat 0x08 - Report Swap Status

pCurrentSwapMode	uint8_t*	Current Swap Mode: 0x00 - Uninitialized 0x01 - Ready 0x02 - Update 0x03 - Update-Erased 0x04 - Complete
pCurrentSwapBlockStatus	uint8_t*	Current Swap Block Status indicates which program flash block is currently located at relative flash address 0x0_0000 0x00 - Program flash block 0 0x01 - Program flash block 1
pCurrentSwapBlockStatus	uint8_t*	Next Swap Block Status indicates which program flash block will be located at relative flash address 0x0_0000 after the next reset. 0x00 - Program flash block 0 0x01 - Program flash block 1
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.9.28.4 Return Values

Table 65: Return Values for PFlashSwapCtl ()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.28.5 Comments

The swap indicator supplied to this API can be any address which belongs to lower half block and not in flash configuration field area. There is no constrain to match to the one provided in the previous swap command calling.

2.9.29 PFlashSwap()

2.9.29.1 Overview

This API is used to swap between the two half of total logical P-Flash memory blocks within the memory map.

2.9.29.2 Prototype

```
uint32_t PFlashSwap(PFLASH_SSD_CONFIG pSSDConfig, \
                   uint32_t addr, \
                   PFLASH_SWAP_CALLBACK pSwapCallback, \
                   pFLASHCOMMANDSEQUENCE pFlashCommandSequence)
```

2.9.29.3 Arguments

Table 66: Arguments for PFlashSwap()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3 .

addr	uint32_t	Address for storing swap indicator.
pSwapCallBack	PFLASH_SWAP_CALLBACK	Callback to do specific task while the swapping is being performed. Refer to section 2.8.29.5 for more description.
pFlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.9.29.4 Return Values

Table 67: Return Values for PFlashSwap()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.9.29.5 Comments

The swap API provides to user with an ability to interfere in a swap progress by letting the user code knows about the swap state in each phase of the process. This is done via '*pSwapCallBack()*' parameter. If user wants to stop at each intermediate swap state, just needs to set return value of this callback function to FALSE. If user wants to complete swap process within a single call, just needs to set return value of this function to TRUE.

It is very important that user needs to erase the non-active swap indicator in somewhere of his application code or in within this swap call back function when swap system is in UPDATE state.

In addition, if user does not want to use the swap call back parameter, just pass NULL_SWAP_CALLBACK as a null pointer. In a such situation, the '*PFlashSwap()*' will operate as in case setting return value of '*pSwapCallBack*' to TRUE and user doesn't need to care about erasing the non-active swap indicator when swap system is in UPDATE state.

The flow of this function describes in following figure :

