# Standard Software Driver for C90TFS/FTFx Flash

# User's Manual

# REVISION LIST

| Version No. | Date | Author | Description |
|---|---|---|---|
| 1.1.0 | 15-10-2014 | FPT Team | First version of SDK C90TFS flash driver which is inherited from BM C90TFS flash driver v1.02 (08.04.2014, FPT Team) |

# TABLE OF CONTENTS

## LIST OF TABLES

# 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.

## 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 2.7 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 | 7.2.0 |
| 2 | CW for MCU | Development tool | 10.4 & 10.5 |
| 2 | ARM GCC | Development tool | 4.8.3 2014q1 or later |
| 3 | Keil MDK | Development tool | 5.11 or later |
| 4 | Kinetis Design Studio (KDS) | Development tool | v1.0.2 or later |

| | | | |
|---|---|---|---|
| 5 | TrueSTUDIO_for_ARM_Pro_win32 | Development tool | v5.1.1_20140820-1543 |
| 6 | JLink ARM | Debugger tool | n/a |
| 7 | PE micro Multilink universal | Debugger tool | n/a |
| 8 | Mini/micro USB cable | Debugger tool | n/a |
| 9 | Hardware (tower/base board,…) for specific device | Hardware | n/a |

## 1.5 Documentation References

**Table 2: References**

| No | Document Name | Version | Document Identifier |
|---|---|---|---|
| 1 | FSL_eNVM_FTFx_UM.doc | 3.4 | |

