Standard Software Driver for C90TFS/FTFx Flash

User's Manual

© Copyright Freescale Semiconductor 2012.
All Rights Reserved

REVISION LIST

Version No.	Date	Author	Description
1.0	06-07-2010	FPT Team	Initial Version
1.1	09-22-2010	FPT Team	Update for the FTFx_KX_256K_256K_4K_2K_2K derivative
1.2	09-30-2010	FPT Team	Update performance result for the FTFx_KX_256K_256K_4K_2K_2K derivative
1.3	02-24-2011	FPT Team	Update for the FTFx_KX_512K_0K_0K_2K_0K derivative
1.4	03-16-2011	FPT Team	Update for the FTFx_JX_128K_32K_2K_1K_1K derivative
1.5	05-05-2011	FPT Team	Update for the FTFx_FX_256K_32K_2K _1K_1K derivative Update callback time for FTFx_JX_128K_32K_2K_1K_1K
1.6	10-31-2011	FPT Team	Update for the FTFx_KX_512K_512K_16K_4K_4K
1.7	11-17-2011	FPT Team	Update for the FTFx_KX_1024K_0K_16K_4K_0K
1.8	12-27-2011	FPT Team	Update list of derivative in table 7. Add comment for FlashReadResource() function.
1.9	02-21-2012	FPT Team	Update to support more Kinetis derivatives. Update PFlashSwap function prototype. Add performance/code size for FTFx_KX_128K_32K_2K_1K_1K derivative (K20).

TABLE OF CONTENTS

1 INTRODUCTION 1
1.1 Document Overview 1
1.2 System Overview 1
1.3 Features 1
1.4 System Requirements 2
1.5 Documentation References 3
1.6 Terms 3
1.7 Acronyms 4
2 API SPECIFICATION 5
2.1 General Overview 6
2.2 General Type Definitions 6
2.3 Configuration parameters 7
2.4 Configuration Macros 7
2.5 CallBack function 8
2.6 Return Codes 9
2.7 Macros 9
2.8 SSD Functions 9
2.8.1 FlashInit() 9
2.8.1.1 Overview 9
2.8.1.2 Activation 9
2.8.1.3 Prototype 10
2.8.1.4 Arguments 10
2.8.1.5 Return Values 10
2.8.1.6 Troubleshooting 10
2.8.1.7 Affected Registers 10
2.8.1.8 Comments 10
2.8.1.9 Assumptions 10
2.8.2 PFlashGetProtection() 10
2.8.2.1 Overview 10
2.8.2.2 Activation 10
2.8.2.3 Prototype 10
2.8.2.4 Arguments 11
2.8.2.5 Return Values 11
2.8.2.6 Troubleshooting 11
2.8.2.7 Affected Registers 11
2.8.2.8 Comments 11
2.8.2.9 Assumptions 11
2.8.3 PFlashSetProtection() 11
2.8.3.1 Overview 11
2.8.3.2 Activation 11
2.8.3.3 Prototype 11
2.8.3.4 Arguments 11
2.8.3.5 Return Values 12
2.8.3.6 Troubleshooting 12
2.8.3.7 Affected Registers 12
2.8.3.8 Comments 12
2.8.3.9 Assumptions 12
2.8.4 DFlashGetProtection() 12
2.8.4.1 Overview 12
2.8.4.2 Activation 12
2.8.4.3 Prototype 12
2.8.4.4 Arguments 12
2.8.4.5 Return Values 12
2.8.4.6 Troubleshooting 13

2.8.4.7 Affected Registers 13
2.8.4.8 Comments 13
2.8.4.9 Assumptions 13
2.8.5 DFlashSetProtection() 13
2.8.5.1 Overview 13
2.8.5.2 Activation 13
2.8.5.3 Prototype 13
2.8.5.4 Arguments 13
2.8.5.5 Return Values 13
2.8.5.6 Troubleshooting 14
2.8.5.7 Affected Registers 14
2.8.5.8 Comments 14
2.8.5.9 Assumptions 14
2.8.6 EERAMGetProtection() 14
2.8.6.1 Overview 14
2.8.6.2 Activation 14
2.8.6.3 Prototype 14
2.8.6.4 Arguments 14
2.8.6.5 Return Values 14
2.8.6.6 Troubleshooting 14
2.8.6.7 Affected Registers 15
2.8.6.8 Comments 15
2.8.6.9 Assumptions 15
2.8.7 EERAMSetProtection() 15
2.8.7.1 Overview 15
2.8.7.2 Activation 15
2.8.7.3 Prototype 15
2.8.7.4 Arguments 15
2.8.7.5 Return Values 15
2.8.7.6 Troubleshooting 15
2.8.7.7 Affected Registers 16
2.8.7.8 Comments 16
2.8.7.9 Assumptions 16
2.8.8 FlashGetInterruptEnable() 16
2.8.8.1 Overview 16
2.8.8.2 Activation 16
2.8.8.3 Prototype 16
2.8.8.4 Arguments 16
2.8.8.5 Return Values 16
2.8.8.6 Troubleshooting 16
2.8.8.7 Affected Registers 16
2.8.8.8 Comments 17
2.8.8.9 Assumptions 17
2.8.9 FlashSetInterruptEnable() 17
2.8.9.1 Overview 17
2.8.9.2 Activation 17
2.8.9.3 Prototype 17
2.8.9.4 Arguments 17
2.8.9.5 Return Values 17
2.8.9.6 Troubleshooting 17
2.8.9.7 Affected Registers 17
2.8.9.8 Comments 17
2.8.9.9 Assumptions 18
2.8.10 FlashGetSecurityState() 18
2.8.10.1 Overview 18
2.8.10.2 Activation 18

2.8.10.3 Prototype 18
2.8.10.4 Arguments 18
2.8.10.5 Return Values 18
2.8.10.6 Troubleshooting 18
2.8.10.7 Affected Registers 18
2.8.10.8 Comments 18
2.8.10.9 Assumptions 18
2.8.11 FlashCommandSequence() 19
2.8.11.1 Overview 19
2.8.11.2 Activation 19
2.8.11.3 Prototype 19
2.8.11.4 Arguments 19
2.8.11.5 Return Values 19
2.8.11.6 Troubleshooting 19
2.8.11.7 Affected Registers 19
2.8.11.8 Comments 19
2.8.11.9 Assumptions 20
2.8.12 FlashSecurityBypass() 20
2.8.12.1 Overview 20
2.8.12.2 Activation 20
2.8.12.3 Prototype 20
2.8.12.4 Arguments 20
2.8.12.5 Return Values 20
2.8.12.6 Troubleshooting 20
2.8.12.7 Affected Registers 21
2.8.12.8 Comments 21
2.8.12.9 Assumptions 21
2.8.13 FlashEraseAllBlock() 21
2.8.13.1 Overview 21
2.8.13.2 Activation 21
2.8.13.3 Prototype 21
2.8.13.4 Arguments 21
2.8.13.5 Return Values 21
2.8.13.6 Troubleshooting 22
2.8.13.7 Affected Registers 22
2.8.13.8 Comments 22
2.8.13.9 Assumptions 22
2.8.14 FlashEraseBlock() 22
2.8.14.1 Overview 22
2.8.14.2 Activation 22
2.8.14.3 Prototype 22
2.8.14.4 Arguments 22
2.8.14.5 Return Values 23
2.8.14.6 Troubleshooting 23
2.8.14.7 Affected Registers 23
2.8.14.8 Comments 24
2.8.14.9 Assumptions 24
2.8.15 FlashEraseSector() 24
2.8.15.1 Overview 24
2.8.15.2 Activation 24
2.8.15.3 Prototype 24
2.8.15.4 Arguments 24
2.8.15.5 Return Values 24
2.8.15.6 Troubleshooting 24
2.8.15.7 Affected Registers 25
2.8.15.8 Comments 25

2.8.15.9 Assumptions 25
2.8.16 FlashEraseSuspend() 25
2.8.16.1 Overview 25
2.8.16.2 Activation 25
2.8.16.3 Prototype 25
2.8.16.4 Arguments 25
2.8.16.5 Return Values 26
2.8.16.6 Troubleshooting 26
2.8.16.7 Affected Registers 26
2.8.16.8 Comments 26
2.8.16.9 Assumptions 26
2.8.17 FlashEraseResume() 26
2.8.17.1 Overview 26
2.8.17.2 Activation 26
2.8.17.3 Prototype 26
2.8.17.4 Arguments 26
2.8.17.5 Return Values 26
2.8.17.6 Troubleshooting 26
2.8.17.7 Affected Registers 27
2.8.17.8 Comments 27
2.8.17.9 Assumptions 27
2.8.18 FlashProgramSection() 27
2.8.18.1 Overview 27
2.8.18.2 Activation 27
2.8.18.3 Prototype 27
2.8.18.4 Arguments 27
2.8.18.5 Return Values 27
2.8.18.6 Troubleshooting 28
2.8.18.7 Affected Registers 28
2.8.18.8 Comments 28
2.8.18.9 Assumptions 29
2.8.19 FlashProgramLongword() 29
2.8.19.1 Overview 29
2.8.19.2 Activation 29
2.8.19.3 Prototype 29
2.8.19.4 Arguments 29
2.8.19.5 Return Values 29
2.8.19.6 Troubleshooting 29
2.8.19.7 Affected Registers 30
2.8.19.8 Comments 30
2.8.19.9 Assumptions 30
2.8.20 FlashProgramPhrase() 30
2.8.20.1 Overview 30
2.8.20.2 Activation 30
2.8.20.3 Prototype 31
2.8.20.4 Arguments 31
2.8.20.5 Return Values 31
2.8.20.6 Troubleshooting 31
2.8.20.7 Affected Registers 32
2.8.20.8 Comments 32
2.8.20.9 Assumptions 32
2.8.21 FlashCheckSum() 32
2.8.21.1 Overview 32
2.8.21.2 Activation 32
2.8.21.3 Prototype 32
2.8.21.4 Arguments 32

2.8.21.5 Return Values 33
2.8.21.6 Troubleshooting 33
2.8.21.7 Affected Registers 33
2.8.21.8 Comments 33
2.8.21.9 Assumptions 33
2.8.22 FlashVerifyAllBlock() 33
2.8.22.1 Overview 33
2.8.22.2 Activation 33
2.8.22.3 Prototype 33
2.8.22.4 Arguments 33
2.8.22.5 Return Values 34
2.8.22.6 Troubleshooting 34
2.8.22.7 Affected Registers 34
2.8.22.8 Comments 34
2.8.22.9 Assumptions 34
2.8.23 FlashVerifyBlock() 34
2.8.23.1 Overview 34
2.8.23.2 Activation 34
2.8.23.3 Prototype 34
2.8.23.4 Arguments 34
2.8.23.5 Return Values 35
2.8.23.6 Troubleshooting 35
2.8.23.7 Affected Registers 35
2.8.23.8 Comments 35
2.8.23.9 Assumptions 36
2.8.24 FlashVerifySection() 36
2.8.24.1 Overview 36
2.8.24.2 Activation 36
2.8.24.3 Prototype 36
2.8.24.4 Arguments 36
2.8.24.5 Return Values 36
2.8.24.6 Troubleshooting 36
2.8.24.7 Affected Registers 37
2.8.24.8 Comments 37
2.8.24.9 Assumptions 37
2.8.25 FlashReadOnce() 37
2.8.25.1 Overview 37
2.8.25.2 Activation 37
2.8.25.3 Prototype 37
2.8.25.4 Arguments 37
2.8.25.5 Return Values 38
2.8.25.6 Troubleshooting 38
2.8.25.7 Affected Registers 38
2.8.25.8 Comments 38
2.8.25.9 Assumptions 38
2.8.26 FlashProgramOnce() 38
2.8.26.1 Overview 38
2.8.26.2 Activation 38
2.8.26.3 Prototype 39
2.8.26.4 Arguments 39
2.8.26.5 Return Values 39
2.8.26.6 Troubleshooting 39
2.8.26.7 Affected Registers 39
2.8.26.8 Comments 39
2.8.26.9 Assumptions 39
2.8.27 FlashProgramCheck() 40

2.8.27.1 Overview ... 40
2.8.27.2 Activation ... 40
2.8.27.3 Prototype ... 40
2.8.27.4 Arguments ... 40
2.8.27.5 Return Values ... 40
2.8.27.6 Troubleshooting ... 41
2.8.27.7 Affected Registers ... 41
2.8.27.8 Comments ... 41
2.8.27.9 Assumptions ... 41
2.8.28 FlashReadResource() ... 41
2.8.28.1 Overview ... 41
2.8.28.2 Activation ... 41
2.8.28.3 Prototype ... 41
2.8.28.4 Arguments ... 42
2.8.28.5 Return Values ... 42
2.8.28.6 Troubleshooting ... 42
2.8.28.7 Affected Registers ... 42
2.8.28.8 Comments ... 42
2.8.28.9 Assumptions ... 43
2.8.29 DEFlashPartition() ... 43
2.8.29.1 Overview ... 43
2.8.29.2 Activation ... 43
2.8.29.3 Prototype ... 43
2.8.29.4 Arguments ... 43
2.8.29.5 Return Values ... 43
2.8.29.6 Troubleshooting ... 43
2.8.29.7 Affected Registers ... 44
2.8.29.8 Comments ... 44
2.8.29.9 Assumptions ... 44
2.8.30 SetEEEEnable() ... 44
2.8.30.1 Overview ... 44
2.8.30.2 Activation ... 44
2.8.30.3 Prototype ... 44
2.8.30.4 Arguments ... 44
2.8.30.5 Return Values ... 45
2.8.30.6 Troubleshooting ... 45
2.8.30.7 Affected Registers ... 45
2.8.30.8 Comments ... 45
2.8.30.9 Assumptions ... 45
2.8.31 EEEWrite() ... 45
2.8.31.1 Overview ... 45
2.8.31.2 Activation ... 45
2.8.31.3 Prototype ... 46
2.8.31.4 Arguments ... 46
2.8.31.5 Return Values ... 46
2.8.31.6 Troubleshooting ... 46
2.8.31.7 Affected Registers ... 46
2.8.31.8 Comments ... 46
2.8.31.9 Assumptions ... 47
2.8.32 PFlashGetSwapStatus() ... 47
2.8.32.1 Overview ... 47
2.8.32.2 Activation ... 47
2.8.32.3 Prototype ... 47
2.8.32.4 Arguments ... 47
2.8.32.5 Return Values ... 47
2.8.32.6 Troubleshooting ... 48

