Standard Software Driver for C90TFS/FTFx Flash

User's Manual

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	- Customize the content of document by removing unnecessary sections. - Update functional prototype of below APIs to match to latest driver version (v034): FlashProgramLongWord, FlashProgramPhrase, EEEWrite, FLashReadResource, 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

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's 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 seperate 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
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 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 detail 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	8-bits	signed char
VINT8	8-bits	volatile signed char
UINT8	8-bits	unsigned char
VUINT8	8-bits	volatile unsigned char
INT16	16-bits	signed short
VINT16	16-bits	volatile signed short
UINT16	16-bits	unsigned short
VUINT16	16-bits	volatile unsigned short
INT32	32-bits	signed long
VINT32	32-bits	volatile signed long
UINT32	32-bits	unsigned long
VUINT32	32-bits	volatile unsigned long

2.3 Configuration parameter

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

```
typedef  struct   _ssd_config
{
        UINT32                 ftfxRegBase;
        UINT32                 PFlashBlockBase;
        UINT32                 PFlashBlockSize;
        UINT32                 DFlashBlockBase;
        UINT32                 DFlashBlockSize;
        UINT32                 EERAMBlockBase;
        UINT32                 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	The register base address of C90TFS/FTFx module.
PFlashBlockBase	UINT32	The base address of P-Flash block.
PFlashBlockSize	UINT32	The size of P-Flash block.
DFlashBlockBase	UINT32	For FlexNVM device, this is the base address of D-Flash block (FlexNVM block). For non-FlexNVM device, this field is unused.
DFlashBlockSize	UINT32	For FlexNVM device, this is the size of area which is used as D-Flash from FlexNVM block. For non-FlexNVM device, this field is unused.
EERAMBlockBase	UINT32	The base address of FlexRAM (for FlexNVM device) or acceleration RAM block (for non-FlexNVM device).
EEEBlockSize	UINT32	For FlexNVM device, this is the size 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: - 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

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 block size.
CC: FlexNVM block 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 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.

2.6 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. If it is not necessary to provide the CallBack service, the user will be able to disable it by a NULL function macro.

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

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

2.7 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 will consist 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	Can’t 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 SSD General Functions

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

2.8.1 FlashInit()

2.8.1.1 Overview

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

2.8.1.2 Prototype

UINT32 FlashInit (PFLASH_SSD_CONFIG PSSDConfig)

2.8.1.3 Arguments

Table 9: Arguments for FlashInit()

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

2.8.1.4 Return Values

Table 10: Return Values for FlashInit()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.1.5 Comments

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

2.8.2 PFlashGetProtection()

2.8.2.1 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.

2.8.2.2 Prototype

UINT32 PFlashGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT32* protectStatus)

2.8.2.3 Arguments

Table 11: Arguments for PFlashGetProtection()

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

protectStatus	UINT32*	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: - 0: this area is protected. - 1: this area is unprotected.

2.8.2.4 Return Values

Table 12: Return Values for PFlashGetProtection()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.2.5 Comments

None.

2.8.3 PFlashSetProtection()

2.8.3.1 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 won't be changed.

2.8.3.2 Prototype

UINT32 PFlashSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT32 protectStatus)

2.8.3.3 Arguments

Table 13: Arguments for PFlashSetProtection ()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
protectStatus	UINT32	The expected protect status user wants to set to P-Flash protection register. Refer to section 2.8.2.3 for more details.

2.8.3.4 Return Values

Table 14: Return Values for PFlashSetProtection ()

Type	Description	Possible Values
UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_CHANGEPROT

2.8.3.5 Comments

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

2.8.4 DFlashGetProtection()

2.8.4.1 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.

2.8.4.2 Prototype

UINT32 DFlashGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8* protectStatus)

2.8.4.3 Arguments

Table 15: Arguments for DFlashGetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
protectStatus	UINT8*	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: - 0: this area is protected. - 1: this area is unprotected.

2.8.4.4 Return Values

Table 16: Return Values for DFlashGetProtection()

Type	Description	Possible Values
UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_EFLASHONLY

2.8.4.5 Comments

None.

2.8.5 DFlashSetProtection()

2.8.5.1 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 won't be changed.

2.8.5.2 Prototype

UINT32 DFlashSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8 protectStatus)

2.8.5.3 Arguments

Table 17: Arguments for DFlashSetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
protectStatus	UINT8	The expected protect status user wants to set to D-Flash protection register. Refer to section 2.8.4.3 for more details.

2.8.5.4 Return Values

Table 18: Return Values for DFlashSetProtection()

Type	Description	Possible Values
UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_EFLASHONLY FTFx_ERR_CHANGEPROT

2.8.5.5 Comments

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

2.8.6 EERAMGetProtection()

2.8.6.1 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.

2.8.6.2 Prototype

UINT32 EERAMGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8* protectStatus)

2.8.6.3 Arguments

Table 19: Arguments for EERAMGetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
protectStatus	UINT8*	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: - 0: this area is protected. - 1: this area is unprotected.

2.8.6.4 Return Values

Table 20: Return Values for EERAMGetProtection()

Type	Description	Possible Values
UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_NOEEE

2.8.6.5 Comments

None.

2.8.7 EERAMSetProtection()

2.8.7.1 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 won't be changed.

2.8.7.2 Prototype

UINT32 EERAMSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8 protectStatus)

2.8.7.3 Arguments

Table 21: Arguments for EERAMSetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
protectStatus	UINT8	The intended protection status value should be written to the EEPROM Protection Register. Refer to section 2.8.6.3 for more details.

2.8.7.4 Return Values

Table 22: Return Values for EERAMSetProtection()

Type	Description	Possible Values
UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_NOEEE FTFx_ERR_CHANGEPROT

2.8.7.5 Comments

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

2.8.8 FlashGetSecurityState()

2.8.8.1 Overview

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

2.8.8.2 Prototype

UINT32 FlashGetSecurityState (PFLASH_SSD_CONFIG PSSDConfig, UINT8* securityState)

2.8.8.3 Arguments

Table 23: Arguments for FlashGetSecurityState()

Argument	Type	Description

PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
securityState	UINT8*	To return the current security status code. • 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.

2.8.8.4 Return Values

Table 24: Return Values for FlashGetSecurityState()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.8.5 Comments

None.

2.8.9 FlashSecurityBypass()

2.8.9.1 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.

2.8.9.2 Prototype