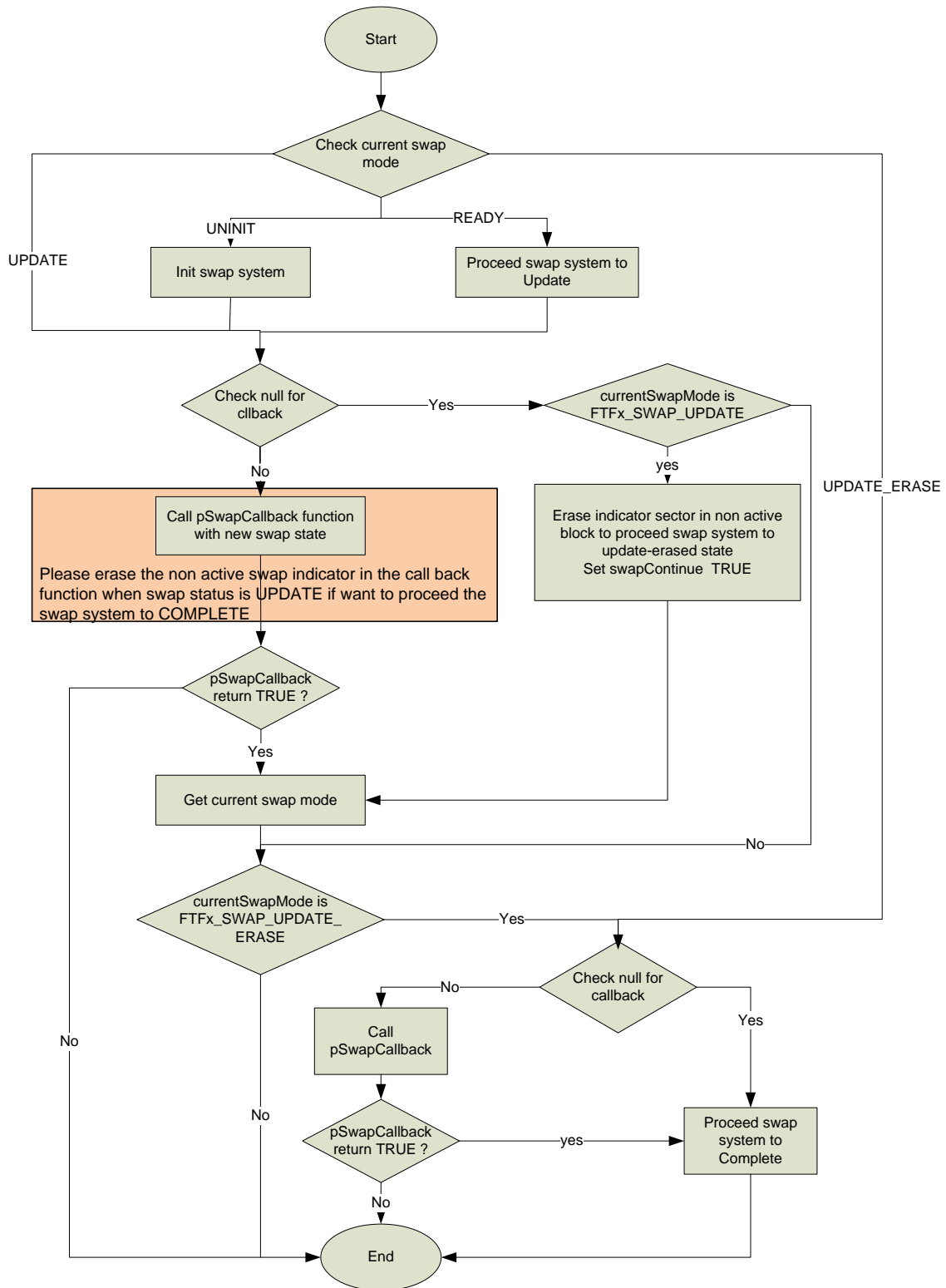


Figure 1 : PFlashSwap flow chart

Below is an example to show how to implement a swap callback:

```
bool PFlashSwapCallback(uint8_t currentSwapMode)
```

```

{
    switch (currentSwapMode)
    {
        case FTFx_SWAP_UNINI:
            /* Put your application-specific code here */
            break;

        case FTFx_SWAP_READY: /* Ready */
            /* Put your application-specific code here */
            break;

        case FTFx_SWAP_UPDATE:
            /* Put your application-specific code here */
            /* for example, erase non-active swap indicator here */
            break;

        case FTFx_SWAP_UPDATE_ERASED:
            /* Put your application-specific code here */
            /* for example, erase non-active swap indicator here */
            break;

        case FTFx_SWAP_COMPLETE:
            /* Put your application-specific code here */
            break;

        default:
            break;
    }

    return TRUE; /* Return FALSE to stop at intermediate swap state */
}

```

For details, refer to swap demo included in this release package.

The swap indicator provided by user must be within the lower half of P-Flash block but not in flash configuration area. If P-Flash block has two logical blocks, then swap indicator must be in P-Flash block 0. If P-Flash block has four logical blocks, then swap indicator can be in block 0 or block 1. Of course, it must not be in flash configuration field.

User must use the same swap indicator for all swap control code except report swap status once swap system has been initialized. To refresh swap system to un-initialization state, just needs to use FlashEraseAllBlock() to clean up swap environment.

2.9.30 RelocateFunction()

2.9.30.1 Overview

This function provides user a facility to relocate a function from one location to another location in RAM.

