Boot FST

Module Design Specification

Version: 1.1.1RC Status: Draft

Date: 07-January-2021

Document Author: Amarnath C

Document List of Approvers:

Firmware Architect: Rahil Malhotra

Validation Architect: Mahaveer Addanki

Functional Safety Architect: Srikanth Kaniyanoor Srinivasan

Functional Safety Manager: Francesco Pingitore

Applicable Standard: ISO 26262 and IEC 61508

Template Version: FuSa\_Documentation\_Word\_Template.docx v0.5

FuSa Lifecycle Phase: FS2

Intel Top Secret

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: [http://](http://www.intel.com/design/literature.htm)www.intel.com/design/literature.htm

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at h[ttp://www.intel.com/](http://www.intel.com/) or from the OEM or retailer.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2021, Intel Corporation. All rights reserved.

Contents

[1.0 Introduction 9](#_Toc62480264)

[1.1 Purpose 9](#_Toc62480265)

[1.2 Audience 9](#_Toc62480266)

[1.3 Acronyms and Terminology 9](#_Toc62480267)

[1.4 Reference Documents 10](#_Toc62480268)

[2.0 Document Scope 12](#_Toc62480269)

[2.1 Prerequisite Documents 12](#_Toc62480270)

[2.2 In Scope 12](#_Toc62480271)

[2.3 Out of Scope 12](#_Toc62480272)

[2.4 Known Gaps/Opens 12](#_Toc62480273)

[2.5 ISO26262 and IEC61508 Fulfillment Matrix 12](#_Toc62480274)

[2.5.1 ISO26262 Fulfillment Matrix 12](#_Toc62480275)

[2.5.2 IEC61508 Fulfillment Matrix 13](#_Toc62480276)

[3.0 Global Design Decisions 14](#_Toc62480277)

[4.0 Functional Description 15](#_Toc62480278)

[4.1 Overview 15](#_Toc62480279)

[4.2 Type Definitions 16](#_Toc62480280)

[4.2.1 Local type definitions 16](#_Toc62480281)

[4.3 Configuration Parameters 24](#_Toc62480282)

[4.4 API Functions 25](#_Toc62480283)

[4.4.1 Boot FST initialization 25](#_Toc62480284)

[4.4.2 Boot FST stop 26](#_Toc62480285)

[4.4.3 Boot FST application 28](#_Toc62480286)

[4.5 Interrupt Handler 31](#_Toc62480287)

[4.6 Local functions 31](#_Toc62480288)

[4.6.1 Boot FST timer expiry callback function 31](#_Toc62480289)

[4.6.2 Boot FST bios boot callback function 32](#_Toc62480290)

[4.6.3 Boot FST POSC results callback function 34](#_Toc62480291)

[4.6.4 Boot FST STL results callback function 36](#_Toc62480292)

[4.6.5 Boot FST ODCC SS callback function 38](#_Toc62480293)

[4.6.6 Boot FST override config param callback function 40](#_Toc62480294)

[4.6.7 Clock monitor Error Injection(EI) test 42](#_Toc62480295)

[4.6.8 Clock monitor pre-mask status check 45](#_Toc62480296)

[4.6.9 Clock monitor count done status check 46](#_Toc62480297)

[4.6.10 Clock monitor PPM range status check 48](#_Toc62480298)

[4.6.11 Clock monitor count error injection test 50](#_Toc62480299)

[4.6.12 Clock monitor lock error injection test 54](#_Toc62480300)

[4.6.13 Clock monitor dead clock error injection test 56](#_Toc62480301)

[4.6.14 Root parity Error Injection(EI) test 59](#_Toc62480302)

[4.6.15 ROM ECC error injection test 62](#_Toc62480303)

[4.6.16 Boot FST frCPU check 63](#_Toc62480304)

[4.6.17 NVIC interrupt priority register check 65](#_Toc62480305)

[4.6.18 Async commands register 66](#_Toc62480306)

[4.6.19 Async commands de-register 68](#_Toc62480307)

[4.6.20 Host boot monitoring 70](#_Toc62480308)

[4.6.21 Host BIOS boot monitor 73](#_Toc62480309)

[4.6.22 POSC results packet validate 75](#_Toc62480310)

[4.6.23 POSC – get number of cores enabled 77](#_Toc62480311)

[4.6.24 POSC results respond 79](#_Toc62480312)

[4.6.25 POSC test results monitor 80](#_Toc62480313)

[4.6.26 RAVDM test 82](#_Toc62480314)

[4.6.27 RAVDM ping test 83](#_Toc62480315)

[4.6.28 RAVDM ITD parameters fetch 86](#_Toc62480316)

[4.6.29 PMC IPC test 88](#_Toc62480317)

[4.6.30 ODCC test – Dummy snapshot compare 91](#_Toc62480318)

[4.6.31 Override config param data set 93](#_Toc62480319)

[4.6.32 Override config param handle 95](#_Toc62480320)

[4.6.33 STL results packet validation 96](#_Toc62480321)

[4.6.34 Uncore STL execution check 98](#_Toc62480322)

[4.6.35 Core STL execution check 100](#_Toc62480323)

[4.6.36 System time set 101](#_Toc62480324)

[4.6.37 STL results respond 103](#_Toc62480325)

[4.6.38 Host startup STL results monitor 104](#_Toc62480326)

[4.7 Imported functions 106](#_Toc62480327)

[4.7.1 GP timer expiry callback function 106](#_Toc62480328)

[4.7.2 Periodic timer expiry callback function 108](#_Toc62480329)

[4.7.3 RTOS timer error handler 109](#_Toc62480330)

[4.7.4 Timer Error Injection(EI) test 110](#_Toc62480331)

[4.7.5 WDT alarm mask and wdt start 113](#_Toc62480332)

[4.7.6 WDT configure and start 114](#_Toc62480333)

[4.7.7 WDT difference error handle 116](#_Toc62480334)

[4.7.8 WDT error handler 117](#_Toc62480335)

[4.7.9 WDT NOK status error handle 118](#_Toc62480336)

[4.7.10 WDT test 119](#_Toc62480337)

[4.7.11 GP timer difference calculation 122](#_Toc62480338)

[4.7.12 GP timer test 123](#_Toc62480339)

[4.7.13 Periodic timer difference calculation 126](#_Toc62480340)

[4.7.14 Periodic timer test 127](#_Toc62480341)

[4.7.15 FMM Scratchpad register test 129](#_Toc62480342)

[4.7.16 Multiplier and Divider check 131](#_Toc62480343)

[4.7.17 Bus Matrix check 134](#_Toc62480344)

[4.7.18 MPU and NVIC registers read 138](#_Toc62480345)

[4.8 Imported Types 139](#_Toc62480346)

[5.0 Data Description 141](#_Toc62480347)

Figures

[Figure 1: Boot FST initialization flow 27](#_Toc62480348)

[Figure 2: Boot FST stop flow 29](#_Toc62480349)

[Figure 3: Boot FST app flow 31](#_Toc62480350)

[Figure 4: Boot fst Timer expiry callback function flow 33](#_Toc62480351)

[Figure 5 : Bios boot complete callback function flow 35](#_Toc62480352)

[Figure 6 : POSC results callback function flow 37](#_Toc62480353)

[Figure 7 : STL results callback function flow 39](#_Toc62480354)

[Figure 8 : ODCC ss callback function flow 41](#_Toc62480355)

[Figure 9 : Override config data callback function flow 43](#_Toc62480356)

[Figure 10: Clock monitor error injection check function flow 45](#_Toc62480357)

[Figure 11 : Clock monitor pre-mask status check function flow 47](#_Toc62480358)

[Figure 12 : Clock monitor count done status function flow 49](#_Toc62480359)

[Figure 13 : Clock monitor PPM range status check function flow 51](#_Toc62480360)

[Figure 14: Clock monitor count error injection function flow 54](#_Toc62480361)

[Figure 15: Clock monitor lock error injection function flow 57](#_Toc62480362)

[Figure 16: Clock monitor dead clock error injection function flow 59](#_Toc62480363)

[Figure 17: Root parity error injection check function flow 62](#_Toc62480364)

[Figure 18: Boot FST frCPU check function flow 65](#_Toc62480365)

[Figure 19: NVIC interrupt priority register check flow 67](#_Toc62480366)

[Figure 20 : Async commands register function flow 69](#_Toc62480367)

[Figure 21 : Async commands de-register function flow 71](#_Toc62480368)

[Figure 22: Host monitoring application thread function flow 74](#_Toc62480369)

[Figure 23: Host BIOS boot monitor overall flow diagram 76](#_Toc62480370)

[Figure 24 : POSC results packet validate function flow 78](#_Toc62480371)

[Figure 25 : POSC- get number of cores function flow 80](#_Toc62480372)

[Figure 26: Host POSC results monitor flow diagram 83](#_Toc62480373)

[Figure 27: RAVDM check function flow 84](#_Toc62480374)

[Figure 28: RAVDM ping test function flow 86](#_Toc62480375)

[Figure 29: Fetch ITD parameters function flow 88](#_Toc62480376)

[Figure 30: PMC IPC test flow diagram 91](#_Toc62480377)

[Figure 31: ODCC dummy snapshot compare flow diagram 93](#_Toc62480378)

[Figure 32 : Override config data set 95](#_Toc62480379)

[Figure 33 : Override config param function flow 97](#_Toc62480380)

[Figure 34 : STL results packet validation function flow 99](#_Toc62480381)

[Figure 35 : Uncore STL execution check function flow 100](#_Toc62480382)

[Figure 36 : Core STL execution check function flow 102](#_Toc62480383)

[Figure 37 : Syatem time set function flow 104](#_Toc62480384)

[Figure 38: Host boot STLs monitor flow diagram 107](#_Toc62480385)

[Figure 39: GP Timer expiry callback function flow 108](#_Toc62480386)

[Figure 40 : Periodic Timer expiry callback function flow 110](#_Toc62480387)

[Figure 41: RTOS timer test flow diagram 113](#_Toc62480388)

[Figure 42 : WDT alarm mask and wdt start function 115](#_Toc62480389)

[Figure 43 : WDT configure and start function flow 117](#_Toc62480390)

[Figure 44: WDT test flow diagram 122](#_Toc62480391)

[Figure 45 : GP timer difference calculation function flow 124](#_Toc62480392)

[Figure 46: GP timer test flow diagram 126](#_Toc62480393)

[Figure 47 : Periodic timer difference calculation function flow 128](#_Toc62480394)

[Figure 48: Periodic timer test flow diagram 130](#_Toc62480395)

[Figure 49: FMM scratchpad register check flow diagram 132](#_Toc62480396)

[Figure 50: frCPU Multipler and Divider check flow 134](#_Toc62480397)

[Figure 51: frCPU bus matrix check flow 138](#_Toc62480398)

[Figure 52: frCPU MPU and NVIC register read function flow 140](#_Toc62480399)

Tables

[Table 1 : Terminology 9](#_Toc62480400)

[Table 2 : Reference Documents 10](#_Toc62480401)

[Table 3 : Open & Follow up Needs 12](#_Toc62480402)

[Table 4 : ISO26262 Fulfillment Matrix 12](#_Toc62480403)

[Table 5 : IEC61508 Fulfillment Matrix 13](#_Toc62480404)

[Table 6 : Boot FST PCIE message format for BIOS boot complete 16](#_Toc62480405)

[Table 7 : Boot FST PCIE message format for POSC test results 17](#_Toc62480406)

[Table 8 : Boot FST PCIE message format for startup STL results 18](#_Toc62480407)

[Table 9 : Boot FST PCIE response message format 19](#_Toc62480408)

[Table 10 : Boot FST PCIE message format for ODCC snapshot 19](#_Toc62480409)

[Table 11 : Boot FST PCIE response message format for ODCC snapshot 20](#_Toc62480410)

[Table 12 : Boot FST PCIE message format for override config data 20](#_Toc62480411)

[Table 13 : Boot FST PCIE response message format for override config data 21](#_Toc62480412)

[Table 14 : Boot FST PUINT request message format 22](#_Toc62480413)

[Table 15 : Boot FST PUINT response message format 23](#_Toc62480414)

[Table 16 : Boot FST PUINT ITD response data 0 23](#_Toc62480415)

[Table 17 : Boot FST PUINT ITD response data 1 23](#_Toc62480416)

[Table 18 : Boot FST PUINT ITD response data 2 24](#_Toc62480417)

[Table 19 : Configuaration parameters 24](#_Toc62480418)

[Table 20 : Boot FST initialization API 25](#_Toc62480419)

[Table 21 : Boot FST stop API 26](#_Toc62480420)

[Table 22 : Boot FST application thread function 28](#_Toc62480421)

[Table 23 : Boot fst timer expiry callback function 31](#_Toc62480422)

[Table 24 : Boot fst bios boot callback function 32](#_Toc62480423)

[Table 25 : Boot fst POSC results callback function 34](#_Toc62480424)

[Table 26 : Boot fst STL results callback function 36](#_Toc62480425)

[Table 27 : Boot fst ODCC SS callback function 38](#_Toc62480426)

[Table 28 : Boot fst override config param callback function 40](#_Toc62480427)

[Table 29 : Clock monitor error injection check function 42](#_Toc62480428)

[Table 30 : Clock monitor pre-mask status check function 45](#_Toc62480429)

[Table 31 : Clock monitor count done status check function 46](#_Toc62480430)

[Table 32 : Clock monitor PPM range status check function 48](#_Toc62480431)

[Table 33 : Clock monitor count error injection function 50](#_Toc62480432)

[Table 34 : Clock monitor lock error injection function 54](#_Toc62480433)

[Table 35 : Clock monitor dead clock error injection function 56](#_Toc62480434)

[Table 36 : Root parity error injection check function 59](#_Toc62480435)

[Table 37 : ROM ECC error injection check function 62](#_Toc62480436)

[Table 38 : Boot FST frCPU check function 63](#_Toc62480437)

[Table 39 : frCPU NVIC interrupt priority register check function 65](#_Toc62480438)

[Table 40 : Async commands register function 66](#_Toc62480439)

[Table 41 : Async commands de-register function 68](#_Toc62480440)

[Table 42 : Host monitoring application thread function 70](#_Toc62480441)

[Table 43 : Host BIOS boot monitor function 73](#_Toc62480442)

[Table 44 : POSC results packet validate function 75](#_Toc62480443)

[Table 45 : POSC – get number of cores enabled function 77](#_Toc62480444)

[Table 46 : POSC results respond function 79](#_Toc62480445)

[Table 47 : POSC test results monitor function 80](#_Toc62480446)

[Table 48 : RAVDM check function 82](#_Toc62480447)

[Table 49 : RAVDM ping test function 83](#_Toc62480448)

[Table 50 : Fetch ITD parameters function 86](#_Toc62480449)

[Table 51 : PMC IPC check function 88](#_Toc62480450)

[Table 52 : ODCC dummy snapshot compare 91](#_Toc62480451)

[Table 53 : Override config param data set function 93](#_Toc62480452)