```
UINT32 FlashSecurityBypass (PFLASH_SSD_CONFIG PSSDConfig, \
                            UINT8* keyBuffer, \
                            pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.9.3 Arguments

Table 25: Arguments for FlashSecurityBypass()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
keyBuffer	UINT8*	Point to the user buffer containing the backdoor key.
FlashComma-ndSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.9.4 Return Values

Table 26: Return Values for FlashSecurityBypass()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.9.5 Comments

None.

2.8.10 FlashEraseAllBlock()

2.8.10.1 Overview

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

2.8.10.2 Prototype

**UINT32 FlashEraseAllBlock (PFLASH_SSD_CONFIG PSSDConfig, \
pFLASHCOMMANDSEQUENCE FlashCommandSequence)**

2.8.10.3 Arguments

Table 27: Arguments for FlashEraseAllBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.10.4 Return Values

Table 28: Return Values for FlashEraseAllBlock()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0 FTFx_ERR_ACCERR

2.8.10.5 Comments

None.

2.8.11 FlashEraseBlock()

2.8.11.1 Overview

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

2.8.11.2 Prototype

**UINT32 FlashEraseBlock (PFLASH_SSD_CONFIG PSSDConfig, \
UINT32 destination, \
pFLASHCOMMANDSEQUENCE FlashCommandSequence)**

2.8.11.3 Arguments

Table 29: Arguments for FlashEraseBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended erase operation.

FlashComm-andSequence	pFLASHCOMMANDSE-QUENCE	Pointer to the flash command sequence function.

2.8.11.4 Return Values

Table 30: Return Values for FlashEraseBlock()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.8.11.5 Comments

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

2.8.12 FlashEraseSector()

2.8.12.1 Overview

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

2.8.12.2 Prototype

**UINT32 FlashEraseSector(PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT32 size, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.12.3 Arguments

Table 31: Arguments for FlashEraseSector()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Address in the first sector to be erased.
size	UINT32	Size to be erased in bytes. It is used to determine number of sectors to be erased.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.12.4 Return Values

Table 32: Return Values for FlashEraseSector()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_SIZE FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.8.12.5 Comments

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

2.8.13 FlashEraseSuspend()

2.8.13.1 Overview

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

2.8.13.2 Prototype

UINT32 FlashEraseSuspend (PFLASH_SSD_CONFIG PSSDConfig)

2.8.13.3 Arguments

Table 33: Arguments for FlashEraseSuspend()

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

2.8.13.4 Return Values

Table 34: Return Values for FlashEraseSuspend()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.13.5 Comments

None.