2.8.32.7 Affected Registers 48
2.8.32.8 Comments 48
2.8.32.9 Assumptions 48
2.8.33 PFlashSwap() 48
2.8.33.1 Overview 48
2.8.33.2 Activation 48
2.8.33.3 Prototype 48
2.8.33.4 Arguments 48
2.8.33.5 Return Values 49
2.8.33.6 Troubleshooting 49
2.8.33.7 Affected Registers 49
2.8.33.8 Comments 50
2.8.33.9 Assumptions 50
APPENDIX A: PERFORMANCE DATA 50
A.1 Code Size and Stack Usage 50
A.2 Write/Erase Times 55
A.3 CallBack Time Period 58

LIST OF TABLES

Table 1: System Requirements 2
Table 2: References 3
Table 3: Terms 3
Table 4: Acronyms 4
Table 5: Type Definitions 6
Table 6: SSD Configuration Structure Field Definition 7
Table 7: Derivative Specific Macro values for C90TFS/FTFx 8
Table 8: Return Codes 9
Table 9: Arguments for FlashInit() 10
Table 10: Return Values for FlashInit() 10
Table 11: Table Registers affected in FlashInit() 10
Table 12: Arguments for PFlashGetProtection() 11
Table 13: Return Values for PFlashGetProtection() 11
Table 14: Registers Affected in PFlashGetProtection() 11
Table 15: Arguments for PFlashSetProtection () 11
Table 16: Return Values for PFlashSetProtection () 12
Table 17: Troubleshooting for PFlashSetProtection() 12
Table 18: Register affected in PFlashSetProtection() 12
Table 19: Arguments for DFlashGetProtection() 12
Table 20: Return Values for DFlashGetProtection() 12
Table 21: Troubleshooting for DFlashGetProtection() 13
Table 22: Register affected in DFlashGetProtection() 13
Table 23: Arguments for DFlashSetProtection() 13
Table 24: Return Values for DFlashSetProtection() 13
Table 25: Troubleshooting for DFlashSetProtection() 14
Table 26: Register affected in DFlashSetProtection() 14
Table 27: Arguments for EERAMGetProtection() 14
Table 28: Return Values for EERAMGetProtection() 14
Table 29: Troubleshooting for EERAMGetProtection() 14
Table 30: Register affected in EERAMGetProtection() 15
Table 31: Arguments for EERAMSetProtection() 15
Table 32: Return Values for EERAMSetProtection() 15
Table 33: Troubleshooting for EERAMSetProtection() 15
Table 34: Register affected in EERAMSetProtection() 16
Table 35: Arguments for FlashGetInterruptEnable() 16
Table 36: Return Values for FlashGetInterruptEnable() 16
Table 37: Register affected in FlashGetInterruptEnable() 16
Table 38: Arguments for FlashSetInterruptEnable() 17
Table 39: Return Values for FlashSetInterruptEnable() 17
Table 40: Register affected in FlashSetInterruptEnable() 17
Table 41: Arguments for FlashGetSecurityState() 18
Table 42: Return Values for FlashGetSecurityState() 18
Table 43: Register affected in FlashGetSecurityState() 18
Table 44: Arguments for FlashCommandSequence() 19
Table 45: Return Values for FlashCommandSequence() 19
Table 46: Registers affected in FlashCommandSequence() 19
Table 47: Arguments for FlashSecurityBypass() 20
Table 48: Return Values for FlashSecurityBypass() 20
Table 49: Troubleshooting for FlashSecurityBypass() 20
Table 50: Register affected in FlashSecurityBypass() 21
Table 51: Arguments for FlashEraseAllBlock() 21
Table 52: Return Values for FlashEraseAllBlock() 21
Table 53: Troubleshooting for FlashEraseAllBlock() 22
Table 54: Registers affected in FlashEraseAllBlock() 22

Table 55: Arguments for FlashEraseBlock() 22
Table 56: Return Values for FlashEraseBlock() 23
Table 57: Troubleshooting for FlashEraseBlock() 23
Table 58: Registers affected in FlashEraseBlock() 23
Table 59: Arguments for FlashEraseSector() 24
Table 60: Return Values for FlashEraseSector() 24
Table 61: Troubleshooting for FlashEraseSector() 25
Table 62: Registers affected in FlashEraseSector() 25
Table 63: Arguments for FlashEraseSuspend() 25
Table 64: Return Values for FlashEraseSuspend() 26
Table 65: Registers affected in FlashEraseSuspend() 26
Table 66: Arguments for FlashEraseResume() 26
Table 67: Return Values for FlashEraseResume() 26
Table 68: Registers affected in FlashEraseResume() 27
Table 69: Arguments for FlashProgramSection() 27
Table 70: Return Values for FlashProgramSection() 27
Table 71: Troubleshooting for FlashProgramSection() 28
Table 72: Registers affected in FlashProgramSection() 28
Table 73: Arguments for FlashProgramLongword() 29
Table 74: Return Values for FlashProgramLongword() 29
Table 75: Troubleshooting for FlashProgramLongword() 30
Table 76: Registers affected in FlashProgramLongword() 30
Table 77: Arguments for FlashProgramPhrase() 31
Table 78: Return Values for FlashProgramPhrase() 31
Table 79: Troubleshooting for FlashProgramPhrase() 31
Table 80: Registers affected in FlashProgramPhrase() 32
Table 81: Arguments for FlashCheckSum() 32
Table 82: Return Values for FlashCheckSum() 33
Table 83: Troubleshooting for FlashCheckSum() 33
Table 84: Arguments for FlashVerifyAllBlock() 33
Table 85: Return Values for FlashVerifyAllBlock() 34
Table 86: Troubleshooting for FlashVerifyAllBlock() 34
Table 87 Registers affected in FlashVerifyAllBlock() 34
Table 88: Arguments for FlashVerifyBlock() 34
Table 89: Return Values for FlashVerifyBlock() 35
Table 90: Troubleshooting for FlashVerifyBlock() 35
Table 91: Registers affected in FlashVerifyBlock() 35
Table 92: Arguments for FlashVerifySection() 36
Table 93: Return Values for FlashVerifySection() 36
Table 94: Troubleshooting for FlashVerifySection() 36
Table 95: Registers affected in FlashVerifySection() 37
Table 96: Arguments for FlashReadOnce() 37
Table 97: Return Values for FlashReadOnce() 38
Table 98: Troubleshooting for FlashReadOnce() 38
Table 99: Registers affected in FlashReadOnce() 38
Table 100: Arguments for FlashProgramOnce() 39
Table 101: Return Values for FlashProgramOnce() 39
Table 102: Troubleshooting for FlashProgramOnce() 39
Table 103: Registers affected in FlashProgramOnce() 39
Table 104: Arguments for FlashProgramCheck() 40
Table 105: Return Values for FlashProgramCheck() 40
Table 106: Troubleshooting for FlashProgramCheck() 41
Table 107: Registers affected in FlashProgramCheck() 41
Table 108: Arguments for FlashReadResource() 42
Table 109: Return Values for FlashReadResource() 42
Table 110: Troubleshooting for FlashReadResource() 42

Table 111: Registers affected in FlashReadResource() 42
Table 112: Arguments for DEFlashPartition() 43
Table 113: Return Values for DEFlashPartition() 43
Table 114: Troubleshooting for DEFlashPartition() 43
Table 115: Registers affected in DEFlashPartition() 44
Table 116: Arguments for SetEEEEnable() 44
Table 117: Return Values for SetEEEEnable() 45
Table 118: Troubleshooting for SetEEEEnable() 45
Table 119: Registers affected in SetEEEEnable() 45
Table 120: Arguments for EEEWrite() 46
Table 121: Return Values for EEEWrite() 46
Table 122: Troubleshooting for EEEWrite() 46
Table 123: Registers affected in EEEWrite() 46
Table 124: Arguments for PFlashGetSwapStatus() 47
Table 125: Return Values for PFlashGetSwapStatus() 47
Table 126: Troubleshooting for PFlashGetSwapStatus() 48
Table 127: Registers affected in PFlashGetSwapStatus() 48
Table 128: Arguments for PFlashSwap() 48
Table 129: Return Values for PFlashSwap() 49
Table 130: Troubleshooting for PFlashSwap() 49
Table 131: Registers affected in PFlashSwap() 49
Table 132: Code Size and Stack Usage for FTFx_KX_256K_256K_4K_2K_2K derivative 50
Table 133: Code Size and Stack Usage for FTFx_KX_512K_0K_0K_2K_0K derivative 51
Table 134: Code Size and Stack Usage for FTFx_JX_128K_32K_2K_1K_1K derivative 52
Table 135: Code Size and Stack Usage for FTFx_FX_256K_32K_2K_1K_1K derivative 53
Table 136: Code Size and Stack Usage for FTFx_KX_512K_512K_16K_4K_4K derivative 53
Table 137: Code Size and Stack Usage for FTFx_KX_128K_32K_2K_1K_1K derivative 54
Table 138: Write/Erase Times for FTFx_KX_256K_256K_4K_2K_2K derivative 55
Table 139: Write/Erase Times for FTFx_KX_512K_0K_0K_2K_0K derivative 55
Table 140: Write/Erase Times for FTFx_JX_128K_32K_2K_1K_1K derivative 56
Table 141: Write/Erase Times for FTFx_FX_256K_32K_2K_1K_1K derivative 56
Table 142: Write/Erase Times for FTFx_KX_512K_512K_16K_4K_4K derivative 57
Table 143: Write/Erase Times for FTFx_KX_128K_32K_2K_1K_1K derivative 57
Table 144: CallBack Time Period for FTFx_KX_256K_256K_4K_2K_2K derivative 58
Table 145: CallBack Time Period for FTFx_KX_512K_0K_0K_2K_0K derivative 58
Table 146: CallBack Time Period for FTFx_JX_128K_32K_2K_1K_1K derivative 59
Table 147: CallBack Time Period for FTFx_FX_256K_32K_2K_1K_1K derivative 59
Table 148: CallBack Time Period for FTFx_KX_512K_512K_16K_4K_4K derivative 60
Table 149: CallBack Time Period for FTFx_KX_128K_32K_2K_1K_1K derivative 60

LIST OF FIGURES

Figure 1 SSD Usage Diagram 5

1 INTRODUCTION

1.1 Document Overview

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

Section 1.2 provides a brief overview of the system for general background knowledge of the project.
Section 1.3 shows the features of the driver.
Section 1.4 details the system requirement for the driver development.
Section 1.5 and Section 1.6 lists the documents referred and terms used in making of this document.
Section 1.7 lists the acronyms used.
Section 2 describes the API specifications. In this section there are many sub sections, which describe the different aspects of the driver.
Section 2.1 provides a general overview of the driver.
Section 2.2 mentions about the type definitions used for the driver.
Section 2.3 and Section 2.4 mention the driver configuration parameters and Configuration Macros respectively.
Section 2.5, Section 2.6 and Section 2.7 describe the Callback notifications, return codes and user configurable macros used for the driver.
Section 2.8 provides the detailed description of standard software Driver function.

1.2 System Overview

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

The C90TFS/FTFx System Software Driver:

- Provides separate APIs to support P-Flash, FlexNVM(may be partitioned as D-Flash and/or E-Flash), FlexRAM(may be used as traditional RAM or as high-endurance EEPROM storage).
- Exposes the hardware features of P-Flash, FlexNVM and FlexRAM memory block in FTFx to achieve high performance. These features include the program, sector erase, block erase, erase verify, program verify etc.
- Supports different modes of operation. It will work on NVM special mode when Export is enabled or background debug mode is enabled. Otherwise, it will work on NVM normal mode.
- Be re-locatable and be easily integrated into a user's application.

1.3 Features

C90TFS/FTFx SSD provides the following features:

- Small code size and high speed program and erase operations
- Ready-to-use demos illustrates the usage of the driver
- Support multi-task via call back function
- Source code releases

- Supports different FTFx derivatives (for example: FTFx_JX_128K_32K_2K_1K_1K, FTFx_KX_256K_256K_4K_2K_2K, FTFx_KX_512K_0K_0K_2K_0K, etc…Refer to table 7 for more details)

1.4 System Requirements

This SSD works on the different derivatives of C90TFS/FTFx. Before using this SSD, user has to provide the specific information to the derivative through a macro of derivative selection.