## 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 devices since associate features are not available on them. For instance, the APIs relating to swap feature are not available on devices 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 bellow:

```
typedef  struct   _ssd_config
{
        uint32_t            ftfxRegBase;
        uint32_t            PFlashBase;
        uint32_t            PFlashSize;
        uint32_t            DFlashBase;
        uint32_t            DFlashSize;
        uint32_t            EERAMBase;
        uint32_t            EEESize;
        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*, *PFlashBase*, *PFlashSize*, *DFlashBase*, *EERAMBase*, *DebugEnable* and *CallBack* before passing the structure to SSD functions. The rest of parameters such as *DFlashSize*, and *EEESize* will be initialized in '*FlashInit()'* automatically. The pointer to *CallBack* has to be initialized either for null callback or for 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. |
| PFlashBase | uint32_t | The base address of P-Flash memory. |
| PFlashSize | uint32_t | The size in byte of P-Flash memory. |
| DFlashBase | uint32_t | For FlexNVM device, this is the base address of D-Flash memory (FlexNVM memory). For non-FlexNVM device, this field is unused. |
| DFlashSize | 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. |
| EERAMBase | uint32_t | The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device). |
| EEESize | 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:<br>- TRUE: enable debug mode.<br>- FALSE: disable debug mode.<br>Refer to section 4 for more information. |
| CallBack | Function pointer | Call back function to service the time critical events. |

## 2.4 Flash Interrupt Macros

The driver 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 ftfxRegBase: 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.

## 2.5 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)**

## 2.6 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 7: 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 |

## 2.7 SSD General Functions

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

### 2.7.1 FlashInit()

#### 2.7.1.1 Overview

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

#### 2.7.1.2 Prototype

**uint32_t FlashInit (PFLASH_SSD_CONFIG pSSDConfig)**

#### 2.7.1.3 Arguments

**Table 8: Arguments for FlashInit()**

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

#### 2.7.1.4 Return Values

**Table 9: Return Values for FlashInit()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |

#### 2.7.1.5 Comments

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

### 2.7.2 PFlashGetProtection()

#### 2.7.2.1 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.

#### 2.7.2.2 Prototype

**uint32_t PFlashGetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint32_t* protectStatus)**

#### 2.7.2.3 Arguments

**Table 10: 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:<br>- 0: this area is protected.<br>- 1: this area is unprotected. |

#### 2.7.2.4 Return Values

**Table 11: Return Values for PFlashGetProtection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |

#### 2.7.2.5 Comments

None.

### 2.7.3 PFlashSetProtection()

#### 2.7.3.1 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.

#### 2.7.3.2 Prototype

uint32_t **PFlashSetProtection (PFLASH_SSD_CONFIG pSSDConfig,** uint32_t **protectStatus)**

#### 2.7.3.3 Arguments

**Table 12: 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 2.7.2.3 for more details. |

#### 2.7.3.4 Return Values

**Table 13: Return Values for PFlashSetProtection ()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_CHANGEPROT |

#### 2.7.3.5 Comments

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

### 2.7.4 DFlashGetProtection()

#### 2.7.4.1 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.

#### 2.7.4.2 Prototype

**uint32_t DFlashGetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t* protectStatus)**

#### 2.7.4.3 Arguments

**Table 14: 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:<br>- 0 : this area is protected.<br>- 1 : this area is unprotected. |

#### 2.7.4.4 Return Values

**Table 15: Return Values for DFlashGetProtection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_EFLASHONLY |

#### 2.7.4.5 Comments

None.

### 2.7.5 DFlashSetProtection()

#### 2.7.5.1 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.

#### 2.7.5.2 Prototype

**uint32_t DFlashSetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)**

#### 2.7.5.3 Arguments

**Table 16: 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 2.7.4.3 for more details. |

#### 2.7.5.4 Return Values

**Table 17: Return Values for DFlashSetProtection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_EFLASHONLY<br>FTFx_ERR_CHANGEPROT |

#### 2.7.5.5 Comments

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

### 2.7.6 EERAMGetProtection()

#### 2.7.6.1 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.

#### 2.7.6.2 Prototype

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

#### 2.7.6.3 Arguments

**Table 18: 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:<br>- 0: this area is protected.<br>- 1: this area is unprotected. |

#### 2.7.6.4 Return Values

**Table 19: Return Values for EERAMGetProtection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_NOEEE |

#### 2.7.6.5 Comments

None.

### 2.7.7 EERAMSetProtection()

#### 2.7.7.1 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.

#### 2.7.7.2 Prototype

**uint32_t EERAMSetProtection (PFLASH_SSD_CONFIG pSSDConfig, uint8_t protectStatus)**

#### 2.7.7.3 Arguments

**Table 20: 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 2.7.6.3 for more details. |

#### 2.7.7.4 Return Values

**Table 21: Return Values for EERAMSetProtection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_NOEEE<br>FTFx_ERR_CHANGEPROT |

#### 2.7.7.5 Comments

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

### 2.7.8 FlashGetSecurityState()

#### 2.7.8.1 Overview

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