[Table 54 Override config param handle function 95](#_Toc62480453)

[Table 55 : STL results packet validation function 96](#_Toc62480454)

[Table 56 : Uncore STL execution check function 98](#_Toc62480455)

[Table 57 : Core STL execution check function 100](#_Toc62480456)

[Table 58 : System time set function 101](#_Toc62480457)

[Table 59 : STL results respond function 103](#_Toc62480458)

[Table 60 : Host startup STL results monitor function 104](#_Toc62480459)

[Table 61 : RTOS timer error handler function 109](#_Toc62480460)

[Table 62 : Timer error injection check function 110](#_Toc62480461)

[Table 63 : WDT alarm mask and wdt start function 113](#_Toc62480462)

[Table 64 : WDT configure and start function 114](#_Toc62480463)

[Table 65 : WDT difference error handle function 116](#_Toc62480464)

[Table 66 : WDT error handler function 117](#_Toc62480465)

[Table 67 : WDT NOK status error handle function 118](#_Toc62480466)

[Table 68 : WDT check function 119](#_Toc62480467)

[Table 69 : GP timer difference calculation function 122](#_Toc62480468)

[Table 70 : GP timer check function 123](#_Toc62480469)

[Table 71 : Periodic timer difference calculation function 126](#_Toc62480470)

[Table 72 : Periodic timer check function 127](#_Toc62480471)

[Table 73 : FMM scratchpad register check function 129](#_Toc62480472)

[Table 74 : Multiplier and Divider check function 131](#_Toc62480473)

[Table 75 : Bus Matrix check function 134](#_Toc62480474)

[Table 76 : frCPU MPU and NVIC register read function 138](#_Toc62480475)

[Table 77 : ThreadX errors 140](#_Toc62480476)

[Table 78 : Global Data Description 141](#_Toc62480477)

[Table 79 : Macros Data Description 142](#_Toc62480478)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Author | Description |
| 05/21/18 | 0.30 | Amarnath C | Initial release for architectural and validation teams feedback |
| 06/27/18 | 0.31 | Amarnath C | Added flows for timer tests and host monitoring. |
| 11/02/18 | 0.50 | Amarnath C | Added more details |
| 12/14/18 | 0.55 | Amarnath C | Added more details |
| 01/28/19 | 0.60 | Amarnath C | 1. Updated PMC IPC test 2. Updated GP timer test |
| 02/08/19 | 0.8RC | Amarnath C | 1. Updated init, application and stop functions. 2. Added Clock monitor, Root parity and ROM ECC error injection tests. 3. Removed host monitor thread. 4. Added host monitor function, updated BIOS boot monitor, added POSC results monitor and updated startup STL results monitor. 5. Added periodic timer test and periodic timer expiry handle callback function. 6. Renamed RTOS timer test to Timer error injection test, updated details. 7. Updated details in WDT test. |
| 03/25/19 | 0.8RC | Amarnath C | Updated clock monitor test, root parity error injection test and RAVDM test. |
| 07/28/2020 | 1.0RC | Unmesh, Hanamshet | Updated flow diagrams and description for new implementation. |
| 01/07/2021 | 1.1RC | Amarnath C | Large functions are broken into small function, updated new functions added. |
| 25/02/2021 | 1.1.1RC | Kavya Rajesh Vembar | Updated clockmon error enjection test check flowchart |

# Introduction

## Purpose

This document describes the design details of platform boot FST. It is responsible for ensuring that the various Platform boot components are up.

SCI is a safety related sub-system present in platforms designed for industrial use such as Elkhart Lake based on which further discussions are carried out.

## Audience

The audience for this document is as follows:

* Safety auditors and reviewers for SCI
* Product design and development team for Elkhart Lake platform
* SCI firmware development and validation teams
* FUSA validation team for SCI

## Acronyms and Terminology

Table 1 : Terminology

| Term | Description |
| --- | --- |
| SCI | SCI stands for Sycamore Island, which is the 0.6 generation of Intel Safety Island. This conforms to IEC61508 standards and is meant primarily for industrial use-cases. |
| FST | Functional Safety Test |
| STL | Software Test Library |
| RAVDM | Register Access over Vendor Defined Message |
| FMM | Fault Management Module |

## Reference Documents

Table 2 : Reference Documents

| Document | Document No./Location |
| --- | --- |
| SCI Software Architecture Specification | <https://sharepoint.amr.ith.intel.com/sites/FUSA_GlobalDomain/ISI/Sycamore%20Island%20ISI%2006/Engineering/Specifications/Sycamore%20Island%20SAS%20v0.8.pdf> |
| JSL FuSa PM HAS for RAVDM formats | <https://sharepoint.gar.ith.intel.com/sites/jasperlake/Shared%20Documents/Arch/HAS/PowerManagement/HAS/FuSa/JSL_FuSa_PM_HAS.html> |
| fRCPU armcm3 SS A1.0 - SW and Application Safety Manual of fRCPU armcm3 product | <https://sharepoint.amr.ith.intel.com/sites/FUSA_GlobalDomain/ISI/Safety%20Island%20WIP/Yogitech%20ISI/fRCPU_armcm3_SS_A1.0.pdf> |
| Timer Service MDS | https://sharepoint.gar.ith.intel.com/sites/QSD/SitePages/Home.aspx?RootFolder=%2Fsites%2FQSD%2FShared%20Documents%2FISI%5Fsw%5Fcommon%5Fdata%2FTIMER%20MDS&FolderCTID=0x012000A4BA3FDD6692FE4DBE8956F8E2A8E438&View={7D387DC1-8317-4B5B-B1DB-E35DAE752468} |
| WDT, RTOS and GP timer MDS | https://sharepoint.gar.ith.intel.com/sites/QSD/SitePages/Home.aspx?RootFolder=%2Fsites%2FQSD%2FShared%20Documents%2FISI%5Fsw%5Fcommon%5Fdata%2FTIMER%20MDS%2FTIMER%20DRIVER%20MDS&FolderCTID=0x012000A4BA3FDD6692FE4DBE8956F8E2A8E438&View={7D387DC1-8317-4B5B-B1DB-E35DAE752468} |
| ISI diagnostic interface specifications | <https://sharepoint.amr.ith.intel.com/sites/FUSA_GlobalDomain/ISI-FSXT/ISI%20Gen%2005%20Work%20Products/Forms/AllItems.aspx?RootFolder=%2fsites%2fFUSA_GlobalDomain%2fISI-FSXT%2fISI%20Gen%2005%20Work%20Products%2fExternal%2fISI%20Diagnostic%20Interface&FolderCTID=0x0120002BFEC25E27A6C749AC3DA4BFEDC77CE0> |
| SCI HAS and SCI components HAS  SCI Clock Monitor MAS | https://sharepoint.amr.ith.intel.com/sites/FUSA\_GlobalDomain/ISI/Safety%20Island%20WIP/Forms/AllItems.aspx?RootFolder=%2Fsites%2FFUSA%5FGlobalDomain%2FISI%2FSafety%20Island%20WIP%2FGen%200%2E6%20%28EHL%2DMCC%29%2FIP%20TR%20WG%2FSCI%5FHAS%2FLatest |
|  |  |

# Document Scope

## Prerequisite Documents

* Sycamore Island Software Architecture Specification documentation

## In Scope

This document is intended to capture the design details of boot FST for SCI.

## Out of Scope

## Known Gaps/Opens

Table 3 : Open & Follow up Needs

|  |  |
| --- | --- |
| **Open and follow-up needs** | **Notes** |
| 1. The flow shown for Error Handling is temporary. 2. [TBD] sharing the VID table to periodic FST. |  |

## ISO26262 and IEC61508 Fulfillment Matrix

This section identifies the content of this document which fulfills the identified safety requirements from the “In Scope” Section.

### ISO26262 Fulfillment Matrix

Table 4 : ISO26262 Fulfillment Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S. No** | **Technique/Measure** | **ASILD** | **Used** | **Section numbers within this document** | **Details** |
| 1a | Natural language | ++ | Yes | All the figures from 1 to 15. | The complete document is written by combination of natural language and semi formal notations used for flow diagrams in the document. |
| 1c | Semi-formal notations | ++ | Yes | All the figures from 1 to 15. | Flow diagrams made in visio are using UML are semi formal methods. |

### IEC61508 Fulfillment Matrix

Table 5 : IEC61508 Fulfillment Matrix

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table A4 - Part 3** | | | | | | |
| **S. No** |  | **Technique/Measure** | **SIL 3** | **Used** | **Section numbers within this document** | **Details** |
| 1 |  | Semi-formal methods | HR | Yes | All the figures from 1 to 15. | Flow diagrams made in visio are using UML are semi-formal methods |
| 2 |  | Computer-aided design tools | R | Yes | All the figures from 1 to 15. | Visio is the computer aided design tool used for making flow diagrams. |
| 3 |  | Defensive programming | R | Yes | -- | Coding guidelines will impose restrictions to do defensive programming |
| 4 |  | Modular approach | HR | Yes | -- | The design of a module in broken down in smaller functional blocks(C functions, threads) responsible for performing a distinct functionality. |
| 5 |  | Design and coding standards | HR | Yes | -- | Coding guidelines address this measure |
| 6 |  | Structured programming | HR | Yes | -- | Coding guidelines address this measure |
| 7 |  | Use of trusted/verified software elements (if available) | HR | Yes | -- | Module uses APIs fromThreadX. ThreadX is compliant software developed as per IEC standards and qualified by TUV SUD or the same. |
| 8 |  | Forward traceability between the software safety requirements specification and software design | R | Yes | -- | Will be established in DNG |

# Global Design Decisions

Boot FST is the first FST to execute after SCI boot. The boot FST thread create and start is done by FSTM.

# Functional Description

## Overview

The Boot FST is launched by FSTM during boot and it is one of the first FSTs to run on SCI. This FST is responsible for carrying out:

1. Proof test

It always checks the state of diagnostic virtual wires to CSE, if it is asserted (done in previous boot cycle by SCI), Boot FST will follow proof test flow and wait for results of proof test from CSE. If the wire is not asserted, then Boot FST follows its normal boot flow starting with next step below.

TODO: Update once all the details are available.

1. Running various SCI Boot Safety Tests. These include:
   1. Timer functional test(GP timer and periodic timer)
   2. WDT functional test
   3. Timer fault injection test
   4. Clock monitor fault injection test
   5. Root parity fault injection test
   6. ROM ECC error injection test
   7. FMM Scratch pad test
   8. fRCPU Boot test
2. Host boot monitoring
   1. Boot complete from BIOS
      1. Host BIOS on its boot completion sends a message to SCI via PCIe MB. On receiving boot complete message SCI will send the acknowledgement with success response code to host. If host does not receive acknowledgement in time, host will resend the boot complete message to SCI. This message is used as trigger point to fetch the ITD parametrers from PUINT.
   2. POSC test results:
      1. Host sends the Pre-OS checker test results message to SCI. This message informs SCI about the number of cores from whom STL results to be expected. SCI will send the acknowledgement with success response code to host. If host does not receive acknowledgement in time, host will resend the POSC test results to SCI.
   3. RAVDM test
      1. Send ping command to check communication between SCI and PUNIT. Receive the ping command response from PUNIT.
      2. Also, send command to get ITD parameter and calculate the voltage values corresponding to temperature.
   4. PMC IPC test
      1. To test the PMC IPC communication, send the get platform transition information Query Command to PMC and receive the response.
   5. ODCC test
      1. Compare ODCC snapshots coming from host test application if it comes while boot FST is executing. Host can use this to test ODCC software safety mechanism. ODCC comparison should detect mismatch in incoming snapshots if incoming snapshots do not match.

TODO: Update once all the details are available.

* 1. Startup STL results
     1. On host boot complete, it sends the startup STL results to SCI from cores (as informed by POSC test results). SCI sends PST and DTI configurations to host safety application running startup STL in response to STL results.

If Boot FST finds out any error in any of these Host Boot Tests and SCI Boot STL, it will signal the FSTM about it. FSTM will assert NOK, invoke required error handling and not get SCI out of the Safe State.

Refer ISI diagnostic interface specifications for host message and response format.

Refer JSL FuSa PM HAS for RAVDM formats.

## Type Definitions

### Local type definitions

Table 6 : Boot FST PCIE message format for BIOS boot complete

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_bios\_boot\_msg |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for BIOS boot complete. | |
| Members: | uint32\_t version : 4U; | Version of specifications. |
| uint32\_t reserved\_1 : 4U; | Reserved. |
| uint32\_t reserved\_2 : 6U; | Reserved. |
| uint32\_t flags : 2U; | Flag for response bit and retry bit. |
| uint32\_t length : 8U; | Message length in DWORDs. |
| uint32\_t reserved\_3 : 8U; | Reserved. |
| uint32\_t seq\_num : 16U; | Sequence number. |
| uint32\_t wl\_address : 8U; | Workload address. |
| uint32\_t isi\_address : 8U; | ISI address. |
| uint32\_t crc : 16U; | CRC value. |
| uint32\_t command : 16U; | BIOS boot complete command. |

Table 7 : Boot FST PCIE message format for POSC test results

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_posc\_res\_msg |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for POSC test results. | |
| Members: | uint32\_t version : 4U; | Version of specifications. |
| uint32\_t reserved\_1 : 4U; | Reserved. |
| uint32\_t reserved\_2 : 6U; | Reserved. |
| uint32\_t flags : 2U; | Flag for response bit and retry bit. |
| uint32\_t length : 8U; | Message length in DWORDs. |
| uint32\_t reserved\_3 : 8U; | Reserved. |
| uint32\_t seq\_num : 16U; | Sequence number. |
| uint32\_t wl\_address : 8U; | Workload address. |
| uint32\_t isi\_address : 8U; | ISI address. |
| uint32\_t crc : 16U; | CRC value. |
| uint32\_t command : 16U; | POSC test results command. |
| uint32\_t posc\_res : 16U; | POSC result. |
| uint32\_t posc\_maj : 8U; | POSC major version. |
| uint32\_t posc\_min : 8U; | POSC minor version. |
| uint32\_t core\_mask\_1; | Core STL executed or not executed. |
| uint32\_t core\_mask\_2; | Core STL executed or not executed. |
|  | uint32\_t avx\_data\_valid; |  |
| uint32\_t sse\_freq; |  |
| uint32\_t avx\_freq; |  |
| uint32\_t avx2\_freq; |  |
| uint32\_t core\_mask1\_sse\_0\_15; | Core mask |
| uint32\_t core\_mask2\_sse\_16\_31; | Core mask |
| uint32\_t core\_mask1\_avx\_0\_15; | Core mask |
| uint32\_t core\_mask2\_avx\_16\_31; | Core mask |
| uint32\_t core\_mask1\_avx2\_0\_15; | Core mask |
| uint32\_t core\_mask2\_avx2\_16\_31; | Core mask |
| uint32\_t core\_mask1\_red\_green1\_0\_15; | Core mask |
| uint32\_t core\_mask2\_red\_green2\_16\_31; | Core mask |