Table 1: System Requirements

No.	Derivative	Tool Name	Description	Version No
1.	FTFx_KX_256K_256K_4K_2K_2K	IAR Embedded Workbench for ARM (Kickstart)	Development tool	5.4
		TWR-K40X256	Evaluation board	Rev X1
		SEGGER J-Link	Debugger	8.0
2.	FTFx_KX_512K_0K_0K_2K_0K	IAR Embedded Workbench for ARM (Kickstart)	Development tool	6.0
		K60 Evaluation board	Evaluation board	
		SEGGER J-Link	Debugger	8.0
3.	FTFx_JX_128K_32K_2K_1K_1K	CodeWarrior	Development tool	10.0
		MCF51JF128 Evaluation board	Evaluation board	
		P&E	Debugger	
4.	FTFx_FX_256K_32K_2K_1K_1K	CodeWarrior	Development tool	10.0
		MCF51FD256 Evaluation board	Evaluation board	
		P&E	Debugger	
5.	FTFx_KX_512K_512K_16K_4K_4K	CodeWarrior	Development tool	10.1
		TWR_K70FN1M Evaluation board	Evaluation board	
		OSBMD	Debugger	
6.	FTFx_KX_1024K_0K_16K_4K_0K	CodeWarrior	Development tool	10.1
		TWR_K70FN1M Evaluation board	Evaluation board	
		OSBMD	Debugger	
7.	FTFx_KX_32K_0K_0K_1K_0K	CodeWarrior	Development tool	10.2
8.	FTFx_KX_32K_32K_2K_1K_1K	CodeWarrior	Development tool	10.2

9.	FTFx_KX_64K_0K_0K_1K_0K	CodeWarrior	Development tool	10.2
10.	FTFx_KX_64K_32K_2K_1K_1K	CodeWarrior	Development tool	10.2
11.	FTFx_KX_128K_0K_0K_1K_0K	CodeWarrior	Development tool	10.2
12.	FTFx_KX_128K_32K_2K_1K_1K	CodeWarrior	Development tool	10.2
		TWR_K20DX50 Evaluation board	Evaluation board	
		OSBMD	Debugger	
13.	FTFx_KX_64K_32K_2K_2K_1K	CodeWarrior	Development tool	10.2
14.	FTFx_KX_128K_32K_2K_2K_1K	CodeWarrior	Development tool	10.2
15.	FTFx_KX_256K_32K_2K_2K_1K	CodeWarrior	Development tool	10.2

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

1.6 Terms

Table 3: Terms

Term	Definition
Flash Block	A macro within the FTFx module which provides the nonvolatile memory storage. The overall size of the Flash memory provided by the module is adjusted by instantiating more or fewer Flash blocks.
P-Flash	The P-Flash memory provides nonvolatile storage for vectors and code store.
P-Flash Sector	The smallest portion of the P-Flash memory (consecutive addresses) that can be erased.
D/E-Flash Block (FlexNVM)	The D/E-Flash block can be configured to be used as either D-Flash memory or E-Flash memory, or a combination of both.
D-Flash Sector	The D-Flash sector is the smallest portion of the D-Flash memory that can be erased.

EERAM (FlexRAM)	The EERAM refers to a RAM, dedicated to the FTFx module that can be configured to store EEE data or to be used as traditional RAM. When configured for EEE, valid writes to the EERAM will generate a new EEE Data Record stored in the E-Flash memory.
Word	A word is 16 bits of data with an aligned word having byte-address[0] = 0
Phrase	64 bits of data with an aligned phrase having byte-address [2:0] = 000.
Double-Phrase	128 bits of data with an aligned phrase having byte-address[3:0] = 0000.
Longword	A longword is 32 bits of data with an aligned longword having byte-address[1:0] = 00
EEE	Using a built-in fling system, the FTFx module emulates the characteristics of an EEPROM by effectively providing a high-endurance, byte-writeable (program and erase) NVM.
Section program buffer	Lower half of the EERAM allocated for storing large amounts of data for programming via the Program Section command

1.7 Acronyms

Table 4: Acronyms

Abbreviation	Complete Name
API	Application Programmer Interface
SRS	Software Requirement Specification
SSD	Standard Software Driver
EEE	EEPROM Emulation
RWW	Read While Write

2 API SPECIFICATION

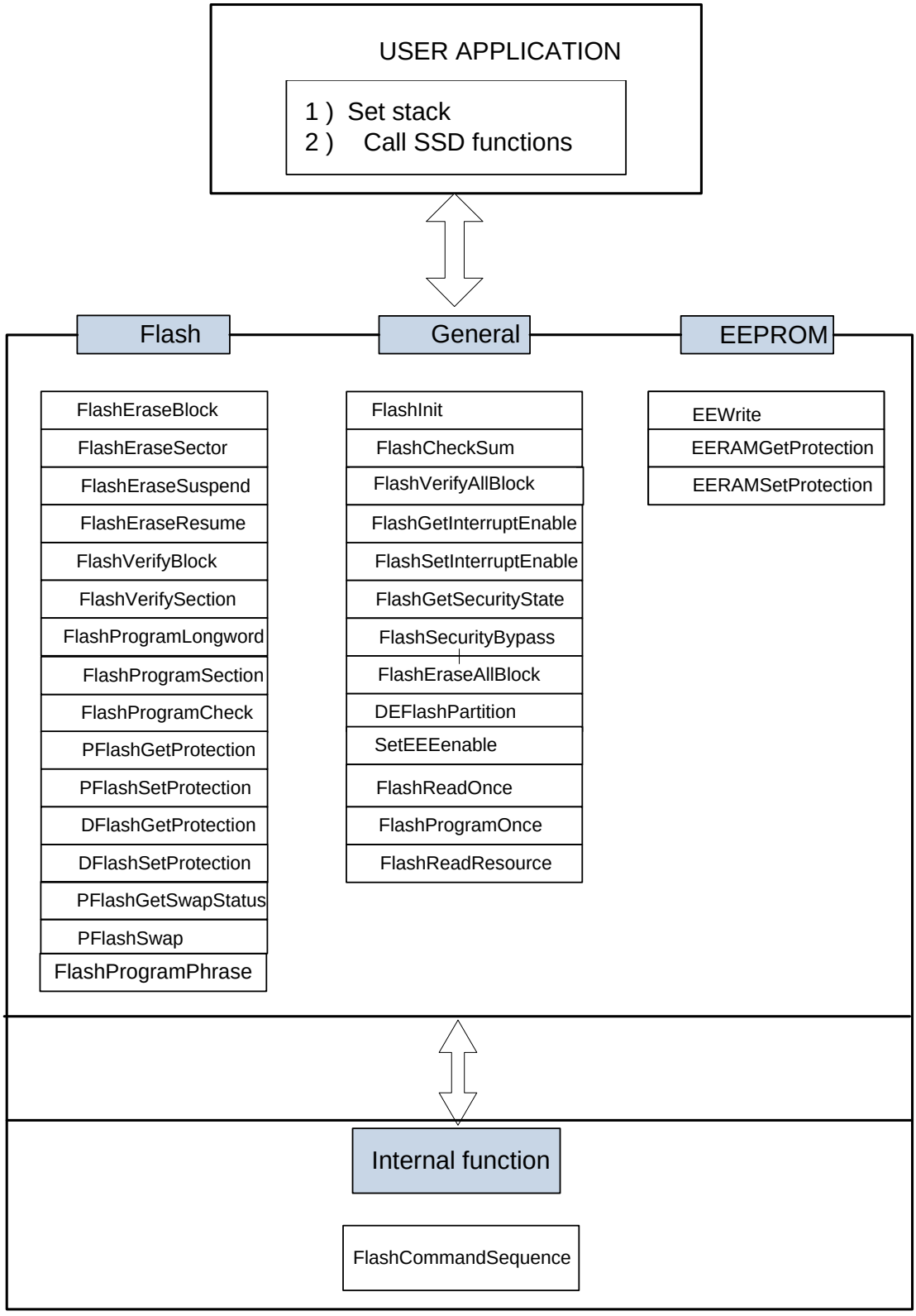


Figure 1 SSD Usage Diagram

The SSD will have 3 types of function based on their functionalities. These are:

- Flash functions – These functions are used for P-Flash or D-Flash operations.
- General functions – These functions are used to provide some general functionality for all EEPROM, P-Flash and D-Flash.
- EEPROM function – These functions are used only for the EEPROM operation.

All Flash functions require the input of SSD Flash configuration structure. It includes both static Flash module parameters and driver configuration parameters.

Note:
PFlashGetSwapStatus() and PFlashSwap() are available on derivatives supporting swap feature only.
FlashProgramLongWord() is available on derivatives supporting program long word feature only.
FlashProgramPhrase() is available on derivatives supporting program phrase feature only.
The below functions are not available on derivatives that don't have FlexNVM block:

- DEFlashPartition
- SetEEEEnable
- EEEWrite
- EERAMSetProtection
- EERAMGetProtection
- DFlashSetProtection
- DFlashGetProtection

2.1 General Overview

The C90TFS/FTFx SSD provides APIs to handle the erase, program, erase verify and program verify operations on the P-Flash, D-Flash and EERAM. Apart from these, it also provides the feature for protection settings, security bypass and interrupts enabling. This SSD also provides the feature for Check sum and supports some of the controller specific commands e.g. Program Once, Read Once... The SSD has a total of 32 APIs including '*FlashCommandSequence*()', which is used for internal operations only.

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

Derived type	Size	C language type Description
VUINT32	32-bits	volatile unsigned long

2.3 Configuration parameters

The configuration parameters used for the SSD are given in this section. The configuration parameters are handled as structure. The structure (FLASH_SSD_CONFIG) includes the static parameters for FTFx which are chip-dependent. The user should correctly initialize the fields including *ftfxRegBase, PFlashBlockBase, PFlashBlockSize, DFlashBlockBase, EERAMBlockBase, EERAMBlockSize, BDMEnable 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 base address of FTFx module registers.
PFlashBlockBase	UINT32	The base address of PFlash block
PFlashBlockSize	UINT32	The size of PFlash block
DFlashBlockBase	UINT32	The base address of DFlash block
DFlashBlockSize	UINT32	The size of DFlash block
EERAMBlockBase	UINT32	The base address of EERAM block
EERAMBlockSize	UINT32	The size of EERAM block
EEEBlockSize	UINT32	The size of EEE block
BDMEnable	BOOL	Defines the state of background debug mode (enable /disable)
CallBack	Function pointer	Call back function to service the time critical events.

The type definition for the structure is given below.

```
typedef  struct   _ssd_config
{
        UINT32          ftfxRegBase;
        UINT32          PFlashBlockBase;
        UINT32          PFlashBlockSize;
        UINT32          DFlashBlockBase;
        UINT32          DFlashBlockSize;
        UINT32          EERAMBlockBase;
        UINT32          EERAMBlockSize;
        UINT32          EEEBlockSize;
        BOOL            BDMEnable;
        PCALLBACK       CallBack;
} FLASH_SSD_CONFIG, *PFLASH_SSD_CONFIG;
```

2.4 Configuration Macros

The C90TFS/FTFx Standard Software Driver has a configuration macro for the selection of C90TFS/FTFx derivatives. This macro is placed in the SSD_FTFx.h. Depending upon the macro selected, they will be changed in some internal macros and source codes.

Based on the value assigned to the macro **FLASH_DERIVATIVE**, the other macros will be given the corresponding values.
The derivative name will be defined based on the following rule:
FTFx_MCUtype_PFlashSize_FlexNVMSize_EEPROMSize_PFlashSectorSize_DFlashSectorSize
The values given to the macros for each derivative is shown in the following table.

Table 7: Derivative Specific Macro values for C90TFS/FTFx

No.	Derivative	MCU	P-Flash Size (KB)	FlexNVM Size (KB)	EEPROM Size (KB)	PFlash sector size (KB)	DFlash Sector size (KB)
1	FTFx_KX_256K_256K_4K_2K_2K	Kinetis	256	256	4	2	2
2	FTFx_KX_512K_0K_0K_2K_0K	Kinetis	512	0	0	2	0
3	FTFx_JX_128K_32K_2K_1K_1K	ColdFire	128	32	2	1	1
4	FTFx_FX_256K_32K_2K_1K_1K	ColdFire	256	32	2	1	1
5	FTFx_KX_512K_512K_16K_4K_4K	Kinetis	512	512	16	4	4
6	FTFx_KX_1024K_0K_16K_4K_0K	Kinetis	1024	0	16	4	0
7	FTFx_KX_32K_0K_0K_1K_0K	Kinetis	32	0	0	1	0
8	FTFx_KX_32K_32K_2K_1K_1K	Kinetis	32	32	2	1	1
9	FTFx_KX_64K_0K_0K_1K_0K	Kinetis	64	0	0	1	0
10	FTFx_KX_64K_32K_2K_1K_1K	Kinetis	64	32	2	1	1
11	FTFx_KX_128K_0K_0K_1K_0K	Kinetis	128	0	0	1	0
12	FTFx_KX_128K_32K_2K_1K_1K	Kinetis	128	32	2	1	1
13	FTFx_KX_64K_32K_2K_2K_1K	Kinetis	64	32	2	2	1
14	FTFx_KX_128K_32K_2K_2K_1K	Kinetis	128	32	2	2	1
15	FTFx_KX_256K_32K_2K_2K_1K	Kinetis	256	32	2	2	1

Note: *EEPROM Size* is the maximum size of FlexRAM can be configured as EEPROM.

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 structure 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.