2.9.30.2 Prototype

```
uint32_t RelocateFunction(uint32_t dest, uint32_t size, uint32_t src)
```

2.9.30.3 Arguments

Table 68: Arguments for RelocateFunction ()

Argument	Type	Description
dest	uint32_t	Destination address where you want to place the function

size	uint32_t	Size of the function
src	uint32_t	Address of the function will be relocated

2.9.30.4 Return Values

Table 69: Return Values for RelocateFunction ()

Type	Description
uint32_t	Relocated address of the function

2.9.30.5 Comments

For Nevis2 and Anguilla Silver derivatives, user needs to provide two following macros to make the function compliable

- PROGRAM_RAM_SPACE_BASE: It specifies start address of ram region in program space
- DATA_SPACE_BASE: It specifies base address of memory region which stores the driver code in data space. For example, if dest is in program flash, value of the macro shall be the base address of program flash in data memory space.

2.10 SSD Internal Functions

This section provides information about internal APIs which are designed to be invoked in SSD general APIs themselves. Use can also call those APIs in his application for his purpose.

2.10.1 FlashCommandSequence()

2.10.1.1 Overview

This API is used to perform command write sequence on the flash.

2.10.1.2 Prototype

uint32_t FlashCommandSequence (PFLASH_SSD_CONFIG pSSDConfig)

2.10.1.3 Arguments

Table 70: Arguments for FlashCommandSequence()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3

2.10.1.4 Return Values

Table 71: Return Values for FlashCommandSequence()

Type	Description	Possible Values
uint32_t	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.10.1.5 Comments

In order to avoid RWW error, this API must not be placed on the same flash block on which program or erase operation is going on.

3 TROUBLESHOOTING

The troubleshooting comprises of hardware error due to each flash command and software error due to input validation checking. For hardware error, refer to error handling section at the end of each command description in the associate reference manual to know the root cause.

Table 72: Hardware error troubleshooting

No.	Error code	Value	Possible causes	Solution
1	FTFx_ERR_ACCERR	0x0001	ACCERR bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons.	Provide valid input parameters for each API according to specific flash module.
2	FTFx_ERR_PVIOL	0x0010	FPVIOL bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons	The flash location targeted to program/erase operation must be unprotected. Swap indicator must not be programmed/erased except in Update or Update-Erase state.
3	FTFx_ERR_MGSTAT0	0x0020	MGSTAT0 bit in FSTAT register is set. Refer to corresponding command description of each API on reference manual to get detail reasons	Hardware error

Table 73: Software error troubleshooting

No.	Error code	Value	Possible causes	Solution
1	FTFx_ERR_SIZE	0x2000	The size provided by user is misaligned.	Size must be an aligned value according to specific constrain of each API.
2	FTFx_ERR_RANGE	0x1000	The size or destination provided by user makes start address or end address out of valid range.	The destination and (destination + size) must fall within valid address range.

3	FTFx_ERR_CHANGEPROT	0x0100	Violate protection transition.	In NVM normal mode, protection size cannot be decreased. So, only increasing protection size is permitted if the device is operating in this mode.
5	FTFx_ERR_NOEEE	0x0200	User accesses to EEPROM operation but there is no EEPROM backup enabled.	Need to enable EEPROM by partitioning FlexNVM to have EEPROM backup and/or enable it by SetEEEnable API.
6	FTFx_ERR_EFLASHONLY	0x0400	User accesses to D-Flash operation but there is no D-Flash on FlexNVM.	Need to partition FlexNVM to have D-Flash.
7	FTFx_ERR_RAMRDY	0x0800	User invokes flash program section command but FlexRam is being set for EEPROM emulation.	Need to set FlexRam as traditional Ram by SetEEEnable API.

4 IMPORTANT NOTE

- The *DebugEnable* field of FLASH_SSD_CONFIG structure shall allow user to use this driver in background debug mode, without returning to the calling function, but returning to debug mode instead. To enable this feature, *DebugEnable* must be set to TRUE and macro C90TFS_ENABLE_DEBUG must be defined to 1.
- If user utilizes callback in his application, this callback function must not be placed in the same flash block in which a program/erase operation is going on to avoid RWW error.
- If user wants to suspend the sector erase operation, for simple method, just invoke FlashEraseSuspend function within callback of FlashEraseSector. In this case, FlashEraseSuspend must not be place in the same block in which flash erase sector command is going on.
- To guarantee the correct execution of this driver, the flash cache in the flash memory controller module should be disabled before invoking any API. The normal demo included in the release package provides the code section to disable/enable this flash cache as well.
- For derivatives which access to memory via both byte and word address spaces such as DSC core:
 - ✓ It is recommended to use directly available type as mentioned in functional prototype without any conversion to avoid conflict.
 - ✓ This driver was tested on Large Data Memory model only.
 - ✓ All address provided by user must be in word address space.