#### 2.7.8.2 Prototype

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

#### 2.7.8.3 Arguments

**Table 22: 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.<br>• FLASH_NOT_SECURE - 0x01, Flash currently not in secure state;<br>• FLASH_SECURE_BACKDOOR_ENABLED – 0x02, Flash is secured and backdoor key access enabled;<br>• FLASH_SECURE_BACKDOOR_DISABLED – 0x04, Flash is secured and backdoor key access disabled. |

#### 2.7.8.4 Return Values

**Table 23: Return Values for FlashGetSecurityState()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |

#### 2.7.8.5 Comments

None.

### 2.7.9 FlashSecurityBypass()

#### 2.7.9.1 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.

#### 2.7.9.2 Prototype

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

#### 2.7.9.3 Arguments

**Table 24: 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 |
| pFlashComm -andSequence | pFLASHCOMMANDSEQUENCE | Pointer to the flash command sequence function |

#### 2.7.9.4 Return Values

**Table 25: Return Values for FlashSecurityBypass()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR |

#### 2.7.9.5 Comments

None

### 2.7.10 FlashEraseAllBlock()

#### 2.7.10.1 Overview

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

#### 2.7.10.2 Prototype

**uint32_t FlashEraseAllBlock (PFLASH_SSD_CONFIG pSSDConfig, **
**pFLASHCOMMANDSEQUENCE pFlashCommandSequence)**

#### 2.7.10.3 Arguments

**Table 26: Arguments for FlashEraseAllBlock()**

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

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

#### 2.7.10.4 Return Values

**Table 27: Return Values for FlashEraseAllBlock()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_PVIOL<br>FTFx_ERR_MGSTAT0<br>FTFx_ERR_ACCERR |

#### 2.7.10.5 Comments

None

### 2.7.11 FlashEraseBlock()

#### 2.7.11.1 Overview

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

#### 2.7.11.2 Prototype

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

#### 2.7.11.3 Arguments

**Table 28: 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. |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.11.4 Return Values

**Table 29: Return Values for FlashEraseBlock()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_PVIOL<br>FTFx_ERR_MGSTAT0 |

#### 2.7.11.5 Comments

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

### 2.7.12 FlashEraseSector()

#### 2.7.12.1 Overview

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

#### 2.7.12.2 Prototype

**uint32_t FlashEraseSector(PFLASH_SSD_CONFIG pSSDConfig, **
**uint32_t dest, **
**uint32_t size, **
**pFLASHCOMMANDSEQUENCE pFlashCommandSequence)**

#### 2.7.12.3 Arguments

**Table 30: 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. |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.12.4 Return Values

**Table 31: Return Values for FlashEraseSector()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_SIZE<br>FTFx_ERR_PVIOL<br>FTFx_ERR_MGSTAT0 |

#### 2.7.12.5 Comments

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

### 2.7.13 FlashEraseSuspend()

#### 2.7.13.1 Overview

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

#### 2.7.13.2 Prototype

**uint32_t FlashEraseSuspend (PFLASH_SSD_CONFIG pSSDConfig)**

#### 2.7.13.3 Arguments

**Table 32: Arguments for FlashEraseSuspend()**

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

#### 2.7.13.4 Return Values

**Table 33: Return Values for FlashEraseSuspend()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |

#### 2.7.13.5 Comments

This function must be called from either RAM or different Flash blocks that one is being targeted for erasing.

### 2.7.14 FlashEraseResume()

#### 2.7.14.1 Overview

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

#### 2.7.14.2 Prototype

**uint32_t FlashEraseResume (PFLASH_SSD_CONFIG pSSDConfig)**

#### 2.7.14.3 Arguments

**Table 34: Arguments for FlashEraseResume()**

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

#### 2.7.14.4 Return Values

**Table 35: Return Values for FlashEraseResume()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |

#### 2.7.14.5 Comments

This function must be called from either RAM or different Flash blocks that one is being targeted for erasing.

### 2.7.15 FlashProgramSection()

#### 2.7.15.1 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.

#### 2.7.15.2 Prototype

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

#### 2.7.15.3 Arguments

**Table 36: 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 |

#### 2.7.15.4 Return Values

**Table 37: Return Values for FlashProgramSection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_PVIOL<br>FTFx_ERR_MGSTAT0<br>FTFx_ERR_RAMRDY |

#### 2.7.15.5 Comments

For devices 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.

### 2.7.16 FlashProgram ()

#### 2.7.16.1 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.

#### 2.7.16.2 Prototype

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

#### 2.7.16.3 Arguments

**Table 38: 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 |

#### 2.7.16.4 Return Values

**Table 39: Return Values for FlashProgramLongword()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_PVIOL<br>FTFx_ERR_SIZE<br>FTFx_ERR_MGSTAT0 |

#### 2.7.16.5 Comments

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

### 2.7.17 FlashCheckSum()

#### 2.7.17.1 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.

#### 2.7.17.2 Prototype

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

#### 2.7.17.3 Arguments

**Table 40: 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 |

#### 2.7.17.4 Return Values

**Table 41: Return Values for FlashCheckSum()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_RANGE |

#### 2.7.17.5 Comments

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

### 2.7.18 FlashVerifyAllBlock()

#### 2.7.18.1 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.

#### 2.7.18.2 Prototype

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