2.6 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 will consist of following values:

Table 8: Return Codes

Name	Value	Description
FTFx_OK	0x0000	Function executes successfully.
FTFx_ERR_SIZE	0x0001	Misaligned size.
FTFx_ERR_RANGE	0x0002	Address is out of the valid range.
FTFx_ERR_ACCERR	0x0004	Access error is set in the FSTAT register.
FTFx_ERR_PVIOL	0x0008	Protection violation is set in FSTAT register.
FTFx_ERR_MGSTAT0	0x0010	MGSTAT 0 bit is set in the FSTAT register.
FTFx_ERR_CHANGEPROT	0x0020	Can't change protection status.
FTFx_ERR_EEESIZE	0x0040	Set the EEE Data Set Size with an invalid value.
FTFx_ERR_EFLASHSIZE	0x0080	Set the D/E Flash Partition Code with an invalid value.
FTFx_ERR_ADDR	0x0100	Misaligned destination.
FTFx_ERR_NOEEE	0x0200	EERAM is not set for EEPROM Emulation.
FTFx_ERR_EFLASHONLY	0x0400	D/E-Flash is set for full E-Flash.
FTFx_ERR_DFLASHONLY	0x0800	D/E-Flash is set for full D-Flash.
FTFx_ERR_RDCOLERR	0x1000	Read Collision error detected.
FTFx_ERR_RAMRDY	0x2000	EERAM is available as direct RAM only.

2.7 Macros

The following macros are user configurable and have to be initialized with correct values before using the SSD.

- FLASH_CALLBACK_CS – This macro is used as a counter value for the '*CallBack*()' function calling in the check sum API. This value can be changed as per the user requirement. User should take care that this value adheres to the maximum permissible CallBack time period.

2.8 SSD Functions

2.8.1 FlashInit()

2.8.1.1 Overview

This function checks and initializes Flash module for the other Flash APIs. It initializes Flash status register, read the D-Flash IFR fields if Dflash is available on the selected derivative.

2.8.1.2 Activation

This API will be the first to be invoked by user application before any other APIs of SSD are invoked.

2.8.1.3 Prototype

UINT32 FlashInit (PFLASH_SSD_CONFIG PSSDConfig)

2.8.1.4 Arguments

Table 9: Arguments for FlashInit()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.

2.8.1.5 Return Values

Table 10: Return Values for FlashInit()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.1.6 Troubleshooting

None.

2.8.1.7 Affected Registers

Table 11: Table Registers affected in FlashInit()

Register	Bits	Action
FSTAT	RDCOLERR, ACCERR, FPVIOL	Write
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOBi	All	Write
FCCOBm,FCCOBn	All	Read

i = 0x4, m = 0xA, n = 0xB: FTFx_KX_512K_512K_16K_4K_4K and FTFx_KX_1024K_0K_16K_4K_0K derivatives.
i = 0x8, m = 0x6, n = 0x7: other derivatives

2.8.1.8 Comments

The '*FlashInit*()' will be synchronous in behavior and it will not support re-entrance.
The flash module cannot be initialized correctly in low power mode.

2.8.1.9 Assumptions

None.

2.8.2 PFlashGetProtection()

2.8.2.1 Overview

This function retrieves current P-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.

2.8.2.2 Activation

This API will be invoked by user application.

2.8.2.3 Prototype

UINT32 PFlashGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT32* protectStatus)

2.8.2.4 Arguments

Table 12: Arguments for PFlashGetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT32*	To return the current value of the P-Flash Protection.

2.8.2.5 Return Values

Table 13: Return Values for PFlashGetProtection()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.2.6 Troubleshooting

None.

2.8.2.7 Affected Registers

Table 14: Registers Affected in PFlashGetProtection()

Register	Bits	Action
FPROT0, FPROT1, FPROT2, FPROT3	All	Read

2.8.2.8 Comments

None.

2.8.2.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.3 PFlashSetProtection()

2.8.3.1 Overview

This function sets the P-Flash protection to the intended protection status. Setting P-Flash protection status is subject to a protection transition restriction. So if there is any setting violation, it will return failed information and the current protection status won't be changed.

2.8.3.2 Activation

This API will be invoked by user application.

2.8.3.3 Prototype

UINT32 PFlashSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT32 protectStatus)

2.8.3.4 Arguments

Table 15: Arguments for PFlashSetProtection ()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT32	The intended protection status value should be written to the P-Flash Protection Registers.

2.8.3.5 Return Values

Table 16: Return Values for PFlashSetProtection ()

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

2.8.3.6 Troubleshooting

Table 17: Troubleshooting for PFlashSetProtection()

Returned Error Bits	Description	Solution
FTFx_ERR_CHANGEPROT	Fail to change protection mode.	Protection cannot be changed while command is in progress

2.8.3.7 Affected Registers

Table 18: Register affected in PFlashSetProtection()

Register	Bits	Action
FPROT0, FPROT1, FPROT2, FPROT3	All	Read, Write

2.8.3.8 Comments

None.

2.8.3.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.4 DFlashGetProtection()

2.8.4.1 Overview

This function 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 Activation

This API will be invoked by user application.

2.8.4.3 Prototype

UINT32 DFlashGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8* protectStatus)

2.8.4.4 Arguments

Table 19: Arguments for DFlashGetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT8*	To return the current value of the D-Flash Protection Register.

2.8.4.5 Return Values

Table 20: Return Values for DFlashGetProtection()

Type	Description	Possible Values

UINT32	Successful completion.	FTFx_OK
	Error value	FTFx_ERR_EFLASHONLY

2.8.4.6 Troubleshooting

Table 21: Troubleshooting for DFlashGetProtection()

Returned Error Bits	Description	Solution
FTFx_ERR_EFLASHONLY	D/EFlash is partitioned as full E-Flash (no D-Flash)	D-Flash should be partitioned by calling *DEFlashPartition()*

2.8.4.7 Affected Registers

Table 22: Register affected in DFlashGetProtection()

Register	Bits	Action
FDPROT	All	Read

2.8.4.8 Comments

None.

2.8.4.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.5 DFlashSetProtection()

2.8.5.1 Overview

This function sets the D-Flash protection to the intended protection status. Setting D-Flash protection status is subject to a protection transition restriction. So if there is any setting violation, it will return failed information and the current protection status won't be changed.

2.8.5.2 Activation

This API will be invoked by user application.

2.8.5.3 Prototype

UINT32 DFlashSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8 protectStatus)

2.8.5.4 Arguments

Table 23: Arguments for DFlashSetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT8	The intended protection status value should be written to the D-Flash Protection Register.

2.8.5.5 Return Values

Table 24: Return Values for DFlashSetProtection()

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

2.8.5.6 Troubleshooting

Table 25: Troubleshooting for DFlashSetProtection()

Returned Error Bits	Description	Solution
FTFx_ERR_EFLASHONLY	D/EFlash is partitioned as full E-Flash (no D-Flash)	D-Flash should be partitioned by calling *DEFlashPartition()*
FTFx_ERR_CHANGEPROT	Fail to change protection mode.	Protection cannot be changed while command is in progress

2.8.5.7 Affected Registers

Table 26: Register affected in DFlashSetProtection()

Register	Bits	Action
FDPROT	All	Read, Write

2.8.5.8 Comments

None.

2.8.5.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.6 EERAMGetProtection()

2.8.6.1 Overview

This function retrieves current EERAM 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.

2.8.6.2 Activation

This API will be invoked by user application.

2.8.6.3 Prototype

UINT32 EERAMGetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8* protectStatus)

2.8.6.4 Arguments

Table 27: Arguments for EERAMGetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT8*	To return the current value of the EERAM Protection Register.

2.8.6.5 Return Values

Table 28: Return Values for EERAMGetProtection()

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

2.8.6.6 Troubleshooting

Table 29: Troubleshooting for EERAMGetProtection()

Returned Error Bits	Description	Solution
FTFx_ERR_NOEEE	EERAM is set for traditional RAM	EEE should be set by calling *SetEEEEnable()* function

2.8.6.7 Affected Registers

Table 30: Register affected in EERAMGetProtection()

Register	Bits	Action
FEPROT	All	Read
FCNFG	EEERDY	Read

2.8.6.8 Comments

None.

2.8.6.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.7 EERAMSetProtection()

2.8.7.1 Overview

This function sets the EERAM protection to the intended protection status. Setting EERAM protection status is subject to a protection transition restriction. So if there is any setting violation, it will return failed information and the current protection status won't be changed.

2.8.7.2 Activation

This API will be invoked by user application.

2.8.7.3 Prototype

UINT32 EERAMSetProtection (PFLASH_SSD_CONFIG PSSDConfig, UINT8 protectStatus)

2.8.7.4 Arguments

Table 31: Arguments for EERAMSetProtection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
protectStatus	UINT8	The intended protection status value should be written to the EERAM Protection Register.

2.8.7.5 Return Values

Table 32: Return Values for EERAMSetProtection()

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

2.8.7.6 Troubleshooting

Table 33: Troubleshooting for EERAMSetProtection()

Returned Error Bits	Description	Solution
FTFx_ERR_NOEEE	EERAM is set for	EEE should be set by calling

Returned Error Bits	Description	Solution
	traditional RAM	*SetEEEEnable()* function
FTFx_ERR_CHANGEPROT	Fail to change protection mode.	Protection cannot be changed while command is in progress

2.8.7.7 Affected Registers

Table 34: Register affected in EERAMSetProtection()

Register	Bits	Action
FEPROT	All	Read, Write
FCNFG	EEERDY	Read

2.8.7.8 Comments

None.

2.8.7.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.8 FlashGetInterruptEnable()

2.8.8.1 Overview

This function will read the FCNFG register and return the interrupt enable states for Flash interrupt types.

2.8.8.2 Activation

This API will be invoked by user application.

2.8.8.3 Prototype

UINT32 FlashGetInterruptEnable (PFLASH_SSD_CONFIG PSSDConfig, UINT8* interruptState)

2.8.8.4 Arguments

Table 35: Arguments for FlashGetInterruptEnable()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
interruptState	UINT8*	To return the current interrupt states. Bit 7 (FCNFG): Command Complete Interrupt Enable Bit 6 (FCNFG): Read Collision Error Enable

2.8.8.5 Return Values

Table 36: Return Values for FlashGetInterruptEnable()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.8.6 Troubleshooting

None.

2.8.8.7 Affected Registers

Table 37: Register affected in FlashGetInterruptEnable()

Register	Bits	Action
FCNFG	CCIE, RDCOLLIE	Read

2.8.8.8 Comments

None.

2.8.8.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.9 FlashSetInterruptEnable()

2.8.9.1 Overview

This function sets the Flash interrupt enable bits in the Flash module configuration register. Other bits in the FCNFG register won't be affected.

2.8.9.2 Activation

This API will be invoked by user application.

2.8.9.3 Prototype

UINT32 FlashSetInterruptEnable (PFLASH_SSD_CONFIG PSSDConfig, UINT8 interruptState)

2.8.9.4 Arguments

Table 38: Arguments for FlashSetInterruptEnable()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
interruptState	UINT8	The bit-mapped value of desired interrupt enabled state. Bit-mapped value corresponding to the Flash interrupts enabling bits are, Bit 7 (FCNFG): Command Complete Interrupt Enable Bit 6 (FCNFG): Read Collision Error Enable

2.8.9.5 Return Values

Table 39: Return Values for FlashSetInterruptEnable()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.9.6 Troubleshooting

None.

2.8.9.7 Affected Registers

Table 40: Register affected in FlashSetInterruptEnable()

Register	Bits	Action
FCNFG	CCIE, RDCOLLIE	Write

2.8.9.8 Comments

None.

2.8.9.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.10 FlashGetSecurityState()

2.8.10.1 Overview

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

2.8.10.2 Activation

This API will be invoked by user application.

2.8.10.3 Prototype

UINT32 FlashGetSecurityState (PFLASH_SSD_CONFIG PSSDConfig, UINT8* securityState)

2.8.10.4 Arguments

Table 41: Arguments for FlashGetSecurityState()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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.

2.8.10.5 Return Values

Table 42: Return Values for FlashGetSecurityState()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.10.6 Troubleshooting

None.

2.8.10.7 Affected Registers

Table 43: Register affected in FlashGetSecurityState()

Register	Bits	Action
FSEC	KEYEN, SEC[1:0]	Read

2.8.10.8 Comments

None.

2.8.10.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.11 FlashCommandSequence()

2.8.11.1 Overview

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

2.8.11.2 Activation

This API will be invoked by the functions which have MGATE operation.

2.8.11.3 Prototype

**UINT32 FlashCommandSequence (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT8 index, **
UINT8* pCommandArray)

2.8.11.4 Arguments

Table 44: Arguments for FlashCommandSequence()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
index	UINT8	Maximum index value of the command array.
pCommandArray	UINT8*	Pointer to the array which contains the data to be written to the FCCOB registers.

2.8.11.5 Return Values

Table 45: Return Values for FlashCommandSequence()

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

2.8.11.6 Troubleshooting

Since this function is transparent to user, this section is kept blank intentionally. The corresponding trouble shooting is given in the respective APIs.

2.8.11.7 Affected Registers

Table 46: Registers affected in FlashCommandSequence()

Register	Bits	Action
FSTAT	ACCERR	Read, Write
	FPVIOL	Read, Write
	MGSTAT0	Read
	RDCOLERR	Read, Write
	CCIF	Read, Write
FCCOB0…FCCOBB (depend on specific API)	[7:0]	Write

2.8.11.8 Comments

The '*FlashCommandSequence*()' is an internal function of the SSD. User should not call this API from the application.

The *FlashCommandSequence()* must not be placed in the target on which user wants to implement. For example, if the API function is employed in P-Flash, then the *FlashCommandSequence()* must not be placed in P-Flash and so on.

2.8.11.9 Assumptions

None.

2.8.12 FlashSecurityBypass()

2.8.12.1 Overview

If the MCU is secured state, this function will unsecure the MCU by comparing the provided backdoor key with ones in the Flash Configuration Field.

2.8.12.2 Activation

This API will be invoked by user application.