Table 8 : Boot FST PCIE message format for startup STL results

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_stl\_res\_msg |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for startup STL results. | |
| Members: | uint32\_t version : 4U; | Version of specifications. |
| uint32\_t reserved\_1 : 4U; | Reserved. |
| uint32\_t reserved\_2 : 6U; | Reserved. |
| uint32\_t flags : 2U; | Flag for response bit and retry bit. |
| uint32\_t length : 8U; | Message length in DWORDs. |
| uint32\_t reserved\_3 : 8U; | Reserved. |
| uint32\_t seq\_num : 16U; | Sequence number. |
| uint32\_t wl\_address : 8U; | Workload address. |
| uint32\_t isi\_address : 8U; | ISI address. |
| uint32\_t crc : 16U; | CRC value. |
| uint32\_t command : 16U; | POSC test results command. |
| uint32\_t sys\_time\_validity : 8U; | System time valid or not. |
| uint32\_t stl\_res\_validity : 8U; | STL results valid or not. |
| uint32\_t stl\_res : 8U; | STL result. |
| uint32\_t uncore\_mask : 8U; | Uncore mask is enable/disable. |
| uint32\_t core\_mask\_1 | Core STL executed or not executed. |
| uint32\_t core\_mask\_2 | Core STL executed or not executed. |
| uint32\_t sys\_time; | System time in POSIX format. |

Table 9 : Boot FST PCIE response message format

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_pcie\_res\_s |
| Type: | Struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB response message format. | |
| Members: | uint32\_t version : 4U; | Version of specifications. |
| uint32\_t reserved\_1 : 4U; | Reserved. |
| uint32\_t reserved\_2 : 6U; | Reserved. |
| uint32\_t flags : 2U; | Flag for response bit and retry bit. |
| uint32\_t length : 8U; | Message length in DWORDs. |
| uint32\_t reserved\_3 : 8U; | Reserved. |
| uint32\_t seq\_num : 16U; | Sequence number. |
| uint32\_t wl\_address : 8U; | Workload address. |
| uint32\_t isi\_address : 8U; | ISI address. |
| uint32\_t crc : 16U; | CRC value. |
| uint32\_t command : 16U; | command. |
| uint32\_t res\_code : 16U; | Response code. |
| uint32\_t reserved\_4 : 16U; | Reserved |
| uint32\_t pst : 16U; | PST value. |
| uint32\_t dti : 16U; | DTI value. |

Table 10 : Boot FST PCIE message format for ODCC snapshot

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_odcc\_ss\_msg |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for ODCC snapshot. | |
|  | struct tpf\_host\_mcu\_packet\_header packet\_header | ODCC packet header. |
| uint32\_t channel\_id : 8U | Channel Id |
| uint32\_t workload\_id : 8U | Workload Id |
| uint32\_t snapshot\_id : 8U | Snapshot Id |
| uint32\_t reserved\_3 : 8U | Reserved |
| uint32\_t reserved\_4 | Reserved |
| uint32\_t signature | ODCC Signature |

Table 11 : Boot FST PCIE response message format for ODCC snapshot

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_odcc\_ss\_res |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for ODCC snapshot response. | |
|  | struct tpf\_host\_mcu\_packet\_header packet\_header | ODCC packet header. |
| uint32\_t channel\_id : 8 | Channel Id |
| uint32\_t workload\_id : 8 | Workload Id |
| uint32\_t snapshot\_id : 8 | Snapshot Id |
| uint32\_t response : 8 | Response code |

Table 12 : Boot FST PCIE message format for override config data

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_write\_config\_data\_cmd\_s |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB message format for override config data. | |
|  | struct tpf\_host\_mcu\_packet\_header hdr | Override config data packet header. |
| uint32\_t pst | Payload data received. |
| uint32\_t prochot : 8 |
| uint32\_t log\_level : 8 |
| uint32\_t mission\_mode : 8 |
| uint32\_t rsvd\_1 : 8 |
| uint32\_t all\_sci\_warn\_thresh : 8 |
| uint32\_t all\_sci\_warn\_cnt\_ref\_interval : 16 |
| uint32\_t rsvd\_2 : 8 |
| uint32\_t pch\_err1\_thresh : 8 |
| uint32\_t pch\_err1\_cnt\_ref\_interval : 16 |
| uint32\_t rsvd\_3 : 8 |
| uint32\_t pch\_err0\_thresh : 8 |
| uint32\_t pch\_err0\_cnt\_ref\_interval : 16 |
| uint32\_t rsvd\_4 : 8 |
|  | uint32\_t ibecc\_dram\_err\_thresh : 8 |
|  | uint32\_t ibecc\_dram\_err\_cnt\_ref\_interval : 16 |
|  | uint32\_t rsvd\_5 : 8 |
|  | uint32\_t cmci\_err\_thresh : 8 |
|  | uint32\_t cmci\_err\_cnt\_ref\_interval : 16 |
|  | uint32\_t rsvd\_6 : 8 |

Table 13 : Boot FST PCIE response message format for override config data

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_write\_config\_data\_resp\_s |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PCIe MB response message format for override config data. | |
|  | struct tpf\_host\_mcu\_packet\_header hdr | Override config data packet header. |
|  | struct  {  uint32\_t resp\_code : 16  uint32\_t pst\_invalid : 1  uint32\_t prochot\_out\_of\_range : 1  uint32\_t log\_level\_invalid : 1  uint32\_t mission\_mode\_invalid : 1  uint32\_t all\_sci\_warn\_thresh\_out\_of\_range : 1  uint32\_t all\_sci\_warn\_ref\_int\_out\_of\_range : 1  uint32\_t pch\_err1\_thresh\_out\_of\_range : 1  uint32\_t pch\_err1\_cnt\_ref\_int\_out\_of\_range : 1  uint32\_t pch\_err0\_thresh\_out\_of\_range : 1  uint32\_t pch\_err0\_cnt\_ref\_int\_out\_of\_range : 1  uint32\_t ibecc\_dram\_err\_thresh\_out\_of\_range : 1  uint32\_t ibecc\_dram\_err\_cnt\_ref\_int\_out\_of\_range : 1  uint32\_t cmci\_err\_thresh\_out\_of\_range : 1  uint32\_t cmci\_err\_cnt\_ref\_int\_out\_of\_range : 1  uint32\_t rsvd : 2  } payload; | Payload data. |

Table 14 : Boot FST PUINT request message format

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_punit\_request | |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | PUNIT request message format for PING test and fetching ITD parameters. | |
| Members: | uint32\_t cmd\_val : 8 | Command value. |
| uint32\_t seqno : 4 | Sequence number. |
| uint32\_t reserved1 : 4 | Reserved area. |
| uint32\_t crc : 8 | CRC value. |
| uint32\_t reserved2 : 4 | Reserved area. |
| uint32\_t reserved3 : 4 | Reserved area. |
| uint32\_t data | Data. |

Table 15 : Boot FST PUINT response message format

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_punit\_response | |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | Response message from PUINT. | |
| Members: | uint32\_t cmd\_val : 8 | Command value. |
| uint32\_t seqno : 4 | Sequence number. |
| uint32\_t res\_code : 4 | Response code. |
| uint32\_t num\_dwords : 8 | Number of DWORDS in message payload. |
| uint32\_t reserved1 : 4 | Reserved area. |
| uint32\_t reserved2 : 4 | Reserved area. |
| uint32\_t data[BOOT\_FST\_PUNIT\_RES\_DATA\_SIZE = 16] | Data. |
| uint32\_t crc | CRC value. |

Table 16 : Boot FST PUINT ITD response data 0

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_itd\_res\_data\_0 | |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | ITD response data 0. | |
| Members: | uint32\_t itd\_cutoff\_tj\_val : 7U | ITD\_CUTOFF\_TJ. |
| uint32\_t resvd1 : 9U | Reserved area. |
| uint32\_t gauranteed\_vid : 16U | Domain's Guaranteed VID. |

Table 17 : Boot FST PUINT ITD response data 1

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_itd\_res\_data\_1 | |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | ITD response data 1. | |
| Members: | uint32\_t itd\_cutoff\_v\_val : 9U | ITD\_CUTOFF\_V. |
| uint32\_t resvd1 : 7U | Reserved area. |
| uint32\_t itd\_cutoff\_v2\_val : 9U | ITD\_CUTOFF\_V2. |
| uint32\_t resvd2 : 7U | Reserved area. |

Table 18 : Boot FST PUINT ITD response data 2

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_itd\_res\_data\_2 | |
| Type: | struct | |
| File: | boot\_fst.c | |
| Range: |  | |
| Description: | ITD response data 2. | |
| Members: | uint32\_t slope : 8U | SLOPE. |
| uint32\_t resvd1 : 8U | Reserved area. |
| uint32\_t slope2 : 8U | SLOPE2. |
| uint32\_t resvd2 : 8U | Reserved area. |

## Configuration Parameters

Table 19 : Configuaration parameters

|  |  |
| --- | --- |
| Syntax: | struct boot\_fst\_config\_s |
| Type: | struct | |
| File: | Platform\_config\_defs.h (TBD: move it to boot\_fst.h) | |
| Range: |  | |
| Description: | Boot FST configuration parameters. TODO: Finalize members. | |
|  | uint32\_t bios\_boot\_time; | BIOS boot time. |
| uint32\_t posc\_results\_time; | Getting POSC results time. |
| uint32\_t stl\_results\_time; | Getting startup STL results time. |

## API Functions

### Boot FST initialization

Table 20 : Boot FST initialization API

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_init | | |
| Syntax: | uint32\_t boot\_fst\_init (  void \*config\_param  ); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*config\_param | void | The void \* pointer points to the config\_param corresponding to this FST, as set by the FSTM. The FSTM itself gets the Config Parameter from the SRAM, placed by SCI ROM. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Boot FST init successful. |
| STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_INIT, tx\_status) | | On event flag creation/byte allocation/queue creation failure. |
| STATUS\_FIRMWARE(M\_FST\_BOOT, E\_IN\_INIT, E\_INVALID\_ARGUMENT); | | On host monitor timer value equal to zero. |
| STATUS\_FIRMWARE(M\_FST\_BOOT, E\_IN\_INIT, E\_INVALID\_PTR) | | On NULL input parameter. |
| Description: | This API performs the initialization of various globals and config parameters corresponding to this FST as sent in the void \*config\_param from the FSTM. | | |
| Preconditions: | Boot FST must be created. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful initialization, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.   1. Validate the input parameter. 2. Validate the host monitor timer timeout value in input configurations. Store the input configurations. 3. Create event flag for monitoring the host boot. 4. Allocate memory to boot FST queue and create the queue. 5. If any error in above steps, report to upper layer. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 1: Boot FST initialization flow



### Boot FST stop

Table 21 : Boot FST stop API

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_stop | | |
| Syntax: | uint32\_t boot\_fst\_stop (  void  ); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Boot FST stop successful. |
| STATUS\_FIRMWARE | | On firmware failure. |
| STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_STOP, tx\_status) | | On BIOS stage2 timer deactivate or delete failed. |
| Description: | This API terminates and deletes every resource created by boot FST and related memory allocated. Also gets the fst app and host monitoring app threads out of any blocked state, so that the boot FST thread can be terminated and deleted by FSTM. | | |
| Preconditions: | Boot FST must be created and initialized. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful stop, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API deletes the host monitor event flag, boot FST queue and releases the memory allocated for queue. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 2: Boot FST stop flow



### Boot FST application

Table 22 : Boot FST application thread function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_app | | |
| Syntax: | void boot\_fst\_app (  uint32\_t thread\_input  ); | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | thread\_input | uint32\_t | Zeror. |
| Parameters (out): | void |  |  |
| Return value: | void | |  |
| Description: | This API implements the actual Boot FST functionality. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This API performs the following operations.   1. Calls fst\_comman\_rtos\_timer\_check() to test the RTOS timer. 2. Calls fst\_common\_gp\_timer\_check() to test the GP timer. 3. Calls fst\_common\_prd\_timer\_check() to test the periodic timer. 4. Call fst\_common\_wdt\_check() to test the WDT. 5. Call fst\_common\_fmm\_sp\_reg\_check() to test FMM scratchpad register. 6. Call boot\_fst\_frcpu\_check() to run frCPU tests. 7. Calls boot\_fst\_clk\_mon\_ei\_check() to test clock monitor error injection. 8. Calls boot\_fst\_root\_parity\_ei\_check() to test root parity error injection. 9. Mask WDT timeout FMM alarm. 10. Calls boot\_fst\_host\_monitor() to monitor host tests. 11. Start WDT and unmask WDT timeout FMM alarm. 12. Report status to FSTM program flow monitoring. 13. Calls fw\_internal\_error\_handler API if status is failure. | | |
| Design Decisions |  | | |
| SAS traceability | 590398, 850153 | | |

Figure 3: Boot FST app flow



## Interrupt Handler

NA

## Local functions

### Boot FST timer expiry callback function

Table 23 : Boot fst timer expiry callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_timer\_expiry\_fn | | |
| Syntax: | static void boot\_fst\_timer\_expiry\_fn(  timer\_id\_e id  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | id | timer\_id\_e | Timer id. |
| Parameters (out): | void |  |  |
| Return value: | void | |  |
| Description: | This API is called on boot FST timer expiry. | | |
| Preconditions: | This API must be registered while creating the boot fst timers. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This API validates the timer expired, updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 4: Boot fst Timer expiry callback function flow



### Boot FST bios boot callback function

Table 24 : Boot fst bios boot callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | Boot FST bios boot callback function | | |
| Syntax: | static void boot\_fst\_bios\_boot\_cb (  void \*data,  uint32\_t len)  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*data | void | Pointer to input data. |
|  | len | Uint32\_t | Length of input data. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful data receiving. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK1, tx\_status) | | On queue send failure. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK2, tx\_status) | | On event flag set failure. |
| Description: | This API is called on receiving the async message from host. | | |
| Preconditions: | This API must be registered with FSTM async interface layer. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful receiving data, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_set() to notify bios boot message reception.  Calls tx\_queue\_send() to send the data to boot fst thread. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 5 : Bios boot complete callback function flow



### Boot FST POSC results callback function

Table 25 : Boot fst POSC results callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | Boot FST POSC results callback function | | |
| Syntax: | static void boot\_fst\_posc\_results\_cb (  void \*data,  uint32\_t len)  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*data | void | Pointer to input data. |
|  | len | Uint32\_t | Length of input data. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful data receiving. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK1, tx\_status) | | On queue send failure. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK2, tx\_status) | | On event flag set failure. |
| Description: | This API is called on receiving the async message from host. | | |
| Preconditions: | This API must be registered with FSTM async interface layer. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful receiving data, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_set() to notify POSC results message reception.  Calls tx\_queue\_send() to send the data to boot fst thread. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 6 : POSC results callback function flow