APPENDIX A: PERFORMANCE DATA

Code Size

Since code size of each API depends on the following factors:

- Which compiler is using. Typically, it will depend on specific MCU type.
- Which tool is used (CW or IAR) and on which optimization level.
- Value of internal macros which cause to enable/disable some code sections.

This section will provide code size of some derivatives which are typical for above factors. This performance is collected under below conditions:

- Tool version: CW 10.4 (GCC build tool) or IAR 6.4.2.
- Code is built on RAM target.

Table 74: Code size configuration

Configuration	Hardware	Derivative	Compiler	Optimization level
Cfg1	PK21DN512	FTFx_KX_512K_0K_4K_2K_0K	IAR	0
Cfg2	PK21DN512	FTFx_KX_512K_0K_4K_2K_0K	CW	4 (Code Size)
Cfg3	PK21DX128	FTFx_KX_128K_64K_4K_2K_2K	IAR	0
Cfg4	PK21DX128	FTFx_KX_128K_64K_4K_2K_2K	CW	4 (Code Size)
Cfg5	MCF51JF128	FTFx_JX_128K_32K_2K_1K_1K	CW	4 (Code Size)
Cfg6	PC56F84789	FTFx_DX_256K_32K_2K_2K_1K	CW	4 (Code Size)
Cfg7	PKL25Z128	FTFx_LX_128K_0K_0K_1K_0K	CW	4 (Code Size)
Cfg8	PKL25Z128	FTFx_LX_128K_0K_0K_1K_0K	IAR	0

Table 75: Code size for different configurations

API name	Code Size (In Bytes)							
	Cfg 1	Cfg 2	Cfg 3	Cfg 4	Cfg 5	Cfg 6	Cfg 7	Cfg 8
DEFlashPartition	n/a	n/a	74	36	104	34	n/a	n/a
DFlashGetProtection	n/a	n/a	42	32	70	23	n/a	n/a
DFlashSetProtection	n/a	n/a	54	40	80	28	n/a	n/a
EEWrite	n/a	n/a	168	160	272	120	n/a	n/a
EERAMGetProtection	n/a	n/a	54	32	86	30	n/a	n/a
EERAMSetProtection	n/a	n/a	72	48	98	39	n/a	n/a
FlashCheckSum	118	116	134	144	230	101	116	120
FlashCommandSequence	64	44	64	44	104	39	48	58
FlashEraseAllBlock	42	28	42	28	70	24	28	42
FlashEraseBlock	114	72	140	100	218	93	n/a	n/a
FlashEraseResume	74	56	74	56	112	40	60	64
FlashEraseSector	164	132	198	152	300	126	140	154
FlashEraseSuspend	60	36	60	36	80	34	44	50
FlashGetSecurityState	62	48	62	48	86	35	52	60
FlashInit	22	20	564	292	574	337	20	24
FlashProgramCheck	226	220	278	260	398	180	200	230
FlashProgram	198	160	226	184	348	144	168	196

FlashProgramOnce	106	56	106	56	136	48	60	86
FlashProgramSection	178	104	206	124	284	119	n/a	n/a
FlashReadOnce	112	60	112	60	144	56	64	90
FlashReadResource	190	116	218	136	296	123	108	162
FlashSecurityBypass	94	76	94	76	120	43	80	74
FlashVerifyAllBlock	62	32	62	32	86	29	36	54
FlashVerifyBlock	136	84	164	104	234	98	n/a	n/a
FlashVerifySection	166	96	194	120	274	110	92	148
PFlashGetProtection	72	40	72	40	120	44	40	72
PFlashSetProtection	164	88	164	88	198	92	108	144
PFlashSwapCtl	196	108	n/a	n/a	n/a	n/a	n/a	n/a
PFlashSwap	392	328		n/a	n/a	n/a	n/a	n/a
SetEEEEEnable	n/a	n/a	62	32	86	29	n/a	n/a
WaitEEWriteToFinish (within EEWrite)	n/a	n/a	148	128	262	86	n/a	n/a

n/a: not available

Timing

The main factors, impact to timing of each API are system clock on which the device is operating, memory configuration and input parameters provided by user. The other factors which have less impact to these timings are compiler, internal macros which enable/disable some code section, optimization level, etc.