2.8.12.3 Prototype

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

2.8.12.4 Arguments

Table 47: Arguments for FlashSecurityBypass()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
keyBuffer	UINT8*	Point to the user buffer containing the backdoor key.
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.12.5 Return Values

Table 48: Return Values for FlashSecurityBypass()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.12.6 Troubleshooting

Table 49: Troubleshooting for FlashSecurityBypass()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if supplied key is all 0s or all Fs	Supplied key should not be all 0s or all Fs.
	Set if an incorrect Backdoor Key is supplied	Provide the correct Backdoor Key
	Set if Backdoor Key access has not been enabled (KEYEN[1:0] != 10)	Unsecuring is impossible if KEYEN[1:0] != 10.
	Set if this command is launched and the backdoor key	Reset the device and call the function with a valid key.

Returned Error Bits	Possible causes	Solution
	has mismatched since the last power down reset	

2.8.12.7 Affected Registers

Table 50: Register affected in FlashSecurityBypass()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	CCIF	Read, Write
FSEC	[7:0]	Read
FCCOB0…FCCOBB	[7:0]	Write

2.8.12.8 Comments

The '*FlashSecurityBypass*()' will be synchronous in behavior and it will not support re-entrance The verification of backdoor key command is executed in this function by calling the '*FlashCommandSequence*()' function.

2.8.12.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.13 FlashEraseAllBlock()

2.8.13.1 Overview

The Erase All Blocks operation will erase all Flash memory, initialize the EERAM, verify all memory contents, then release MCU security.

2.8.13.2 Activation

This API will be invoked by user application.

2.8.13.3 Prototype

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

2.8.13.4 Arguments

Table 51: Arguments for FlashEraseAllBlock()

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

2.8.13.5 Return Values

Table 52: Return Values for FlashEraseAllBlock()

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

2.8.13.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash erase all block command.

Table 53: Troubleshooting for FlashEraseAllBlock()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_PVIOL	Set if any region of P-Flash, D-Flash or EERAM is protected	Write to a protected area is not possible. The global address provided should point to an unprotected area.
FTFx_ERR_MGSTAT0	Set if erase verify fails	Hardware Error

2.8.13.7 Affected Registers

Table 54: Registers affected in FlashEraseAllBlock()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCCOB0	[7:0]	Write

2.8.13.8 Comments

The '*FlashEraseAllBlock*()' will be synchronous in behavior and it will not support re-entrance.
The *FlashEraseAllBlock*() must not be placed in to the target on which this function will run.
For derivative of FTFx_FX_256K_32K_2K_1K_1K , '*FlashEraseAllBlock*' will not be available if swap command has been called at least once.
For derivatives of FTFx_KX_512K_512K_16K_4K_4K or FTFx_KX_1024K_0K_16K_4K_0K, '*FlashEraseAllBlock*' will be used to place the swap system in the uninitialized state.

2.8.13.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.14 FlashEraseBlock()

2.8.14.1 Overview

The Erase Flash Block operation will erase all addresses in a single P-Flash or D-Flash block.

2.8.14.2 Activation

This API will be invoked by user application.

2.8.14.3 Prototype

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

2.8.14.4 Arguments

Table 55: Arguments for FlashEraseBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended erase operation.
FlashComm-andSequence	pFLASHCOMMANDSE-QUENCE	Pointer to the flash command sequence function.

2.8.14.5 Return Values

Table 56: Return Values for FlashEraseBlock()

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

2.8.14.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash Erase Block command. Apart from these the input based errors handling is also mentioned.

Table 57: Troubleshooting for FlashEraseBlock()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if D-Flash is selected with EEE enabled	Disable EEE if D-Flash is selected.
	Set if command not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_PVIOL	Set if any region of selected flash is protected in NVM normal mode of NVM special mode	Write to a protected area is not possible. The global address provided should point to an unprotected area.
FTFx_ERR_ADDR	Destination is not longword / double-phrase aligned.	Check the passed destination parameter. It has to be longword / double-phrase aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	Set if erase verify fails	Hardware Error

2.8.14.7 Affected Registers

Table 58: Registers affected in FlashEraseBlock()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCCOB0, FCCOB1, FCCOB2, FCCOB3	[7:0]	Write

2.8.14.8 Comments

The '*FlashEraseBlock*()' will be synchronous in behavior and it will not support re-entrance.
The *FlashEraseBlock*() must not be placed in to the target on which this function will run.
With the derivatives including multiply logical P-Flash blocks, *FlashEraseBlock()* just erases a single block in a single call.
For derivatives including swap feature, '*FlashEraseBlock*' will not be available at the block including swap indicator if swap command has been called at least once.

2.8.14.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.15 FlashEraseSector()

2.8.15.1 Overview

The Erase Flash Sector operation will erase one or more sectors in P-Flash or D-Flash.

2.8.15.2 Activation

This API will be invoked by user application.

2.8.15.3 Prototype

```
UINT32 FlashEraseSector(PFLASH_SSD_CONFIG PSSDConfig, \
                        UINT32 destination, \
                        UINT32 size, \
                        pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.15.4 Arguments

Table 59: Arguments for FlashEraseSector()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended erase operation.
size	UINT32	Size to be erased in bytes
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.15.5 Return Values

Table 60: Return Values for FlashEraseSector()

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

2.8.15.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash Erase Sector command. Apart from these the input based errors handling is also mentioned.

Table 61: Troubleshooting for FlashEraseSector()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_PVIOL	Set if the global address [22:0] points to a protected area	Write to a protected area is not possible. The global address provided should point to an unprotected area.
FTFx_ERR_SIZE	Size is not sector aligned	Check the passed size parameter. It has to be sector aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	Erase verify fails	Hardware Error
FTFx_ERR_ADDR	Destination is not phrase / double-phrase aligned.	Check the passed destination parameter. It has to be phrase / double-phrase aligned.

2.8.15.7 Affected Registers

Table 62: Registers affected in FlashEraseSector()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCCOB0, FCCOB1, FCCOB2, FCCOB3	[7:0]	Write

2.8.15.8 Comments

The '*FlashEraseSector*()' will be synchronous in behavior and it will not support re-entrance.
The *FlashEraseSector()* must not be placed in to the target on which this function will run.
For derivatives including swap feature, the sector containing swap indicator will not be erased if swap command has been called at least once.

2.8.15.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.16 FlashEraseSuspend()

2.8.16.1 Overview

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

2.8.16.2 Activation

This API will be invoked by user application.

2.8.16.3 Prototype

UINT32 FlashEraseSuspend (PFLASH_SSD_CONFIG PSSDConfig)

2.8.16.4 Arguments

Table 63: Arguments for FlashEraseSuspend()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.

2.8.16.5 Return Values

Table 64: Return Values for FlashEraseSuspend()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.16.6 Troubleshooting

None

2.8.16.7 Affected Registers

Table 65: Registers affected in FlashEraseSuspend()

Register	Bits	Action
FSTAT	CCIF	Read
FCNFG	ERSSUSP	Write

2.8.16.8 Comments

None.

2.8.16.9 Assumptions

'FlashInit' function needs to be called before calling this function. *FlashEraseSector()* should be taking place while calling this function.

2.8.17 FlashEraseResume()

2.8.17.1 Overview

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

2.8.17.2 Activation

This API will be invoked by user application.

2.8.17.3 Prototype

UINT32 FlashEraseResume (PFLASH_SSD_CONFIG PSSDConfig)

2.8.17.4 Arguments

Table 66: Arguments for FlashEraseResume()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.

2.8.17.5 Return Values

Table 67: Return Values for FlashEraseResume()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK

2.8.17.6 Troubleshooting

None.

2.8.17.7 Affected Registers

Table 68: Registers affected in FlashEraseResume()

Register	Bits	Action
FCNFG	ERSSUSP	Read
FSTAT	CCIF	Write

2.8.17.8 Comments

None.

2.8.17.9 Assumptions

'FlashInit' function needs to be called before calling this function.

2.8.18 FlashProgramSection()

2.8.18.1 Overview

This function 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 EERAM while it is set to function as a RAM. The Section Program Buffer is limited to the lower half of the EERAM.

2.8.18.2 Activation

This API will be invoked by user application.

2.8.18.3 Prototype

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

2.8.18.4 Arguments

Table 69: Arguments for FlashProgramSection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended program operation.
Number	UINT16	Number of phrase/longword to be programmed.
FlashComma-ndSequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.18.5 Return Values

Table 70: Return Values for FlashProgramSection()

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

2.8.18.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash Program section command. Apart from these the input based errors handling is also mentioned.

Table 71: Troubleshooting for FlashProgramSection()

Returned Error Bits	Possible Causes	Solution
FTFx_ERR_ACCERR	Set if request number of phrase/longword is zero.	Check the passed number of phrase/longword parameter. It has to be not zero.
	Set if command not available in current mode	The current mode restricts the command launched in the API.
	Set if the space required to store data for the request number of phrase/longword is more than half the size of EERAM	Check the passed number of phrase/longword parameter. It has to be less than half the size of EERAM.
	Set if requested section crosses a program flash sector boundary	The requested section must lie within one sector.
FTFx_ERR_PVIOL	Set if flash address points to a protected area	Write to a protected area is not possible. The flash address provided should point to an unprotected area.
FTFx_ERR_RAMRDY	EERAM is not set to function as traditional RAM (RAMRDY=0)	EERAM must be set as traditional RAM by calling SetEEEEnable() function..
FTFx_ERR_ADDR	Destination is misaligned on double-phrase/phrase/longword	Check the passed destination parameter. It has to be double-phrase/phrase/longword aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	erase verify fails	Hardware Error

2.8.18.7 Affected Registers

Table 72: Registers affected in FlashProgramSection()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCNFG	RAMRDY	Read
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4, FCCOB5	[7:0]	Write

2.8.18.8 Comments

A Flash memory location must be in the erased state before being programmed.

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.

2.8.18.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.19 FlashProgramLongword()

2.8.19.1 Overview

This function is used to perform program operation on P-flash or D-Flash block according to longword alignment.

2.8.19.2 Activation

This API will be invoked by user application.

2.8.19.3 Prototype

```
UINT32 FlashProgramLongword (PFLASH_SSD_CONFIG PSSDConfig, \
                             UINT32 destination, \
                             UINT32 size, \
                             UINT32 source, \
                             pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.19.4 Arguments

Table 73: Arguments for FlashProgramLongword()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended program operation.
size	UINT32	Size to be programmed
source	UINT32	Source address from which data has to be taken for program operation.
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.19.5 Return Values

Table 74: Return Values for FlashProgramLongword()

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

2.8.19.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash Program longword command. Apart from these, the input based errors handling is also mentioned.

Table 75: Troubleshooting for FlashProgramLongword()

Returned Error Bits	Possible Causes	Solution
FTFx_ERR_ACCERR	Set if command not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_PVIOL	Set if flash address points to a protected area	Write to a protected area is not possible. The flash address provided should point to an unprotected area.
FTFx_ERR_SIZE	Size is misaligned on longword	Check the passed size parameter. It has to be longword aligned.
FTFx_ERR_ADDR	Destination is misaligned on longword	Check the passed destination parameter. It has to be longword aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	Verify fails	Hardware Error

2.8.19.7 Affected Registers

Table 76: Registers affected in FlashProgramLongword()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4, FCCOB5, FCCOB6, FCCOB7	[7:0]	Write

2.8.19.8 Comments

This function is only available in the derivatives supporting Flash program longword command.
A Flash memory location must be in the erased state before being programmed.
For derivatives including swap feature, the swap indicator address is protected from '*ProgramLongword*'.

2.8.19.9 Assumptions

'FlashInit()' function must be called before invoking this function.

2.8.20 FlashProgramPhrase()

2.8.20.1 Overview

This function is used to perform program operation on P-flash or D-Flash block according to phrase alignment.

2.8.20.2 Activation

This API will be invoked by user application.

2.8.20.3 Prototype

**UINT32 FlashProgramPhrase (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT32 size, **
**UINT32 source, **
pFLASHCOMMANDSEQUENCE FlashCommandSequence)

2.8.20.4 Arguments

Table 77: Arguments for FlashProgramPhrase()

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

2.8.20.5 Return Values

Table 78: Return Values for FlashProgramPhrase()

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

2.8.20.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to Flash Program Phrase command. Apart from these, the input based errors handling is also mentioned.

Table 79: Troubleshooting for FlashProgramPhrase()

Returned Error Bits	Possible Causes	Solution
FTFx_ERR_ACCERR	Set if command not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_PVIOL	Set if flash address points to a protected area	Write to a protected area is not possible. The flash address provided should point to an unprotected area.
FTFx_ERR_SIZE	Size is misaligned on phrase	Check the passed size parameter. It has to be phrase aligned.
FTFx_ERR_ADDR	Destination is misaligned on phrase	Check the passed destination parameter. It has to be phrase aligned.
FTFx_ERR_RANGE	Size or destination or end	The size or destination or end

Returned Error Bits	Possible Causes	Solution
	address for erase exceeds the valid range	address provided should be within the valid range
FTFx_ERR_MGSTAT0	Verify fails	Hardware Error

2.8.20.7 Affected Registers

Table 80: Registers affected in FlashProgramPhrase()

Register	Bits	Action
FSTAT	ACCERR	Write, Read
	FPVIOL	Write, Read
	MGSTAT0	Read
	CCIF	Read, Write
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4, FCCOB5, FCCOB6, FCCOB7, FCCOB8, FCCOB9, FCCOBA, FCCOBB	[7:0]	Write

2.8.20.8 Comments

This function is only available in the derivatives supporting Flash program phrase command.
A Flash memory location must be in the erased state before being programmed.
For derivatives including swap feature, the swap indicator address is protected from '*ProgramPhrase*'.