#### 2.7.18.3 Arguments

**Table 42: 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:<br>• marginLevel = 0x0: use 'Normal' read level<br>• marginLevel = 0x1: use the 'User' read<br>• marginLevel = 0x2: use the 'Factory' read |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.18.4 Return Values

**Table 43: Return Values for FlashVerifyAllBlock()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.18.5 Comments

None

### 2.7.19 FlashVerifyBlock()

#### 2.7.19.1 Overview

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

#### 2.7.19.2 Prototype

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

#### 2.7.19.3 Arguments

**Table 44: 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:<br>• marginLevel = 0x0: use 'Normal' read level<br>• marginLevel = 0x1: use the 'User' read<br>• marginLevel = 0x2: use the 'Factory' read |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function. |

#### 2.7.19.4 Return Values

**Table 45: Return Values for FlashVerifyBlock()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.19.5 Comments

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

### 2.7.20 FlashVerifySection()

#### 2.7.20.1 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.

#### 2.7.20.2 Prototype

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

#### 2.7.20.3 Arguments

**Table 46: 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:<br>• marginLevel = 0x0: use 'Normal' read level<br>• marginLevel = 0x1: use the 'User' read<br>• marginLevel = 0x2: use the 'Factory' read |
| pFlashComma-ndSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.20.4 Return Values

**Table 47: Return Values for FlashVerifySection()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.20.5 Comments

None

### 2.7.21 FlashReadOnce()

#### 2.7.21.1 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.

#### 2.7.21.2 Prototype

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

#### 2.7.21.3 Arguments

**Table 48: 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 device. |
| pDataArray | uint8_t* | Pointer to the array to return the data read by the read once command |
| pFlashComma-ndSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.21.4 Return Values

**Table 49: Return Values for FlashReadOnce()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR |

#### 2.7.21.5 Comments

None.

### 2.7.22 FlashProgramOnce()

#### 2.7.22.1 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.

#### 2.7.22.2 Prototype

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

#### 2.7.22.3 Arguments

**Table 50: 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 2.7.21.3 |
| pDataArray | uint8_t* | Pointer to the array from which data will be taken for program once command. |
| pFlashComma-ndSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.22.4 Return Values

**Table 51: Return Values for FlashProgramOnce()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.22.5 Comments

None.

### 2.7.23 FlashProgramCheck()

#### 2.7.23.1 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.

#### 2.7.23.2 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)**

#### 2.7.23.3 Arguments

**Table 52: 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:<br>• marginLevel = 0x1: read at 'User' margin 1/0 level.<br>• marginLevel = 0x2: read at 'Factory' margin 1/0 level. |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function. |

#### 2.7.23.4 Return Values

**Table 53: Return Values for FlashProgramCheck()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.23.5 Comments

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

### 2.7.24 FlashReadResource()

#### 2.7.24.1 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.

#### 2.7.24.2 Prototype

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

#### 2.7.24.3 Arguments

**Table 54: 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. |
| resourceSelect Code | uint8_t | Read resource select code:<br>• 0: Flash IFR<br>• 1: Version ID |
| pFlashComm-andSequence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.24.4 Return Values

**Table 55: Return Values for FlashReadResource()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR |

#### 2.7.24.5 Comments

None.

### 2.7.25 DEFlashPartition()

#### 2.7.25.1 Overview

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

#### 2.7.25.2 Prototype

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

#### 2.7.25.3 Arguments

**Table 56: 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. |
| pFlashCommandS-equence | pFLASHCOMMANDS-EQUENCE | Pointer to the flash command sequence function |

#### 2.7.25.4 Return Values

**Table 57: Return Values for DEFlashPartition()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.25.5 Comments

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

### 2.7.26 SetEEEEnable()

#### 2.7.26.1 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.

#### 2.7.26.2 Prototype

**uint32_t SetEEEEnable (PFLASH_SSD_CONFIG pSSDConfig, uint8_t EEEEnable)**

#### 2.7.26.3 Arguments

**Table 58: Arguments for SetEEEEnable()**

| Argument | Type | Description |
|---|---|---|
| pSSDConfig | PFLASH_SSD_CONFIG | The SSD configuration structure pointer mentioned in section 2.3 |
| EEEEnable | uint8_t | FlexRam function control code. It can be:<br>- 0xFF: make FlexRam available for RAM.<br>- 0x00: make FlexRam available for EEPROM. |

#### 2.7.26.4 Return Values

**Table 59: Return Values for SetEEEEnable()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR |

#### 2.7.26.5 Comments

None.

### 2.7.27 EEEWrite()

#### 2.7.27.1 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.

#### 2.7.27.2 Prototype

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

#### 2.7.27.3 Arguments

**Table 60: Arguments for EEEWrite()**

| 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. |

#### 2.7.27.4 Return Values

**Table 61: Return Values for EEEWrite()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_RANGE<br>FTFx_ERR_NOEEE<br>FTFx_ERR_PVIOL |

#### 2.7.27.5 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.

### 2.7.28 PFlashSwapCtl ()

#### 2.7.28.1 Overview

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

#### 2.7.28.2 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)
```