### Boot FST STL results callback function

Table 26 : Boot fst STL results callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | Boot FST STL results callback function | | |
| Syntax: | static void boot\_fst\_stl\_results\_cb (  void \*data,  uint32\_t len)  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*data | void | Pointer to input data. |
|  | len | Uint32\_t | Length of input data. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful data receiving. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK1, tx\_status) | | On queue send failure. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK2, tx\_status) | | On event flag set failure. |
| Description: | This API is called on receiving the async message from host. | | |
| Preconditions: | This API must be registered with FSTM async interface layer. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful receiving data, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_set() to notify STL results message reception.  Calls tx\_queue\_send() to send the data to boot fst thread. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 7 : STL results callback function flow



### Boot FST ODCC SS callback function

Table 27 : Boot fst ODCC SS callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | Boot FST ODCC Snap Shot callback function | | |
| Syntax: | static void boot\_fst\_odcc\_ss\_cb (  void \*data,  uint32\_t len)  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*data | void | Pointer to input data. |
|  | len | Uint32\_t | Length of input data. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful data receiving. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK1, tx\_status) | | On queue send failure. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK2, tx\_status) | | On event flag set failure. |
| Description: | This API is called on receiving the async message from host. | | |
| Preconditions: | This API must be registered with FSTM async interface layer. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful receiving data, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_set() to notify ODCC SS message reception.  Calls tx\_queue\_send() to send the data to boot fst thread. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 8 : ODCC ss callback function flow



### Boot FST override config param callback function

Table 28 : Boot fst override config param callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | Boot FST override config param callback function | | |
| Syntax: | static void boot\_fst\_override\_config\_param\_cb (  void \*data,  uint32\_t len)  ) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*data | void | Pointer to input data. |
|  | len | Uint32\_t | Length of input data. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful data receiving. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK1, tx\_status) | | On queue send failure. |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CALLBACK2, tx\_status) | | On event flag set failure. |
| Description: | This API is called on receiving the async message from host. | | |
| Preconditions: | This API must be registered with FSTM async interface layer. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful receiving data, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_set() to notify override config param message reception.  Calls tx\_queue\_send() to send the data to boot fst thread. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 9 : Override config data callback function flow



### Clock monitor Error Injection(EI) test

Table 29 : Clock monitor error injection check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_clk\_mon\_ei\_check | | |
| Syntax: | static uint32\_t boot\_fst\_clk\_mon\_ei\_check (void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor error injection check successful. |
| STATUS\_FIRMWARE (M\_FST\_BOOT,  E\_IN\_CLK\_MONITOR\_ERR\_INJECTION,  E\_INVALID\_CLK\_MONITOR\_ID) | | Clock monitor error injection check failed. |
| Description: | This API performs the Clock monitor error injection test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful Clock monitor error injection check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  In a for loop of NUM\_CLK\_MONITOR (defined in local control driver) perform the below steps.  Calls lc\_osdl\_clkmon\_status\_set() to enable clock monitor.  Calls the boot\_fst\_count\_ei(clock monitor id, sample window, target count and error count) to test the count error injection.  Calls the boot\_fst\_lock\_ei (clock monitor id, sample window, target count) to test the lock error injection.  Calls the boot\_fst\_dead\_clock\_ei (clock monitor id) to test the dead clock error injection.  Call the lc\_osdl\_clkmon\_set\_val(CLK\_MONITOR\_2, CLKMON\_SAMPLE\_WINDOW, 192302U) and lc\_osdl\_clkmon\_set\_val( CLK\_MONITOR\_2, CLKMON\_TARGET\_COUNT, 1001573U) to set the sample window and target count  Call the lc\_osdl\_clkmon\_set\_val(CLK\_MONITOR\_3, CLKMON\_SAMPLE\_WINDOW, 1001573U) and lc\_osdl\_clkmon\_set\_val( CLK\_MONITOR\_3, CLKMON\_TARGET\_COUNT, 192302U) to set the sample window and target count  Calls lc\_osdl\_clkmon\_status\_set() to disable clock monitor  If any error in above steps, return error . | | |
| Design Decisions |  | | |
| SAS traceability | 850142 | | |

Figure 10: Clock monitor error injection check function flow



### Clock monitor pre-mask status check

Table 30 : Clock monitor pre-mask status check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_cm\_pre\_mask\_status\_check | | |
| Syntax: | static uint32\_t boot\_fst\_cm\_pre\_mask\_status\_check (  enum clk\_monitor\_id\_e clk\_mon\_id,  uint32\_t sample\_window,  uint32\_t target\_count,  uint32\_t error\_count,  uint8\_t ei\_type  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor pre-mask status successful. |
| Return value of errmgt\_firmware\_internal\_error() API. | |  |
| Description: | This API gets pre-mask alarm status for clock mon id..  Note: The flow is covered as part of calling function flow diagram. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful status check, else it will error handling API to assert NOK. | | |
| Implementation comments | This API performs the following operations.  Calls lc\_osdl\_clkmon\_status\_get() using parameters clk\_mon\_id, CLKMON\_PRE\_MASK\_ALARM, &comp\_status to get the pre mask alarm status.  If alarm is not set, test failed, update the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 11 : Clock monitor pre-mask status check function flow



### Clock monitor count done status check

Table 31 : Clock monitor count done status check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_cm\_count\_done\_status\_check | | |
| Syntax: | static uint32\_t boot\_fst\_cm\_count\_done\_status\_check (  enum clk\_monitor\_id\_e clk\_mon\_id,  uint32\_t sample\_window,  uint32\_t target\_count,  uint32\_t error\_count  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor count done status get successful. |
| Return value of errmgt\_firmware\_internal\_error() API. | |  |
| Description: | This API waits for count done status and checks count done status.  Note: The flow is covered as part of calling function flow diagram. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful status check, else it will error handling API to assert NOK. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_thread\_sleep to wait for count done status to set.  Calls lc\_osdl\_clkmon\_status\_get() using parameters clk\_mon\_id, CLKMON\_COUNT\_DONE\_STATUS, &comp\_status to get the count done status.  If status is not set, test failed, update the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 12 : Clock monitor count done status function flow



### Clock monitor PPM range status check

Table 32 : Clock monitor PPM range status check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_cm\_ppm\_range\_status\_check | | |
| Syntax: | static uint32\_t boot\_fst\_cm\_ppm\_range\_status\_check (  enum clk\_monitor\_id\_e clk\_mon\_id,  uint32\_t sample\_window,  uint32\_t target\_count,  uint32\_t error\_count  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor ppm range status get successful. |
| Return value of errmgt\_firmware\_internal\_error() API. | |  |
| Description: | This API gets PPM range status if not set calls error hanlding API.  Note: The flow is covered as part of calling function flow diagram. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful status check, else it will error handling API to assert NOK. | | |
| Implementation comments | This API performs the following operations.  Calls lc\_osdl\_clkmon\_status\_get() using parameters clk\_mon\_id, CLKMON\_PPM\_IN\_RANGE\_STATUS, &comp\_status to get PPM IN range status.  If status is not set, test failed, update the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 13 : Clock monitor PPM range status check function flow



### Clock monitor count error injection test

Table 33 : Clock monitor count error injection function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_count\_ei | | |
| Syntax: | static uint32\_t boot\_fst\_count\_ei(  enum clk\_monitor\_id\_e clk\_mon\_id,  uint32\_t sample\_window,  uint32\_t target\_count,  uint32\_t error\_count  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor count error injection check successful. |
| STATUS\_FIRMWARE ( M\_FST\_BOOT, E\_IN\_COUNT\_ERR\_INJECTION, E\_STATUS\_BITS\_MISMATCH) | | Clock monitor count error injection check failed. |
| Description: | This API performs the Clock monitor count error injection test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful Clock monitor count error injection check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls the lc\_osdl\_clkmon\_set\_val() using parameters clk\_mon\_id, CLKMON\_SAMPLE\_WINDOW and sample\_window to program sample window.  Calls the lc\_osdl\_clkmon\_set\_val() using parameters clk\_mon\_id, CLKMON\_TARGET\_COUNT and target\_count to program target count.  Calls lc\_osdl\_clkmon\_set\_val() parameters clk\_mon\_id, CLKMON\_ERR\_COUNT, error\_count to program error count.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_CNT\_ERR\_INJ\_EN, True to enable the count error injection.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_STOP\_REPEATand False to disable repeat mode.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ andTrue to enable NOK mask.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE and True to enable clock monitor.  Calls boot\_fst\_cm\_count\_done\_status\_check() API to check count done status.  Calls boot\_fst\_cm\_ppm\_range\_status\_check() API to check ppm range status.  Calls boot\_fst\_cm\_pre\_mask\_status\_check() API to check pre-mask status.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE, False to disable clock monitor.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_CNT\_ERR\_INJ\_EN, False to disable count error injection.  Calls lc\_osdl\_clkmon\_status\_set() using clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ, False to disable NOK mask.  If any error in above steps, return error . | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 14: Clock monitor count error injection function flow



### Clock monitor lock error injection test

Table 34 : Clock monitor lock error injection function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_lock\_ei | | |
| Syntax: | static uint32\_t boot\_fst\_lock\_ei (  enum clk\_monitor\_id\_e clk\_mon\_id,  uint32\_t sample\_window,  uint32\_t target\_count  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor lock error injection check successful. |
| STATUS\_FIRMWARE | | Clock monitor lock error injection check failed. |
| Description: | This API performs the Clock monitor lock error injection test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful clock monitor lock error injection check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls the lc\_osdl\_clkmon\_set\_val() using parameters clk\_mon\_id, CLKMON\_SAMPLE\_WINDOW, sample\_window to program sample window.  Calls the lc\_osdl\_clkmon\_set\_val() using parameters clk\_mon\_id, CLKMON\_TARGET\_COUNT, target\_count to program target count.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ, True to enable NOK mask.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE, True to enable clock monitor.  Wait for minimum of 10 period clocks.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_LOCK\_ERR\_INJ\_EN, True to enable the lock error injection.  Calls lc\_osdl\_clkmon\_status\_get() using parameters clk\_mon\_id, CLKMON\_LOCK\_ERROR\_STATUS, &comp\_status to get the lock error status. If status is equal to 0x2 then test pass else failed.  Calls boot\_fst\_cm\_pre\_mask\_status\_check() API to check pre-mask status.s  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE, False to disable clock monitor.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_LOCK\_ERR\_INJ\_EN, False to disable lock error injection.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ, False to disable NOK mask.  If any error in above steps, return error . | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 15: Clock monitor lock error injection function flow



### Clock monitor dead clock error injection test

Table 35 : Clock monitor dead clock error injection function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_dead\_clock\_ei | | |
| Syntax: | static uint32\_t boot\_fst\_dead\_clock\_ei (  enum clk\_monitor\_id\_e clk\_mon\_id,  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Clock monitor dead clock error injection check successful. |
| STATUS\_FIRMWARE | | Clock monitor dead clock error injection check failed. |
| Description: | This API performs the Clock monitor dead clock error injection test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful clock monitor dead clock error injection check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_DCD\_ERROR\_INJ\_EN, True to enable the dead clock error injection.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ, True) to enable NOK mask.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE, True to enable clock monitor.  Wait for timeout period.  Calls lc\_osdl\_clkmon\_status\_get() using parameters clk\_mon\_id, CLKMON\_DCD\_ERROR\_STATUS, &comp\_status to get the dead clock error status. If error status is 0x2 then test is pass else fail.  Calls boot\_fst\_cm\_pre\_mask\_status\_check() API to check pre-mask status.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_CTRL\_IN\_ENABLE, False to disable clock monitor.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_DCD\_ERROR\_INJ\_EN, False to disable dead clock error injection.  Calls lc\_osdl\_clkmon\_status\_set() using parameters clk\_mon\_id, CLKMON\_MASK\_NOK\_ERR\_INJ, False) to disable NOK mask.  If any error in above steps, return error . | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 16: Clock monitor dead clock error injection function flow



### Root parity Error Injection(EI) test

Table 36 : Root parity error injection check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_root\_parity\_ei\_check | | |
| Syntax: | static uint32\_t boot\_fst\_root\_parity\_ei\_check (void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Root parity error injection check successful. |
| STATUS\_FIRMWARE | | Root parity error injection check failed. |
| Description: | This API performs the root parity error injection test for DMA, RTOS, WDT, PSS, FMM, GPIO, Global control and local control sub modules. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful root parity error injection check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls fmm\_parity\_alarm\_mask\_unmask() to mask root parity alarm.  Calls fmm\_parity\_alarm\_inject() to enable root parity error injection.  Calls fmm\_parity\_alarm\_premask\_status\_get() to get the pre-mask alarm status bit. If pre-mask alarm is generated, test pass, else test failed. If test failed update the status with failure value.  Calls fmm\_parity\_alarm\_inject\_disable() to disable root parity error injection.  Calls fmm\_parity\_alarm\_clear() to clear the pre-mask alarm.  Calls fmm\_parity\_alarm\_mask\_unmask() to unmask root parity alarm.  If any error in above steps, return error.  Repeate the above steps for all sub-modules in a for loop. | | |
| Design Decisions |  | | |
| SAS traceability | 850142 | | |

Figure 17: Root parity error injection check function flow



### ROM ECC error injection test

Table 37 : ROM ECC error injection check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_rom\_ecc\_ei\_check | | |
| Syntax: | static uint32\_t boot\_fst\_rom\_ecc\_ei\_check (  void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | frCPU check successful. |
| STATUS\_FIRMWARE | | frCPU check failed. |
| Description: | This API performs the ROM ECC error injection test.  Note: This requirement should come from CCB. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful frCPU check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | The proposal is to perform following operations.   1. Mask FMM’s ROM\_ECC\_ucerr\_alarm. 2. Program DMA for ROM data access to the single/multi bit error location. 3. Wait DMA interrupt for completed transfer then check/clear AXI\_Status\_reg in DMA (how is that cleared?). 4. Check FMM’s ROM\_ECC\_ucerr\_alarm pre\_status to make sure it is set (or check Alarm Status Register in MEMSS), NOK if not set. 5. Clear Alarm Status Register and Info Status Register in MEMSS. 6. Unmask FMM’s ROM\_ECC\_ucerr\_alarm | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### Boot FST frCPU check

Table 38 : Boot FST frCPU check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_frcpu\_check | | |
| Syntax: | static uint32\_t boot\_fst\_frcpu\_check(  void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | frCPU check successful. |
| STATUS\_FIRMWARE | | frCPU check failed. |
| Description: | This API performs the frCPU tests. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful frCPU check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.   1. Calls the fst\_common\_mul\_div\_check () function to perform multiplier and divider check. 2. Calls the fst\_common\_bus\_matrix\_check () function to perform bus matrix check. 3. Calls the boot\_fst\_nvic\_int\_prio\_check() function to perform NVIC interrupt priority check. 4. Calls the fst\_common\_mpu\_nvic\_reg\_read() function to read the MPU and NVIC registers. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 18: Boot FST frCPU check function flow