2.8.20.9 Assumptions

'FlashInit()' function must be called before invoking this function.

2.8.21 FlashCheckSum()

2.8.21.1 Overview

This function is used to calculate checksum value for the specified flash range in byte.

2.8.21.2 Activation

This API will be invoked by user application.

2.8.21.3 Prototype

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

2.8.21.4 Arguments

Table 81: Arguments for FlashCheckSum()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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

2.8.21.5 Return Values

Table 82: Return Values for FlashCheckSum()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_RANGE

2.8.21.6 Troubleshooting

The troubleshooting mentioned below is the input based error.

Table 83: Troubleshooting for FlashCheckSum()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range

2.8.21.7 Affected Registers

None.

2.8.21.8 Comments

None.

2.8.21.9 Assumptions

'FlashInit()' function must be called before invoking this function.

.

2.8.22 FlashVerifyAllBlock()

2.8.22.1 Overview

This function will check to see if the P-Flash and D-Flash blocks as well as EERAM, E-Flash records, and D-Flash IFR have been erased to the specified read margin level, if applicable, and will release security if the readout passes.

2.8.22.2 Activation

This API will be invoked by user application.

2.8.22.3 Prototype

```
UINT32 FlashVerifyAllBlock (PFLASH_SSD_CONFIG PSSDConfig, \
                            UINT8 marginLevel, \
                            pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.22.4 Arguments

Table 84: Arguments for FlashVerifyAllBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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.

2.8.22.5 Return Values

Table 85: Return Values for FlashVerifyAllBlock()

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

2.8.22.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash verify all block command.

Table 86: Troubleshooting for FlashVerifyAllBlock()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if invalid margin choice specified	Margin must be one of three values: 0x00, 0x01, 0x02
FTFx_ERR_MGSTAT0	Set if erase verify fails	Hardware error

2.8.22.7 Affected Registers

Table 87 Registers affected in FlashVerifyAllBlock()

Register	Bits	Action
FSTAT	ACCERR, MGSTAT0	Read

2.8.22.8 Comments

None.

2.8.22.9 Assumptions

'FlashInit()' function must be called before invoking this function.

2.8.23 FlashVerifyBlock()

2.8.23.1 Overview

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

2.8.23.2 Activation

This API will be invoked by user application.

2.8.23.3 Prototype

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

2.8.23.4 Arguments

Table 88: Arguments for FlashVerifyBlock()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended erase 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.

2.8.23.5 Return Values

Table 89: Return Values for FlashVerifyBlock()

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

2.8.23.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash verify block command. Apart from this, the input based error handling is also mentioned.

Table 90: Troubleshooting for FlashVerifyBlock()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode.	The current mode restricts the command launched in the API.
	Set if invalid margin choice specified.	Margin must be one of three values: 0x00, 0x01, 0x02
	Set if D-Flash is selected with EEE enable.	Disable EEE by calling SetEEEEnable() function
FTFx_ERR_ADDR	Destination is misaligned on longword/double-phrase.	Destination must be longword/double-phrase aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range.	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	Erase verify fails.	Hardware error.

2.8.23.7 Affected Registers

Table 91: Registers affected in FlashVerifyBlock()

Register	Bits	Action
FSTAT	ACCERR, MGSTAT0	Read
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4	[7..0]	Write

2.8.23.8 Comments

With the derivative including multiply logical P-Flash blocks, 'FlashVerifyBlock' just verify a single block in a single call.

2.8.23.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.24 FlashVerifySection()

2.8.24.1 Overview

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

2.8.24.2 Activation

This API will be invoked by user application.

2.8.24.3 Prototype

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

2.8.24.4 Arguments

Table 92: Arguments for FlashVerifySection()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended erase operation.
Number	UINT16	Number of double phrase/phrase/longword to be verified.
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.

2.8.24.5 Return Values

Table 93: Return Values for FlashVerifySection()

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

2.8.24.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash verify section command. Apart from this, the input based error handling is also mentioned.

Table 94: Troubleshooting for FlashVerifySection()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not	The current mode restricts the

Returned Error Bits	Possible causes	Solution
	available in current mode	command launched in the API.
	Set if invalid margin specified	Provide valid margin. It must be one of three values: 0x00, 0x01, 0x02
	Set if request number of double phrase/ phrase/longword is zero	The request number of double phrase/ phrase/longword must be not zero
FTFx_ERR_ADDR	Destination is misaligned on double-phrase/ phrase/ longword.	Destination must be double-phrase/ phrase/longword aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_MGSTAT0	Erase verify fails	Hardware error.

2.8.24.7 Affected Registers

Table 95: Registers affected in FlashVerifySection()

Register	Bits	Action
FSTAT	ACCERR, MGSTAT0	Read
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4, FCCOB5, FCCOB6	[7..0]	Write

2.8.24.8 Comments

None.

2.8.24.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.25 FlashReadOnce()

2.8.25.1 Overview

This function is used to read access to a reserved 64 byte field located in the P-Flash IFR. With the derivative of FTFx_KX_512K_512K_16K_4K_4K or FTFx_KX_1024K_0K_16K_4K_0K, access to this field is via 8 records, each of which is 8 bytes long. Other derivatives, access to the read once field is via 16 records, each of which is 4 bytes long.

2.8.25.2 Activation

This API will be invoked by user application.

2.8.25.3 Prototype

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

2.8.25.4 Arguments

Table 96: Arguments for FlashReadOnce()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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.

2.8.25.5 Return Values

Table 97: Return Values for FlashReadOnce()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.25.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash read once command.

Table 98: Troubleshooting for FlashReadOnce()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.

2.8.25.7 Affected Registers

Table 99: Registers affected in FlashReadOnce()

Register	Bits	Action
FSTAT	ACCERR	Read
FCCOB0, FCCOB1	[7..0]	Write
FCCOB4, FCCOB5,…,FCCOBn	[7..0]	Read

n = 0xB: FTFx_KX_512K_512K_16K_4K_4K or FTFx_KX_1024K_0K_16K_4K_0K derivative
n = 0x7: other derivatives

2.8.25.8 Comments

None.

2.8.25.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.26 FlashProgramOnce()

2.8.26.1 Overview

This API is used to program data into a dedicated 64 bytes region in the P-Flash IFR which stores critical information for the user.

2.8.26.2 Activation

This API will be invoked by user application.

2.8.26.3 Prototype

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

2.8.26.4 Arguments

Table 100: Arguments for FlashProgramOnce()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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.

2.8.26.5 Return Values

Table 101: Return Values for FlashProgramOnce()

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

2.8.26.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash program once command.

Table 102: Troubleshooting for FlashProgramOnce()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.
FTFx_ERR_MGSTAT0	Set if any error has been encountered during verify operation	Hardware error

2.8.26.7 Affected Registers

Table 103: Registers affected in FlashProgramOnce()

Register	Bits	Action
FSTAT	ACCERR, MGSTAT0	Read
FCCOB0, FCCOB1	[7..0]	Write
FCCOB4, FCCOB5, …, FCCOBn	[7..0]	Write

n = 0xB: FTFx_KX_512K_512K_16K_4K_4K or FTFx_KX_1024K_0K_16K_4K_0K derivative
n = 0x7: other derivatives

2.8.26.8 Comments

None.

2.8.26.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.27 FlashProgramCheck()

2.8.27.1 Overview

The Program Check command tests a previously programmed P-Flash or D-Flash longword to see if it reads correctly at the specified margin level.

2.8.27.2 Activation

This API will be invoked by user application.

2.8.27.3 Prototype

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

2.8.27.4 Arguments

Table 104: Arguments for FlashProgramCheck()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended check operation.
pExpectedData	UINT8*	The pointer to the expected data.
pFailAddr	UINT32*	Returned failing address.
pFailData	UINT8*	With FTFx_KX_512K_512K_16K, pFailData is always equal to zero. With other derivatives, this is returned failing data.
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.
FlashCommand-Sequence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.27.5 Return Values

Table 105: Return Values for FlashProgramCheck()

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

2.8.27.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to flash program check command. Apart from these, the input based error handling is also mentioned.

Table 106: Troubleshooting for FlashProgramCheck()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.
	Set if invalid margin choice specified	Provide valid margin choice (must be 0x01 or 0x02).
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range.
FTFx_ERR_MGSTAT0	Erase verify fails	Hardware error.
FTFx_ERR_SIZE	Size is misaligned on longword.	Size must be longword aligned.
FTFx_ERR_ADDR	Destination is misaligned on longword.	Destination must be longword aligned.

2.8.27.7 Affected Registers

Table 107: Registers affected in FlashProgramCheck()

Register	Bits	Action
FSTAT	CCIF, ACCERR, MGSTAT0	Read
FCCOB0, FCCOB1, FCCOB2, FCCOB3	[7..0]	Write
FCCOB4	[7..0]	Read, Write
FCCOB5, FCCOB6, FCCOB7	[7..0]	Read
FCCOB8, FCCOB9, FCCOBA, FCCOBB	[7..0]	Write

2.8.27.8 Comments

None.

2.8.27.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.28 FlashReadResource()

2.8.28.1 Overview

This function is provided for the user to read data from P-Flash IFR and D-Flash IFR space.

2.8.28.2 Activation

This API will be invoked by user application.

2.8.28.3 Prototype

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

2.8.28.4 Arguments

Table 108: Arguments for FlashReadResource()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
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.

2.8.28.5 Return Values

Table 109: Return Values for FlashReadResource()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR FTFx_ERR_ADDR FTFx_ERR_RANGE

2.8.28.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to both flash read resource command. Apart from these, the input based error handling is also mentioned.

Table 110: Troubleshooting for FlashReadResource()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.
	Set if invalid resource code entered	Provide valid resource code
FTFx_ERR_ADDR	Destination is misaligned on longword/phrase	Destination must be longword/phrase aligned.
FTFx_ERR_RANGE	Size or destination or end address for erase exceeds the valid range	The size or destination or end address provided should be within the valid range

2.8.28.7 Affected Registers

Table 111: Registers affected in FlashReadResource()

Register	Bits	Action
FSTAT	CCIF, ACCERR	Read
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOBm	[7..0]	Write
FCCOB4, FCCOB5, …, FCCOBn	[7..0]	Read

n = 0xB, m = 0x4: FTFx_KX_512K_512K_16K_4K_4K or FTFx_KX_1024K_0K_16K_4K_0K derivative
n = 0x7, m = 0x8: other derivatives

2.8.28.8 Comments

This function is to read the data from special purpose memory resource such as Program flash IFR, Data flash IFR.

It is assumed that the valid address range for each special purpose memory is the offset address to each flash space (Program flash or Data flash). Thus, if user wants to read the data from Data flash IFR, he must provide the address belonging to valid Data flash range. And if user wants to read the data from Program flash, he must provide the address belonging to valid Program flash range.

2.8.28.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.29 DEFlashPartition()

2.8.29.1 Overview

This function prepares the D/E-Flash block for use as D-Flash, E-Flash or a combination of both and initializes the EERAM.

2.8.29.2 Activation

This API will be invoked by user application.

2.8.29.3 Prototype

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

2.8.29.4 Arguments

Table 112: Arguments for DEFlashPartition()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
EEEDataSizeCode	UINT8	EEE Data Size Code. Refer to reference manual of corresponding derivative for more details.
DEPartitionCode	UINT8	D/E-Flash Partition Code. Refer to reference manual of corresponding derivative for more details.
FlashCommandS-equence	pFLASHCOMMANDS-EQUENCE	Pointer to the flash command sequence function.

2.8.29.5 Return Values

Table 113: Return Values for DEFlashPartition()

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

2.8.29.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to program partition command.

Table 114: Troubleshooting for DEFlashPartition()

Returned Error Bits	Possible causes	Solution

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API
	Set if EEE data size code and D/E flash partition code are not 0xFF initially	EEE data size code and D/E Flash partition code in D-Flash IFR must be erased in advance
	Set if invalid EEE data size code entered	Provide valid EEE data size code
	Set if invalid D/E Flash partition code entered	Provide valid D/E Flash partition code
	Set if D/E flash partition code is set for full D-Flash (no EEE) and EEE data size code allocate EERAM for EEE	EFlash must be partitioned if there is corresponding EERAM for EEE and vice versa
	Set if D/E flash partition code allocate space for EFlash but EEE data size code allocate no EERAM for EEE	EFlash must be partitioned if there is corresponding EERAM for EEE and vice versa
FTFx_ERR_MGSTAT0	Set if any error has been encountered during verify operation	Hardware error

2.8.29.7 Affected Registers

Table 115: Registers affected in DEFlashPartition()

Register	Bits	Action
FSTAT	CCIF, ACCERR, MGSTAT0	Read
FCCOB0, FCCOB4, FCCOB5	[7..0]	Write

2.8.29.8 Comments

None.

2.8.29.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.30 SetEEEEnable()

2.8.30.1 Overview

This function is used to change the function of the EERAM. When not partitioned for EEE, the EERAM is typically used as traditional RAM. When partitioned for EEE, the EERAM is typically used to store EEE data.

2.8.30.2 Activation

This API will be invoked by user application.

2.8.30.3 Prototype

UINT32 SetEEEEnable (PFLASH_SSD_CONFIG PSSDConfig, UINT8 EEEEnable)

2.8.30.4 Arguments

Table 116: Arguments for SetEEEEnable()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
EEEEnable	UINT8	EERAM function control code.

2.8.30.5 Return Values

Table 117: Return Values for SetEEEEnable()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ACCERR

2.8.30.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due set EEE enable command.

Table 118: Troubleshooting for SetEEEEnable()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ACCERR	Set if command is not available in current mode	The current mode restricts the command launched in the API.
	Set if EERAM function control code is not defined	Provide valid EERAM function control code (must be either 0x00 or 0xFF)
	Set if EERAM function control code is set to make EERAM available for EEE but D/E flash is not partitioned for EEE	Partition D/E flash for EEE in case of enabling EERAM for EEE

2.8.30.7 Affected Registers

Table 119: Registers affected in SetEEEEnable()

Register	Bits	Action
FSTAT	CCIF, ACCERR	Read
FCCOB0, FCCOB1	[7..0]	Write

2.8.30.8 Comments

None.

2.8.30.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.31 EEEWrite()

2.8.31.1 Overview

This function is used to write data to EERAM when it is used as EEPROM emulator.

2.8.31.2 Activation

This API will be invoked by user application.

2.8.31.3 Prototype

**UINT32 EEEWrite (PFLASH_SSD_CONFIG PSSDConfig, **
**UINT32 destination, **
**UINT32 size, **
UINT32 source)

2.8.31.4 Arguments

Table 120: Arguments for EEEWrite()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
destination	UINT32	Start address for the intended write operation.
size	UINT32	Size to be written.
source	UINT32	Source address from which data has to be taken for writing operation.

2.8.31.5 Return Values

Table 121: Return Values for EEEWrite()

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

2.8.31.6 Troubleshooting

The troubleshooting mentioned below comprises of the input based errors handling.

Table 122: Troubleshooting for EEEWrite()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_RANGE	Size or destination or end address for writing exceeds the valid range	The size or destination or end address provided should be within the valid range
FTFx_ERR_NOEEE	EERAM is not available for EEE	Enable EEE by calling SetEEEEnable() function
FTFx_ERR_PVIOL	EEPROM is protected for write	Call EERAMSetProtection() function to unprotect it, then call EEEWrite() again.

2.8.31.7 Affected Registers

Table 123: Registers affected in EEEWrite()

Register	Bits	Action
FCNFG	EEERDY	Read
FSTAT	FPVIOL	Read

2.8.31.8 Comments

None.

2.8.31.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.32 PFlashGetSwapStatus()

2.8.32.1 Overview

This function is used to retrieve status of the swap system at runtime.

2.8.32.2 Activation

This API will be invoked by user application.

2.8.32.3 Prototype