2.8.14 FlashEraseResume()

2.8.14.1 Overview

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

2.8.14.2 Prototype

UINT32 FlashEraseResume (PFLASH_SSD_CONFIG PSSDConfig)

2.8.14.3 Arguments

Table 35: Arguments for FlashEraseResume()

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

2.8.14.4 Return Values

Table 36: Return Values for FlashEraseResume()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.14.5 Comments

None.

2.8.15 FlashProgramSection()

2.8.15.1 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.

2.8.15.2 Prototype

**UINT32 FlashProgramSection (PFLASH_SSD_CONFIG PSSDConfig, \
UINT32 destination, \
UINT16 Number, \
pFLASHCOMMANDSEQUENCE FlashCommandSequence)**

2.8.15.3 Arguments

Table 37: Arguments for FlashProgramSection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended program operation.
Number	UINT16	Number of alignment unit to be programmed. Refer to associate reference manual to get correct value of this alignment constrain.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.15.4 Return Values

Table 38: Return Values for FlashProgramSection()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0 FTFx_ERR_RAMRDY

2.8.15.5 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.

2.8.16 FlashProgram ()

2.8.16.1 Overview

This API is used to program 4 consecutive bytes (for program long word command) and 8 consecutive bytes (for program phrase command) on P-flash or D-Flash block.

2.8.16.2 Prototype

**UINT32 FlashProgram(PFLASH_SSD_CONFIG PSSDConfig, **

**UINT32 destination, **
**UINT32 size, **
**UINT8* pData, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.16.3 Arguments

Table 39: Arguments for FlashProgramLongword()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended program operation.
size	UINT32	Size in byte to be programmed
pData	UINT8*	Pointer of source address from which data has to be taken for program operation.
FlashComm-andSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.16.4 Return Values

Table 40: Return Values for FlashProgramLongword()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_SIZE FTFx_ERR_MGSTAT0

2.8.16.5 Comments

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

2.8.17 FlashCheckSum()

2.8.17.1 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.

2.8.17.2 Prototype

**UINT32 FlashCheckSum (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT32 size, **
UINT32* pSum)

2.8.17.3 Arguments

Table 41: Arguments for FlashCheckSum()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address of the Flash range to be summed

size	UINT32	Size in byte of the flash range to be summed
PSum	UINT32 *	To return the sum value

2.8.17.4 Return Values

Table 42: Return Values for FlashCheckSum()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_RANGE

2.8.17.5 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.

2.8.18 FlashVerifyAllBlock()

2.8.18.1 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.

2.8.18.2 Prototype

**UINT32 FlashVerifyAllBlock (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT8 marginLevel, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.18.3 Arguments

Table 43: Arguments for FlashVerifyAllBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
marginLevel	UINT8	Read Margin Choice as follows: • marginLevel = 0x0: use 'Normal' read level • marginLevel = 0x1: use the 'User' read • marginLevel = 0x2: use the 'Factory' read
FlashComm-andSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.18.4 Return Values

Table 44: Return Values for FlashVerifyAllBlock()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.18.5 Comments

None.

2.8.19 FlashVerifyBlock()

2.8.19.1 Overview

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

2.8.19.2 Prototype

**UINT32 FlashVerifyBlock(PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT8 marginLevel, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.19.3 Arguments

Table 45: Arguments for FlashVerifyBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended verify operation.
marginLevel	UINT8	Read Margin Choice as follows: • marginLevel = 0x0: use 'Normal' read level • marginLevel = 0x1: use the 'User' read • marginLevel = 0x2: use the 'Factory' read
FlashComm-andSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.19.4 Return Values

Table 46: Return Values for FlashVerifyBlock()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.19.5 Comments

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

2.8.20 FlashVerifySection()

2.8.20.1 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.

2.8.20.2 Prototype

**UINT32 FlashVerifySection (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT16 Number, **
**UINT8 marginLevel, **

pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.20.3 Arguments

Table 47: Arguments for FlashVerifySection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended verify operation.
Number	UINT16	Number of alignment unit to be verified. Refer to corresponding reference manual to get correct information of alignment constrain.
marginLevel	UINT8	Read Margin Choice as follows: • marginLevel = 0x0: use 'Normal' read level • marginLevel = 0x1: use the 'User' read • marginLevel = 0x2: use the 'Factory' read
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.20.4 Return Values

Table 48: Return Values for FlashVerifySection()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.20.5 Comments

None.