This section provides performance data for some typical derivatives on normal operating frequency of bus clock.

Below are some constrains in measurement:

- Programmable size: this is the supported program size for specific flash module. It can be long word or phrase.
- Programmed data: The data written to memory is 0x11223344 for long word programming and is 0x1122334455667788 for phrase programming.
- For all data regarding to EEPROM backup or D flash, it is available on device with FlexNVM only.

For each specific API, the timing is collected with below conditions:

- **FlashEraseAllBlock():**
 - ✓ EEPROM disable: Erase all flash memory in advance and then program entire P flash and D flash (if available) with programmed data. Note that, for P flash, only program from the third sector to avoid flash configuration field sector for safety reason.
 - ✓ EEPROM enabled: Partition with 50% FlexRam for EEPROM use and 50% FlexNVM for EEPROM backup. Then, program data to P flash and D flash as in case of EEPROM disable. Finally, fill up entire FlexRam section which is used for EEPROM use with programmed data.
- **EEWrite():** Write to FlexRAM with 1 programmed data.
- **FlashVerifyAllBlock():** Collect timing under different margin levels after erasing entire memory via FlashEraseAllBlock().
- **FlashEraseBlock():**
 - ✓ On P flash: Program from the third sector to end of this block with programmed data.
 - ✓ On D flash: Partition with no EEPROM backup. And then program to entire D flash with programmed data.

- **FlashVerifyBlock():** Collect timing under different margin levels after erasing this block via FlashEraseBlock()
- **FlashProgram():** program 1 programmed data.
- **FlashProgramCheck():** check for 1 programmed data after it is written to flash with different margin levels
- **CheckSum():** Do check sum for 1 programmed data on P flash and D flash. The constant FLASH_CALLBACK_CS is set to 1.
- **FlashEraseSector():** Erase 1 sector on P flash or D flash after programming on entire this sector with programmed data.
- **FlashVerifySection():** Do verifying for 1 sector with different margin levels on P flash and D flash after it is fully erased.
- **FlashProgramSection():** Program 1 number of long word/phrase with programmed data to P flash and D flash.

Below table provides timing data for some typical configurations with mentioned above conditions.

Note: The timing is collected when code is executed from ram

Table 76: Timing configuration

Configuration	Hardware	System Clock (MHz)	Flash Clock (MHz)
Cfg1	PKL25Z128	48	24
Cfg2	PK70FX512	120	30
Cfg3	PK21DN512	50	25
Cfg4	PCF51JF128	50	25
Cfg5	PC56F84789	100	25

Table 77: Timing for different configurations

Function/condition		Derivative					
		Cfg1 (ms)	Cfg2 (ms)	Cfg3 (ms)	Cfg4 (ms)	Cfg5 (ms)	
FlashEraseAllBlock	EEPROM enabled	n/a	731.696	n/a	155.259	124.752	
	EEPROM disabled	52.970	729.423	145.258	157.933	123.056	
EEWrite		n/a	1.030	n/a	0.540	0.524	
FlashVerifyAllBlock	Normal margin	1.464	3.143	1.552	1.558	1.798	
	User margin	1.494	3.289	1.609	1.638	1.839	
	Factory margin	1.495	3.290	1.610	1.609	1.842	
FlashEraseBlock	P flash	n/a	179.234	72.130	124.907	81.920	
	D flash	n/a	181.081	n/a	32.781	36.192	
FlashVerifyBlock	P flash	Normal margin	n/a	0.809	1.540	1.547	1.765
		User margin	n/a	0.837	1.564	1.578	1.778
		Factory margin	n/a	0.837	1.565	1.580	1.778
	D flash	Normal margin	n/a	0.803	n/a	0.411	0.464
		User margin	n/a	0.803	n/a	0.412	0.463
		Factory margin	n/a	0.803	n/a	0.412	0.463