```
UINT32 PFlashGetSwapStatus (PFLASH_SSD_CONFIG PSSDConfig, \
                            UINT32 flashAddress, \
                            UINT8* pCurrentSwapMode, \
                            UINT8* pCurrentSwapBlockStatus, \
                            UINT8* pNextSwapBlockStatus, \
                            pFLASHCOMMANDSEQUENCE FlashCommandSequence)
```

2.8.32.4 Arguments

Table 124: Arguments for PFlashGetSwapStatus()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
flashAddress	UINT32	Address of swap indicator. In case PFlashSwap has never been called, use any valid value.
pCurrentSwapMode	UINT8*	Returned current swap mode. Possible values are: ▪ FTFx_SWAP_UNINIT ▪ FTFx_SWAP_INIT ▪ FTFx_SWAP_UPDATE (*) ▪ FTFx_SWAP_UPDATE_ERASED (*) ▪ FTFx_SWAP_COMPLETE (*) – Only abnormal swap operation can cause swap system having these modes.
pCurrentSwapBlockStatus	UINT8*	Returned current swap block status. If the logical lower half of P-Flash block is active, 0 will be returned. Otherwise 1 will be returned.
pNextSwapBlockStatus	UINT8*	Returned next swap block status. If the logical lower half of P-Flash block will be the active block after performing swap and reset, 0 will be returned. Otherwise 1 will be returned.
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.32.5 Return Values

Table 125: Return Values for PFlashGetSwapStatus()

Type	Description	Possible Values
UINT32	Successful completion	FTFx_OK
	Error value	FTFx_ERR_ADDR FTFx_ERR_RANGE

2.8.32.6 Troubleshooting

The troubleshooting mentioned below comprises the input based errors handling

Table 126: Troubleshooting for PFlashGetSwapStatus()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ADDR	Input flash address is not longword aligned	Input flash address should be longword aligned.
FTFx_ERR_RANGE	Input flash address is ▪ not in the lower half of P-Flash block. ▪ in Flash Configuration Field of lower half P flash block.	Input flash address should be in the lower half of P-Flash block, but outside the Flash Configuration Field.

2.8.32.7 Affected Registers

Table 127: Registers affected in PFlashGetSwapStatus()

Register	Bits	Action
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4	[7..0]	Write
FCCOB5, FCCOB6, FCCOB7	[7..0]	Read

2.8.32.8 Comments

Currently, the FTFx_FX_256K_32K_2K_1K_1K, FTFx_KX_512K_512K_16K_4K_4K and FTFx_KX_1024K_0K_16K_4K_0K derivatives support swap feature.

2.8.32.9 Assumptions

'FlashInit()' must be called before invoking this function.

2.8.33 PFlashSwap()

2.8.33.1 Overview

This function is used to swap the two logical P-Flash memory blocks within the memory map.

2.8.33.2 Activation

This API will be invoked by user application.

2.8.33.3 Prototype

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

2.8.33.4 Arguments

Table 128: Arguments for PFlashSwap()

Argument	Type	Description
PSSDConfig	PFLASH_SSD_CONFIG	The SSD configuration structure pointer.
flashAddress	UINT32	Address for storing swap indicator.
pSwapCallBack	PFLASH_SWAP_CALLBACK	Callback to report swap status while the swapping is being performed.
FlashCommandSequence	pFLASHCOMMANDSEQUENCE	Pointer to the flash command sequence function.

2.8.33.5 Return Values

Table 129: Return Values for PFlashSwap()

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

2.8.33.6 Troubleshooting

The troubleshooting mentioned below comprises of hardware errors due to swap command. Apart from these, the input based errors handling is also mentioned.

Table 130: Troubleshooting for PFlashSwap()

Returned Error Bits	Possible causes	Solution
FTFx_ERR_ADDR	Input flash address is not longword aligned	Input flash address should be longword aligned.
FTFx_ERR_RANGE	Input flash address is ▪ not in the lower half of P-Flash block. ▪ in Flash Configuration Field	Input flash address should be in the lower half of P-Flash block, but outside the Flash Configuration Field.
FTFx_ERR_ACCERR	Flash address does not match the swap indicator address in IFR	The flash address provided to function must be the same with the one when call this function at the first time.
FTFx_ERR_MGSTAT0	Any error encounting during swap determination and program verify operation or any brownout detected during swap determination.	Hardware error.

2.8.33.7 Affected Registers

Table 131: Registers affected in PFlashSwap()

Register	Bits	Action
FCCOB0, FCCOB1, FCCOB2, FCCOB3, FCCOB4	[7..0]	Write
FCCOB5, FCCOB6, FCCOB7	[7..0]	Read

2.8.33.8 Comments

Currently, the FTFx_FX_256K_32K_2K_1K_1K, FTFx_KX_512K_512K_16K_4K_4K, FTFx_KX_1024K_0K_16K_4K_0K, FTFx_KX_64K_32K_2K_2K_1K, FTFx_KX_128K_32K_2K_2K_1K and FTFx_KX_256K_32K_2K_2K_1K derivatives support swap feature.

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 callback. Below is an example to show how to implement a swap callback. Also, if user does not want to use the swap callback parameter, just use NULL_CALLBACK as a null pointer.

```
BOOL PFlashSwapCallback(UINT8 currentSwapMode)
{
    switch (currentSwapMode)
    {
    case FTFx_SWAP_UNINI:
        /* Put your application-specific code here */
        break;