2.8.21 FlashReadOnce()

2.8.21.1 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.

2.8.21.2 Prototype

```
UINT32 FlashReadOnce (PFLASH_SSD_CONFIG PSSDConfig, \
                      UINT8 recordIndex,\
                      UINT8* pDataArray, \
                      pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.21.3 Arguments

Table 49: Arguments for FlashReadOnce()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
recordIndex	UINT8	The record index will be read. It can be from 0x0 to 0x7 or from 0x0 to 0xF according to specific derivative.

pDataArray	UINT8*	Pointer to the array to return the data read by the read once command.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.21.4 Return Values

Table 50: Return Values for FlashReadOnce()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.21.5 Comments

None.

2.8.22 FlashProgramOnce()

2.8.22.1 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.

2.8.22.2 Prototype

```
UINT32 FlashProgramOnce (PFLASH_SSD_CONFIG PSSDConfig, \
                         UINT8 recordIndex,\
                         UINT8* pDataArray, \
                         pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.22.3 Arguments

Table 51: Arguments for FlashProgramOnce()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
recordIndex	UINT8	Refer to section 2.8.21.3.
pDataArray	UINT8*	Pointer to the array from which data will be taken for program once command.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.22.4 Return Values

Table 52: Return Values for FlashProgramOnce()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.22.5 Comments

None.

2.8.23 FlashProgramCheck()

2.8.23.1 Overview

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

2.8.23.2 Prototype

```
UINT32 FlashProgramCheck (PFLASH_SSD_CONFIG PSSDConfig, \
                          UINT32 destination, \
                          UINT32 size, \
                          UINT8* pExpectedData, \
                          UINT32* pFailAddr, \
                          UINT8 marginLevel, \
                          pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.23.3 Arguments

Table 53: Arguments for FlashProgramCheck()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended program check operation.
pExpectedData	UINT8*	The pointer to the expected data.
pFailAddr	UINT32*	Returned the first aligned failing address.
marginLevel	UINT8	Read margin choice as follows: • marginLevel = 0x1: read at 'User' margin 1/0 level. • marginLevel = 0x2: read at 'Factory' margin 1/0 level.
FlashComman-dSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.23.4 Return Values

Table 54: Return Values for FlashProgramCheck()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.23.5 Comments

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

2.8.24 FlashReadResource()

2.8.24.1 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.

2.8.24.2 Prototype

**UINT32 FlashReadResource (PFLASH_SSD_CONFIG PSSDConfig, \
UINT32 destination, \
UINT8* pDataArray, \
pFLASHCOMMANDSEQUENCE FlashCommandSequence)**

2.8.24.3 Arguments

Table 55: Arguments for FlashReadResource()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended read operation.
pDataArray	UINT8*	Pointer to the data returned by the read resource command.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.24.4 Return Values

Table 56: Return Values for FlashReadResource()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.24.5 Comments

None.

2.8.25 DEFlashPartition()

2.8.25.1 Overview

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

2.8.25.2 Prototype

**UINT32 DEFlashPartition (PFLASH_SSD_CONFIG PSSDConfig, \
UINT8 EEEDataSizeCode, \
UINT8 DEPartitionCode, \
pFLASHCOMMANDSEQUENCE FlashCommandSequence)**

2.8.25.3 Arguments

Table 57: Arguments for DEFlashPartition()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
EEEDataSizeCode	UINT8	EEPROM Data Size Code. Refer to associate reference manual for more details.
DEPartitionCode	UINT8	FlexNVM Partition Code. Refer to associate reference manual for more details.
FlashCommandS-equence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.25.4 Return Values

Table 58: Return Values for DEFlashPartition()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.25.5 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.

2.8.26 SetEEEEnable()

2.8.26.1 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.

2.8.26.2 Prototype

UINT32 SetEEEEnable (PFLASH_SSD_CONFIG PSSDConfig, UINT8 EEEEnable)

2.8.26.3 Arguments

Table 59: Arguments for SetEEEEnable()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
EEEEnable	UINT8	FlexRam function control code. It can be: - 0xFF: make FlexRam available for RAM. - 0x00: make FlexRam available for EEPROM.

2.8.26.4 Return Values

Table 60: Return Values for SetEEEEnable()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.26.5 Comments

None.

2.8.27 EEEWrite()

2.8.27.1 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.

2.8.27.2 Prototype