### NVIC interrupt priority register check

Table 39 : frCPU NVIC interrupt priority register check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_nvic\_int\_prio\_check | | |
| Syntax: | static void boot\_fst\_nvic\_int\_prio\_check (void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | Void | |  |
| Description: | This API performs the NVIC interrupt priority register check. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful frCPU check, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  In a for loop (for 8 interrupt priority registers), read all the used priority values from ARM Cortex-M3 NVIC priority registers, bit-wise XORing them. This is the current parity and store it. | | |
| Design Decisions |  | | |
| SAS traceability | 850153 | | |

Figure 19: NVIC interrupt priority register check flow



### Async commands register

Table 40 : Async commands register function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_async\_cmd\_reg | | |
| Syntax: | Static uint32\_t boot\_fst\_async\_cmd\_reg (void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful registrations. |
|  | Return value of register\_callback\_cmd() API. | |  |
| Description: | This function registers boot FST async commands with FSTM async interface layer. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful registration, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.   1. Register BIOS boot complete callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, command = 0x206U, fst id = BOOT\_FST\_ID and callback = boot\_fst\_bios\_boot\_cb. 2. Register POSC results callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, command = 0x218U, fst id = BOOT\_FST\_ID and callback = boot\_fst\_posc\_results\_cb. 3. Register STL results callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, command = 0x23EU, fst id = BOOT\_FST\_ID and callback = boot\_fst\_stl\_results\_cb. 4. Register ODCC dummy snapshot callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, command = 0x039EU, fst id = BOOT\_FST\_ID and callback = boot\_fst\_odcc\_ss\_cb. 5. Register override config parameter callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, command = 0x4D0U, fst id = BOOT\_FST\_ID and callback = boot\_fst\_override\_config\_param\_cb. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 20 : Async commands register function flow



### Async commands de-register

Table 41 : Async commands de-register function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_async\_cmd\_de\_reg | | |
| Syntax: | Static uint32\_t boot\_fst\_async\_cmd\_de\_reg (void); | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful registrations. |
|  | Return value of de\_register\_callback\_cmd () API. | |  |
| Description: | This function de-registers boot FST async commands with FSTM async interface layer. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful de-registration, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.   1. De-register BIOS boot complete callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, fst id = BOOT\_FST\_ID. 2. De-register POSC results callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, fst id = BOOT\_FST\_ID. 3. De-register STL results callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, fst id = BOOT\_FST\_ID. 4. De-register ODCC dummy snapshot callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, fst id = BOOT\_FST\_ID. 5. De-register override config parameter callback with FSTM async interface. With interface number = PCIE\_FSTM\_INTERFACE, fst id = BOOT\_FST\_ID. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 21 : Async commands de-register function flow



### Host boot monitoring

The asynchronous data received from host to SCI FSTs is routed through FSTM. Refer SCI SAS and FSTM MDS for more details.

Table 42 : Host monitoring application thread function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_host\_monitor | | |
| Syntax: | Static uint32\_t boot\_fst\_host\_monitor (void); | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful host monitor. |
|  | STATUS\_FIRMWARE/STATUS\_THREADX | | On host monitor failure. |
| Description: | This function calls the host monitoring APIs. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful host monitoring, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | Initially, host\_monitor\_timeout is set to BIOS boot timeout value.  This function performs the following operations.   1. Calls boot\_fst\_async\_cmd\_reg to register async commands. 2. Calls fst\_lib\_timer\_create() to create host monitor timer. 3. Calls fst\_lib\_timer\_start() to start host monitor timer. 4. Calls tx\_event\_flags\_get(BOOT\_FST\_TIMER\_EXPIRED | BOOT\_FST\_BIOS\_BOOT\_COM\_MSG\_RECEIVED | BOOT\_FST\_POSC\_RESULTS\_RECEIVED | BOOT\_FST\_STL\_RESULTS\_RECEIVED | BOOT\_FST\_ODCC\_SS\_RECEIVED). 5. If event flag BOOT\_FST\_BIOS\_BOOT\_COM\_MSG\_RECEIVED is set, calls the boot\_fst\_bios\_boot\_monitor() function to check the host BIOS boot status. On success, set the bios\_boot\_completed to 1 and host\_monitor\_timeout to posc results timeout. 6. If event flag BOOT\_FST\_POSC\_RESULTS\_RECEIVED is set and bios\_boot\_completed equal to 1, calls boot\_fst\_posc\_results\_monitor() function to check the POSC test results. On Success set posc\_res\_rec\_completed to 1 and   host\_monitor\_timeout to stl results timeout.   1. If event flag BOOT\_FST\_ODCC\_SS\_RECEIVED is set, calls boot\_fst\_odcc\_check() function to comapre the ODCC dummy snapshot test results. 2. If event flag BOOT\_FST\_STL\_RESULTS\_RECEIVED is set and posc\_res\_rec\_completed is equal to 1, calls boot\_fst\_stl\_results\_monitor () function to check the startup STL results. On status and response code equal to success, if host\_boot\_monitor\_completed equal to zero set host\_boot\_monitor\_completed to 1. 3. If event flag BOOT\_FST\_TIMER\_EXPIRED is set, then update the status with timeout error. 4. Calls fst\_lib\_timer\_stop() to stop the host monitor timer. On success, calls fst\_lib\_timeout\_change() to configure host\_monitor\_timeout. 5. Repeat from step 6 to 13 in while loop, until status equal to success and host\_boot\_monitor\_completed equal to 0. 6. If status is success, calls boot\_fst\_ravdm\_check() function to test SCI communication with PUNIT. And calls boot\_fst\_pmc\_ipc\_check() function to test SCI communication with PMC. 7. Calls boot\_fst\_async\_cmd\_de\_reg() API to de-register async commands. 8. If timer status is success, calls fst\_lib\_timer\_delete() to delete host monitor timer. 9. Return with status. | | |
| Design Decisions |  | | |
| SAS traceability | 590419, 590782 | | |

Figure 22: Host monitoring application thread function flow



### Host BIOS boot monitor

Table 43 : Host BIOS boot monitor function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_bios\_boot\_monitor | | |
| Syntax: | Static uint32\_t boot\_fst\_bios\_boot\_monitor (void) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful BIOS boot monitor. |
| STATUS\_FIRMWARE | | On BIOS boot monitor failure. |
| Description: | This API performs the BIOS boot monitor. Host BIOS on its boot completion sends a message to SCI via PCIe MB and SCI send acknowledgement to host. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful host BIOS boot monitor, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | On receiving BIOS boot complete message, boot\_fst\_bios\_boot\_cb is called. It calls tx\_queue\_send(boot\_fst\_queue) and set the event flag to notify boot FST thread waiting for messag  This function performs the following operations.   1. Calls tx\_queue\_receive() to receive the BIOS boot complete message 2. Checks if bios\_boot\_msg->length equal to BOOT\_FST\_BIOS\_BOOT\_REC\_LEN, if not update the response status as invalid length. 3. Checks if bios\_boot\_msg->isi\_bdf\_valid is equal to 0XAAu, if not update the response code status with BDF mismatch. 4. Update the response packet and calls tl\_respond() to send the Acknowledgement to host with response code. If any error in ack then host will send the BIOS boot complete message again. 5. If any error in above steps report to FSTM. | | |
| Design Decisions |  | | |
| SAS traceability | 971267, 971269, 971270, 850143, 850153 | | |

The sequence diagram for host BIOS boot monitor is shown below.

Figure 23: Host BIOS boot monitor overall flow diagram



### POSC results packet validate

Table 44 : POSC results packet validate function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_posc\_results\_pkt\_validate | | |
| Syntax: | static uint32\_t boot\_fst\_posc\_results\_pkt\_validate(  const struct boot\_fst\_posc\_res\_msg \*result\_msg,  uint16\_t \*res\_code\_ptr  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \* result\_msg | struct boot\_fst\_posc\_res\_msg | Pointer to POSC results message. |
|  | \*res\_code\_ptr | Uint16\_t | Pointer to response code. |
| Parameters (out): | \*res\_code\_ptr | Uint16\_t | Pointer to response code. |
| Return value: | STATUS\_SUCCESS | | On successful POSC packet validation. |
| STATUS\_FIRMWARE(M\_FST\_BOOT, E\_IN\_POSC\_RES\_MONITOR,  E\_INVALID\_SSEAVX\_DATA\_VALID\_FIELD) | | If invalid AVX validate data is received.. |
|  | STATUS\_FIRMWARE(  M\_FST\_BOOT, E\_IN\_POSC\_RES\_MONITOR, E\_POSC\_RES\_FAIL) | | If POSC results is received is fail. |
| Description: | This API validates the POSC results message and update the status and response code if any error.. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful validation, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Validate POSC results length, if mismatch then update response code.  Validate avx\_data\_valid field, if value is not INVALID then update status and response code.  Validate posc result field, if result is fail then update the status and response code. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 24 : POSC results packet validate function flow



### POSC – get number of cores enabled

Table 45 : POSC – get number of cores enabled function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_num\_cores\_enabled\_get | | |
| Syntax: | static uint16\_t boot\_fst\_num\_cores\_enabled\_get(  uint32\_t core\_mask  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | core\_mask | Uint32\_t | Core mask value received in POSC results message. |
| Parameters (out): | void |  |  |
| Return value: | BOOT\_FST\_PCIE\_SUCCESS | | On success. |
| BOOT\_FST\_PCIE\_UNKNOWN\_CORE\_MASK | | Response code on failure. |
| Description: | This API will parse the 32 bits core mask(antivalent bits format) value to get the cores enabled. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful parsing, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  In a for loop of 16, parse the core mask value received (2 bit antivalent value) if core mask corresponding to core is enabled then increment the boot\_fst\_num\_stl\_res\_expected.  If parsed values is not equal to enable or disable then update the response code with BOOT\_FST\_PCIE\_UNKNOWN\_CORE\_MASK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 25 : POSC- get number of cores function flow



### POSC results respond

Table 46 : POSC results respond function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_posc\_results\_respond | | |
| Syntax: | static uint32\_t  boot\_fst\_posc\_results\_respond(  struct boot\_fst\_pcie\_res\_s \*posc\_res,  uint32\_t res\_len,  uint16\_t response\_code) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \* result\_msg | struct boot\_fst\_posc\_res\_msg | Pointer to POSC results message. |
|  | res\_len | uint32\_t | POSC results response length |
|  | response\_code | uint16\_t | Response code. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful sending response. |
| Return value of tl\_respond | |  |
| Description: | This API updates the POSC results response fields and respond to POSC results message. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Updates the flags, length and response code POSC response fileds.  Calls tl\_respond to send response to POSC results message. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### POSC test results monitor

Table 47 : POSC test results monitor function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_posc\_results\_monitor | | |
| Syntax: | Static uint32\_t boot\_fst\_posc\_results\_monitor (void) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful host STL manager startup monitor. |
| STATUS\_FIRMWARE | | On host STL manager startup monitor failure. |
| Description: | This API performs the POSC test results monitor. | | |
| Preconditions: | Host BIOS boot complete message received and ack sent successfully. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful POSC results monitor, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | On receiving POSC results message, boot\_fst\_ posc\_results\_cb is called. It calls the tx\_queue\_send(boot\_fst\_queue) and set the event flag to notify boot FST thread waiting for message.  This function performs the following operations.  Calls tx\_queue\_receive() to receive the POSC results message.  Calls boot\_fst\_posc\_results\_pkt\_validate() to validate POSC results message.  Calls boot\_fst\_num\_cores\_enabled\_get() API to get number of cores enabled.  Copy the POSC results message header into posc results response header and calls boot\_fst\_posc\_results\_respond() API to respond to POSC results message..  If any error in above steps calls errmgt\_firmware\_internal\_error() to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | 971267, 971269, 971270, 850143, 850157, 850153 | | |

The sequence diagram for POSC results monitor is shown below.

Figure 26: Host POSC results monitor flow diagram



### RAVDM test

To test the RAVDM, perform RAVDM ping test and fetch the ITD parameters from PUINT.

Table 48 : RAVDM check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_ravdm\_check | | |
| Syntax: | static uint32\_t boot\_fst\_ravdm\_check (void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful RAVDM test. |
| STATUS\_FIRMWARE | | On RAVDM check failure. |
| Description: | This API send the ping command to PUNIT and fetch the ITD parameters from PUINT via RAVDM. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful RAVDM test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Calls boot\_fst\_ravdm\_ping\_check() to perform RAVDM ping test.  Calls boot\_fst\_ravdm\_itd\_fetch() to fetch the ITD parameters from PUINT via RAVDM.  If any error in above steps, return error. | | |
| Design Decisions |  | | |
| SAS traceability | 971267, 971269, 971270, 590363, 850153 | | |

Figure 27: RAVDM check function flow



### RAVDM ping test

Table 49 : RAVDM ping test function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_ravdm\_ping\_check | | |
| Syntax: | static uint32\_t boot\_fst\_ravdm\_ping\_check (void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful RAVDM ping test. |
| STATUS\_FIRMWARE | | On RAVDM ping test failure. |
| Description: | This API sends the ping command and receives the response to/from PUNIT via RAVDM. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful RAVDM ping test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Prepares the ping message with PING command.  Calls tl\_send(PUNIT, ping message, timeout) to send the ping message to PUINT via RAVDM.  Calls tl\_receive(PUNIT, ping message, timeout) to receive the ping response from PUINT.  Check the response code, if error response code, return error. And check number of DWORDS if not ,matching return error.  If any error in above steps, return error. | | |
| Design Decisions |  | | |
| SAS traceability | 850155 | | |

Figure 28: RAVDM ping test function flow



### RAVDM ITD parameters fetch

Table 50 : Fetch ITD parameters function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_ravdm\_itd\_fetch | | |
| Syntax: | static uint32\_t boot\_fst\_ravdm\_itd\_fetch (void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful fectching ITD parameters. |
| STATUS\_FIRMWARE | | On ITD parameters fetching failure. |
| Description: | This API sends the get ITD parametes command and receives the response to/from PUNIT via RAVDM. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful fetching the ITD parameters, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Prepares the get ITD parameters message with get ITD parametsrs command and domain.  Calls tl\_send(PUNIT, message, timeout) to send the get ITD parameter message to PUINT via RAVDM.  Calls tl\_receive(PUNIT, ping message, timeout) to receive the response from PUINT.  Check the response code, if error response code, return error. And check number of DWORDS if not ,matching return error.  Copy the ITD parameters into ITD values global variable which will be shared with periodic voltage monitoring FST.  If any error in above steps, return error.  Repeate the above steps for all the domains (VCCCORE, VCCL2, VCCRING and VCCSA). | | |
| Design Decisions |  | | |
| SAS traceability | 971282, 971283, 971284, 850155 | | |

Figure 29: Fetch ITD parameters function flow



### PMC IPC test

Table 51 : PMC IPC check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_pmc\_ipc\_check | | |
| Syntax: | Static uint32\_t boot\_fst\_pmc\_ipc\_check (void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful PMC IPC test. |
| STATUS\_FIRMWARE | | On PMC IPC test failure. |
| Description: | This API performs the PMC IPC test by sending platform transition information query and receiving the respose from PMC via IPC.  TODO: Check if opcode validation is required, under discussion. Current driver implementation does not pass opcode to application. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful PMC IPC test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.   1. Updates the PMC IPC test monitoring timer structure with timeout value, reload type and callback function. 2. Calls the fst\_lib\_timer\_create() to create the PMC IPC test monitoring timer. 3. Calls the fst\_lib\_timer\_start() to start the timer. If timer start failed then call timer delete API. 4. Calls the pmc\_ipc\_osdl\_send() to send platform transition information query. Platform transition query opcode (0x0009) and send length is one. Refer PMC IPC driver MDS for more information on APIs. 5. Calls pmc\_ipc\_osdl\_receive() to receive the response from PMC. 6. Calls fst\_lib\_timer\_stop() to stop timer and fst\_lib\_timer\_delete() to delete timer. Call these APIs irrespective of status from above steps. If both PMC IPC send/receive and timer stop/delete failed then return PMC IPC error. 7. If any error in above steps return error status. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 30: PMC IPC test flow diagram



### ODCC test – Dummy snapshot compare

While startup tests are executing if host test application sends the ODCC snapshots then compare the dummy snapshots received.