#### 2.7.28.3 Arguments

**Table 62: 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:<br>• 0x01 - Initialize Swap System<br>• 0x02 - Set Swap in Update State<br>• 0x04 - Set Swap in Complete Stat<br>• 0x08 - Report Swap Status |

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

#### 2.7.28.4 Return Values

**Table 63: Return Values for PFlashSwapCtl ()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.28.5 Comments

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

### 2.7.29 PFlashSwap()

#### 2.7.29.1 Overview

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

#### 2.7.29.2 Prototype

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

#### 2.7.29.3 Arguments

**Table 64: 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 2.7.29.5 for more description. |
| pFlashCommandSequence | pFLASHCOMMANDSEQUENCE | Pointer to the flash command sequence function |

#### 2.7.29.4 Return Values

**Table 65: Return Values for PFlashSwap()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_MGSTAT0 |

#### 2.7.29.5 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:

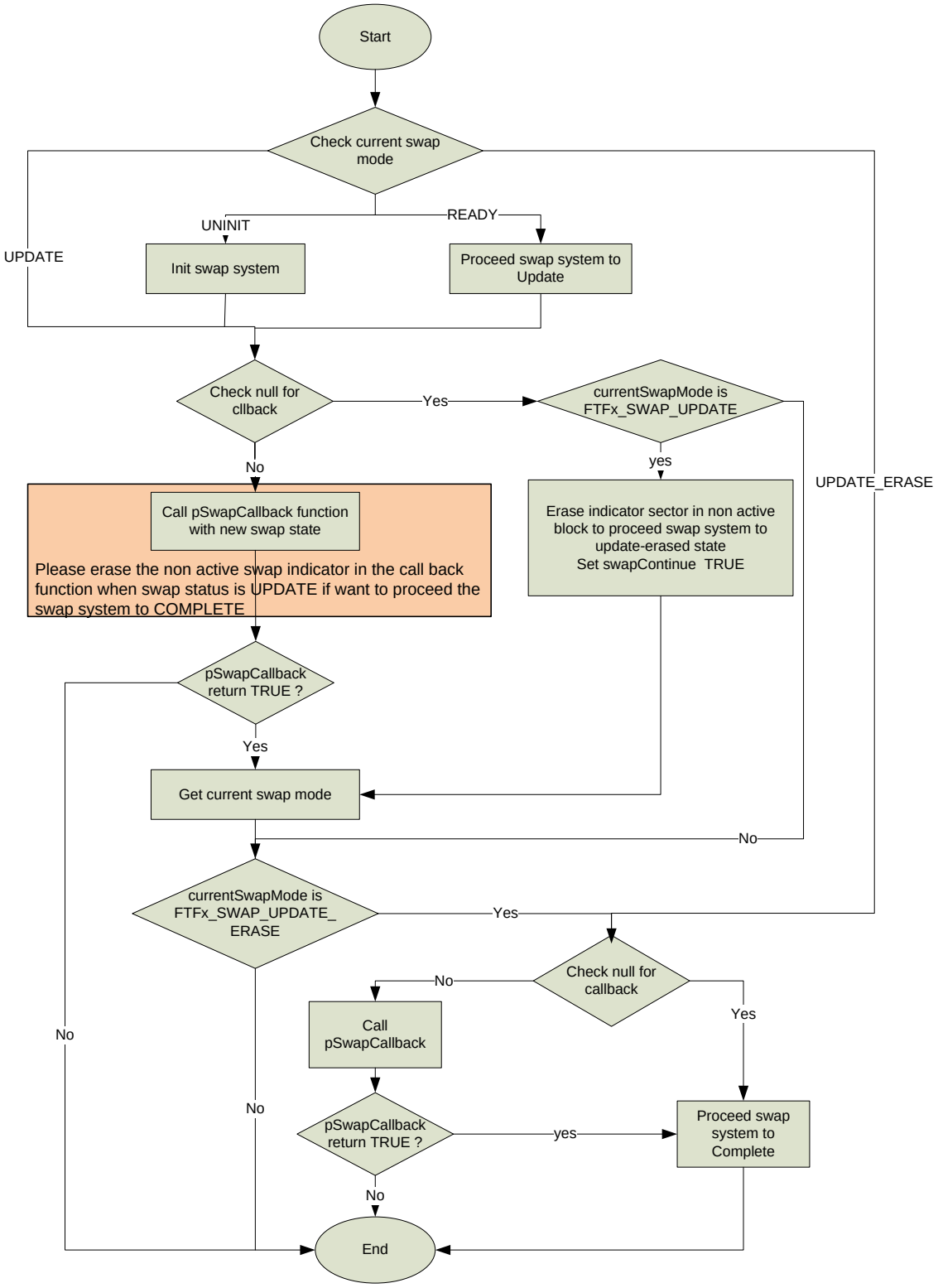


**Figure 1 : PFlashSwap flow chart**

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 memory but not in flash configuration area. If P-Flash memory has two logical blocks, then swap indicator must be in P-Flash block 0. If P-Flash memory 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.