FlashProgram	P flash		0.068	0.120	0.113	0.096	0.073
	D flash		n/a	0.112	n/a	0.068	0.070
FlashProgramCheck	P flash	User margin	0.062	0.154	0.071	0.066	0.050
	D flash	Factory margin	0.062	0.144	0.071	0.066	0.049
Checksum	P flash		0.003	0.002	0.002	0.004	0.002
	D flash		n/a	0.002	n/a	0.005	0.003
FlashEraseSector	P flash		9.354	6.651	7.585	7.630	7.481
	D flash		n/a	7.819	n/a	7.657	7.476
FlashProgramSection	P flash		n/a	0.280	0.133	0.120	0.130
	D flash		n/a	0.261	n/a	0.082	0.095
FlashVerifySection	P flash	Normal margin	0.036	0.053	0.047	0.043	0.047
		User margin	0.062	0.082	0.074	0.071	0.058
		Factory margin	0.062	0.082	0.074	0.071	0.058
	D flash	Normal margin	n/a	0.048	n/a	0.043	0.045
		User margin	n/a	0.077	n/a	0.071	0.056
		Factory margin	n/a	0.077	n/a	0.071	0.056

n/a: not available

Callback Time Periods

Callback time period is time interval between two consecutive invoking call back function. This data will have significant meaning to help user to setup properly watchdog time period such that the device will never be reset by watchdog.

The callback time period will depend on the system clock and time interval of callback function. Thus, the callback time period will be the same for all APIs using flash command with the same callback function. For this reason, this section will provide data for FlashChecksum() which does not use flash command and other data which is common used for all APIs using flash command. Below are assumptions for collecting this data:

- The constant FLASH_CALLBACK_CS is set to 1 for FlashChecksum().
- The time interval of callback is no longer than the callback time period.

Refer to [section 5.2](#) for more information about clock setting as well as hardware used to collect this data of each configuration.

Table 78: Callback time period for different configurations

Function \ Derivative	Cfg1 (us)	Cfg2 (us)	Cfg3 (us)	Cfg4 (us)	Cfg5 (us)
FlashChecksum	2.896	2.500	4.280	3.980	0.600
APIs using flash command	0.792	0.650	1.280	0.680	0.120

APPENDIX B: DERIVATIVE AND DEVICE MAPPING

Below table provides mapping between derivative selection and corresponding device. In addition, it gives information of which device was picked up to tested for each derivative. When the information shows as n/a, it means this derivative has not been tested on any device.

Note: refer to [section 2.4](#) for definition of AA, BB, CC, DD, EE, FF on below table.

Table 79: Mapping between derivative and device

No.	Derivative	AA	BB	CC	DD	EE	FF	Supported Device	Tested Device
1	FTFx_KX_256K_256K_4K_2K_2K	Kinetis	256	256	4	2	2	MK10DX256xxx10 MK20DX256xxx10 MK30DX256xxx10 MK40DX256xxx10 MK50DX256xxx10 MK51DX256xxx10 MK53DX256xxx10 MK53DX256Zxxx10 MK60DX256xxx10	n/a
2	FTFx_KX_512K_0K_4K_2K_0K	Kinetis	512	0	4	2	0	MK10DN512xxx10 MK20 DN512xxx10 MK30 DN512xxx10 MK40 DN512xxx10 MK50 DN512xxx10 MK51 DN512xxx10 MK52 DN512xxx10 MK53 DN512xxx10 MK53 DN512Zxxx10 MK60 DN512xxx10 MK21DN512xxx5	MK21DN512xxx5
3	FTFx_KX_512K_512K_16K_4K_4K	Kinetis	512	512	16	4	4	MK60FX512xxx12 MK61FX512xxx12 MK70FX512xxx12 MK70FX512xxx15 MK10FX512xxx12 MK20FX512xxx12	MK70FX512xxx15
4	FTFx_KX_1024K_0K_16K_4K_0K	Kinetis	1024	0	16	4	0	MK60FN1M0xxx12 MK61 FN1M0xxx12 MK70 FN1M0xxx12 MK10 FN1M0xxx12 MK20 FN1M0xxx12	n/a
5	FTFx_KX_32K_0K_2K_1K_0K	Kinetis	32	0	2	1	0	MK10DN32xxx5 MK20DN32xxx5	n/a
6	FTFx_KX_32K_32K_2K_1K_1K	Kinetis	32	32	2	1	1	MK10DX32xxx5 MK20DX32xxx5	n/a
7	FTFx_KX_64K_0K	Kinetis	64	0	2	1	0	MK10DN64xxx5	n/a