Table 52 : ODCC dummy snapshot compare

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_odcc\_check | | |
| Syntax: | Static uint32\_t boot\_fst\_odcc\_check (void) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful ODCC snapshot camparision. |
| Return value of odcc\_dummy\_ss\_comparison() API. | |  |
|  | STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_SNAPHSHOT\_COMPARISON, tx\_status) | | On queue receive failure. |
| Description: | This API receives the two dummy snapshots from host and calls odcc\_dummy\_ss\_comparison() API to compare the snapshots. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful host boot STLs monitor, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | On receiving ODCC snapshot message, boot\_fst\_odcc\_ss\_cb is called. It calls the tx\_queue\_send(boot\_fst\_queue) and set the event flag to notify boot FST thread waiting for message.  This function performs the following operations.  Calls tx\_queue\_receive() to receive the ODCC snapshot.  If boot\_fst\_first\_odcc\_ss\_received equal to zero then stop ODCC results in boot\_fst\_odcc\_ss[0] local variable else store in boot\_fst\_odcc\_ss[1] and call the odcc\_dummy\_ss\_comparison(boot\_fst\_odcc\_ss, 2).  Report status to caller function. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 31: ODCC dummy snapshot compare flow diagram



### Override config param data set

Table 53 : Override config param data set function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_override\_data\_set | | |
| Syntax: | static void boot\_fst\_override\_data\_set(  const struct boot\_fst\_write\_config\_data\_cmd\_s \*cmd\_prt,  struct boot\_fst\_write\_config\_data\_resp\_s \*resp\_ptr  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*cmd\_prt | struct boot\_fst\_write\_config\_data\_cmd\_s | Pointer to override config param message. |
|  | \*resp\_ptr | struct boot\_fst\_write\_config\_data\_resp\_s | Pointer to override config param response message. |
| Parameters (out): | \*resp\_ptr | struct boot\_fst\_write\_config\_data\_resp\_s | Pointer to override config param response message. |
| Return value: | void | |  |
| Description: | This API sets the configuration data and update the payload in response message. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  If the override parameter is not equal to 0xFFFFFFFFU for any of the payload member of boot\_fst\_write\_config\_data\_cmd\_s, calls corresponding global config set API. The return value of the set API is updated in corresponding response message members. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 32 : Override config data set



### Override config param handle

Table 54 Override config param handle function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_override\_config\_param | | |
| Syntax: | static uint32\_t boot\_fst\_override\_config\_param(void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful sending response. |
| STATUS\_THREADX(M\_FST\_BOOT, E\_IN\_CONFIG\_PARSING, tx\_status) | | For queue receive failure. |
|  | Return value of isi\_memcpy\_s API or tl\_send API. | |  |
| Description: | This API process the override config data and sets it using glocal config APIs. And form the respons packet and responds. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful processing, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Calls tx\_queue\_receive() API to receive override config data.  In response packet sets the default value for payload fields.  Calls boot\_fst\_override\_data\_set() API to set the data.  Updates the response code payload fields updated in boot\_fst\_override\_data\_set() API.  Capies the header data from received message to response message.  Calls tl\_send() API to send the response to override config message. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 33 : Override config param function flow



### STL results packet validation

Table 55 : STL results packet validation function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_stl\_results\_pkt\_validate | | |
| Syntax: | static uint16\_t boot\_fst\_stl\_results\_pkt\_validate(  const struct boot\_fst\_stl\_res\_msg \*result\_msg  ) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*result\_msg | struct boot\_fst\_stl\_res\_msg | Pointer to STL results message. |
| Parameters (out): | void |  |  |
| Return value: | BOOT\_FST\_PCIE\_SUCCESS | | On successful sending response. |
| BOOT\_FST\_PCIE\_INVALID\_LENGTH | | For Invalid length |
| BOOT\_FST\_PCIE\_UNKNOWN\_SYS\_TIME\_VALID | | For unknown system time valid field value.. |
| BOOT\_FST\_PCIE\_UNKNOWN\_STL\_RES\_VALID | | For unknown STL results valid field value. |
| BOOT\_FST\_PCIE\_UNKNOWN\_STL\_RES | | For unknown STL results value. |
| Description: | This API validates the STL results message received. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful validation, else it will return response code. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Validates length of message received. If not matching updated response code.  Validates system time validity bits, if invalid then updates response code.  Validates STL results validity bits, if invalid then updates response code.  Validates STL results bits, if unknown then updates response code.  If STL results received is valid and results is FAIL then update diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 34 : STL results packet validation function flow



### Uncore STL execution check

Table 56 : Uncore STL execution check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_uncore\_stl\_exec\_check | | |
| Syntax: | static uint16\_t boot\_fst\_uncore\_stl\_exec\_check(  uint8\_t stl\_core\_mask) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | stl\_core\_mask | uint8\_t | Uncore mask received. |
| Parameters (out): | void |  |  |
| Return value: | BOOT\_FST\_PCIE\_SUCCESS | | On successful checking the uncore mask. |
| BOOT\_FST\_PCIE\_UNKNOWN\_UNCORE\_MASK | | If core mask is unknown. |
| Description: | This API validates the uncore mask value received and checks if STL executed or not. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful checking uncore mask, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  If STL is executed on uncore If executed update response code to success and increment boot\_fst\_num\_stl\_res\_received variable by 1.  Else if STL is not executed then update the response code to success.  Else uncore mask value is unknown so update the response code with BOOT\_FST\_PCIE\_UNKNOWN\_UNCORE\_MASK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 35 : Uncore STL execution check function flow



### Core STL execution check

Table 57 : Core STL execution check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: |  | | |
| Syntax: | static uint16\_t boot\_fst\_stl\_exec\_check(  uint32\_t stl\_core\_mask) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | stl\_core\_mask | Uint32\_t | core mask received. |
| Parameters (out): | void |  |  |
| Return value: | BOOT\_FST\_PCIE\_SUCCESS | | On successful checking the core mask. |
| BOOT\_FST\_PCIE\_UNKNOWN\_CORE\_MASK | | If core mask is unknown. |
| Description: | This API checks the execution status of core mask. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful checking execution status, else it will return response code. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  In a for loop of 16 the below operations are executed.  Fetch the 2 bit core mask from stl\_core\_mask using the index.  If core mask is unknown then update the response code to BOOT\_FST\_PCIE\_UNKNOWN\_CORE\_MASK.  Else if core mask STL status is executed then increment the boot\_fst\_num\_stl\_res\_received by 1.  Else update the response code with success. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 36 : Core STL execution check function flow



### System time set

Table 58 : System time set function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_system\_time\_set | | |
| Syntax: | static uint32\_t  boot\_fst\_system\_time\_set(  const struct boot\_fst\_stl\_res\_msg \*result\_msg) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*result\_msg | struct boot\_fst\_stl\_res\_msg | Pointer to STL results message. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful setting system time. |
| Return value of rtos\_timer\_to\_sec\_get | |  |
| Description: | This API sets the system time received in STL results mesasage and also sets the RTC time in seconds. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  In STL message if the system time is valid then calls gc\_system\_time\_set() to set the time.  Calls rtos\_timer\_to\_sec\_get() to get the RTOS time in seconds and calls gc\_rtc\_time\_set () to set RTC time.  If system time is invalid the calls gc\_system\_time\_set() and gc\_rtc\_time\_set() APIs to set time to zero. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 37 : Syatem time set function flow



### STL results respond

Table 59 : STL results respond function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_stl\_results\_respond | | |
| Syntax: | static void boot\_fst\_stl\_results\_respond(  struct boot\_fst\_pcie\_res\_s \*stl\_res,  uint16\_t response\_code) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \* stl\_res | struct boot\_fst\_stl\_res\_msg | Pointer to STL results message. |
|  | response\_code | Uint32\_t | Response code. |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Void. | | |
| Description: | This API respond to STL results message by updating the response message fields. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will call fw\_internal\_error\_handler() API to assrt NOK. | | |
| Implementation comments | This function performs the following operations.  Update the response message fields.  Calls the tl\_respond() API to respond to STL results message. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### Host startup STL results monitor

Table 60 : Host startup STL results monitor function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | boot\_fst\_stl\_results\_monitor | | |
| Syntax: | Static uint32\_t boot\_fst\_stl\_results\_monitor (uint16\_t \*hbc\_response\_code) | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*hbc\_response\_code | Uint16\_t | Pointer to response code. |
| Parameters (out): | \*hbc\_response\_code | Uint16\_t | Pointer to response code. |
| Return value: | STATUS\_SUCCESS | | On successful host boot STLs monitor. |
| STATUS\_FIRMWARE | | On host boot STLs monitor failure. |
| Description: | This API performs the host boot STLs monitor. Making sure that the Host Boot STLs are run and they pass. Disassociate startup STL results from POSC Core count. If STL results is PASS then check if STL executed on at least on one core/uncore, if not assert NOK. All STL results are received only in one host boot complete message.. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful host boot STLs monitor, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | On receiving STL results(Host boot complete) message, boot\_fst\_stl\_results\_cb is called. It calls the tx\_queue\_send(boot\_fst\_queue) and set the event flag to notify boot FST thread waiting for message.  This function performs the following operations.  Calls tx\_queue\_receive() to receive the STL results message.  Calls boot\_fst\_stl\_results\_pkt\_validate() to validate packet fields.  Calls boot\_fst\_uncore\_stl\_exec\_check() to check uncore execution status.  Calls boot\_fst\_stl\_exec\_check() to check core execution status for core mask 1.  Calls boot\_fst\_stl\_exec\_check() to check core execution status for core mask 2.  As part of STL results message atleast one core/uncore mask must be set else it’s a NOK condition. If boot\_fst\_num\_stl\_res\_received is equal to zero then update the response code with BOOT\_FST\_STL\_NOT\_EXE\_ON\_ALL\_CORES. Also update the status and diagnostic data to assert NOK.  Calls boot\_fst\_system\_time\_set() API to set system time.  Update the response packet and calls tl\_respond() to send the Acknowledgement to host with response code. If any error in ack then host will send the STL results message again.  If any error in above steps then calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | 971269, 971270, 850143, 850157, 850153 | | |

The sequence diagram for Host boot STLs run and pass test is shown below.

Figure 38: Host boot STLs monitor flow diagram



## Imported functions

### GP timer expiry callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_gp\_timer\_cb | | |
| Syntax: | static void fst\_common\_gp\_timer\_cb(  timer\_id\_e id  ) | | |
| File: | Common\_checks.c | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | id | timer\_id\_e | Timer id. |
| Parameters (out): | void |  |  |
| Return value: | void | |  |
| Description: | This API is called on GP timer expiry. | | |
| Preconditions: | This API must be registered while creating the GP timers. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | On GP timer expiry, it sets the FST\_COMMON\_TIMER\_EXPIRED event flag. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 39: GP Timer expiry callback function flow



### Periodic timer expiry callback function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_prd\_timer\_cb | | |
| Syntax: | static void fst\_common\_periodic\_timer\_cb(  void  ) | | |
| File: | Common\_checks.c | | |
| Sync/Async: | Asynchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | void | |  |
| Description: | This API is called on periodic timer expiry. | | |
| Preconditions: | This API must be registered while configuring the periodic timers. | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | On Periodic timer expiry, it sets the FST\_COMMON\_PERIODIC\_TIMER\_EXPIRED event flag. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 40 : Periodic Timer expiry callback function flow



### RTOS timer error handler

Table 61 : RTOS timer error handler function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_rtos\_error\_handler | | |
| Syntax: | static uint32\_t fst\_common\_rtos\_error\_handler(uint8\_t event\_data1,  uint8\_t event\_data2,  uint8\_t event\_data3,  rtos\_timer\_counter\_t \*rtos\_test\_value1,  rtos\_timer\_counter\_t \*rtos\_test\_value2,  rtos\_timer\_counter\_t \*rtos\_test\_value3) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | event\_data1 | uint8\_t | Event data 1. |
|  | event\_data2 | uint8\_t | Event data 2. |
|  | event\_data3 | uint8\_t | Event data 3. |
|  | \*rtos\_test\_value1 | rtos\_timer\_counter\_t | RTOS timer values. |
|  | \*rtos\_test\_value2 | rtos\_timer\_counter\_t | RTOS timer values. |
|  | \*rtos\_test\_value3 | rtos\_timer\_counter\_t | RTOS timer values. |
| Parameters (out): | void |  |  |
| Return value: | Return value of errmgt\_firmware\_internal\_error() API. | | If any error while asserting NOK. |
| Description: | This API updates the diagnostic data and asserts NOK for RTOS timer error. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Update the diagnostic data using input parameters.  Calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### Timer Error Injection(EI) test

Table 62 : Timer error injection check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_rtos\_timer\_check | | |
| Syntax: | uint32\_t fst\_common\_timer\_ei\_check (void) | | |
| File: | boot\_fst\_timer\_tests.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful timer test. |
| STATUS\_FIRMWARE | | On timer test failure. |
| Description: | This API performs the timer error injection test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful RTOS timer test, else it will return errors for general errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls fmm\_alarm\_mask\_unmask() to mask RTOS timer alarm.  Calls rtos\_timer\_osdl\_ei\_counters\_load() function to load the error injection counters with different values.  Calls rtos\_timer\_osdl\_ei\_enable() function to enable the error injection. Soon after the enabling the error injection, free running counter NOK is set.  Calls the rtos\_timer\_osdl\_frc\_nok\_sts\_get() function to get the NOK status. If NOK is set, timer test is pass, else calls fst\_common\_rtos\_error\_handler() API to assert NOK.  Calls fmm\_alarm\_premask\_status\_get() to get premask status. If status is 1 then calls fst\_common\_rtos\_error\_handler() API to assert NOK.  Calls rtos\_timer\_osdl\_ei\_disable() function to disable error injection.  Calls rtos\_timer\_osdl\_frc\_reset() function to reset the free running counter. This will clear the NOK status.  Calls fmm\_alarm\_premask\_status\_get() to get the pre mask status. If status is not 1 then calls fst\_common\_rtos\_error\_handler() API to assert NOK.  Calls fmm\_alarm\_mask\_unmask() to unmask RTOS timer alarm. | | |
| Design Decisions |  | | |
| SAS traceability | 850142, 850153 | | |