## 2.7.30 RelocateFunction()

### 2.7.30.1 Overview

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

### 2.7.30.2 Prototype

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

### 2.7.30.3 Arguments

**Table 66: Arguments for RelocateFunction ()**

| Argument | Type | Description |
|---|---|---|
| dest | uint32_t | Destination address in RAM where you want to place the function |

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

#### 2.7.30.4 Return Values

**Table 67: Return Values for RelocateFunction ()**

| Type | Description |
|---|---|
| uint32_t | Relocated address of the function |

#### 2.7.30.5 Comments

None

## 2.8 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.

### 2.8.1 FlashCommandSequence()

#### 2.8.1.1 Overview

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

#### 2.8.1.2 Prototype

**uint32_t FlashCommandSequence (PFLASH_SSD_CONFIG pSSDConfig)**

#### 2.8.1.3 Arguments

**Table 68: Arguments for FlashCommandSequence()**

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

#### 2.8.1.4 Return Values

**Table 69: Return Values for FlashCommandSequence()**

| Type | Description | Possible Values |
|---|---|---|
| uint32_t | Successful completion | FTFx_OK |
| | Error value | FTFx_ERR_ACCERR<br>FTFx_ERR_PVIOL<br>FTFx_ERR_MGSTAT0 |

#### 2.8.1.5 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 70: 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 programed/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 71: 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.

# 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 72: Code size configuration**

| Configuration | Hardware | Compiler | Optimization level |
|---|---|---|---|
| Cfg1 | PK21DN512 | IAR | 0 |
| Cfg2 | PK21DN512 | CW | 4 (Code Size) |
| Cfg3 | PK21DX128 | IAR | 0 |
| Cfg4 | PK21DX128 | CW | 4 (Code Size) |
| Cfg7 | PKL25Z128 | CW | 4 (Code Size) |
| Cfg8 | PKL25Z128 | IAR | 0 |

**Table 73: Code size for different configurations**

| API name | Code Size (In Bytes) | | | | | |
|---|---|---|---|---|---|---|
| | Cfg 1 | Cfg 2 | Cfg 3 | Cfg 4 | Cfg 7 | Cfg 8 |
| DEFlashPartition | n/a | n/a | 74 | 36 | n/a | n/a |
| DFlashGetProtection | n/a | n/a | 42 | 32 | n/a | n/a |
| DFlashSetProtection | n/a | n/a | 54 | 40 | n/a | n/a |
| EEEWrite | n/a | n/a | 168 | 160 | n/a | n/a |
| EERAMGetProtection | n/a | n/a | 54 | 32 | n/a | n/a |
| EERAMSetProtection | n/a | n/a | 72 | 48 | n/a | n/a |
| FlashCheckSum | 118 | 116 | 134 | 144 | 116 | 120 |
| FlashCommandSequence | 64 | 44 | 64 | 44 | 48 | 58 |
| FlashEraseAllBlock | 42 | 28 | 42 | 28 | 28 | 42 |
| FlashEraseBlock | 114 | 72 | 140 | 100 | n/a | n/a |
| FlashEraseResume | 74 | 56 | 74 | 56 | 60 | 64 |
| FlashEraseSector | 164 | 132 | 198 | 152 | 140 | 154 |
| FlashEraseSuspend | 60 | 36 | 60 | 36 | 44 | 50 |
| FlashGetSecurityState | 62 | 48 | 62 | 48 | 52 | 60 |
| FlashInit | 22 | 20 | 564 | 292 | 20 | 24 |
| FlashProgramCheck | 226 | 220 | 278 | 260 | 200 | 230 |
| FlashProgram | 198 | 160 | 226 | 184 | 168 | 196 |
| FlashProgramOnce | 106 | 56 | 106 | 56 | 60 | 86 |
| FlashProgramSection | 178 | 104 | 206 | 124 | n/a | n/a |