```
UINT32 EEEWrite (PFLASH_SSD_CONFIG PSSDConfig, \
                 UINT32 destination, \
                 UINT32 size, \
                 UINT8* pData)
```

2.8.27.3 Arguments

Table 61: Arguments for EEEWrite()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
destination	UINT32	Start address for the intended write operation.
size	UINT32	Size in byte to be written.
pData	UINT8*	Pointer to source address from which data has to be taken for writing operation.

2.8.27.4 Return Values

Table 62: Return Values for EEEWrite()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_RANGE FTFx_ERR_NOEEE FTFx_ERR_PVIOL

2.8.27.5 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.

2.8.28 PFlashSwapCtl ()

2.8.28.1 Overview

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

2.8.28.2 Prototype

```
UINT32 PFlashSwapCtl(PFLASH_SSD_CONFIG pSSDConfig,
                     UINT32 addr,
                     UINT8 swapcmd,
                     UINT8* pCurrentSwapMode,
                     UINT8* pCurrentSwapBlockStatus,
```

UINT8* pNextSwapBlockStatus,
pFLASHCOMMANDSEQUENCE pFlashCommandSequence)

2.8.28.3 Arguments

Table 63: Arguments for PFlashSwapCtl ()

Argument	Type	Description
pSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
addr	UINT32	Address of swap indicator.
swapcmd	UINT8	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*	Current Swap Mode: 0x00 - Uninitialized 0x01 - Ready 0x02 - Update 0x03 - Update-Erased 0x04 - Complete
pCurrentSwapBlockStatus	UINT8*	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*	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
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.28.4 Return Values

Table 64: Return Values for PFlashSwapCtl ()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.28.5 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.

2.8.29 PFlashSwap()

2.8.29.1 Overview

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

2.8.29.2 Prototype

**UINT32 PFlashSwap(PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 flashAddress, **
**PFLASH_SWAP_CALLBACK pSwapCallback, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.29.3 Arguments

Table 65: Arguments for PFlashSwap()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer mentioned in section 2.3.
flashAddress	UINT32	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.
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.29.4 Return Values

Table 66: Return Values for PFlashSwap()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_MGSTAT0

2.8.29.5 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()'*. If user wants to stop at each intermediate swap state, just needs to set return value of this callback function to FALSE. In this case, he needs to erase the non-active swap indicator in somewhere of user's application code. If user wants to complete swap process within a single call, just needs to set return value of this function to TRUE. And in this case, he needs to erase the non-active swap indicator within this swap call back function based on the *'currentSwapMode'* input parameter.
In addition, if user does not want to use the swap call back parameter, just use NULL_CALLBACK as a null pointer. In a such situation, the *'PFlashSwap()'* will operate as in case setting return value of *'pSwapCallBack'* to FALSE.
Below is an example to show how to implement a swap callback:

```
BOOL PFlashSwapCallback(UINT8 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.

2.8.30 RelocateFunction()

2.8.30.1 Overview

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

2.8.30.2 Prototype

UINT32 RelocateFunction(UINT32 dest, UINT32 size, UINT32 src)

2.8.30.3 Arguments

Table 67: Arguments for RelocateFunction ()

Argument	Type	Description
dest	UINT32	Destination address where you want to place the function
size	UINT32	Size of the function
src	UINT32	Address of the function will be relocated

2.8.30.4 Return Values

Table 68: Return Values for RelocateFunction ()

Type	Description
UINT32	Relocated address of the function

2.8.30.5 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.

2.9 SSD Internal Functions

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

2.9.1 FlashCommandSequence()

2.9.1.1 Overview

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

2.9.1.2 Prototype

UINT32 FlashCommandSequence (PFLASH_SSD_CONFIG PSSDConfig)

2.9.1.3 Arguments

Table 69: Arguments for FlashCommandSequence()

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

2.9.1.4 Return Values

Table 70: Return Values for FlashCommandSequence()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_PVIOL FTFx_ERR_MGSTAT0

2.9.1.5 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 71: 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 72: 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.
- 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's recommend 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.