Figure 41: RTOS timer test flow diagram



### WDT alarm mask and wdt start

Table 63 : WDT alarm mask and wdt start function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_alarm\_mask\_and\_start | | |
| Syntax: | static uint32\_t fst\_common\_wdt\_alarm\_mask\_and\_start(void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
|  |  |  |  |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On success. |
| Return value of fmm\_alarm\_mask\_unmask() or wdt\_osdl\_configure() or wdt\_osdl\_start() APIs. | |  |
| Description: | This API mask the WDT timeout alarm, configures the WDt and starts WDT timer. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Calls fmm\_alarm\_mask\_unmask() API to mask WDT alarm.  Calls wdt\_osdl\_configure() API to configure WDT to FST\_COMMON\_WDT\_TIMEOUT\_TICKS.  Calls wdt\_osdl\_start() to start WDT timer. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 42 : WDT alarm mask and wdt start function



### WDT configure and start

Table 64 : WDT configure and start function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_config\_start | | |
| Syntax: | static uint32\_t fst\_common\_wdt\_config\_start(void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On success. |
| Return value of fmm\_alarm\_premask\_status\_get() or wdt\_osdl\_configure() or errmgt\_firmware\_internal\_error() APIs. | |  |
| Description: | This API | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful respond, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This function performs the following operations.  Calls fmm\_alarm\_premask\_status\_get() API to get premask status.  If pre-mask status is not equal to 1 then wdt\_osdl\_configure(WDT\_DEFAULT\_VALUE) and calls wdt\_osdl\_start() to configure and start WDT.  If pre-mask status is equal to 1 then updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 43 : WDT configure and start function flow



### WDT difference error handle

Table 65 : WDT difference error handle function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_diff\_error\_handler | | |
| Syntax: | static uint32\_t fst\_common\_wdt\_diff\_error\_handler(uint64\_t rtos\_timer\_diff) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | rtos\_timer\_diff | Uint64\_t | RTOS timer difference value. |
|  |  |  |  |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Return value of errmgt\_firmware\_internal\_error(). | |  |
|  | |  |
| Description: | This API asserts nok for WDT if mismatch in RTOS timer difference value. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  Updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### WDT error handler

Table 66 : WDT error handler function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_error\_handler | | |
| Syntax: | static uint32\_t fst\_common\_wdt\_error\_handler(void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Return value of errmgt\_firmware\_internal\_error() API. | |  |
|  | |  |
| Description: | This API asserts NOK if premask status is not equal to 1. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  Updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### WDT NOK status error handle

Table 67 : WDT NOK status error handle function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_status\_nok\_error | | |
| Syntax: | static uint32\_t fst\_common\_wdt\_status\_nok\_error(void) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Return value of errmgt\_firmware\_internal\_error() API. | |  |
|  | |  |
| Description: | This API asserts NOK if WDT NOK is not set after error injection. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  Updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

### WDT test

Table 68 : WDT check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_wdt\_check | | |
| Syntax: | uint32\_t fst\_common\_wdt\_check (void) | | |
| File: | boot\_fst\_timer\_tests.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful timer test. |
| STATUS\_FIRMWARE | | On timer test failure. |
| Description: | This API performs the WD timer test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful WDT test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls fst\_common\_wdt\_alarm\_mask\_and\_start() to mask alart and start WDT.  Calls rtos\_timer\_osdl\_twall\_clock\_get() to get RTOS time stamp.  In do-while loop with retry count wait for WDT expiry, Calls wdt\_osdl\_nok\_status\_get() function to get the WDT NOK status.  If NOK is set, timer test is pass, else report as test failure by calling fst\_common\_wdt\_status\_nok\_error.  Calls rtos\_timer\_osdl\_twall\_clock\_get () to get RTOS time stamp.  Calculate the RTOS time stamp difference if not matching then calls fst\_common\_wdt\_diff\_error\_handler() to assert NOK.  Calls fst\_common\_wdt\_config\_start() to configure WDT and start it.  Calls fmm\_alarm\_premask\_status\_get() to get premask status. If status is not equal to 1 then calls fst\_common\_wdt\_error\_handler() to assert NOK.  Calls fmm\_alarm\_mask\_unmask() to unmask WDT timeout alarm. | | |
| Design Decisions |  | | |
| SAS traceability | 850142, 850153 | | |

Figure 44: WDT test flow diagram



### GP timer difference calculation

Table 69 : GP timer difference calculation function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_gp\_timer\_diff\_cal | | |
| Syntax: | static uint32\_t fst\_common\_gp\_timer\_diff\_cal(  rtos\_timer\_val\_t \*ts\_start,  timer\_id\_e id) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*ts\_start | rtos\_timer\_val\_t | Pointer to start RTOS time stamp. |
|  | id | timer\_id\_e | Timer id |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Return value of rtos\_timer\_osdl\_twall\_clock\_get() or errmgt\_firmware\_internal\_error() APIs. | |  |
|  | |  |
| Description: | This API calculates the RTOS time stamp difference for GP timer and asserts nok if mismatch in RTOS timer difference value. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  Calls rtos\_timer\_osdl\_twall\_clock\_get() to get stop RTOS time stamp.  Calculate the RTOS time stamp difference and convert to micro seconds and check the difference value with configured and tolerance value.  If mismatch in RTOS time difference, updates the diagnostic data and calls errmgt\_firmware\_internal\_error() API. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 45 : GP timer difference calculation function flow



### GP timer test

Table 70 : GP timer check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_gp\_timer\_check | | |
| Syntax: | uint32\_t fst\_common\_gp\_timer\_check (void) | | |
| File: | boot\_fst\_timer\_tests.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful timer test. |
| STATUS\_FIRMWARE | | On timer test failure. |
| Description: | This API performs the GP timer test. Test uses, FST timer library APIs exposed by timer service (uses GP timer driver APIs). Refer timer service, RTOS and GP timer driver MDS for details. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful GP timer test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | To test all instances of GP timer, create and start AX\_GP\_TIMER\_HW\_NUM number of timers in a for loop. Do not delete the timer until all the timer instances are tested. Once all instances are tested then stop and delete all the used timers.  This API performs the following operations  Create event flag to get notification of GP timer expiry interrupt.  In a for loop:  Updates fst\_timer\_t structure variable with timeout value, reload type and fst\_common\_gp\_timer\_cb callback.  Calls fst\_lib\_timer\_create() function to create timer. This API internally updates the timer id in fst\_timer\_t structure variable. Use the timer id to start, stop and delete the timer.  Calls the fst\_lib\_timer\_start(id) function to start the timer.  Calls rtos\_timer\_osdl\_twall\_clock\_get(pointer to start time stamp stucture) function to get the RTOS timer time stamp.  Calls tx\_event\_flags\_get() function to wait (do not wait forever) for FST\_COMMON\_TIMER\_EXPIRED event flag to be set. The event flag is set in timer expiry callback function registered above.  If the FST\_COMMON\_TIMER\_EXPIRED event flag is set successfully within the time.  Calls fst\_common\_gp\_timer\_diff\_cal() to calculate RTOS timestamp difference and assert NOK if any mismatch.  In a for loop, calls fst\_lib\_timer\_stop(id) and fst\_lib\_timer\_delete(id) functions to stop and delete the timer used above.  Calls tx\_event\_flags\_delete() function to delete the event flag created in first step.  If any error in above steps return error. | | |
| Design Decisions |  | | |
| SAS traceability | 850142, 850153 | | |

The sequence diagram for GP timer test is shown below.

Figure 46: GP timer test flow diagram



### Periodic timer difference calculation

Table 71 : Periodic timer difference calculation function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_prd\_timer\_diff\_cal | | |
| Syntax: | static uint32\_t fst\_common\_prd\_timer\_diff\_cal(rtos\_timer\_val\_t \*ts\_start,  rtos\_timer\_val\_t \*ts\_stop) | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*ts\_start | rtos\_timer\_val\_t | Pointer to start time stamp. |
|  | \*ts\_stop | rtos\_timer\_val\_t | Pointer to stop time stamp. |
|  |  |  |  |
| Parameters (out): | void |  |  |
| Return value: | Return value of errmgt\_firmware\_internal\_error() API. | |  |
|  | |  |
| Description: | This API calculates the RTOS time stamp difference and assert NOK if any mismatch. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This function performs the following operations.  Calculates the RTOS timestamp difference using input parameters.  Convert the difference time to micro seconds.  If any mismatch in RTOS time stamp difference then update the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | Private Function | | |

Figure 47 : Periodic timer difference calculation function flow



### Periodic timer test

Table 72 : Periodic timer check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_prd\_timer\_check | | |
| Syntax: | uint32\_t fst\_common\_periodic\_timer\_check (void) | | |
| File: | boot\_fst\_timer\_tests.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On successful timer test. |
| STATUS\_FIRMWARE | | On timer test failure. |
| Description: | This API performs the periodic timer test. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful periodic timer test, else it will return errors. The upper layer will have to take care of error returned. | | |
| Implementation comments | This API performs the following operations.  Calls tx\_event\_flags\_create() to create fst\_common\_event\_flags.  Updates periodic\_timer\_config\_s structure variable with timeout value and periodic timer expiry callback function.  Calls periodic\_timer\_osdl\_configure() function to configure timeout and callback function.  Calls the periodic\_timer\_osdl\_start() function to start the periodic timer.  Calls rtos\_timer\_osdl\_twall\_clock\_get(pointer to start time stamp stucture) function to get the RTOS timer time stamp.  Calls tx\_event\_flags\_get() function to wait (do not wait forever) for FST\_COMMON\_PERIODIC\_TIMER\_EXPIRED event flag to be set. The event flag is set in timer expiry callback function registered above.  If the FST\_COMMON\_PERIODIC\_TIMER\_EXPIRED event flag is set successfully within the time then calls rtos\_timer\_osdl\_twall\_clock\_get(pointer to stop time stamp) function to get the RTOS timer time stamp.  Calls fst\_common\_prd\_timer\_diff\_cal(&ts\_start, &ts\_stop) to handle time stamp.  Calls periodic\_timer\_osdl\_stop() to stop periodic timer.  Calls tx\_event\_flags\_delete() to delete event flags.  If any error in above steps return error. | | |
| Design Decisions |  | | |
| SAS traceability | 850142, 850153 | | |

Figure 48: Periodic timer test flow diagram



### FMM Scratchpad register test

Table 73 : FMM scratchpad register check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_fmm\_sp\_reg\_check | | |
| Syntax: | uint32\_t fst\_common\_fmm\_sp\_reg\_check (void) | | |
| File: | boot\_fst\_timer\_tests.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | On FMM scratchpad register check successful. |
| STATUS\_FIRMWARE | | On FMM scratchpad register check failure. |
| Description: | This API performs the FMM scratchpad register check. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful FMM scratchpad register check, else it will call error handling API to assert NOK. | | |
| Implementation comments | It performs the following operations.   1. Call the fmm\_scratchpad\_reg\_rw () function to write 0xA5A5A5A5U to FMM scratchpad register. The function returns the read back value from scratchpad register, store it in read\_data\_1. 2. Call the fmm\_scratchpad\_reg\_rw () function to write 0x5A5A5A5AU to FMM scratchpad register. The function returns the read back value from scratchpad register, store it in read\_data\_2. 3. Compare write and read data return success if matching. If mismatch the update the diagnostic data and calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | 850139, 850153 | | |

The sequence diagram for FMM scratchpad register test is shown below.

Figure 49: FMM scratchpad register check flow diagram



### Multiplier and Divider check

Table 74 : Multiplier and Divider check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_mul\_div\_check | | |
| Syntax: | uint32\_t fst\_common\_mul\_div\_check (void); | | |
| File: | fst\_common\_checks.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Multiplier and divider check successful. |
| STATUS\_FIRMWARE | | Multiplier and divider check failed. |
| Description: | This API implements the multiplier and divider check functionality. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful multiplier and divider check, else it will call error handling API to assert NOK. | | |
| Implementation comments | It performs the following operations.   1. Multiplication in first “for loop”: 2. Perform the multiplication of input vector 1 and input vector 2 index elements. The result is 32 bit truncated value. 3. For each vector index, multiplication should be performed three times. First time, multiply positive numbers, second and third time, negating one of the two input vectors. 4. Compare the result with corresponding expected output vector index element. If mismatch in comparision, break the loop and return error to calling function. 5. Division in second “for loop”: 6. Perform the division of input vector 1 and input vector 2 index elements (input vectors are different for division). The result is quotient of signed division. 7. For each vector index, division should be performed three times. First time, divide positive numbers, second and third time, negating one of the two input vectors. 8. Compare the result with corresponding expected output vector index element. If mismatch in comparision, break the loop and return error to calling function. 9. Multiply and accumulate in third “for loop”: 10. Perform unsigned long long multiply and accumulate. 11. For each vector index, each number from first multiplication vector and first element of the second multiplication vector are multiplied and accumulated. 12. Compare the multiply and accumulate result with expected result. If mismatch in comparision, calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | 850153 | | |

Figure 50: frCPU Multipler and Divider check flow



### Bus Matrix check

Table 75 : Bus Matrix check function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_bus\_matrix\_check | | |
| Syntax: | uint32\_t fst\_common\_bus\_matrix\_check (uint8\_t \*mem\_loc); | | |
| File: | fst\_common\_checks.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | \*mem\_loc | uint8\_t | Pointer to single port or dual port memory location. |
| Parameters (out): | void |  |  |
| Return value: | STATUS\_SUCCESS | | Bus matrix check successful. |
| STATUS\_FIRMWARE | | Bus matrix check failed. |
| Description: | This API implements the bus matrix check functionality. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: | The function will return success on successful bus matrix check, else it will call error handling API to assert NOK. | | |
| Implementation comments | Note: Memory location may change.  The memory locations of global variable is used for single port memory regions.  The memory locations used for dual port memory regions are:   * mem\_loc\_0 = 0x8FFF8U * mem\_loc\_1 = 0x8FFF9U * mem\_loc\_2 = 0x8FFFAU * mem\_loc\_3 = 0x8FFFBU * mem\_loc\_4 = 0x8FFFCU   This API performs the following operations.   1. Word write-read-compare operation. 2. Write 0xAAAAAAAA to SRAM location mem\_loc\_0 and write 0x55555555 to SRAM location mem\_loc\_4. 3. Read back the SRAM location mem\_loc\_0 and mem\_loc\_4, compare with the write data. If mismatch in comparision, return error to calling function. 4. Aligned and unaligned accesses of variouse data size to fixed SRAM locations. 5. Two aligned word write to consecutive SRAM locations.    * Write 0x12345678U to SRAM location mem\_loc\_0 and mem\_loc\_4.    * Read three unaligned word from SRAM location mem\_loc\_1, mem\_loc\_2 and mem\_loc\_3. Compare with expected results, if mismatch, return error to calling function.    * Read two unaligned half-word from SRAM location mem\_loc\_1 and mem\_loc\_3. Compare with expected results, if mismatch, return error to calling function. 6. Three unaligned word write to SRAM location.  * Write 0x12345678U to SRAM locations mem\_loc\_1, mem\_loc\_2 and mem\_loc\_3. * Read three bytes from SRAM locations mem\_loc\_1, mem\_loc\_2 and mem\_loc\_3 respectively. Compare with expected results, if mismatch, return error to calling function.  1. Two unaligned half-word write to SRAM.  * Write half-word 0x1234U to SRAM location mem\_loc\_1 and mem\_loc\_3.  1. Two byte write to SRAM location.  * Write byte 0x12U to SRAM location mem\_loc\_1 and mem\_loc\_2.  1. Aligned word read and compare.  * Read words from SRAM location mem\_loc\_0 and mem\_loc\_4. Compare with expected results, if mismatch, return error to calling function.   If any error in above steps calls errmgt\_firmware\_internal\_error() API to assert NOK. | | |
| Design Decisions |  | | |
| SAS traceability | 848945, 850168, 850153 | | |