	_2K_1K_0K							MK20DN64xxx5	
8	FTFx_KX_64K_32 K_2K_1K_1K	Kinetis	64	32	2	1	1	MK10DX64xxx5 MK20DX64xxx5	n/a
9	FTFx_KX_128K_0 K_2K_1K_0K	Kinetis	128	0	2	1	0	MK10DN128xxx5 MK20DN128xxx5	MK20DN1 28xxx5
10	FTFx_KX_128K_32 K_2K_1K_1K	Kinetis	128	32	2	1	1	MK10DX128xxx5 MK20DX128xxx5	n/a
11	FTFx_KX_64K_32 K_2K_2K_1K	Kinetis	64	32	2	2	1	MK10DX64xxx7 MK20DX64xxx7 MK30DX64xxx7 MK40DX64xxx7	n/a
12	FTFx_KX_128K_32 K_2K_2K_1K	Kinetis	128	32	2	2	1	MK10DX128xxx7 MK20DX128xxx7 MK30DX128xxx7 MK40DX128xxx7 MK50DX128xxx7 MK51DX128xxx7	n/a
13	FTFx_KX_256K_32 K_2K_2K_1K	Kinetis	256	32	2	2	1	MK10DX256xxx7 MK20DX256xxx7 MK30DX256xxx7 MK40DX256xxx7 MK50DX256xxx7 MK51DX256xxx7	MK40DX2 56xxx7
14	FTFx_KX_1024K_0 K_4K_4K_0K	Kinetis	1024	0	4	4	0	MK21FN1M0xxx10 MK22FN1M0xxx10	MK21FN1 M0xxx10
15	FTFx_KX_512K_12 8K_4K_4K_4K	Kinetis	512	12 8	4	4	4	MK21FX512xxx10 MK22FX512xxx10	MK21FX5 12xxx10
16	FTFx_KX_256K_64 K_4K_2K_2K	Kinetis	256	64	4	2	2	MK21DX256xxx5	
17	FTFx_KX_128K_64 K_4K_2K_2K	Kinetis	128	64	4	2	2	MK21DX128xxx5	MK21DX1 28xxx5 MK20DX1 28xxx5
18	FTFx_KX_256K_0 K_4K_2K_0K	Kinetis	256	0	4	2	0	MK51DN256xxx10 MK60DN256xxx10	n/a
19	FTFx_KX_128K_12 8K_4K_2K_2K	Kinetis	128	12 8	4	2	2	MK10DX128xxx10 MK20DX128xxx10 MK30DX128xxx10 MK40DX128xxx10	n/a
20	FTFx_NX_256K_32 K_2K_2K_1K	Nevis2	256	32	2	2	1	MC56F84585 MC56F84587 MC56F84786 MC56F84 789	MC56F847 89
21	FTFx_NX_128K_32 K_2K_2K_1K	Nevis2	128	32	2	2	1	MC56F84462 MC56F84565 MC56F84567 MC56F84763 MC56F84766 MC56F84769	n/a