5 APPENDIX A: PERFORMANCE DATA

5.1 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 73: 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	0
Cfg3	PK21DX128	FTFx_KX_128K_64K_4K_2K_2K	IAR	0
Cfg4	PK21DX128	FTFx_KX_128K_64K_4K_2K_2K	CW	0
Cfg5	MCF51JF128	FTFx_JX_128K_32K_2K_1K_1K	CW	0
Cfg6	PC56F84789	FTFx_DX_256K_32K_2K_2K_1K	CW	0
Cfg7	PKL25Z128	FTFx_LX_128K_0K_0K_1K_0K	CW	0
Cfg8	PKL25Z128	FTFx_LX_128K_0K_0K_1K_0K	IAR	0

Table 74: 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	0x34	0x60	0x6e	0x30	n/a	n/a
DFlashGetProtection	n/a	n/a	0x24	0x48	0x46	0x23	n/a	n/a
DFlashSetProtection	n/a	n/a	0x34	0x60	0x5a	0x2f	n/a	n/a
EEEWrite	n/a	n/a	0x12a	0xf0	0x13c	0x83	n/a	n/a
EERAMGetProtection	n/a	n/a	0x22	0x54	0x4e	0x3f	n/a	n/a
EERAMSetProtection	n/a	n/a	0x46	0x6c	0x90	0x6b	n/a	n/a
FlashCheckSum	0x78	0xb8	0x88	0xd4	0x10e	0x80	0xa4	0x7a
FlashCommandSequence	0x36	0x58	0x36	0x58	0x60	0x2f	0x58	0x36
FlashEraseAllBlock	0x24	0x40	0x24	0x40	0x4c	0x20	0x3c	0x2a
FlashEraseBlock	0x5e	0xa8	0x7a	0xd4	0x10e	0x7a	n/a	n/a
FlashEraseResume	0x24	0x70	0x24	0x70	0x2e	0x19	0x84	0x24
FlashEraseSector	0x8c	0xe8	0xae	0x118	0x166	0x9c	0xd4	0x8c
FlashEraseSuspend	0x28	0x60	0x28	0x60	0x50	0x24	0x58	0x2c
FlashGetSecurityState	0x38	0x60	0x38	0x60	0x68	0x30	0x60	0x3e
FlashInit	0x16	0x34	0x20e	0x2bc	0x332	0x1ae	0x28	0x18
FlashProgramCheck	0xbe	0x120	0xee	0x16c	0x1d0	0xd6	0x11c	0xb4
FlashProgram	0xaa	0x110	0xc6	0x13c	0x18e	0xb6	0x108	0xa6

FlashProgramOnce	0x56	0x80	0x56	0x80	0x90	0x47	0x94	0x60
FlashProgramSection	0x74	0xe4	0x92	0x110	0x14e	0x9a	n/a	n/a
FlashReadOnce	0x60	0x88	0x60	0x88	0x8e	0x4e	0x9c	0x4c
FlashReadResource	0x94	0xe4	0xb0	0x110	0x156	0x9f	0xec	0x86
FlashSecurityBypass	0x54	0x70	0x54	0x70	0x94	0x41	0x7c	0x94
FlashVerifyAllBlock	0x2a	0x50	0x2a	0x50	0x5e	0x28	0x54	0x2a
FlashVerifyBlock	0x62	0xb4	0x80	0xdc	0x11a	0x80	n/a	n/a
FlashVerifySection	0x70	0xd4	0x8e	0xfc	0x142	0x93	0xc8	0x70
PFlashGetProtection	0x40	0x7c	0x40	0x7c	0x72	0x3f	0x68	0x3e
PFlashSetProtection	0x6a	0x0c	0x6a	0xc4	0xcc	0x6b	0xd8	0x6e
PFlashSwapCtl	0x6c	0xd0	n/a	n/a	n/a	n/a	n/a	n/a
PFlashSwap	0x100	0x138		n/a	n/a	n/a	n/a	n/a
SetEEEEnable	n/a	n/a	0x2a	0x50	0x5e	0x28	n/a	n/a
WaitEEWriteToFinish (within EEEWrite)	n/a	n/a	0xa0	0xfc	0x114	0x7e	n/a	n/a

n/a: not available

5.2 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.