Figure 51: frCPU bus matrix check flow



### MPU and NVIC registers read

Table 76 : frCPU MPU and NVIC register read function

|  |  |  |  |
| --- | --- | --- | --- |
| Service name: | fst\_common\_mpu\_nvic\_reg\_read | | |
| Syntax: | void fst\_common\_mpu\_nvic\_reg\_read (void); | | |
| File: | fst\_common\_checks.c | | |
| Sync/Async: | Synchronous | | |
| Reentrancy: | Reentrant | | |
| Parameters (in): | void |  |  |
| Parameters (out): | void |  |  |
| Return value: | void | |  |
| Description: | This API reads the MPU and NVIC registers. | | |
| Preconditions: |  | | |
| Timing Constraints: |  | | |
| Caveats: |  | | |
| Configuration: |  | | |
| Error Handling: |  | | |
| Implementation comments | This API performs the following operations.   1. Reads the following registers.   MPU registers to read:   * MPU Type Register, address 0xE000ED90U * MPU Control Register, address 0xE000ED94U * MPU Region Base Address register, address 0xE000ED9CU   NVIC registers to read:   * Configuration Control Register, address 0xE000ED14U * System Handler Control and State Register, address 0xE000ED24U * System Handler Priority Registers 1, address 0xE000ED18U * System Handler Priority Registers 2 lower halfword, address 0xE000ED20U * Debug Halting Control and Status Register, address 0xE000EDF0U * Debug Exception and Monitor Control Register, address 0xE000EDFCU * Vector Table Offset Register, address 0xE000ED08U * Application Interrupt and Reset Control Register, address 0xE000ED0CU | | |
| Design Decisions |  | | |
| SAS traceability | 850153 | | |

Figure 52: frCPU MPU and NVIC register read function flow



## Imported Types

|  |  |  |
| --- | --- | --- |
| Syntax: | struct boot\_fst\_config\_s | |
| Type: | struct | |
| File: | boot\_fst.h | |
| Range: |  | |
| Description: | Boot FST configuration parameters. | |
| Members: | uint32\_t bios\_stage2\_timer\_val | Bios stage 2 timer time value. |

Table 77 : ThreadX errors

|  |  |
| --- | --- |
| Byte allocate errors | TX\_SUCCESS, TX\_DELETED, TX\_NO\_MEMORY, TX\_WAIT\_ABORTED, TX\_POOL\_ERROR, TX\_PTR\_ERROR, TX\_SIZE\_ERROR |
| Mutex create/get/put errors | TX\_SUCCESS, TX\_MUTEX\_ERROR, TX\_CALLER\_ERROR, TX\_INHERIT\_ERROR,TX\_DELETED, TX\_WAIT\_ERROR, TX\_WAIT\_ABORTED,TX\_NOT\_AVAILABLE,TX\_NOT\_OWNED |
| Thread create errors | TX\_SUCCESS, TX\_THREAD\_ERROR, TX\_PTR\_ERROR, TX\_SIZE\_ERROR, TX\_PRIORITY\_ERROR, TX\_THRESH\_ERROR, TX\_START\_ERROR, TX\_CALLER\_ERROR |
| Queue create/receive | TX\_SUCCESS, TX\_QUEUE\_ERROR, TX\_PTR\_ERROR, TX\_SIZE\_ERROR, TX\_CALLER\_ERROR, TX\_DELETED, TX\_QUEUE\_EMPTY, TX\_WAIT\_ABORTED, TX\_WAIT\_ERROR |
| Event flags create/set/get/delete | TX\_SUCCESS, TX\_GROUP\_ERROR, TX\_CALLER\_ERROR, TX\_DELETED, TX\_NO\_EVENTS, TX\_WAIT\_ABORTED, TX\_GROUP\_ERROR, TX\_PTR\_ERROR, TX\_WAIT\_ERROR, TX\_OPTION\_ERROR. |

# Data Description

Table 78 : Global Data Description

| **Varialbles** | **Type** | **Description** |
| --- | --- | --- |
| boot\_fst\_bios\_boot\_msg | struct | PCIe MB message format for BIOS boot complete |
| boot\_fst\_posc\_res\_msg | struct | PCIe MB message format for POSC test results. |
| boot\_fst\_stl\_res\_msg | struct | PCIe MB message format for startup STL results. |
| boot\_fst\_pcie\_res\_s | struct | PCIe MB response message format |
| boot\_fst\_odcc\_ss\_msg | struct | PCIe MB message format for ODCC SnapShot(SS) |
| boot\_fst\_odcc\_ss\_res | struct | PCIe MB response message format for ODCC SnapShot(SS). |
| boot\_fst\_punit\_response | struct | Response message from PUINT. |
| boot\_fst\_itd\_res\_data\_0 | struct | ITD response data 0. |
| boot\_fst\_itd\_res\_data\_1 | struct | ITD response data 1. |
| boot\_fst\_itd\_res\_data\_2 | struct | ITD response data 2. |
| boot\_stored\_parity | uint32\_t | NVIC interrupt priority register stored parity. |
| fst\_bm\_sp\_sram\_loc[8U] | Volatile uint8\_t | Single Port SRAM memory location |
| fst\_bm\_dp\_sram\_loc[8U] \_\_attribute\_\_((at(0x8FFF8U))) | volatile uint8\_t | Dual Port SRAM memory location. |

Table 79 : Macros Data Description

|  |  |  |
| --- | --- | --- |
| **Macros** | **Value** | **Discription** |
| BOOT\_FST\_SUCCESS\_RES\_CODE | 0x0U | Response code for success |
| BOOT\_FST\_UNSUP\_CMD\_RES\_CODE | 0x1U | PUINT response code for unsupported command. |
| BOOT\_FST\_PCIE\_SUCCESS | 0U | Boot fst pcie success |
| BOOT\_FST\_PCIE\_POSC\_RES\_SUCCESS | 0U | Boot fst pcie resonse success |
| BOOT\_FST\_PCIE\_INVALID\_LENGTH | 0x600BU | Boot fst invalid length. |
| BOOT\_FST\_PCIE\_UNKNOWN\_SYS\_TIME\_VALID | 0x6538U |  |
| BOOT\_FST\_PCIE\_UNKNOWN\_STL\_RES\_VALID | 0x656FU |  |
| BOOT\_FST\_PCIE\_UNKNOWN\_STL\_RES | 0x6576U |  |
| BOOT\_FST\_PCIE\_UNKNOWN\_CORE\_MASK | 0x6507U | Boot fst unknown core mask |
| BOOT\_FST\_PCIE\_UNKNOWN\_UNCORE\_MASK | 0x65A6U | Boot fst unknown uncore mask |
| BOOT\_FST\_PCIE\_STL\_NOT\_EXECUTED\_ON\_ALL\_CORES | 0x6C4EU | PCIE stl not executed on all cores. |
| BOOT\_FST\_PCIE\_STL\_NOT\_EXECUTED\_ON\_EXPECTED\_CORE\_UNCORE | 0x651EU | PCIE stl not executed on expected core and uncore. |
| BOOT\_FST\_BB\_UNKNOWN\_BDF\_FIELD | 0x6E7EU | Boot fst BDF mismatch. |
| BOOT\_FST\_ODCC\_COMPARE\_PASS | 0x55U | ODCC compare passed. |
| BOOT\_FST\_ODCC\_COMPARE\_FAILED | 0xAAU | ODCC compare failed. |
| BOOT\_FST\_ODCC\_FIRST\_SS\_RECEIVED | 0xA5U | ODCC first snap shot received. |
| BOOT\_FST\_UNCORE\_STL\_EXEC | 0x1U | Boot fst uncore stl execution. |
| BOOT\_FST\_UNCORE\_STL\_NOT\_EXEC | 0x2U | Boot fst uncore stl non execution. |
| BOOT\_FST\_CORE\_STL\_EXEC | 0x1U | Boot fst core stl execution. |
| BOOT\_FST\_CORE\_STL\_NOT\_EXEC | 0x2U | Boot fst core stl non execution. |
| BOOT\_FST\_CM1\_SW | 5868530U | Clock monitor 1 sample window |
| BOOT\_FST\_CM1\_TC | 5000U | Clock monitor 1 target count |
| BOOT\_FST\_CM1\_CE | 10U | Clock monitor 1 Count Error |
| BOOT\_FST\_CM2\_SW | 49U | Clock monitor 2 sample window |
| BOOT\_FST\_CM2\_TC | 254U | Clock monitor 2 target count |
| BOOT\_FST\_CM2\_CE | 30U | Clock monitor 2 Count Error |
| BOOT\_FST\_CM3\_SW | 130U | Clock monitor 3 sample window |
| BOOT\_FST\_CM3\_TC | 25U | Clock monitor 3 target count |
| BOOT\_FST\_CM3\_CE | 27U | Clock monitor 3 Count Error |
| NUM\_CLK\_MONITOR | 3U | Number of clock monitors. |
| BOOT\_FST\_CRC\_CHECK\_FAIL\_RES\_CODE | 0X2U | PUINT response code for CRC check failure. |
| BOOT\_FST\_TIMEOUT\_RES\_CODE | 0X3U | PUINT response code for timeout during execution. |
| BOOT\_FST\_NUM\_ARMCM3\_INT\_PRIO\_REG | 0X8U | Number of ARM CM3 NVIC interrupt priority registers. |
| BOOT\_FST\_NUM\_PRIO\_PER\_REG | 0x4U | Number of priority values per register. |
| BOOT\_FST\_WIDTH\_PRIO\_IN\_REG | 0x8U | Width of priority in register |
| BOOT\_FST\_PTI\_QUERY\_OPCODE | 0x0009U | Platform Transition Information query opcode. |
| BOOT\_FST\_PMC\_IPC\_TIMER\_TIMEOUT | 200000U | PMC IPC communication monitoring timer value. |
| BOOT\_FST\_BIOS\_BOOT\_COM\_CMD | 0x206U | Command for BIOS boot complete message. |
| BOOT\_FST\_POSC\_RES\_CMD | 0x218U | Command for POSC test results message. |
| BOOT\_FST\_HOST\_BOOT\_COM\_CMD | 0x21FU | Command for host boot complete |
| BOOT\_FST\_HOST\_ODCC\_SS\_RES\_CMD | 0x6A8U | Command for ODCC shapshot response. |
| BOOT\_FST\_BIOS\_BOOT\_REC\_LEN | 0x4U | BIOS boot received message length in Dwords. |
| BOOT\_FST\_BIOS\_BOOT\_RES\_LEN | 0x4U | BIOS boot response message length in Dwords. |
| BOOT\_FST\_POSC\_REC\_LEN | 0xFU | POSC results received message length in Dwords. |
| BOOT\_FST\_POSC\_RES\_LEN | 0x4U | POSC results response message length in Dwords. |
| BOOT\_FST\_ODCC\_REC\_LEN | 0x6U | ODCC snapshot received message length in Dwords. |
| BOOT\_FST\_ODCC\_RES\_LEN | 0x4U | ODCC snapshot response message length in Dwords. |
| BOOT\_FST\_STL\_REC\_LEN | 0x7U | STL results received message length in Dwords. |
| BOOT\_FST\_RES\_FLAG | 0X1U | Response flag value. |
| BOOT\_FST\_PCIE\_STL\_RES\_PASS | 0x55U | STL results pass value. |
| BOOT\_FST\_PCIE\_STL\_RES\_FAIL | 0xAAU | STL results fail value. |
| BOOT\_FST\_PCIE\_RES\_VALID | 0x55U | System time or STL results valid. |
| BOOT\_FST\_PCIE\_RES\_INVALID | 0xAAU | ystem time or STL results invalid. |
| BOOT\_FST\_CONST\_FFFF | 0xFFFFU | Macro for constant value. |
| BOOT\_FST\_BYTE\_PER\_DWORD | 0x4U | Number of bytes per Dword. |
| FST\_COMMON\_PRD\_TIME\_DIFF\_TOLERANCE | 30 | Periodic timer timestamp difference tolerance. |
| FST\_COMMON\_GP\_TIME\_DIFF\_TOLERANCE | 65 | GP timer timestamp difference tolerance. |
| FST\_COMMON\_WDT\_TIME\_DIFF\_TOLERANCE | 30 | Watchdog timer timestamp difference tolerance. |
| FST\_COMMON\_WDT\_TIME\_LOW\_TOLERANCE | 1 | Watchdog timer timestamp difference lower tolerance. |
| FST\_COMMON\_GP\_TIMER\_EXPIRED | 0x1 | GP timer expiry event flags value. |
| FST\_COMMON\_PRD\_TIMER\_EXPIRED | 0x2 | Periodic timer expiry event flags value. |
| FST\_COMMON\_GP\_TIMER\_TIMEOUT | 200 | GP timer timeout in micro seconds. |
| FST\_COMMON\_PRD\_TIMER\_TIMEOUT | 250 | Periodic timer timeout in micro seconds. |
| FST\_COMMON\_EVENT\_FLAG\_TIMEOUT | 2000 | Event flags timeout. |
| WDT\_DEFAULT\_VALUE | 0xFFFFFFFF | Default value of WDT. |
| FST\_COMMON\_WDT\_TIMEOUT\_TICKS | 20000 | WDT timeout in ticks. If F = 200Mhz, for 100usec timeout ticks are equal to 20,000. |
| FST\_COMMON\_WDT\_TIMEOUT\_MICRO\_SEC | 100 | WDT timeout in micro seconds.  Note: If ticks are changed then change this value. |
| FST\_COMMON\_WDT\_RETRY\_COUNT | 100000 | WDT error status get retry count. |