22	FTFx_NX_96K_32 K_2K_2K_1K	Nevis2	96	32	2	2	1	MC56F84451 MC56F84452 MC56F84550 MC56F84533	n/a
23	FTFx_NX_64K_32 K_2K_2K_1K	Nevis2	64	32	2	2	1	MC56F84441 MC56F84442 MC56F84540 MC56F84543	n/a
24	FTFx_LX_128K_0 K_0K_1K_0K	Kinetis	128	0	0	1	0	MKL15Z128xxx4 MKL25Z128xxx4 MKL46Z128xxx4	MKL25Z1 28xxx4
25	FTFx_LX_64K_0K _0K_1K_0K	Kinetis	64	0	0	1	0	MKL14Z64xxx4 MKL15Z64xxx4 MKL24Z64xxx4 MKL25Z64xxx4	n/a
26	FTFx_LX_32K_0K _0K_1K_0K	Kinetis	32	0	0	1	0	MKL05Z32xxx4 MKL04Z32xxx4 MKL14Z32xxx4 MKL24Z32xxx4 MKL25Z32xxx4	MKL05Z3 2xxx4
27	FTFx_LX_16K_0K _0K_1K_0K	Kinetis	16	0	0	1	0	MKL05Z16xxx4 MKL04Z16xxx4 MKL02Z16xxx4	MKL02Z1 6xxx4
28	FTFx_LX_8K_0K_ 0K_1K_0K	Kinetis	8	0	0	1	0	MKL05Z8xxx4 MKL04Z8xxx4	n/a
29	FTFx_LX_256K_0 K_0K_1K_0K	Kinetis	256	0	0	1	0	MKL46Z256xxx4	MKL46Z2 56xxx4
30	FTFx_AX_64K_0K _0K_1K_0K	Anguillar Silver	64	0	0	1	0	MC56F82743 MC56F82744 MC56F82746 MC56F82748	MC56F827 48
31	FTFx_AX_48K_0K _0K_1K_0K	Anguillar Silver	48	0	0	1	0	MC56F82733 MC56F82734 MC56F82736 MC56F82738	n/a
32	FTFx_AX_32K_0K _0K_1K_0K	Anguillar Silver	32	0	0	1	0	MC56F82723 MC56F82724 MC56F82726 MC56F82728	n/a
33	FTFx_AX_16K_0K _0K_1K_0K	Anguillar Silver	16	0	0	1	0	MC56F82323 MC56F82326	n/a
34	FTFx_CX_128K_32 K_2K_1K_1K	Coldfire	128	32	2	1	1	MCF51JF128 MCF51JU128	MCF51JF1 28
35	FTFx_CX_64K_32 K_2K_1K_1K	Coldfire	64	32	2	1	1	MCF51JF64 MCF51JU64	n/a
36	FTFx_CX_32K_32 K_2K_1K_1K	Coldfire	32	32	2	1	1	MCF51JF32 MCF51JU32	n/a
37	FTFx_CX_256K_32 K_2K_1K_1K	Coldfire	256	32	2	1	1	MCF51JG256 MCF51FD256	MCF51JG2 56

38	FTFx_MX_64K_0K_0K_1K_0K	Kinetis	64	0	0	1	0	MKM13Z64xxx5 MKM14Z64xxx5 MKM32Z64xxx5 MKM33Z64xxx5	n/a
39	FTFx_MX_128K_0K_0K_1K_0K	Kinetis	128	0	0	1	0	MKM14Z128xxx5 MKM33Z128xxx5 MKM34Z128xxx5	MKM34Z128xxx5
40	FTFx_KX_512K_0K_0K_2K_0K	Kinetis	512	0	0	2	0	MK22FN512xxx12 MKV31F512xxx12	MK22FN512xxx8
41	FTFx_KX_256K_0K_0K_2K_0K	Kinetis	256	0	0	2	0	MK22FN256xxx12 MKV31F256xxx12	n/a
42	FTFx_KX_128K_0K_0K_2K_0K	Kinetis	128	0	0	2	0	MKV30F128xxx10 MK02FN128xxx10 MKV31F128xxx10	n/a
43	FTFx_KX_64K_0K_0K_2K_0K	Kinetis	64	0	0	2	0	MKV30F64xxx10 MK02FN64xxx10	n/a
44	FTFx_KX_256K_0K_0K_4K_0K	Kinetis	256	0	0	4	0	MKV40F256xxx15 MKV45F256xxx15 MKV46F256xxx15 MK24F256xxx12	n/a
45	FTFx_KX_128K_0K_0K_4K_0K	Kinetis	128	0	0	4	0	MKV40F128xxx15 MKV43F128xxx15 MKV44F128xxx15 MKV45F128xxx15 MKV46F128xxx15	n/a
46	FTFx_KX_64K_0K_0K_4K_0K	Kinetis	64	0	0	4	0	MKV40F64xxx15 MKV43F64xxx15 MKV44F64xxx15	n/a
47	FTFx_KX_2048K_0K_4K_4K_0K	Kinetis	2048	0	4	4	0	MK65FN2M0xxx18 MK66FN2M0xxx18	n/a
48	FTFx_KX_1024K_256K_4K_4K_4K	Kinetis	1024	256	4	4	4	MK65FX1M0xxx18 MK66FX1M0xxx18	n/a