<u>Note:</u> *The timing is collected when code is executed from ram*

Table 75: 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 76: Timing for different configurations

Function/condition \ Derivative			Cfg1 (ms)	Cfg2 (ms)	Cfg3 (ms)	Cfg4 (ms)	Cfg5 (ms)
FlashEraseAllBlock	EEPROM enabled		n/a	727.730	n/a	143.900	107.074
	EEPROM disabled		46.549	742.698	196.017	142.035	110.446
EEWrite			n/a	0.844	n/a	0.539	0.476
FlashVerifyAllBlock	Normal margin		1.467	3.226	1.494	1.544	1.479
	User margin		1.517	3.365	1.547	1.638	1.515
	Factory margin		1.1527	3.365	1.548	1.643	1.515
FlashEraseBlock	P flash		n/a	182.693	106.093	30.553	76.625
	D flash		n/a	182.676	n/a	1.403	27.544
FlashVerifyBlock	P flash	Normal margin	n/a	0.829	1.492	1.560	1.452
		User margin	n/a	0.860	1.520	1.590	1.463
		Factory margin		0.859	1.521	1.591	1.462
	D flash	Normal margin	n/a	0.823	n/a	0.407	0.382
		User margin	n/a	0.824	n/a	0.414	0.381
		Factory margin		0.823	n/a	0.414	0.381

FlashProgram	P flash		0.066	0.088	0.076	0.068	0.087
	D flash		n/a	0.079	n/a	0.068	0.059
FlashProgramCheck	P flash	User margin	0.060	0.157	0.069	0.067	0.044
	D flash	Factory margin	0.060	0.147	0.069	0.068	0.044
CheckSum	P flash		0.003	0.002	0.002	0.007	0.003
	D flash		n/a	0.002	n/a	0.007	0.004
FlashEraseSector	P flash		8.255	6.808	7.331	6.499	6.154
	D flash		n/a	6.793	n/a	6.515	6.145
FlashProgramSection	P flash		n/a	0.275	0.128	0.083	0.156
	D flash		n/a	0.175	n/a	0.083	0.114
FlashVerifySection	P flash	Normal margin	0.035	0.054	0.045	0.045	0.040
		User margin	0.061	0.083	0.072	0.072	0.049
		Factory margin	0.061	0.083	0.072	0.072	0.049
	D flash	Normal margin	n/a	0.049	n/a	0.045	0.038
		User margin	n/a	0.078	n/a	0.072	0.047
		Factory margin	n/a	0.078	n/a	0.072	0.047

n/a: not available

5.3 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 77: Callback time period for different configurations

Function \ Derivative	Cfg1 (us)	Cfg2 (us)	Cfg3 (us)	Cfg4 (us)	Cfg5 (us)
FlashCheckSum	2.500	5.792	8.560	12.833	0.800
APIs using flash command	0.650	1.625	2.560	1.583	0.140

6 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 78: 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 K	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 K	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 K	512	512	16	4	4	MK60FX512xxx12 MK61FX512xxx12 MK70FX512xxx12 MK70FX512xxx15 MK10FX512xxx12 MK20FX512xxx12	MK70FX512xxx15
4	FTFx_KX_1024K_0K_16K_4K_0K	Kinetis K	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 K	32	0	2	1	0	MK10DN32xxx5 MK20DN32xxx5	n/a
6	FTFx_KX_32K_32K_2K_1K_1K	Kinetis K	32	32	2	1	1	MK10DX32xxx5 MK20DX32xxx5	n/a
7	FTFx_KX_64K_0K	Kinetis K	64	0	2	1	0	MK10DN64xxx5	n/a