| | | | | | | |
|---|---|---|---|---|---|---|
| FlashReadOnce | 112 | 60 | 112 | 60 | 64 | 90 |
| FlashReadResource | 190 | 116 | 218 | 136 | 108 | 162 |
| FlashSecurityBypass | 94 | 76 | 94 | 76 | 80 | 74 |
| FlashVerifyAllBlock | 62 | 32 | 62 | 32 | 36 | 54 |
| FlashVerifyBlock | 136 | 84 | 164 | 104 | n/a | n/a |
| FlashVerifySection | 166 | 96 | 194 | 120 | 92 | 148 |
| PFlashGetProtection | 72 | 40 | 72 | 40 | 40 | 72 |
| PFlashSetProtection | 164 | 88 | 164 | 88 | 108 | 144 |
| PFlashSwapCtl | 196 | 108 | n/a | n/a | n/a | n/a |
| PFlashSwap | 392 | 328 | | n/a | n/a | n/a |
| SetEEEEnable | n/a | n/a | 62 | 32 | n/a | n/a |
| WaitEEWriteToFinish (within EEEWrite) | n/a | n/a | 148 | 128 | 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 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 74: Timing configuration**

| Configuration | Hardware | System Clock (MHz) | Flash Clock (MHz) |
|---|---|---|---|
| Cfg1 | PKL25Z128 | 48 | 24 |
| Cfg2 | PK70FX512 | 120 | 30 |
| Cfg3 | PK21DN512 | 50 | 25 |

**Table 75: Timing for different configurations**

| Function/condition \ Derivative | | | Cfg1 (ms) | Cfg2 (ms) | Cfg3 (ms) |
|---|---|---|---|---|---|
| FlashEraseAllBlock | EEPROM enabled | | n/a | 731.696 | n/a |
| | EEPROM disabled | | 52.970 | 729.423 | 145.258 |
| EEWrite | | | n/a | 1.030 | n/a |
| FlashVerifyAllBlock | Normal margin | | 1.464 | 3.143 | 1.552 |
| | User margin | | 1.494 | 3.289 | 1.609 |
| | Factory margin | | 1.495 | 3.290 | 1.610 |
| FlashEraseBlock | P flash | | n/a | 179.234 | 72.130 |
| | D flash | | n/a | 181.081 | n/a |
| FlashVerifyBlock | P flash | Normal margin | n/a | 0.809 | 1.540 |
| | | User margin | n/a | 0.837 | 1.564 |
| | | Factory margin | n/a | 0.837 | 1.565 |
| | D flash | Normal margin | n/a | 0.803 | n/a |
| | | User margin | n/a | 0.803 | n/a |
| | | Factory margin | n/a | 0.803 | n/a |
| FlashProgram | P flash | | 0.068 | 0.120 | 0.113 |
| | D flash | | n/a | 0.112 | n/a |
| FlashProgramCheck | P flash | User margin | 0.062 | 0.154 | 0.071 |
| | D flash | Factory margin | 0.062 | 0.144 | 0.071 |
| CheckSum | P flash | | 0.003 | 0.002 | 0.002 |

| | | | | | |
|---|---|---|---|---|---|
| | D flash | | n/a | 0.002 | n/a |
| FlashEraseSector | P flash | | 9.354 | 6.651 | 7.585 |
| | D flash | | n/a | 7.819 | n/a |
| FlashProgramSection | P flash | | n/a | 0.280 | 0.133 |
| | D flash | | n/a | 0.261 | n/a |
| FlashVerifySection | P flash | Normal margin | 0.036 | 0.053 | 0.047 |
| | | User margin | 0.062 | 0.082 | 0.074 |
| | | Factory margin | 0.062 | 0.082 | 0.074 |
| | D flash | Normal margin | n/a | 0.048 | n/a |
| | | User margin | n/a | 0.077 | n/a |
| | | Factory margin | n/a | 0.077 | n/a |

***n/a: not available***

## CallBack Time Periods

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

The callback period will depend on the system clock and time interval of callback function. Thus, the callback 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 period.

Refer to Timing for more information about clock setting as well as hardware used to collect this data of each configuration.

**Table 76: Callback period for different configurations**

| Function \ Derivative | Cfg1 (us) | Cfg2 (us) | Cfg3 (us) |
|---|---|---|---|
| FlashCheckSum | 2.896 | 2.500 | 4.280 |
| APIs using flash command | 0.792 | 0.650 | 1.280 |