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

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

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

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

    default:
        break;
    }

    return TRUE; /* Return FALSE to stop swapping */
}
```

2.8.33.9 Assumptions

'FlashInit()' must be called before invoking this function.

APPENDIX A: PERFORMANCE DATA

A.1 Code Size and Stack Usage

Table 132: Code Size and Stack Usage for FTFx_KX_256K_256K_4K_2K_2K derivative

API Name	Code Size (In Bytes)	Stack Usage

FlashInit()	436	16
DEFlashPartition()	56	28
DFlashGetProtection()	38	12
DFlashSetProtection()	88	20
EEEWrite()	98	20
EERAMGetProtection()	40	12
EERAMSetProtection()	90	20
FlashCheckSum()	138	32
FlashCommandSequence()	282	24
FlashEraseAllBlock()	42	16
FlashEraseBlock()	138	20
FlashEraseResume()	22	8
FlashEraseSector()	200	32
FlashEraseSuspend()	38	8
FlashGetInterruptEnable()	40	12
FlashGetSecurityState()	58	12
FlashProgramCheck()	280	48
FlashProgramLongWord()	226	36
FlashProgramOnce()	128	32
FlashProgramSection()	194	36
FlashReadOnce()	118	28
FlashReadResource()	160	32
FlashSecurityByPass()	114	32
FlashSetInterruptEnable()	36	12
FlashVerifyAllBlock()	46	20
FlashVerifyBlock()	154	28
FlashVerifySection()	176	40
PFlashGetProtection()	66	24
PFlashSetProtection()	22	36
SetEEEnable()	48	16

Table 133: Code Size and Stack Usage for FTFx_KX_512K_0K_0K_2K_0K derivative

API Name	Code Size (In Bytes)	Stack Usage
FlashInit()	26	8
FlashCheckSum()	138	32
FlashCommandSequence()	282	24
FlashEraseAllBlock()	42	16
FlashEraseBlock()	138	20
FlashEraseResume()	22	8
FlashEraseSector()	200	32
FlashEraseSuspend()	38	8
FlashGetInterruptEnable()	40	12
FlashGetSecurityState()	58	12
FlashProgramCheck()	280	48
FlashProgramLongWord()	226	36
FlashProgramOnce()	128	32

FlashProgramSection()	194	36
FlashReadOnce()	118	28
FlashReadResource()	160	32
FlashSecurityByPass()	114	32
FlashSetInterruptEnable()	36	12
FlashVerifyAllBlock()	46	20
FlashVerifyBlock()	154	28
FlashVerifySection()	176	40
PFlashGetProtection()	66	24
PFlashSetProtection()	22	36

Table 134: Code Size and Stack Usage for FTFx_JX_128K_32K_2K_1K_1K derivative

API Name	Code Size (In Bytes)	Stack Usage
FlashInit()	676	10
DEFlashPartition()	86	20
DFlashGetProtection()	58	12
DFlashSetProtection()	138	11
EEEWrite()	194	20
EERAMGetProtection()	66	12
EERAMSetProtection()	146	11
FlashCheckSum()	256	32
FlashCommandSequence()	542	15
FlashEraseAllBlock()	62	13
FlashEraseBlock()	230	20
FlashEraseResume()	64	8
FlashEraseSector()	318	32
FlashEraseSuspend()	64	8
FlashGetInterruptEnable()	70	9
FlashGetSecurityState()	92	9
FlashProgramCheck()	458	45
FlashProgramLongWord()	378	36
FlashProgramOnce()	206	26
FlashProgramSection()	314	28
FlashReadOnce()	214	20
FlashReadResource()	244	29
FlashSecurityByPass()	186	29
FlashSetInterruptEnable()	84	6
FlashVerifyAllBlock()	74	15
FlashVerifyBlock()	242	22
FlashVerifySection()	290	34
PFlashGetProtection()	102	24
PFlashSetProtection()	386	36
SetEEEnable()	64	11

Table 135: Code Size and Stack Usage for FTFx_FX_256K_32K_2K_1K_1K derivative

API Name	Code Size (In Bytes)	Stack Usage
FlashInit()	676	10
DEFlashPartition()	86	20
DFlashGetProtection()	58	12
DFlashSetProtection()	138	11
EEEWrite()	194	20
EERAMGetProtection()	66	12
EERAMSetProtection()	146	11
FlashCheckSum()	256	32
FlashCommandSequence()	542	15
FlashEraseAllBlock()	62	13
FlashEraseBlock()	230	20
FlashEraseResume()	64	8
FlashEraseSector()	318	32
FlashEraseSuspend()	64	8
FlashGetInterruptEnable()	70	9
FlashGetSecurityState()	92	9
FlashProgramCheck()	458	45
FlashProgramLongWord()	378	36
FlashProgramOnce()	206	26
FlashProgramSection()	314	32
FlashReadOnce()	214	20
FlashReadResource()	244	29
FlashSecurityByPass()	186	29
FlashSetInterruptEnable()	84	6
FlashVerifyAllBlock()	74	15
FlashVerifyBlock()	242	22
FlashVerifySection()	290	34
PFlashGetProtection()	102	24
PFlashSetProtection()	386	36
PFlashGetSwapStatus	270	36
PFlashSwap	632	17
SetEEEnable()	64	11

Table 136: Code Size and Stack Usage for FTFx_KX_512K_512K_16K_4K_4K derivative

API Name	Code Size (In Bytes)	Stack Usage
FlashInit()	476	16
DEFlashPartition()	66	28
DFlashGetProtection()	44	12
DFlashSetProtection()	104	20
EEEWrite()	118	20
EERAMGetProtection()	50	12
EERAMSetProtection()	110	20

FlashCheckSum()	134	32
FlashCommandSequence()	282	24
FlashEraseAllBlock()	46	16
FlashEraseBlock()	138	20
FlashEraseResume()	46	8
FlashEraseSector()	180	32
FlashEraseSuspend()	48	8
FlashGetInterruptEnable()	44	12
FlashGetSecurityState()	60	12
FlashProgramCheck()	254	48
FlashProgramPhrase()	228	36
FlashProgramOnce()	184	32
FlashProgramSection()	190	36
FlashReadOnce()	174	28
FlashReadResource()	198	32
FlashSecurityByPass()	186	32
FlashSetInterruptEnable()	42	12
FlashVerifyAllBlock()	54	20
FlashVerifyBlock()	146	28
FlashVerifySection()	174	40
PFlashGetProtection()	62	24
PFlashSetProtection()	232	36
PFlashGetSwapStatus	270	36
PFlashSwap	632	20
SetEEEnable()	54	16

Table 137: Code Size and Stack Usage for FTFx_KX_128K_32K_2K_1K_1K derivative

API Name	Code Size (In Bytes)	Stack Usage
FlashInit()	472	16
DEFlashPartition()	56	28
DFlashGetProtection()	34	12
DFlashSetProtection()	94	20
EEEWrite()	110	20
EERAMGetProtection()	40	12
EERAMSetProtection()	100	20
FlashCheckSum()	124	32
FlashCommandSequence()	282	24
FlashEraseAllBlock()	36	16
FlashEraseBlock()	128	20
FlashEraseResume()	36	8
FlashEraseSector()	186	32
FlashEraseSuspend()	38	8
FlashGetInterruptEnable()	34	12
FlashGetSecurityState()	50	12
FlashProgramCheck()	254	48
FlashProgramLongWord()	198	36

FlashProgramOnce()	126	32
FlashProgramSection()	240	36
FlashReadOnce()	116	28
FlashReadResource()	164	32
FlashSecurityByPass()	186	32
FlashSetInterruptEnable()	32	12
FlashVerifyAllBlock()	44	20
FlashVerifyBlock()	136	28
FlashVerifySection()	224	40
PFlashGetProtection()	52	24
PFlashSetProtection()	224	36
SetEEEnable()	44	16

A.2 Write/Erase Times

Table 138: Write/Erase Times for FTFx_KX_256K_256K_4K_2K_2K derivative

Operation	Time (ms) Flash clock = 25 MHz
EEE Write (FTFx_WORD_SIZE)	0.5
Flash Erase All Block (EEPROM enabled)	294
Flash Erase All Block (EEPROM disabled)	113
D Flash Erase Block	56
P Flash Erase Block	56
D Flash Erase Sector (FTFx_SECTOR_SIZE)	41
P Flash Erase Sector (FTFx_SECTOR_SIZE)	41
D Flash Program Check (FTFx_LONGWORD_SIZE)	0.044
P Flash Program Check (FTFx_LONGWORD_SIZE)	0.044
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.072
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.072
D Flash Program Section	0.113
P Flash Program Section	0.113
Flash Verify All Block	2.1
D Flash Verify Block	1.4
P Flash Verify Block	1.4
D Flash Verify Section	0.042
P Flash Verify Section	0.041
D Flash Check Sum (FTFx_LONGWORD_SIZE)	0.002
P Flash Check Sum (FTFx_LONGWORD_SIZE)	0.002

Table 139: Write/Erase Times for FTFx_KX_512K_0K_0K_2K_0K derivative

Operation	Time (ms) Flash clock = 25 MHz
Flash Erase All Block	71
P Flash Erase Block	34
P Flash Erase Sector (FTFx_SECTOR_SIZE)	31
P Flash Program Check (FTFx_LONGWORD_SIZE)	0.025
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.033

P Flash Program Section (1 phrase)	0.064
Flash Verify All Block	1.2
P Flash Verify Block	0.83
P Flash Verify Section	0.019
P Flash Check Sum (FTFx_LONGWORD_SIZE)	0.001

Table 140: Write/Erase Times for FTFx_JX_128K_32K_2K_1K_1K derivative

Operation	Time (ms) Flash clock = 25 MHz
EEE Write (FTFx_WORD_SIZE)	1.008
Flash Erase All Block (EEPROM enabled)	46.842
Flash Erase All Block (EEPROM disabled)	49.048
D Flash Erase Block	7.797
P Flash Erase Block	38.737
D Flash Erase Sector (FTFx_SECTOR_SIZE)	6.427
P Flash Erase Sector (FTFx_SECTOR_SIZE)	6.297
D Flash Program Check (FTFx_LONGWORD_SIZE)	0.110
P Flash Program Check (FTFx_LONGWORD_SIZE)	1.049
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.988
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.916
D Flash Program Section	1.080
P Flash Program Section	0.992
Flash Verify All Block	2.076
D Flash Verify Block	0.897
P Flash Verify Block	2.456
D Flash Verify Section	0.513
P Flash Verify Section	0.472
D Flash Check Sum (FTFx_LONGWORD_SIZE)	0.125
P Flash Check Sum (FTFx_LONGWORD_SIZE)	0.117

Table 141: Write/Erase Times for FTFx_FX_256K_32K_2K_1K_1K derivative

Operation	Time (ms) Flash clock = 25 MHz
EEE Write (FTFx_WORD_SIZE)	0.74304
Flash Erase All Block (EEPROM enabled)	111.56544
Flash Erase All Block (EEPROM disabled)	114.6384
D Flash Erase Block	11.28608
P Flash Erase Block	51.7296
D Flash Erase Sector (FTFx_SECTOR_SIZE)	10.04416
P Flash Erase Sector (FTFx_SECTOR_SIZE)	6.63456
D Flash Program Check (FTFx_LONGWORD_SIZE)	0.84768
P Flash Program Check (FTFx_LONGWORD_SIZE)	0.84352
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.7808
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.64704
D Flash Program Section	0.72704
P Flash Program Section	0.62944

Flash Verify All Block	2.4384
D Flash Verify Block	0.52864
P Flash Verify Block	2.05728
D Flash Verify Section	0.13088
P Flash Verify Section	0.088
D Flash Check Sum (FTFx_LONGWORD_SIZE)	0.92064
P Flash Check Sum (FTFx_LONGWORD_SIZE)	0.91296

Table 142: Write/Erase Times for FTFx_KX_512K_512K_16K_4K_4K derivative

Operation	Time (ms) Flash clock = 25 MHz
EEE Write (FTFx_WORD_SIZE)	0.246
Flash Erase All Block (EEPROM enabled)	0.9407
Flash Erase All Block (EEPROM disabled)	1.97303
D Flash Erase Block	0.811112
P Flash Erase Block	1.937384
D Flash Erase Sector (FTFx_SECTOR_SIZE)	0.211787
P Flash Erase Sector (FTFx_SECTOR_SIZE)	0.19234
D Flash Program Check (FTFx_PHRASE_SIZE)	0.127864
P Flash Program Check (FTFx_PHRASE_SIZE)	0.13296
D Flash Program Phrase (FTFx_PHRASE_SIZE)	0.112144
P Flash Program Phrase (FTFx_PHRASE_SIZE)	0.115488
D Flash Program Section (1 phrase)	0.224256
P Flash Program Section (1 phrase)	0.231504
Flash Verify All Block	2.090368
D Flash Verify Block	0.675696
P Flash Verify Block	0.6785
D Flash Verify Section (1 phrase)	0.04552
P Flash Verify Section (1 phrase)	0.047816
D Flash Check Sum (FTFx_PHRASE_SIZE)	0.012304
P Flash Check Sum (FTFx_PHRASE_SIZE)	0.011904

Table 143: Write/Erase Times for FTFx_KX_128K_32K_2K_1K_1K derivative

Operation	Time (ms) Flash clock = 25 MHz
EEE Write (FTFx_WORD_SIZE)	0.32308
Flash Erase All Block (EEPROM enabled)	60.58196
Flash Erase All Block (EEPROM disabled)	59.91128
D Flash Erase Block	23.22524
P Flash Erase Block	31.38924
D Flash Erase Sector (FTFx_SECTOR_SIZE)	5.70276
P Flash Erase Sector (FTFx_SECTOR_SIZE)	4.68464
D Flash Program Check (FTFx_LONGWORD_SIZE)	0.07776
P Flash Program Check (FTFx_LONGWORD_SIZE)	0.07668
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.07808
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	0.07756
D Flash Program Section	0.10092

P Flash Program Section	0.09824
Flash Verify All Block	1.3718
D Flash Verify Block	0.3702
P Flash Verify Block	1.36872
D Flash Verify Section	0.04804
P Flash Verify Section	0.04536
D Flash Check Sum (FTFx_LONGWORD_SIZE)	0.0078
P Flash Check Sum (FTFx_LONGWORD_SIZE)	0.00716

A.3 CallBack Time Period

Table 144: CallBack Time Period for FTFx_KX_256K_256K_4K_2K_2K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	435
D Flash Erase Block	435
P Flash Erase Block	435
D Flash Erase Sector (FTFx_SECTOR_SIZE)	435
P Flash Erase Sector (FTFx_SECTOR_SIZE)	435
D Flash Program Check (FTFx_LONGWORD_SIZE)	435
P Flash Program Check (FTFx_LONGWORD_SIZE)	435
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	435
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	435
D Flash Program Section	435
P Flash Program Section	435
Flash Verify All Block	435
D Flash Verify Block	435
P Flash Verify Block	435
D Flash Verify Section	435
P Flash Verify Section	435
D Flash Check Sum (FTFx_LONGWORD_SIZE)	435
P Flash Check Sum (FTFx_LONGWORD_SIZE)	435

Table 145: CallBack Time Period for FTFx_KX_512K_0K_0K_2K_0K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	310
P Flash Erase Block	310
P Flash Erase Sector (FTFx_SECTOR_SIZE)	310
P Flash Program Check (FTFx_LONGWORD_SIZE)	310
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	310
P Flash Program Section (1 phrase)	310
Flash Verify All Block	310
P Flash Verify Block	310
P Flash Verify Section	310

P Flash Check Sum (FTFx_LONGWORD_SIZE)	280

Table 146: CallBack Time Period for FTFx_JX_128K_32K_2K_1K_1K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	1760
D Flash Erase Block	1760
P Flash Erase Block	1760
D Flash Erase Sector (FTFx_SECTOR_SIZE)	1760
P Flash Erase Sector (FTFx_SECTOR_SIZE)	1760
D Flash Program Check (FTFx_LONGWORD_SIZE)	1760
P Flash Program Check (FTFx_LONGWORD_SIZE)	1760
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	1760
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	1760
D Flash Program Section	1760
P Flash Program Section	1760
Flash Verify All Block	1760
D Flash Verify Block	1760
P Flash Verify Block	1760
D Flash Verify Section	1760
P Flash Verify Section	1760
D Flash Check Sum (FTFx_LONGWORD_SIZE)	2560
P Flash Check Sum (FTFx_LONGWORD_SIZE)	2560

Table 147: CallBack Time Period for FTFx_FX_256K_32K_2K_1K_1K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	1760
D Flash Erase Block	1760
P Flash Erase Block	1760
D Flash Erase Sector (FTFx_SECTOR_SIZE)	1760
P Flash Erase Sector (FTFx_SECTOR_SIZE)	1760
D Flash Program Check (FTFx_LONGWORD_SIZE)	1760
P Flash Program Check (FTFx_LONGWORD_SIZE)	1760
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	1760
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	1760
D Flash Program Section	1760
P Flash Program Section	1760
Flash Verify All Block	1760
D Flash Verify Block	1760
P Flash Verify Block	1760
D Flash Verify Section	1760
P Flash Verify Section	1760
D Flash Check Sum (FTFx_LONGWORD_SIZE)	2680
P Flash Check Sum (FTFx_LONGWORD_SIZE)	2680

Table 148: CallBack Time Period for FTFx_KX_512K_512K_16K_4K_4K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	3.84
D Flash Erase Block	3.84
P Flash Erase Block	3.84
D Flash Erase Sector (FTFx_SECTOR_SIZE)	3.84
P Flash Erase Sector (FTFx_SECTOR_SIZE)	3.84
D Flash Program Check (FTFx_PHRASE_SIZE)	3.84
P Flash Program Check (FTFx_PHRASE_SIZE)	3.84
D Flash Program Phrase (FTFx_PHRASE_SIZE)	3.84
P Flash Program Phrase (FTFx_PHRASE_SIZE)	3.84
D Flash Program Section	3.84
P Flash Program Section	3.84
Flash Verify All Block	3.84
D Flash Verify Block	3.84
P Flash Verify Block	3.84
D Flash Verify Section	3.84
P Flash Verify Section	3.84
D Flash Check Sum (FTFx_PHRASE_SIZE)	4.28
P Flash Check Sum (FTFx_PHRASE_SIZE)	4.28

Table 149: CallBack Time Period for FTFx_KX_128K_32K_2K_1K_1K derivative

Operation	Time (ns) Flash clock = 25 MHz
Flash Erase All Block	3.68
D Flash Erase Block	3.68
P Flash Erase Block	3.68
D Flash Erase Sector (FTFx_SECTOR_SIZE)	3.68
P Flash Erase Sector (FTFx_SECTOR_SIZE)	3.68
D Flash Program Check (FTFx_LONGWORD_SIZE)	3.68
P Flash Program Check (FTFx_LONGWORD_SIZE)	3.68
D Flash Program Long Word (FTFx_LONGWORD_SIZE)	3.68
P Flash Program Long Word (FTFx_LONGWORD_SIZE)	3.68
D Flash Program Section	3.68
P Flash Program Section	3.68
Flash Verify All Block	3.68
D Flash Verify Block	3.68
P Flash Verify Block	3.68
D Flash Verify Section	3.68
P Flash Verify Section	3.68
D Flash Check Sum (FTFx_LONGWORD_SIZE)	4.16
P Flash Check Sum (FTFx_LONGWORD_SIZE)	4.16

Note:

1. Callback time period for '*FlashCheckSum*()' is measured with FLASH_CALLBACK_CS (CallBack function period for checksum) value as 1.