	_2K_1K_0K							MK20DN64xxx5	
8	FTFx_KX_64K_32 K_2K_1K_1K	Kinetis K	64	32	2	1	1	MK10DX64xxx5 MK20DX64xxx5	n/a
9	FTFx_KX_128K_0 K_2K_1K_0K	Kinetis K	128	0	2	1	0	MK10DN128xxx5 MK20DN128xxx5	MK20DN1 28xxx5
10	FTFx_KX_128K_32 K_2K_1K_1K	Kinetis K	128	32	2	1	1	MK10DX128xxx5 MK20DX128xxx5	n/a
11	FTFx_KX_64K_32 K_2K_2K_1K	Kinetis K	64	32	2	2	1	MK10DX64xxx7 MK20DX64xxx7 MK30DX64xxx7 MK40DX64xxx7	n/a
12	FTFx_KX_128K_32 K_2K_2K_1K	Kinetis K	128	32	2	2	1	MK10DX128xxx7 MK20DX128xxx7 MK30DX128xxx7 MK40DX128xxx7 MK50DX128xxx7 MK51DX128xxx7	n/a
13	FTFx_KX_256K_32 K_2K_2K_1K	Kinetis K	256	32	2	2	1	MK10DX256xxx7 MK20DX256xxx7 MK30DX256xxx7 MK40DX256xxx7 MK50DX256xxx7 MK51DX256xxx7	MK40DX2 56xxx7
14	FTFx_KX_1024K_0 K_4K_4K_0K	Kinetis K	102 4	0	4	4	0	MK21FN1M0xxx10 MK22FN1M0xxx10	MK21FN1 M0xxx10
15	FTFx_KX_512K_12 8K_4K_4K_4K	Kinetis K	512	128	4	4	4	MK21FX512xxx10 MK22FX512xxx10	MK21FX5 12xxx10
16	FTFx_KX_256K_64 K_4K_2K_2K	Kinetis K	256	64	4	2	2	MK21DX256xxx5	
17	FTFx_KX_128K_64 K_4K_2K_2K	Kinetis K	128	64	4	2	2	MK21DX128xxx5	MK21DX1 28xxx5 MK20DX1 28xxx5
18	FTFx_KX_256K_0 K_4K_2K_0K	Kinetis K	256	0	4	2	0	MK51DN256xxx10 MK60DN256xxx10	n/a
19	FTFx_KX_128K_12 8K_4K_2K_2K	Kinetis K	128	128	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_32K_2K_2K_1K	Nevis2	96	32	2	2	1	MC56F84451 MC56F84452 MC56F84550 MC56F84533	n/a
23	FTFx_NX_64K_32K_2K_2K_1K	Nevis2	64	32	2	2	1	MC56F84441 MC56F84442 MC56F84540 MC56F84543	n/a
24	FTFx_LX_128K_0K_0K_1K_0K	Kinetis L	128	0	0	1	0	MKL15Z128xxx4 MKL25Z128xxx4 MKL46Z128xxx4	MKL25Z128xxx4
25	FTFx_LX_64K_0K_0K_1K_0K	Kinetis L	64	0	0	1	0	MKL14Z64xxx4 MKL15Z64xxx4 MKL24Z64xxx4 MKL25Z64xxx4	n/a
26	FTFx_LX_32K_0K_0K_1K_0K	Kinetis L	32	0	0	1	0	MKL05Z32xxx4 MKL04Z32xxx4 MKL14Z32xxx4 MKL24Z32xxx4 MKL25Z32xxx4	MKL05Z32xxx4
27	FTFx_LX_16K_0K_0K_1K_0K	Kinetis L	16	0	0	1	0	MKL05Z16xxx4 MKL04Z16xxx4 MKL02Z16xxx4	MKL02Z16xxx4
28	FTFx_LX_8K_0K_0K_1K_0K	Kinetis L	8	0	0	1	0	MKL05Z8xxx4 MKL04Z8xxx4	n/a
29	FTFx_LX_256K_0K_0K_1K_0K	Kinetis L	256	0	0	1	0	MKL46Z256xxx4	MKL46Z256xxx4
30	FTFx_AX_64K_0K_0K_1K_0K	Anguillar Silver	64	0	0	1	0	MC56F82743 MC56F82744 MC56F82746 MC56F82748	MC56F82748
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_32K_2K_1K_1K	Coldfire	128	32	2	1	1	MCF51JF128 MCF51JU128	MCF51JF128
35	FTFx_CX_64K_32K_2K_1K_1K	Coldfire	64	32	2	1	1	MCF51JF64 MCF51JU64	n/a
36	FTFx_CX_32K_32K_2K_1K_1K	Coldfire	32	32	2	1	1	MCF51JF32 MCF51JU32	n/a
37	FTFx_CX_256K_32K_2K_1K_1K	Coldfire	256	32	2	1	1	MCF51JG256 MCF51FD256	MCF51JG256

38	FTFx_MX_64K_0K_0K_1K_0K	Kinetis L	64	0	0	1	0	MKM13Z64xxx5 MKM14Z64xxx5 MKM32Z64xxx5 MKM33Z64xxx5	n/a
39	FTFx_MX_128K_0K_0K_1K_0K	Kinetis L	128	0	0	1	0	MKM14Z128xxx5 MKM33Z128xxx5 MKM34Z128xxx5	MKM34Z128xxx5
40	FTFx_KX_512K_0K_0K_2K_0K	Kinetis K	512	0	0	2	0	MK22FN512xxx8	MK22FN512xxx8
41	FTFx_KX_256K_0K_0K_2K_0K	Kinetis K	512	0	0	2	0	MK22FN256xxx8	